TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

Diplomarbeit

SMIng

Eine neue Sprache zur Definition von Management-Informationen

von

cand. inform. Frank Strauß
<strauss@ibr.cs.tu-bs.de>

Aufgabenstellung und Betreuung:
Prof. Dr. H. Langendörfer
Dr. J. Schönwälder

Braunschweig, den 10. Mai 1999

INSTITUT FÜR BETRIEBSSYSTEME UND RECHNERVERBUND
Prof. Dr. H. Langendörfer

Erklärung

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

Braunschweig, den 10. Mai 1999

Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Prof. Dr. H. Langendörfer · Prof. Dr. M. Zitterbart

Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig · Postf. 3329 · D-38023 Braunschweig

Bültenweg 74/75
D-38106 Braunschweig
Telefax: (0531) 391–5936
Telex: 95 25 26 tubsw

Prof. Dr. H. Langendörfer
Telefon: (0531) 391–3283

Braunschweig, den 11.11.1998

Aufgabenstellung für die Diplomarbeit

SMIng – Eine neue Sprache zur Definition von Management-Informationen

vergeben an

Herrn cand. inform. Frank Strauß

Im Internet wurde zusammen mit dem Simple Network Management Protocol (SNMP) eine Sprache zur Beschreibung von Management-Informationen (Structure of Management Information, SMI) entwickelt und standardisiert (RFC 1155, RFC 1212, RFC 1215). Schon nach kurzer Zeit wurden Mängel und Unklarheiten deutlich, die zu einer zweiten Version der SMI geführt haben (RFC 1902, 1903, 1904). Diese zweiter Version wird derzeit überarbeitet, um sie zu einem vollen Internet Standard zu erheben. Bei dieser Überarbeitung können allerdings keine grundsätzlichen Probleme der SMI behandelt werden, da die Kompatibilität zu existierenden MIB-Definitionen gewahrt bleiben muß.

Ziel dieser Diplomarbeit ist die Erarbeitung von Konzepten für eine alternative Definitionssprache (SMIng), die die bisher bekannten Probleme der SMI Version 2 löst und effizient zu implementieren und zu benutzen ist. Die neue Sprache soll soweit kompatibel zur SMI Version 2 sein, daß automatische Übersetzungen möglich sind. Die Entwurfsentscheidungen sollen systematisch und begründet getroffen werden. Dazu ist es erforderlich, sich mit den Konzepten ähnlicher Definitionssprachen wie z.B. GDMO, CIM oder IDL auseinanderzusetzen.

Im praktischen Teil dieser Diplomarbeit sollen Bibliotheken entstehen, die Spezifikationen in der SMI Version 2 und in der neuen Sprache SMIng lesen können und einem Anwendungsprogramm über eine zu definierende Schnittstelle zur Verfügung stellen. Mit Hilfe dieser Bibliotheken sollen Übersetzer realisiert werden, die SMIv2 $\rightarrow$ SMIng und SMIng $\rightarrow$ SMIv2 konvertieren können.

Laufzeit: 6 Monate

Aufgabenstellung und Betreuung: Prof. Dr. H. Langendörfer
Dr. J. Schönwälder

(Prof. Dr. H. Langendörfer)

Kurzfassung

Das systematische Management von Netzwerken beschäftigt verschiedene Standardisierungsgremien seit über zehn Jahren. Dabei enthalten alle Konzepte ein Informationsmodell, das der Strukturierung von Management-Informationen dient und eine Sprache umfaßt, in der formale Beschreibungen von Management-Objekten und -Ereignissen vorgenommen werden.

Im Internet-Management ist diese Sprache die *Structure of Management Information* (SMI). Sie liegt in zwei Versionen vor (SMIv1 und SMIv2), deren Anwendung weite Verbreitung erlangt hat. Dennoch weisen ihre Spezifikationen Defizite hinsichtlich Eindeutigkeit, Verständlichkeit, Vollständigkeit und Erweiterbarkeit auf. Infolgedessen existieren unzureichende Parser-Implementierungen und Beschreibungen von Management-Informationen.

Die vorliegende Arbeit enthält nach einem Vergleich der bekanntesten Informationsmodelle den systematischen Entwurf und die vollständige Spezifikation einer neuen Sprache zur Definition von Management-Informationen im Internet-Management: „SMIng“ beseitigt die bekannten Probleme unter Wahrung der semantischen Kompatibilität. Ferner wird eine Software-Bibliothek zum Zugriff auf Management-Informationen entworfen und implementiert. Mit Hilfe dieser Bibliothek werden Werkzeuge zur Prüfung und Übersetzung von SMIv2 zu SMIng und von SMIng zu SMIv2 entwickelt. Sie untermauern die erreichten Entwurfsziele und ermöglichen eine kosteneffiziente Weiterverwendung bestehender Beschreibungen von Management-Informationen.

Abstract

More than ten years various standardization bodies are concerned with the methodic management of computer networks. All concepts include an information model, that is used for the structuring of management information and contains a language that allows a formal description of managed objects and events.

In the area of internet management, this language is the *Structure of Management Information* (SMI), presently defined in two versions, SMIv1 and SMIv2. They are widely deployed, although their specifications suffer from some drawbacks concerning unambiguousness, clearness, integrity, and extensibility. As a consequence, there are insufficient definitions of management information and erroneous implementations of parsers.

This thesis presents the systematic design and the complete specification of a new language for the description of management information, based on the comparison of the most common information models. “SMIng” eliminates the known problems whilst remaining semantically compatible. Secondly, a software library for management information has been developed. Based on this library tools for checking descriptions and converting from SMIv2 to SMIng and from SMIng to SMIv2 prove the approach of SMIng and allow a cost-efficient re-usability of existing descriptions of management information.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Das Management vernetzter Systeme beschäftigt einige Standardisierungsgremien seit geraumer Zeit und mit zunehmender Bedeutung. Die Grundlagen für ein Management verschiedenster Komponenten sind Standards in mehreren Bereichen: Ein gemeinsam genutztes Management-Protokoll ist erforderlich, um die Kommunikation zwischen den beteiligten Management-Instanzen zu ermöglichen. Dieses Protokoll muß die für ein Management erforderlichen Operationen standardisiert übermitteln. Ferner sollte es den Ansprüchen verschiedener Sicherheitsaspekte genügen. Um aber überhaupt sinnvolle Management-Operationen formulieren, übermitteln und ausführen zu können, bedarf es eines Informationsmodells, das eine eindeutige Beschreibung von Management-Informationen erlaubt und allen beteiligten Entwicklern von Agenten und Management-Applikationen, den anwendenden Personen sowie den genutzten Programmen zur Verfügung steht. Ein wesentlicher Bestandteil eines Informationsmodells ist eine Sprache, in der Management-Informationen in MIB Modulen (*Management Information Base*) lesbar für Personen und Programme formuliert werden.

Im Bereich der Internet Protokollfamilie werden Standardisierungen in Arbeitsgruppen der *Internet Engineering Task Force* (IETF) erarbeitet. Hier wurden seit 1987 verschiedene Ansätze des Netzwerkmanagements verfolgt. Als Informationsmodell haben sich hier im Laufe der Jahre zwei Versionen der *Structure of Management Information* (SMI) etabliert.

1.1. Motivation

Die vorliegende Arbeit verfolgt zwei Kernziele, die Probleme der derzeitig genutzten SMI Versionen aufgreifen: den systematischen Entwurf einer neuen Sprache zur Definition von Management-Informationen und den Entwurf und die Implementierung einer Software-Bibliothek zum Zugriff auf Management-Informationen in MIB Modulen.

1.1.1. Eine neue Sprache zur Definition von Management-Informationen

Die SMI in den Versionen 1 und 2, kurz SMIv1/v2, weist einige Defizite auf, die zwar im Laufe der Zeit identifiziert wurden, aber zu einem großen Teil nicht ohne gravierende Kompatibilitätsprobleme bereinigt werden konnten. Entsprechende inkompatible Änderungen waren bislang aufgrund der weiten Verbreitung von MIB Modulen, die in diesen SMI Notationen erstellt wurden, und der Entwicklung im IETF Standardisierungsprozeß nicht akzeptabel.

Hauptziel dieser Arbeit ist daher die systematische Erarbeitung einer neuen Sprache zur Definition von Management-Informationen unter Berücksichtigung verschiedener Aspekte: Die 'neue SMI' — sie wurde für die Zeit ihrer Entwicklung SMIng getauft — soll dem derzeitigen Informationsmodell des Internet-Management gerecht werden. Eine Konvertierbarkeit von MIBs in der SMIv2 Notation nach SMIng und umgekehrt soll gewährleistet sein.

Ein weiteres Ziel ist die weitgehende Unabhängigkeit der Sprachspezifikation von externen Spezifikationen, die in der IETF nicht verfügbar sind. Dies ist eines der Probleme der SMIv1/v2, die auf der *Abstract Syntax Notation One* (ASN.1), einer für Netzwerkprotokolle häufig genutzten Notation für Syntax-Beschreibungen, basiert. Diese Abhängigkeit von einer veralteten und nicht mehr zugänglichen Spezifikation der ISO ist die Ursache für viele Unklarheiten bei der Auslegung der SMIv1/v2. Die Spezifikation der Syntax von SMIng soll hingegen sehr präzise und ohne externe Grundlagen erfolgen, um die eindeutige Implementierung von korrekten Parsern zu erleichtern. Als Notation für die SMIng Grammatik bietet sich daher die *Augmented Backus-Naur Form* (ABNF) an, die als *Standards Track* RFC (*Request For Comments*) durch die IETF publiziert ist und eine detaillierte Spezifikation bis auf Zeichenebene erlaubt.

Eine wichtige Hilfe für den Entwurf von SMIng sind die Kenntnisse über andere Informationsmodelle und ein Vergleich ihrer Eigenschaften. In diesem Sinne werden neben der SMIv1/v2 das Informationsmodell des OSI Managements *Guidelines for the Definition of Managed Objects* (GDMO), das *Common Information Model* (CIM) der *Desktop Management Task Force* (DMTF) und die *Interface Definition Language* (IDL) der *Object Management Group* (OMG) betrachtet. Aus den Erkenntnissen über diese Modelle werden anschließend Schlüsse für den Entwurf von SMIng gezogen.

1.1.2. Eine Bibliothek zum Zugriff auf Management-Informationen

Das zweite in dieser Arbeit verfolgte Kernziel ist der Entwurf und die Implementierung einer Software-Bibliothek für den Zugriff auf Management-Informationen. Dabei wird auf eine weitgehende Unabhängigkeit der zugrunde liegenden Spezifikationssprache Wert gelegt: Sowohl MIB Module im SMIv1/v2 Format als auch MIB Module im SMIng Format werden von der Bibliothek erfaßt und über eine gemeinsame Schnittstelle Applikationen wie Management-Stationen oder MIB-Compilern verfügbar gemacht.

Die Bibliothek soll modular erweiterbar sein. Neben den zwei im Rahmen dieser Arbeit zu entwickelnden Parsern für SMIv1/v2 und SMIng soll die Bibliothek durch weitere Zugriffsmethoden auf Management-Informationen ergänzt werden können. Um diesem Ziel gerecht zu werden, bietet sich eine zweischichtige Architektur an: Eine obere Schicht bildet sämtliche Management-Informationen auf Datenstrukturen ab und implementiert die Schnittstelle zu Applikationen. Eine untere Schicht besteht aus einem oder mehreren Treibern, die Management-Informationen aus Dateien oder über ein Netzwerk lesen und in den Datenstrukturen der oberen Schicht ablegen.

Die zu erstellenden Parser werden zugunsten der Effizienz und Fehlerfreiheit mit einem Parser-Generator entwickelt werden. Im Falle von SMIng ist dies anhand der o.g. ABNF Grammatik leicht möglich. Für den SMIv1/v2 Parser ist hingegen zunächst eine formale Grammatik anhand der SMIv1/v2 Spezifikationen zu erstellen. Basis hierfür sind ASN.1 Grundlagen, ASN.1 Konstrukte und Macros, sowie darüber hinausgehende informelle Beschreibungen.

Mit Hilfe der Bibliothek werden im Rahmen dieser Arbeit zwei Applikationen erstellt: zum einen sollen Module sowohl in SMIv1/v2 Notation als auch SMIng Module auf syntaktische und semantische

Korrektheit überprüft werden können. Zum anderen soll die im vorangegangenen Abschnitt erwähnte Konvertierbarkeit zwischen SMIv2 und SMIng durch das Erstellen entsprechender Compiler praktisch nachgewiesen werden.

1.2. Begrenzung

Das Problem der Konvertierbarkeit von SMIv1 Modulen wird nicht ausführlich betrachtet. Es findet bereits in den SMIv2 Spezifikationen kaum Beachtung, da SMIv1 von SMIv2 ohnehin abgelöst werden soll. Die Betrachtung der Konvertierung zwischen SMIv1 und SMIng erscheint daher wenig sinnvoll.

Die formale Beschreibung von Operationen auf Objekten wird in SMIng nicht behandelt. Eine umfassende Betrachtung hätte den Rahmen dieser Arbeit gesprengt. Dies sei als Anregung für künftige Arbeiten hier erwähnt.

Die Implementierung der Bibliothek ist als prototypisch anzusehen. Die angestrebten Ziele wurden erreicht. Für ein Publizieren sind jedoch noch verbesserte Fehlererkennungen der Parser und einige weitere Veränderungen angeraten.

1.3. Kernaussage

Die vorliegende Arbeit spezifiziert in sich abgeschlossen eine konsistente und effizient zu implementierende Sprache zur Definition von Management-Informationen, die das Potential zur Nachfolge der derzeitigen SMIv1/v2 aufweist. Daß ein Übergang praktikabel durchführbar ist, wird durch die Implementierung einer Bibliothek, die transparenten Zugriff auf beide Formate zuläßt, und durch Compiler, die zwischen beiden Sprachen übersetzen, untermauert.

1.4. Verwandte Arbeiten und Literatur

Da es einige Defizite in den SMIv2 Spezifikationen gab, wurde Mitte 1998 ein *SMIv2 Design Team* [1] mit der Überarbeitung dieser Spezifikationen betraut. 75 einzelne Punkte an den bis dahin aktuellen Spezifikationen wurden kritisiert und diskutiert. Zahlreiche Änderungen zur Verdeutlichung und auch einige Vereinfachungen wurden vorgenommen, ohne jedoch eine inkompatible Abweichung von den bisherigen Spezifikationen zu bewirken. Im Januar 1999 wurden die überarbeiteten Dokumente [2], [3], [4] von der *Internet Engineering Steering Group* (IESG), dem Rat der IETF Arbeitsbereiche, akzeptiert und im April 1999 schließlich als *Full Standard* publiziert.

Im Rahmen des *SMI Clarification Project* [5] wurde seit 1997 unter Führung von David T. Perkins an Dokumenten gearbeitet, die einige oft mißverstandende Aspekte der SMI klarstellen sollten. Leider kam es bislang nicht zu einer Integration dieser Klarstellungen in die SMI Spezifikationen.

Das vermutlich aufschlußreichste Buch über die SMIv1/v2 ist „Understanding SNMP MIBs“ von David T. Perkins und Evan McGinnis [6]. Es dokumentiert detailliert die verschiedenen SMI Spezifikationen und die Entwicklung von MIB Modulen. Dabei wird bewußt auf die Verwendung von ASN.1 Konstrukten verzichtet. Zahlreiche deutliche Hinweise auf problematische SMI Vorschriften erleichtern das Vermeiden von Fehlern beim Entwurf von Management-Informationen und waren im Rahmen dieser Arbeit eine große Hilfe bei der Entwicklung von SMIng.

1.5. Gliederung dieser Arbeit

Nachdem dieses Kapitel die Entwicklung einer neuen Sprache zur Definition von Management-Informationen motiviert, führt das folgende Kapitel einen Vergleich von vier Informationsmodellen an. Neben dem SMI Modell des Internet Management sind dies die bereits erwähnten Modelle OSI GDMO, DMTF CIM und CORBA IDL. Jedes Modell wird dabei in seiner Entwicklung historisch umrissen und anschließend in seinen wesentlichen Merkmalen und sprachlichen Konstrukten dokumentiert. Beispiele erläutern die praktische Anwendung der wichtigsten Konstrukte. Jeder der vier Abschnitte schließt mit einer kritischen Bewertung des Modells.

Kapitel 3 dokumentiert den Entwurf der neuen Sprache SMIng. Hierzu wurden vom Autor über 50 einzelne Punkte gesammelt, die Aspekte der SMIv2 kritisieren und konstruktive Änderungen vorschlagen oder positive Eigenschaften der in Kapitel 2 kennengelernten Modelle auf SMIng übertragen. Die Punkte, die im Entwurf von SMIng Einzug gehalten haben, werden in einzelnen Abschnitten dieses Kapitels diskutiert. Unter Berücksichtigung dieser Punkte wurde schließlich die Spezifikation „SMIng – A new Structure of Management Information“ (`draft-irtf-nmrg-sming-00`) geschrieben, die den inhaltlichen Kern dieser Arbeit darstellt. Diese Spezifikation wurde nach den Maßgaben der IETF für Internet Standard Dokumente verfaßt und wird voraussichtlich nach Abschluß dieser Diplomarbeit als Internet-Draft publiziert und evtl. im Rahmen der *Internet Research Task Force* (IRTF) Arbeitsgruppe für Netzwerkmanagement (*Network Management Research Group*, NMRG) weiterentwickelt. Da diese Spezifikation ein eigenes Dokument in einem eigenen Format darstellt, wurde sie in den Anhang E verlagert.

In Kapitel 4 wird der Entwurf der Software-Bibliothek `libsmi` dokumentiert, die Applikationen den Zugriff auf Management-Informationen ohne Kenntnis der zugrunde liegenden SMI Version ermöglicht. Nachdem die grundlegende zweischichtige Struktur der Bibliothek erörtert wurde, werden die entworfenen Datenstrukturen und die auf ihnen arbeitenden Funktionen im einzelnen beschrieben. Anschließend werden die mit Hilfe der `libsmi` entwickelten Applikationen `smilint` und `smidump` vorgestellt.

Das abschließende Kapitel 5 faßt die gewonnen Erkenntnisse zusammen und gibt einen Ausblick auf Arbeiten, die die Entwicklung sowohl der entworfenen Sprache SMIng im Internet Management als auch der Bibliothek `libsmi` im Umfeld existierender und künftiger Management-Applikationen vorantreiben könnten.

1.6. Danksagungen

An dieser Stelle möchte ich all denjenigen danken, die zur Entstehung dieser Arbeit beigetragen haben.

Allen voran gilt mein Dank Jürgen Schönwälder, der mich als Betreuer dieser Arbeit stets mit aktuellen Informationen, seien es handfeste Hinweise oder nette Anekdoten, versorgt hat. Ohne seine zahlreichen

Anregungen und manche interessante Diskussion über einzelne knifflige Probleme der SMI wäre diese Arbeit für mich nur halb so innovativ und spannend gewesen.

Dem Institut für Betriebssysteme und Rechnerverbund danke ich für die Bereitstellung und Pflege einer Umgebung, in der ein angenehmes Arbeiten möglich war und Spaß machte.

Für einzelne fachkundige Hinweise danke ich ferner

- Holger Dietrich, der mir als Mitentwickler der CIM MOF Grammatik einige Hinweise zu CIM gab, die MOF Grammatik (Anhang C) zur Verfügung gestellt hat und das Kapitel über CIM mit inhaltlicher Fachkunde gelesen hat,
- Scott Bradner und Deirdre Kostick, die mir die Umstände erläutert haben, die zur Ablehnung der WBEM-Vorschläge in der IETF führten,
- Neill Jones, der mir die LEX/YACC Quellen der OMG IDL Grammatik zur Verfügung gestellt hat,
- Jan Gänsler, der mir die CIM Meta- und Core-Schemata geschickt hat, sowie
- Marshall T. Rose, der mir mit seinen Arbeiten an einem XML-basierten Werkzeug zum Schreiben von Internet Standard Dokumenten die Entwicklung des SMIng Dokumentes sehr erleichtert hat.

Für das Korrekturlesen bedanke ich mich bei Erik Schönfelder und Jürgen Schönwälder sowie meinem Bruder Henry Strauß. Für die verbleibenden Fehler zeichne ich selbstverständlich allein verantwortlich.

Weiteren Freunden, die ich an dieser Stelle nicht im einzelnen aufführen möchte, danke ich dafür, daß es für mich in den zurückliegenden sechs Monaten stets auch andere Ziele und Ablenkungen gab.

2. Management Informationsmodelle

In diesem Kapitel werden vier Modelle zur formellen Beschreibung von Management-Informationen betrachtet. Dies ist zunächst die SMI (*Structure of Management Information*), wie sie derzeit im Rahmen von SNMP-basierten Managementlösungen eingesetzt wird. Der folgende Abschnitt behandelt GDMO, die formale Beschreibung von Management Informationen im OSI-basierten Netzwerk Management. Daran schließt sich ein Abschnitt über CIM (*Common Information Model*) an. Dies ist ein jüngerer Ansatz der DMTF (*Desktop Management Task Force*). Der letzte Abschnitt befaßt sich mit der OMG IDL, der *Interface Definition Language* im Rahmen der OMA (*Object Management Architecture*).

Jedes betrachtete Modell wird zunächst historisch knapp umrissen, um die jeweiligen Entwicklungsschritte besser verstehen und in Relation setzen zu können. Darauf folgen konzeptionelle Grundlagen der Modelle und Beschreibungen der jeweiligen syntaktischen Elemente. Jeder Abschnitt schließt mit einem Beispiel und einer Bewertung des Modells.

Ziel dieser Betrachtungen ist es, im folgenden Kapitel Schlüsse für den Entwurf einer weiterentwickelten SMI zu ziehen.

Die Beschreibungen dieser Modelle mitsamt ihrer sprachlichen Konstrukte wirken etwas langatmig. Sie dienen dem interessierten Leser jedoch, um die aufgeführten Beispiele deuten und selbst bewerten zu können. Für ein zügigeres Lesen kann man sich ggf. auf die abschließenden Bewertungen beschränken. Ist hingegen ein näheres Studium der Modelle gewünscht, so sind in allen Fällen Referenzen auf die zugrunde liegenden Spezifikationen und weiterführende Literatur angegeben.

2.1. SMIv2

2.1.1. Historische Entwicklung

Die Anfänge des Netzwerkmanagement, so wie es heute in Form des SNMP Modells besteht, gehen auf das Jahr 1987 zurück. Damals wurde erstmals der steigende Bedarf an standardisierten Management-Mechanismen für Netzwerkkomponenten erkannt. Einen Ansatz bildete der Entwurf des *High-Level Entity Management System* (HEMS, [7], [8], [9], [10]), der aber über das experimentelle Stadium nicht hinauskam. Neben der Beschreibung der Nachrichtenformate des *High-Level Entity Management Protocol* (HEMP) nutzte HEMS bereits auch für die Modellierung von Management-Informationen in der *HEMS Monitoring and Control Language* eine Syntax, die an die *Abstract Syntax Notation One* (ASN.1, [11]) angelehnt war [9].

Im selben Jahr erschien mit dem *Simple Gateway Monitoring Protocol* (SGMP, [12]) ein zweiter Entwurf eines Management-Modells. Hierbei wurde ASN.1 (damals noch als X.409 referenziert) lediglich zur Definition der Nachrichtenformate verwendet. Die Modellierung der zu kommunizierenden Variablen geschah informell. Ihre Namen wiesen bereits eine hierarchische Struktur auf und wurden in Octet Strings kodiert. Eine ASN.1-basierte Syntax zur Modellierung der Informationen war jedoch hier noch nicht vorhanden.

Beide Ansätze — HEMS und SGMP — sind inzwischen zurückgezogen worden und tragen den Status *Historic*. Der später im April 1989 veröffentlichte und im Oktober 1990 noch einmal fortgeführte Entwurf der *Common Management Information Services and Protocols for the Internet (CMOT and CMIP)*, einer Anwendung des OSI Management-Modells auf TCP/IP Netze, entsprach zwar im Ansatz dem langfristigen Ziel des *Internet Architecture Boards* (IAB) [13], konnte sich aber im Gegensatz zu dem bereits zuvor erarbeiteten Nachfolgemodell von SGMP nicht bewähren. Diese den kurzfristigen Zielen des IAB entsprechenden Entwürfe wurden bereits im August 1988 als SNMP Management-Modell in Form von drei Dokumenten veröffentlicht:

- *Simple Network Management Protocol*, RFC 1067 [14]: die Spezifikation des Protokolls zwischen Managern und Agenten,
- *Structure and Identification of Management Information for TCP/IP-Based Internets*, RFC 1065 [15]: die Spezifikation eines Modells von Management-Informationen, und
- *Management Information Base for Network Management of TCP/IP-Based Internets*, RFC 1066 [16]: die Spezifikation eines Moduls mit einem grundlegenden Umfang von Objekten (MIB).

Die in RFC 1065 [15] beschriebene *SMI* definiert dabei eine formelle Notation zur Modellierung von Management-Informationen und darüber hinausgehende knappe Richtlinien zu ihrer Anwendung und zur Eingliederung in eine Namenshierarchie. Im Mai 1990 konnte das Dokument nach geringfügigen Änderungen als RFC 1155 [17] zum *Standard* Status erhoben werden.

Im März 1991 wurden unter anderem zwei Dokumente verabschiedet, die die grundlegenden Informationen zur Entwicklung von prägnanten SMI-MIBs verdeutlichen sollten und eine neue Spezifikation zur Definition von Traps beinhalteten:

- *Concise MIB Definitions*, RFC 1212 [18],
- *A Convention for Defining Traps for Use with the SNMP*, RFC 1215 [19].

In den folgenden Jahren wurden nicht nur am Management Protokoll sondern auch an der SMI einige Defizite deutlich, die schließlich im April 1993 zur Verabschiedung der inzwischen in drei Teile aufgespaltenen SMI führten:

- *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1442 [20],
- *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1443 [21],

- *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1444 [22].

Vom *Proposed Standard* wurden diese Dokumente im Januar 1996 wiederum nach geringfügigen Detaillierungen in Form der RFCs 1902–1904 [23], [24], [25] zum *Draft Standard* erhoben. Seit Juli 1998 war ein *SMIv2 Design Team* [1] mit der Einarbeitung zahlreicher Detaillierungen und einiger Vereinfachungen befaßt, so daß die Dokumente schließlich im April 1999 als RFCs 2578–2580 [2], [3], [4] mit dem Status *Full Standard* publiziert werden konnten.

Seit der Definition der SMI im Rahmen der SNMPv2 Entwicklung (kurz SMIv2) sind neue Standard MIB Module in dieser neuen SMI zu verfassen. Mit der Erhebung der SMIv2 zum Full Standard ist zudem auch für Autoren weiterer MIB Module die Entscheidung für SMIv2 erleichtert worden. Es ist damit zu rechnen, daß SMIv2 somit verstärkt SMIv1 ablösen wird.

Module in der alten SMI (zur deutlicheren Unterscheidung oft SMIv1 genannt) können von den meisten Applikationen weiterhin verarbeitet werden, sollen aber nicht mehr neu verfaßt werden. Im folgenden wird nur die SMIv2 betrachtet.

Die formelle Beschreibung von Management-Informationen geschieht in einer Syntax, die einer modifizierten Teilmenge der *Abstract Syntax Notation One* (ASN.1, [11]) entspricht. Diese Tatsache beruht auf dem ursprünglichen Ziel des IAB, dieselben Spezifikationen von Management-Informationen zu einem späteren Zeitpunkt auch für das CMIS-basierte Management zu verwenden. Mit der stetigen Weiterentwicklung des SNMP-basierten Managements wurde aber bereits im August 1989 klar, daß dieses Ziel nicht zu erreichen ist, ohne neue Entwicklungen einzuschränken [26]. Trotz dieser sehr frühen Erkenntnis wurde bis heute an dem ASN.1-basierten Modell festgehalten, obgleich einige Defizite dieses Ansatzes lange bekannt sind (siehe 2.1.5).

2.1.2. Das Modell

Ressourcen, die Gegenstand des Netzwerkmanagements sein sollen, werden auf Variablen abgebildet, deren Werte beim Lesen Zustände der Ressourcen widerspiegeln und beim Schreiben Wertänderungen oder andere Aktionen bewirken. Eine Klasse einer solchen Variablen wird als Objekt (*Object* oder *Object Type*) bezeichnet. Eine Instanz eines solchen Objektes ist eine Variable (*Variable* oder *Object Instance*). Eigenschaften eines Objektes wie der Datentyp, die Zugriffsmöglichkeiten, eine eindeutige Identifizierung von Objekt und Instanzen, Darstellungshinweise und eine informelle Beschreibung des Objektes werden in einem geeigneten ASN.1 Makro formuliert.

Es gibt skalare Objekte (*Scalar Objects*), zu denen genau eine Instanz existiert und Spaltenobjekte (*Columnar Objects*), die eine Spalte einer Tabelle darstellen, deren Anzahl von Zeilen variieren kann.

Die eindeutige Identifizierung von Objekten geschieht anhand eines registrierten ASN.1 *Object Identifiers*. Dies ist eine Aneinanderreihung von natürlichen Zahlen, die einen Pfad in einem Hierarchie-Baum beschreibt. Jede Zahl entspricht von der Wurzel bis zum Objekt einem Knoten in der Hierarchie. Ein einmal registriertes Objekt darf nicht mehr entfernt oder in seiner Syntax oder Semantik verändert werden. Im Gegensatz zu ASN.1 begrenzt SMI die Länge eines Object Identifiers auf maximal 128 Subidentifier und den Wert jedes Subidentifiers auf den Bereich von 0 bis $2^{32} - 1 = 4294967295$.

Einige Bereiche des Registrierungsbaumes sind durch Standardisierungen vorgegeben, andere Teilbereiche sind an Hersteller delegiert und wiederum andere dienen experimentellen Zwecken. Der Bereich der ersten vier Ebenen des Baumes ist in Abbildung 2.1 zu sehen.

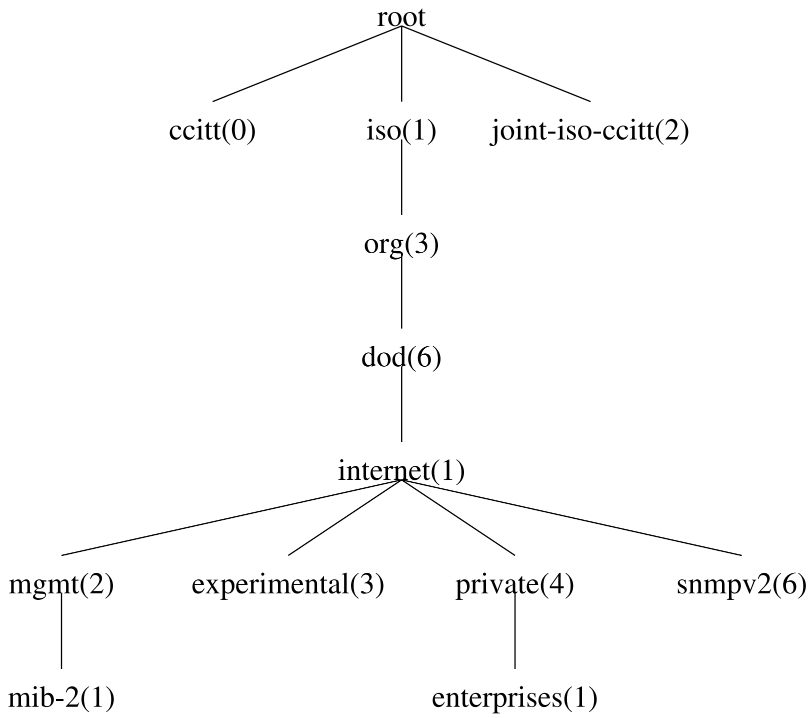

Abbildung 2.1.: OID-Baum der für SNMP wichtigsten Registrierungen

Die eindeutige Identifizierung von Objektinstanzen geschieht bei skalaren Objekten durch Anhängen des Subidentifiers '0' an den Object Identifier des Object Type. Bei Spaltenobjekten wird hingegen eine Subidentifier-Folge angehängt, die eine Tabellenzeile eindeutig identifiziert und aus der Beschreibung eines entsprechenden Zeilenobjektes hervorgeht.

Zusammengehörige Objekte werden in ASN.1 Modulen (*MIB Modules, Management Information Base*) zusammengefaßt und formell nach Regeln der SMI beschrieben. Module enthalten darüber hinaus Definitionen von *Notifications* (vom Agenten initiierte Nachrichten an einen oder mehrere Manager beim Eintreten bestimmter Ereignisse). Über Gruppen von Objekten und Notifications können Anforderungen an Implementierungen formuliert werden (*Conformance Statements*). Andererseits können auch Aussagen über Eigenschaften einer konkreten Implementierung getroffen werden (*Agent Capabilities*).

2.1.3. Sprachelemente

Die SMI definiert acht ASN.1 Makros, die zur formellen Beschreibung von Management-Informationen benutzt werden. Diese Makros werden in den folgenden Abschnitten schematisch knapp beschrieben. Die entsprechenden ASN.1 Definitionen und die exakten Beschreibungen sind in [2], [3] und [4] nachzulesen.

Im Rahmen dieser Arbeit ist ein SMIv1/v2 Parser entstanden, dessen YACC[1]-Grammatik in Anhang B angegeben ist. Die vollständigen Sourcen samt LEX-Spezifikation sind beim Autor und Betreuer dieser Arbeit erhältlich.

2.1.3.1. MODULE-IDENTITY

```
<descriptor> MODULE-IDENTITY
    LAST-UPDATED <timestamp>
    ORGANIZATION <organization>
    CONTACT-INFO <contact-info>
    DESCRIPTION <description>
    [ REVISION <timestamp>
      DESCRIPTION <description>
      [...] ]
    ::= { <object-identifier> }
```

Jedes Modul muß unmittelbar nach den ASN.1-`IMPORTS` Statements genau ein `MODULE-IDENTITY` Makro aufweisen. Hierin werden Angaben zum vorliegenden Modul gemacht: In der `LAST-UPDATED` Klausel wird das Datum der letzten Revision des Moduls notiert. `ORGANIZATION` enthält eine Beschreibung der Organisation, die dieses Modul entwickelt hat. Unter `CONTACT-INFO` sind die Adresse, Telefonnummer und EMail-Adresse der Person zu nennen, an die Anfragen zu diesem Modul zu richten sind. Die `DESCRIPTION` Klausel enthält eine informelle Beschreibung des Moduls. In den abschließenden optionalen Paaren von `REVISION` und `DESCRIPTION` Klauseln werden in zeitlich umgekehrter Folge das Datum und eine Beschreibung jeder Revision des Moduls notiert.

Diese Informationen werden mit einem *Descriptor* versehen und unter einem Object Identifier registriert.

2.1.3.2. OBJECT-TYPE

```
<descriptor> OBJECT-TYPE
    SYNTAX   <type>
           | BITS { <bit-identifier> ( <number> ) [ , ... ] }
    [ UNITS <units> ]
    MAX-ACCESS   not-accessible | accessible-for-notify | read-only
               | read-write | read-create
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    [   INDEX { [ IMPLIED ] <column-descriptor> [ , ... ] }
      | AUGMENTS <index-descriptor> ]
    [ DEFVAL { <value> } ]
    ::= { <object-identifier> }
```

Das `OBJECT-TYPE` Makro dient der Definition und Registrierung eines Objektes unter einem Object Identifier. Die `SYNTAX` Klausel spezifiziert die Werte, die Instanzen dieses Objektes annehmen können. Dies sind die ASN.1 Typen `INTEGER`, `OCTET STRING` und `OBJECT IDENTIFIER`,

[1]Einige Unterschiede machten es erforderlich, die GNU-Variante *bison* zu verwenden.

Typ-Einschränkungen in Form von Größenbeschränkungen (SIZE), Wertebereichen (*Ranges*) oder Aufzählungstypen (*Enumerations*), *Textual Conventions* (siehe 2.1.3.5), durch die SMI definierte ASN.1-*Application Types* oder BITS-Konstrukte.

Durch die optionale UNITS Klausel kann die Einheit der Werte eines Objektes bestimmt werden. Die MAX-ACCESS Klausel gibt die möglichen Zugriffe auf Instanzen diese Objektes an: accessible-for-notify zeigt an, daß das Objekt nur in Notifications übermittelt werden kann. Zugriffe durch *Get*-Requests sind hingegen erst ab read-only möglich. Mit read-write kann der Wert einer Instanz durch *Set*-Requests auch verändert werden. Zeilen von read-create Spaltenobjekten können durch *Set*-Requests sogar neu erzeugt werden.

Die *STATUS* Klausel zeigt an, ob dieses Objekt der aktuellen Spezifikation entspricht und implementiert sein muß (current), ob es nicht mehr unterstützt werden muß, aber aus Gründen der Kompatibilität zu anderen Implementierungen implementiert werden kann (deprecated) oder ob es in älteren Definitionen vorhanden war, aber mittlerweile nicht mehr implementiert werden sollte (obsolete).

Die DESCRIPTION Klausel dient einer informellen Beschreibung des Objektes. Die optionale REFERENCE Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

Die INDEX Klausel dient in Objekten einer Tabellenzeile der Angabe derjenigen Spaltenobjekte, die eine Zeile eindeutig identifizieren (Indizierung, siehe 2.1.2). Das optionale IMPLIED Schlüsselwort kann dabei die Kodierung von Indizierungsspalten um einen Subidentifier verkürzen, wenn die Länge des Spaltenobjektes konstant ist oder das Spaltenobjekt nicht die Länge Null haben kann und das letzte Indizierungsobjekt darstellt. Durch die AUGMENTS Klausel kann eine Tabelle als Ergänzung einer anderen Tabelle definiert werden, indem auf das Zeilenobjekt der ursprünglichen Tabelle verwiesen wird. Instanzen werden somit durch dieselben Indizierungsspalten identifiziert wie in der ursprünglichen Tabelle.

Durch die DEFVAL Klausel kann schließlich ein Wert definiert werden, den Objekte annehmen, die von Agenten dynamisch erzeugt werden oder als Spaltenobjekte durch *Set*-Operationen erzeugt werden, die keinen Wert für diese Spalte aufweisen.

2.1.3.3. OBJECT-IDENTITY

```
<descriptor> OBJECT-IDENTITY
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    ::= { <object-identifier> }
```

Das OBJECT-IDENTITY Makro dient der Registrierung eines Object Identifiers und der Verknüpfung mit einem Descriptor. Dabei werden als zusätzliche Informationen eine Statusangabe, eine Beschreibung und eine optionale informelle Referenzangabe an diesen Object Identifier gebunden.

2.1.3.4. NOTIFICATION-TYPE

```
<descriptor> NOTIFICATION-TYPE
```

```
    OBJECTS { <object-identifier> [ , ... ] }
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    ::= { <object-identifier> }
```

Das NOTIFICATION-TYPE Makro dient der Definition und Registrierung einer Notification. Die in der optionalen OBJECTS Klausel aufgeführten Object Identifier bezeichnen Objekte deren Instanzen in einer entsprechenden Notification enthalten sein müssen.

Die STATUS Klausel zeigt wie im OBJECT-TYPE Makro die Aktualität dieser Definition an. Die DESCRIPTION Klausel dient einer informellen Beschreibung des Objektes. Die optionale REFERENCE Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

2.1.3.5. TEXTUAL-CONVENTION

```
<type> ::= TEXTUAL-CONVENTION
    [ DISPLAY-HINT <format> ]
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    SYNTAX   <type>
           | BITS { <bit-identifier> ( <number> ) [ , ... ] }
```

Das TEXTUAL-CONVENTION Makro hat eine andere Struktur als die anderen Makros, da es keinen Objekt Identifier registriert, sondern der Definition eines Typs dient. Dieser Typ wird durch die optionale DISPLAY-HINT Klausel mit einem Formatierungshinweis versehen, so daß etwa Manager die Möglichkeit haben, OCTET STRINGs in einer für den Anwender lesbaren Form darzustellen. Die STATUS Klausel zeigt wie im OBJECT-TYPE Makro die Aktualität dieser Typ-Definition an. Die DESCRIPTION Klausel dient einer informellen Beschreibung des Typs. Die optionale REFERENCE Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

Die SYNTAX Klausel spezifiziert die Werte, die Instanzen von Objekten dieses Typs annehmen können. Dies sind die ASN.1 Typen INTEGER, OCTET STRING und OBJECT IDENTIFIER, Typ-Einschränkungen in Form von Größenbeschränkungen (SIZE), Wertebereichen (*Ranges*) oder Aufzählungstypen (*Enumerations*), durch die SMI definierte ASN.1-*Application Types* oder BITS-Konstrukte. Eine weitere Textual Convention darf hier nicht angegeben werden.

2.1.3.6. OBJECT-GROUP

```
<descriptor> OBJECT-GROUP
    OBJECTS { <object-identifier> [ , ... ] }
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    ::= { <object-identifier> }
```

Das `OBJECT-GROUP` Makro dient der Zusammenfassung einer Menge von Objekten zu einer Gruppe. Eine so registrierte Gruppe kann somit in *Conformance Statements* verwendet werden (siehe 2.1.3.8, 2.1.3.9).

Die `OBJECTS` Klausel spezifiziert die entsprechende Liste von Objekten. Wie in anderen Makros auch, enthält die `STATUS` Klausel die Aktualität dieser Objekt-Gruppe. Die `DESCRIPTION` Klausel dient einer informellen Beschreibung. Die optionale `REFERENCE` Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

2.1.3.7. NOTIFICATION-GROUP

```
<descriptor> NOTIFICATION-GROUP
    NOTIFICATIONS { <object-identifier> [ , ... ] }
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    ::= { <object-identifier> }
```

Ähnlich wie das `OBJECT-GROUP` Makro dient das `NOTIFICATION-GROUP` Makro der Zusammenfassung einer Menge von Notifications zu einer Gruppe. Eine so registrierte Gruppe kann somit in *Conformance Statements* verwendet werden (siehe 2.1.3.8, 2.1.3.9).

Die `OBJECTS` Klausel spezifiziert die entsprechende Liste von Objekten. Wie in anderen Makros auch, enthält die `STATUS` Klausel den historischen Status dieser Objekt-Gruppe. Die `DESCRIPTION` Klausel dient einer informellen Beschreibung. Die optionale `REFERENCE` Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

2.1.3.8. MODULE-COMPLIANCE

```
<descriptor> MODULE-COMPLIANCE
    STATUS current | deprecated | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    [ MODULE [ <module> ]
             [ MANDATORY-GROUPS { <group-oid> [ , ... ] } ]
             [   GROUP <group-oid>
                 DESCRIPTION <description>
               | OBJECT <object-identifier>
                 [ SYNTAX <syntax> ]
                 [ WRITE-SYNTAX <syntax> ]
                 [ MIN-ACCESS <access> ]
                 DESCRIPTION <description>
               [ , ... ] ] ]
      [ , ... ] ]
    ::= { <object-identifier> }
```

Mit Hilfe des `MODULE-COMPLIANCE` Makros werden Anforderungen an MIB Implementierungen formuliert. Wie in anderen Makros auch, enthält die `STATUS` Klausel den historischen Status dieses *Conformance Statements*. Die `DESCRIPTION` Klausel dient einer informellen Beschreibung. Die optionale `REFERENCE` Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

Der anschließende Abschnitt des Makros kann eine komplexe Gestalt annehmen: Mehrere Blöcke beschreiben Anforderungen an jeweils ein Modul. Beziehen sich diese Angaben auf das lokale Modul, so kann der Modulname in der `MODULE` Klausel entfallen.

Die in der optionalen `MANDATORY-GROUPS` Klausel aufgeführten Gruppen von Objekten und Notifications, wie sie durch `OBJECT-GROUP` und `NOTIFICATION-GROUP` Makros definiert werden, sind zwingend durch eine Implementation zu erbringen, die diesem Conformance Statement genügen soll.

Die nachfolgenden optionalen Paare von `GROUP` und `DESCRIPTION` Klauseln spezifizieren hingegen Gruppen, die optional sind oder nur unter gewissen Bedingungen zu implementieren sind. Diese Bedingungen sind in der jeweiligen `DESCRIPTION` Klausel informell zu beschreiben.

`OBJECT` Klauseln können die möglichen Werte von Objekten (`SYNTAX`), ihre für *Set*-Requests zulässigen Werte und die minimal zu implementierenden Zugriffsrechte (`MIN-ACCESS`) gegenüber der ursprünglichen Objektdefinition verfeinern. Eine informelle Beschreibung dieser Verfeinerung und ggf. Bedingungen, unter denen diese anzuwenden sind, ist in der entsprechenden `DESCRIPTION` Klausel zu notieren.

2.1.3.9. AGENT-CAPABILITIES

```
<descriptor> AGENT-CAPABILITIES
    PRODUCT-RELEASE <text>
    STATUS current | obsolete
    DESCRIPTION <description>
    [ REFERENCE <reference> ]
    [ SUPPORTS <module>
      INCLUDES { <group-oid> [ , ... ] }
      [   VARIATION <notification-oid>
          [ ACCESS not-implemented ]
          DESCRIPTION <description>
        | VARIATION <object-oid>
          [ SYNTAX <syntax> ]
          [ WRITE-SYNTAX <syntax> ]
          [ ACCESS   not-implemented | accessible-for-notify
                   | read-only | read-write
                   | read-create | write-only ]
          [ CREATION-REQUIRES { <object-identifier> [ , ... ] } ]
          [ DEFVAL { <value> } ]
          DESCRIPTION <description>
        [ , ... ] ]
      [ , ... ] ]
    ::= { <object-identifier> }
```

Das AGENT-CAPABILITIES Makro dient nicht der Modellierung von Management-Informationen. Es ermöglicht vielmehr, die Eigenschaften einer MIB-Implementierung hinsichtlich ihres Umfangs und ihrer Abweichungen von den entsprechenden MIB-Definitionen zu beschreiben.

Die PRODUCT-RELEASE Klausel enthält eine informelle Beschreibung des Agenten, der durch dieses AGENT-CAPABILITIES Makro beschrieben wird. Die STATUS Klausel zeigt an, ob dieses Conformance Statement aktuell oder obsolet ist. Durch die DESCRIPTION Klausel wird eine informelle Beschreibung des Statements gegeben. Die optionale REFERENCE Klausel kann Referenzen auf andere Module oder Informationen anderer Quellen aufnehmen.

Die nachfolgenden Abschnitte beschreiben durch eine SUPPORTS Klausel eingeleitet die durch den Agenten unterstützten MIB Module. Hierbei benennt eine INCLUDES Klausel die jeweils vollständig oder teilweise unterstützten Objekt- und Notification-Gruppen. Daran können sich in VARIATION Klauseln formulierte Abweichungen von den jeweiligen MIB Definitionen anschließen. Im Falle von Notifications kann dies lediglich durch ACCESS not-implemented angezeigt eine fehlende Implementierung sein. Bei Objekten können solche Variationen einen anderen Wertebereich (SYNTAX), einen eingeschränkten Wertebereich für *Set*-Requests (WRITE-SYNTAX), eingeschränkte Zugriffsmöglichkeiten (ACCESS), Anforderungen an bestimmte Spalten im Falle von *Set*-Requests zur Erzeugung von Tabellenzeilen und abweichende Default-Werte (DEFVAL) anzeigen. Jede Variation weist eine DESCRIPTION Klausel mit einer informellen Beschreibung auf.

2.1.4. Beispiel

Dieses Beispiel enthält Ausschnitte der Interface MIB (IF-MIB), die zahlreiche Elemente der SMIv2 nutzt. Um das Beispiel zu kürzen, wurden einige Abschnitte wiederholter gleichförmiger Makros entfernt.

```
IF-MIB DEFINITIONS ::= BEGIN

   IMPORTS
       MODULE-IDENTITY, OBJECT-TYPE, Counter32, Gauge32, Counter64,
       Integer32, TimeTicks, mib-2,
       NOTIFICATION-TYPE                        FROM SNMPv2-SMI
       TEXTUAL-CONVENTION, DisplayString,
       PhysAddress, TruthValue, RowStatus,
       TimeStamp, AutonomousType, TestAndIncr   FROM SNMPv2-TC
       MODULE-COMPLIANCE, OBJECT-GROUP          FROM SNMPv2-CONF
       snmpTraps                                FROM SNMPv2-MIB
       IANAifType                               FROM IANAifType-MIB;


   ifMIB MODULE-IDENTITY
       LAST-UPDATED "9611031355Z"
       ORGANIZATION "IETF Interfaces MIB Working Group"
       CONTACT-INFO
               "   Keith McCloghrie
                   Cisco Systems, Inc.
                   170 West Tasman Drive
                   San Jose, CA  95134-1706
                   US

                   408-526-5260
                   kzm@cisco.com"
```

```
    DESCRIPTION
            "The MIB module to describe generic objects for
            network interface sub-layers.  This MIB is an updated
            version of MIB-II's ifTable, and incorporates the
            extensions defined in RFC 1229."
    REVISION      "9602282155Z"
    DESCRIPTION
            "Revisions made by the Interfaces MIB WG."
    REVISION      "9311082155Z"
    DESCRIPTION
            "Initial revision, published as part of RFC 1573."
    ::= { mib-2 31 }

ifMIBObjects OBJECT IDENTIFIER ::= { ifMIB 1 }

interfaces   OBJECT IDENTIFIER ::= { mib-2 2 }


-- [...]


-- InterfaceIndex contains the semantics of ifIndex and
-- should be used for any objects defined on other mib
-- modules that need these semantics.

InterfaceIndex ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "d"
    STATUS       current
    DESCRIPTION
            "A unique value, greater than zero, for each interface
            or interface sub-layer in the managed system.  It is
            recommended that values are assigned contiguously
            starting from 1.  The value for each interface sub-
            layer must remain constant at least from one re-
            initialization of the entity's network management
            system to the next re-initialization."
    SYNTAX       Integer32 (1..2147483647)


-- [...]


ifNumber  OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The number of network interfaces (regardless of their
            current state) present on this system."
    ::= { interfaces 1 }


ifTableLastChange  OBJECT-TYPE
    SYNTAX      TimeTicks
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of sysUpTime at the time of the last
            creation or deletion of an entry in the ifTable.  If
            the number of entries has been unchanged since the
            last re-initialization of the local network management
            subsystem, then this object contains a zero value."
    ::= { ifMIBObjects 5 }
```

```
-- the Interfaces table

-- The Interfaces table contains information on the entity's
-- interfaces.  Each sub-layer below the internetwork-layer
-- of a network interface is considered to be an interface.

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

ifEntry OBJECT-TYPE
    SYNTAX      IfEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "An entry containing management information applicable
            to a particular interface."
    INDEX   { ifIndex }

    ::= { ifTable 1 }

IfEntry ::=
    SEQUENCE {
        ifIndex                 InterfaceIndex,
        ifDescr                 DisplayString,
        ifType                  IANAifType,
        ifMtu                   Integer32,
        ifSpeed                 Gauge32,
        ifPhysAddress           PhysAddress,
        ifAdminStatus           INTEGER,
        ifOperStatus            INTEGER,
        ifLastChange            TimeTicks,
        ifInOctets              Counter32,
        ifInUcastPkts           Counter32,
        ifInNUcastPkts          Counter32,  -- deprecated
        ifInDiscards            Counter32,
        ifInErrors              Counter32,
        ifInUnknownProtos       Counter32,
        ifOutOctets             Counter32,
        ifOutUcastPkts          Counter32,
        ifOutNUcastPkts         Counter32,  -- deprecated
        ifOutDiscards           Counter32,
        ifOutErrors             Counter32,
        ifOutQLen               Gauge32,    -- deprecated
        ifSpecific              OBJECT IDENTIFIER -- deprecated
    }

ifIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A unique value, greater than zero, for each
            interface.  It is recommended that values are assigned
            contiguously starting from 1.  The value for each
            interface sub-layer must remain constant at least from
            one re-initialization of the entity's network
```

```
                management system to the next re-initialization."
        ::= { ifEntry 1 }

    ifDescr OBJECT-TYPE
        SYNTAX      DisplayString (SIZE (0..255))
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "A textual string containing information about the
                interface.  This string should include the name of the
                manufacturer, the product name and the version of the
                interface hardware/software."
        ::= { ifEntry 2 }

    ifType OBJECT-TYPE
        SYNTAX      IANAifType
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The type of interface.  Additional values for ifType
                are assigned by the Internet Assigned Numbers
                Authority (IANA), through updating the syntax of the
                IANAifType textual convention."
        ::= { ifEntry 3 }

    ifMtu OBJECT-TYPE
        SYNTAX      Integer32
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The size of the largest packet which can be
                sent/received on the interface, specified in octets.
                For interfaces that are used for transmitting network
                datagrams, this is the size of the largest network
                datagram that can be sent on the interface."
        ::= { ifEntry 4 }

    ifSpeed OBJECT-TYPE
        SYNTAX      Gauge32
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "An estimate of the interface's current bandwidth in
                bits per second.  For interfaces which do not vary in
                bandwidth or for those where no accurate estimation
                can be made, this object should contain the nominal
                bandwidth.  If the bandwidth of the interface is
                greater than the maximum value reportable by this
                object then this object should report its maximum
                value (4,294,967,295) and ifHighSpeed must be used to
                report the interace's speed.  For a sub-layer which
                has no concept of bandwidth, this object should be
                zero."
        ::= { ifEntry 5 }

    ifPhysAddress OBJECT-TYPE
        SYNTAX      PhysAddress
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The interface's address at its protocol sub-layer.
                For example, for an 802.x interface, this object
                normally contains a MAC address.  The interface's
```

```
            media-specific MIB must define the bit and byte
            ordering and the format of the value of this object.
            For interfaces which do not have such an address
            (e.g., a serial line), this object should contain an
            octet string of zero length."
    ::= { ifEntry 6 }

ifAdminStatus OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),       -- ready to pass packets
                down(2),
                testing(3)   -- in some test mode
            }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
            "The desired state of the interface.  The testing(3)
            state indicates that no operational packets can be
            passed.  When a managed system initializes, all
            interfaces start with ifAdminStatus in the down(2)
            state.  As a result of either explicit management
            action or per configuration information retained by
            the managed system, ifAdminStatus is then changed to
            either the up(1) or testing(3) states (or remains in
            the down(2) state)."
    ::= { ifEntry 7 }


-- [...]


--
--   Extension to the interface table
--
-- This table replaces the ifExtnsTable table.
--

ifXTable        OBJECT-TYPE
    SYNTAX      SEQUENCE OF IfXEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A list of interface entries.  The number of entries
            is given by the value of ifNumber.  This table
            contains additional objects for the interface table."
    ::= { ifMIBObjects 1 }

ifXEntry        OBJECT-TYPE
    SYNTAX      IfXEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "An entry containing additional management information
            applicable to a particular interface."
    AUGMENTS    { ifEntry }
    ::= { ifXTable 1 }


IfXEntry ::=
    SEQUENCE {
        ifName                  DisplayString,
        ifInMulticastPkts       Counter32,
        ifInBroadcastPkts       Counter32,
```

```
        ifOutMulticastPkts      Counter32,
        ifOutBroadcastPkts      Counter32,
        ifHCInOctets            Counter64,
        ifHCInUcastPkts         Counter64,
        ifHCInMulticastPkts     Counter64,
        ifHCInBroadcastPkts     Counter64,
        ifHCOutOctets           Counter64,
        ifHCOutUcastPkts        Counter64,
        ifHCOutMulticastPkts    Counter64,
        ifHCOutBroadcastPkts    Counter64,
        ifLinkUpDownTrapEnable  INTEGER,
        ifHighSpeed             Gauge32,
        ifPromiscuousMode       TruthValue,
        ifConnectorPresent      TruthValue,
        ifAlias                 DisplayString,
        ifCounterDiscontinuityTime TimeStamp
    }

ifName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The textual name of the interface.  The value of this
            object should be the name of the interface as assigned
            by the local device and should be suitable for use in
            commands entered at the device's `console'.  This
            might be a text name, such as `le0' or a simple port
            number, such as `1', depending on the interface naming
            syntax of the device.  If several entries in the
            ifTable together represent a single interface as named
            by the device, then each will have the same value of
            ifName.  Note that for an agent which responds to SNMP
            queries concerning an interface on some other
            (proxied) device, then the value of ifName for such an
            interface is the proxied device's local name for it.

            If there is no local name, or this object is otherwise
            not applicable, then this object contains a zero-
            length string."
    ::= { ifXEntry 1 }


ifInMulticastPkts OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The number of packets, delivered by this sub-layer to
            a higher (sub-)layer, which were addressed to a
            multicast address at this sub-layer.  For a MAC layer
            protocol, this includes both Group and Functional
            addresses.

            Discontinuities in the value of this counter can occur
            at re-initialization of the management system, and at
            other times as indicated by the value of
            ifCounterDiscontinuityTime."
    ::= { ifXEntry 2 }


-- [...]
```

```
-- conformance information

ifConformance OBJECT IDENTIFIER ::= { ifMIB 2 }

ifGroups      OBJECT IDENTIFIER ::= { ifConformance 1 }
ifCompliances OBJECT IDENTIFIER ::= { ifConformance 2 }


-- compliance statements

ifCompliance2 MODULE-COMPLIANCE
    STATUS  current
    DESCRIPTION
            "The compliance statement for SNMPv2 entities which
            have network interfaces."

    MODULE  -- this module
        MANDATORY-GROUPS { ifGeneralInformationGroup, ifStackGroup2,
                           ifCounterDiscontinuityGroup }

        GROUP       ifFixedLengthGroup
        DESCRIPTION
            "This group is mandatory for all network interfaces
            which are character-oriented or transmit data in
            fixed-length transmission units."


-- [...]


        OBJECT       ifAdminStatus
        SYNTAX       INTEGER { up(1), down(2) }
        MIN-ACCESS   read-only
        DESCRIPTION
            "Write access is not required, nor is support for the
            value testing(3)."

        OBJECT       ifAlias
        MIN-ACCESS   read-only
        DESCRIPTION
            "Write access is not required."

    ::= { ifCompliances 2 }

-- units of conformance

ifGeneralInformationGroup    OBJECT-GROUP
    OBJECTS { ifIndex, ifDescr, ifType, ifSpeed, ifPhysAddress,
              ifAdminStatus, ifOperStatus, ifLastChange,
              ifLinkUpDownTrapEnable, ifConnectorPresent,
              ifHighSpeed, ifName, ifNumber, ifAlias,
              ifTableLastChange }
    STATUS  current
    DESCRIPTION
            "A collection of objects providing information
            applicable to all network interfaces."
    ::= { ifGroups 10 }

-- the following five groups are mutually exclusive; at most
-- one of these groups is implemented for any interface
```

```
    ifFixedLengthGroup    OBJECT-GROUP
        OBJECTS { ifInOctets, ifOutOctets, ifInUnknownProtos,
                  ifInErrors, ifOutErrors }
        STATUS  current
        DESCRIPTION
                "A collection of objects providing information
                specific to non-high speed (non-high speed interfaces
                transmit and receive at speeds less than or equal to
                20,000,000 bits/second) character-oriented or fixed-
                length-transmission network interfaces."
        ::= { ifGroups 2 }


    -- [...]


END
```

2.1.5. Bewertung

Das SNMP-basierte Netzwerkmanagement ist derzeit als das erfolgreichste Modell anzusehen. Dieser Erfolg ist auf zwei grundlegende Zielsetzungen zurückzuführen, die sich in den vergangenen zehn Jahren als richtig erwiesen haben: Ein Grundsatz war stets die Einfachheit der Konzepte. Dies spiegelt sich auch im Namen des so entworfenen Protokolls wieder: das **S**imple Network Management Protocol. Daß dies auch für das entwickelte Informationsmodell gilt, wird insbesondere im Vergleich mit den in den folgenden Abschnitten beschriebenen Modellen deutlich.

Der zweite grundlegende Gedanke war es, die entwickelten Konzepte stets leicht erweitern zu können, um die Entwicklung des Protokolls und der SMI nicht zu behindern. Wie in Abschnitt 2.1.1 beschrieben, wurden im Laufe der vergangenen zehn Jahre tatsächlich einige Änderungen erforderlich, was die anfängliche Einfachheit zum Teil aufgehoben hat. Solche Änderungen werden durch den Standardisierungsprozeß allerdings erschwert bzw. verzögert.

ASN.1 als Basis für die SMI zu benutzen, schien anfangs angesichts des langfristigen Ziels, den Übergang zum OSI Management zu ermöglichen, sicherlich richtig. Dies brachte jedoch auch einige Nachteile mit sich. Einige lexikographische und syntaktische Details blieben lange Zeit unbeantwortet und konnten erst durch jüngste Präzisierungen in den entsprechenden RFC-Dokumenten geklärt werden: Sind eingeschobene Kommentare als 'whitespace' zu interpretieren? Wie ist mit Descriptoren umzugehen, die in ASN.1 Schlüsselworte sind? Wie sind Zeilenumbrüche gefolgt von 'whitespace' in Strings zu formatieren?

Erschwerend kommt hinzu, daß die ASN.1 Version, auf der die SMI beruht, heute nicht mehr von der ISO erhältlich ist. Sie stammt vom Dezember 1987. Erhältlich sind lediglich eine Version aus dem Jahr 1990 und die aktuelle vierteilige Version aus dem Jahr 1995, [11]).

In ASN.1 sind Vorwärtsreferenzen zulässig, die die Entwicklung effizienter Parser erschweren. In der SMI wäre dies aber nicht notwendig gewesen. Ein anderer Punkt ist der eingeschränkte Umfang von Typen: Die SMI kennt nur ganzzahlige Typen. Außerdem wurde nach der Spezifikation der SMIv1 deutlich, daß der Wertebereich zu knapp bemessen war, was in der SMIv2 zur Einführung des `Counter64` Typs führte. In ähnlicher Weise ist nun fraglich, ob `Unsigned64` und `Integer64` Typen eingeführt werden sollten.

Da SMIv2 zwar einerseits auf ASN.1 basiert, andererseits gegenüber ASN.1 neben etlichen Einschränkungen auch einige Veränderungen aufweist[2], sind vorhandene ASN.1 Parser als Basis für SMI Compiler nur bedingt nutzbar.

Andere Informationsmodelle, wie GDMO oder CIM (siehe 2.2, 2.3) sind in zweierlei Hinsicht objektorientiert. Zum einen erlauben sie das Kapseln zusammengehöriger Informationen in Klassen. Eine derart modellierte Klasse stellt ein abgeschlossenes Konstrukt dar. Die SMI bildet aufgrund ihrer Einfachheit jede Information auf eine einfache Variable ab. Das Gruppieren von Variablen ist in Form von Tabellen zwar möglich aber aufwendiger, wenig übersichtlich und enthält redundante Informationen. Dadurch, daß die SMI keine Vererbungshierarchie kennt und keine feine Strukturierung wiederverwendbarer Elemente wie die Templates bei GDMO (siehe Abschnitt 2.2) bietet, gibt es relativ wenige Abhängigkeiten zwischen Definitionen innerhalb eines Moduls und über Modulgrenzen hinweg. Das erleichtert Änderungen bestehender MIBs. Mögliche Veränderungen an MIBs und dabei zu befolgende Regeln werden durch die SMI präzise beschrieben.

Die ASN.1 Notation ist im OSI-Bereich zwar durchaus gebräuchlich, für Implementierer jedoch nicht sehr intuitiv verständlich. Notationen wie in CIM oder in der OMG IDL sind stärker an Daten- bzw. Klassenstrukturen gängiger Programmiersprachen angelehnt und erleichtern so das Verständnis und die Übersichtlichkeit.

Diese und weitere kritisierte Punkte und entsprechende Änderungsvorschläge werden in Kapitel 3 detailliert behandelt.

[2] Aus der Einleitung des RFC 1902: These modules are written using an adapted subset of OSI's Abstract Syntax Notation One, ASN.1 (1988).

2.2. GDMO

2.2.1. OSI Management

Das OSI-Referenzmodell (*Open Systems Interconnect*) [29] der *International Organization for Standardization* (ISO) [30] wurde in den 80er Jahren entwickelt. 1989 entstand als Teil 4 des Referenzmodells das *OSI Management Framework* [31], [32]. Die Zusammenhänge der wichtigsten OSI-Dokumente zeigt Abbildung 2.2. Die für das Management-Informationsmodell wichtigen Standards sind die in der Abbildung rechts zu sehenden vier Teile der *Structure of Management Information* (SMI) [33], [34], [35], [36].

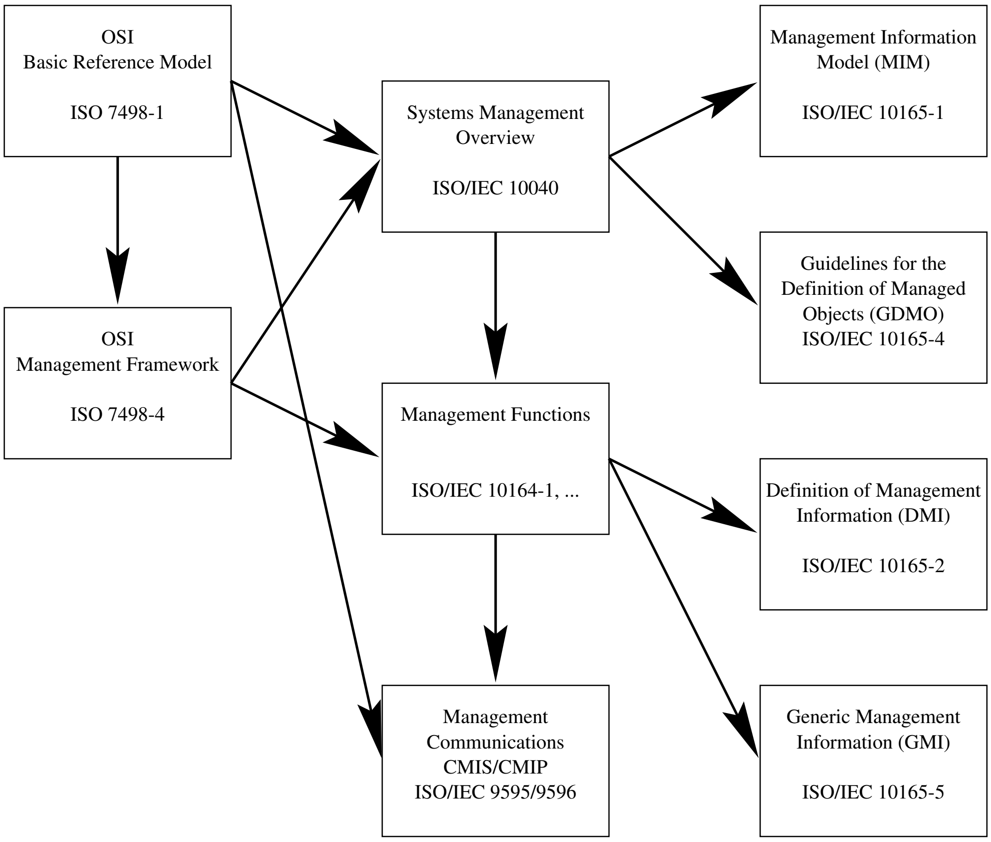

Abbildung 2.2.: OSI Management Standards

Darüber hinaus existieren seit etwa 1996 der Teil 6 (*Requirements and Guidelines for Implementation Conformance Statement Proformas associated with OSI Management*) [37] und der Teil 7 (*Structure of Management Information: General Relationship Model*) [38].

Während das *Management Information Model* (MIM) [33] die grundlegenden Konzepte der *Managed Objects* (MO) beschreibt, sind für das Modellieren von Management-Informationen in erster Linie die *Guidelines for the Definition of Managed Objects* (GDMO) [34], [39] von Bedeutung.

Objekte im Sinne des Management Information Models weisen einige grundlegende Eigenschaften der Objektorientierung auf, wie z.B. das Kapseln zusammengehöriger Informationen und Operationen in Klassen und die Vererbung von Attributen und Operationen einer oder mehrerer Klassen an eine abgeleitete Klasse. Eine Klasse hat eine Anzahl von Attributen (*Attributes*), die in einer Instanz konkrete Werte (*Values*) annehmen. Diese Attribute spiegeln den Zustand eines Objektes wider. Mehrere Attribute können zu Gruppen (*Attribute Groups*) zusammengefaßt werden. Managementoperationen (*Management Operations*) dienen dazu, Aktionen auf Attributen oder Attributgruppen des Objektes oder auf dem Objekt als Ganzes auszuführen. Einem Objekt können *Notifications* zugeordnet sein, die bei Eintreten definierter Ereignisse des Objektes abgegeben werden, ohne daß ein Manager dies initiiert. *Packages* dienen dazu, verschiedene Ausprägungen derselben Klassen zu definieren, ohne eine Vererbungshierarchie aufzubauen, die bei mehrfachen Kriterien sehr aufwendig und unübersichtlich werden könnte. Dabei werden *Mandatory Packages*, die jede Instanz der Klasse aufweisen muß, und *Conditional Packages*, die nur unter gewissen Bedingungen vorhanden sind, unterschieden. *Name Bindings* weisen Attribute zur Identifizierung von Objektinstanzen für die Objekterzeugung und -löschung und für den Zugriff auf Objekte aus. Das Verhalten (*Behaviour*) all dieser Beschreibungselemente kann informell beschrieben werden.

Durch den Mechanismus der Vererbung (*Inheritance*) werden Spezialisierungen (*Specialization*) von Klassen gebildet. Dabei wird die abgeleitete Klasse als *Subclass* und die Elternklasse als *Superclass* bezeichnet. Mehrfachvererbung (*Multiple Inheritance*), also das Ableiten einer Klasse von mehr als einer Elternklasse, ist möglich. Eine ausgezeichnete Klasse (‘Top’, definiert in DMI [35]) bildet die Wurzel der Vererbungshierarchie (*Inheritance Hierarchy*), die jedoch wegen der möglichen Mehrfachvererbung nicht zwangsläufig einen Baum darstellt.

Die Klassenvererbung ist *strict*, d.h. alle Attribute, Operationen und Notifications der Elternklasse werden ohne Ausnahme in die abgeleitete Klasse übernommen. Innerhalb gewisser Regeln können aber Wertebereiche und andere Werteeinschränkungen von Attributen und Operationsparametern und der Umfang von Parametern und Notifications verändert werden. Diese Regeln sind präzise in MIM [33] beschrieben und gewährleisten die Interoperabilität zwischen verschiedenen Implementierungen, die ggf. dieselben Objekt-Instanzen anhand verschiedener Klassenspezialisierungen referenzieren.

Alle Attribute werden durch global eindeutige ASN.1 *Object Identifier* (*Abstract Syntax Notation One*) [11] identifiziert.

Die auf Attributen arbeitenden Management-Operationen sind

- *Get*, um den Wert eines Attributes zu lesen,
- *Replace*, um den Wert eines Attributes zu verändern und
- *Replace with Default*, um den Wert eines Attributes auf einen durch die Definition des Objektes wohlbekannten Wert zurückzusetzen.

Darüber hinaus gibt es zwei Operation, die auf Attribute mit Mengen-Typen (*Set-Valued Attributes*) angewendet werden können:

- *Add*, um Elemente zu einem Mengen-Attribut zu ergänzen und

- *Remove*, um Elemente aus einem Mengen-Attribut zu entfernen.

Zu der Definition einer Klasse gehört die Angabe, welche Operationen auf jedem einzelnen Attribut der Klasse ausgeführt werden können.

Die folgenden Operationen arbeiten auf einer Objekt-Instanz:

- *Create* dient der Erzeugung einer Objekt-Instanz. Dabei werden die Attribute nach bestimmten Regeln, die aus der Klassen-Definition und der Create-Operation hervorgehen, initialisiert.
- *Delete* dient der Entfernung einer Objekt-Instanz.
- *Action* ist eine generische Operation, die genutzt werden kann, um weitere Operationen auf Instanzen ausführen zu können. Die Definition der Action-Operation ist Bestandteil der Klassen-Definition.

Die Menge der Objekt-Instanzen bildet durch eine 'enthalten-in'-Relation eine Baumstruktur (*Containment Tree*). Dies ist nicht zu verwechseln mit der Hierarchie der Klassen-Definitionen. Beispielsweise ist eine Netzverbindung Bestandteil einer Protokoll-Maschine, die wiederum Bestandteil eines Subsystems ist, usw. Die Wurzel dieser Baumstruktur bildet ein *System*-Objekt.

Die eindeutige Identifizierung einer Objekt-Instanz geschieht durch die Aneinanderreihung von einzelnen Namen im Containment Tree (*Relative Distinguished Names*, RDN) von der Wurzel bis zum entsprechenden Objekt. Diese einzelnen relativen Namen bestehen jeweils aus einem in der jeweiligen Klassen ausgezeichneten *Attribute Identifier* und einem dazugehörigen Wert. Dieses Attribut-Wert-Paar (*Attribute Value Assertion*, AVA) muß an jeder Stelle des Pfades das enthaltene Objekt innerhalb des Elternobjektes eindeutig identifizieren. Hieraus setzt sich der lokale Anteil (*Local Form*) eines Objektnamens zusammen. Darüber hinaus ist eine eindeutige globale Identifizierung über Systemgrenzen hinweg notwendig. Dazu dienen die zwei Attribute *systemTitle* und *systemId* der Klasse *System* durch die das System eindeutig identifiziert werden kann. Zusammen bilden alle notwendigen Attribut-Wert-Paare den *Distinguished Name* (DN).

2.2.2. Definition von *Managed Objects*

Die konkrete Definition von *Managed Objects* wird in *Guidelines for the Definition of Managed Objects* (GDMO) [34] beschrieben. Hier finden sich zum einen Richtlinien, nach denen Klassenhierarchien aufgebaut werden sollen und zum anderen konkrete syntaktische Notationen zur Beschreibung von Klassen-Definitionen.

Eine angemessene Verwendung von Strukturierungsmitteln wie Vererbung, Mehrfachvererbung und Packages wird vorgeschrieben, um eine übersichtliche Klassenhierarchie zu gewährleisten und mehrfache Definitionen gleicher Objekte mit unterschiedlichen Ausprägungen zu vermeiden. Die Komplexität von modellierten Objekten soll der Komplexität der zugrundeliegenden Ressourcen entsprechen.

Objekt-Klassen werden im *Registration Tree* registriert, in dem ihnen eindeutige *Object Identifier* zugeordnet werden. Ein Object Identifier ist eine Aneinanderreihung von Integer-Werten (*Arcs*), die von links beginnend eindeutige Knoten im Registrierungsbaum von der Wurzel abwärts darstellen. Dabei ist jeder

Knoten die *Registration Authority* für den darunter registrierten Teilbaum. Dies geschieht in derselben Weise und sogar im selben Registrierungsbaum wie im Falle von SNMP.

Das grundlegende Mittel zur Definition von Objekt-Klassen ist das Ausfüllen von *Templates*. Solche Templates existieren für Klassen, Packages, Parameter, Attribute, Attributgruppen, Behaviours, Actions, Notifications und Name Bindings. Sie sind in [34] ähnlich wie ASN.1 Macros [11] definiert. Die durch GDMO definierte Sprache weist ähnlich wie die SMI im Falle von SNMP zahlreiche Ähnlichkeiten zu ASN.1 auf, ist jedoch nicht vollständig ASN.1 konform ([32], 6.6.1).

Das Template zur Modellierung eines MOs ist `MANAGED OBJECT CLASS`. Dies bedient sich weiterer Templates entweder unmittelbar als *Inline Definition* oder durch Referenzierung separat definierter Templates anhand ihrer Labels. Die Struktur der wichtigsten Template-Abhängigkeiten zur Definition von MOs zeigt Abbildung 2.3.

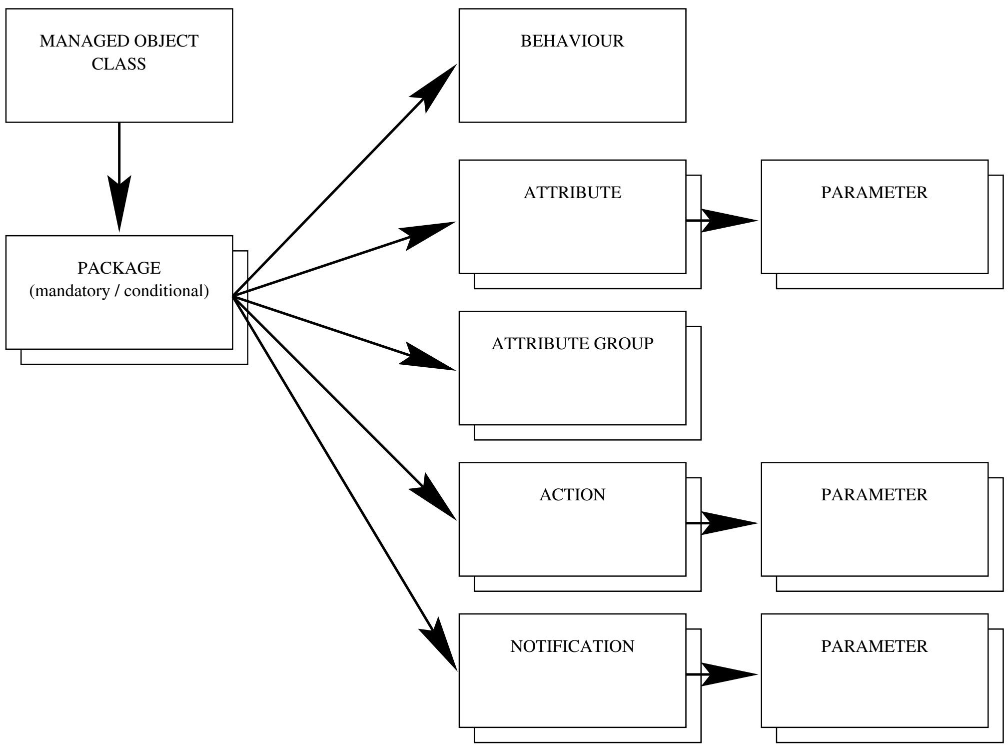

Abbildung 2.3.: Die wichtigsten Abhängigkeiten der GDMO Templates

Nachfolgend sind sämtliche neun Templates schematisch aufgeführt und knapp erläutert. Daran schließt sich ein Beispiel einer Klassendefinition an.

Weitere Artikel, die die GDMO Templates knapp erläutern, sind [40] und [41]. Die vollständige Definition ist [34] zu entnehmen.

2.2.2.1. MANAGED OBJECT CLASS

```
<class-label> MANAGED OBJECT CLASS
  DERIVED FROM <managed-object-class-templates>
  [ CHARACTERIZED BY <mandatory-package-templates> ]
  [ CONDITIONAL PACKAGES <conditional-package-templates>
      PRESENT IF <condition> ]
  REGISTERED AS <class-object-identifier>
```

Das `MANAGED OBJECT CLASS` Template dient der Definition einer neuen Objektklasse. Sie erbt alle Eigenschaften der Klassen `managed-object-class-templates`. Das `DERIVED FROM` Konstrukt ist hier nicht als optional gekennzeichnet, da es lediglich in der einen Definition der Klasse *Top* in DMI nicht vorhanden ist.

Im `CHARACTERIZED BY` Konstrukt werden alle *Mandatory Packages* aufgeführt. In gleicher Weise werden durch `CONDITIONAL PACKAGES` alle unter der Bedingung `condition` geltenden *Conditional Packages* aufgeführt. Diese Bedingungen werden lediglich informell beschrieben.

Die Klasse wird unter dem global eindeutigen Object Identifier `class-object-identifier` registriert, um sie an anderer Stelle und in Operationen eindeutig referenzieren zu können.

2.2.2.2. PACKAGE

```
<package-label> PACKAGE
  [ BEHAVIOUR <behaviour-templates> ]
  [ ATTRIBUTES <attribute-templates> <property-list>
    [ <parameter-templates> ] ]
  [ ATTRIBUTE GROUPS <attribute-group-templates>
    [ <attribute-templates> ] ]
  [ ACTIONS <action-templates>
    [ <paramater-templates> ] ]
  [ NOTIFICATIONS <notification-templates>
    [ <parameter-templates> ] ]
  [ REGISTERED AS <package-object-identifier> ]
```

Das `PACKAGE` Template faßt die logisch zusammengehörigen Verhaltensbeschreibungen (`BEHAVIOUR`), Attribute (`ATTRIBUTE`), Attributgruppen (`ATTRIBUTE GROUPS`), Actions (`ACTIONS`) und Notifications (`NOTIFICATIONS`) zu einem Package zusammen.

Jedes aufgeführte Attribut ist mit einer `property-list` versehen, die

- die auf dem Attribut ausführbaren Operationen ausweist (`GET`, `REPLACE`, `ADD`, `REMOVE`),
- den Default-Wert des Attributs ausweist, der bei einer *Replace with Default* Operation hergestellt wird,
- einen Initialwert des Attributs ausweist, den das Attribut bei der Objektinstanziierung annimmt,
- zulässige Werte des Attributs ausweist, die das Attribut annehmen kann,

- erforderliche Werte des Attributs ausweist, die eine Implementierung unterstützen muß.

Ferner kann ein Attribut ebenso wie das ACTIONS und NOTIFICATIONS Konstrukt Parameter-Erweiterungen aufweisen.

Das Package kann durch das REGISTERED AS Konstrukt unter dem global eindeutigen Object Identifier class-object-identifier registriert werden, um es in anderen CONDITIONAL PACKAGES Konstrukten referenzieren zu können.

2.2.2.3. PARAMETER

```
<parameter-label> PARAMETER
  CONTEXT <context-types>
  [ WITH SYNTAX <asn1-data-type>
    | ATTRIBUTE <attribute-template> ]
  [ BEHAVIOUR <behaviour-templates> ]
  REGISTERED AS <parameter-object-identifier>
```

Das PARAMETER Template dient der Erweiterung von CMIS-Fehlermeldungen und der Ergänzung der Syntax von Operationen, Notifications und Antworten. Das ASN.1 Konstrukt, das zu der Möglichkeit solcher Erweiterungen führt, hat die Gestalt:

```
<Type> ::= SEQUENCE {
  ...
  <label>       <parameter-object-identifier>,
  <extension>   ANY DEFINED BY <label>
  ...
}
```

Das CONTEXT Konstrukt führt eine Reihe von Kontexten auf, in denen dieses PARAMETER Template Anwendung findet. Möglich sind ACTION-INFO, ACTION-REPLY, EVENT-INFO und EVENT-REPLY für eine *Action*-Operation und -Antwort, eine Notification und die Bestätigung einer Notification. SPECIFIC-ERROR ist im Falle von CMIS Fehlermeldungen möglich. Die entsprechende ASN.1 SEQUENCE-Struktur wird gemäß des WITH SYNTAX oder ATTRIBUTE Konstrukts an seiner ANY DEFINED BY Stelle ergänzt.

Durch das REGISTERED AS Konstrukt wird der Parameter unter einen global eindeutigen Object Identifier parameter-object-identifier registriert, um ihn in SEQUENCE-Strukturen referenzieren zu können.

2.2.2.4. ATTRIBUTE

```
<attribute-label> ATTRIBUTE
  [ DERIVED FROM <attribute-template>
    | WITH ATTRIBUTE SYNTAX <asn1-data-type> ]
  [ MATCHES FOR <qualifiers> ]
```

```
  [ BEHAVIOUR <behaviour-templates> ]
  [ PARAMETERS <parameter-templates> ]
  [ REGISTERED AS <attribute-object-identifier> ]
```

Das ATTRIBUTE Template dient der Definition eines Attributes als Spezialisierung eines im DERIVED FROM Konstrukt angegebenen Attributes oder durch Angabe eines konkreten ASN.1 Typs.

Das MATCHES FOR Konstrukt ermöglicht die Angabe von Regeln, die die mögliche Verwendung von CMIS Filtern (siehe [32]) beschreibt.

Das PARAMETERS Konstrukt dient der Erweiterung möglicher CMIS Fehlermeldungen, die dieses Attribut betreffen.

Mit dem REGISTERED AS Konstrukt kann das Attribut unter einem eindeutigen Objekt Identifier attribute-object-identifier registriert werden, um es in anderen Templates referenzieren zu können.

2.2.2.5. ATTRIBUTE GROUP

```
<attribute-group-label> ATTRIBUTE GROUP
  [ GROUP ELEMENTS <attribute-templates> ]
  [ FIXED ]
  [ DESCRIPTION <string> ]
  REGISTERED AS <attribute-group-object-identifier>
```

Das ATTRIBUTE GROUP Template dient der Definition einer Gruppe von Attributen, die im GROUP ELEMENTS Konstrukt anzugeben sind, um auf ihnen leichter gemeinsame Operationen (*Get* und *Replace with Default*) ausführen zu können. Wenn die Gruppe nicht explizit als FIXED ausgezeichnet ist, kann ihr Umfang in weiteren PACKAGE Templates ergänzt werden.

Das DESCRIPTION Konstrukt dient der Angabe einer informellen Beschreibung der Attributgruppe.

Durch das REGISTERED AS Konstrukt wird die Attributgruppe registriert, um sie etwa in CMIS Operationen eindeutig referenzieren zu können.

2.2.2.6. BEHAVIOUR

```
<behaviour-label> BEHAVIOUR
  [ DEFINED AS <string> ]
```

Mit Hilfe des BEHAVIOUR Templates kann das Verhalten verschiedener Elemente beschrieben werden. Derartige Beschreibungen sind rein informell.

Über Weiterentwicklungen von GDMO wird diskutiert, um das BEHAVIOUR Template mit formal beschreibbaren Informationen versehen zu können, so daß entsprechende Compiler dies in verwertbare Informationen überführen können. Ein Entwurf [42] sieht einen regelbasierten Ansatz nach der sog. CRS Technik (Communicating Rule Systems) vor, die bereits in anderen OSI Protokollbeschreibungen Anwendung gefunden hat. Dieser Entwurf ermöglicht detaillierte Verhaltensbeschreibungen und gliedert sich in das Vererbungskonzept ein. Dazu sind jedoch aufwendige Beschreibungen und Erweiterungen der GDMO-Syntax nötig, um die nötigen Schnittstellen zu Variablen des Modells zu erhalten.

2.2.2.7. ACTION

```
<action-label> ACTION
  [ BEHAVIOUR <behaviour-templates> ]
  [ MODE CONFIRMED ]
  [ PARAMETERS <parameter-templates> ]
  [ WITH INFORMATION SYNTAX <asn1-data-type> ]
  [ WITH REPLY SYNTAX <asn1-data-type> ]
  REGISTERED AS <action-object-identifier>
```

Mit dem `ACTION` Template können Operationen beschrieben werden, die über die Funktionalität der einfachen *Get* und *Replace* Operationen hinausgehen.

Durch `MODE CONFIRMED` kann erzwungen werden, daß die Operation stets beantwortet wird. Das `PARAMETERS` Konstrukt dient der Ergänzung möglicher CMIS Fehlermeldungen.

Im `WITH INFORMATION SYNTAX` Konstrukt können zusätzliche Argumente der Operation definiert werden, während im `WITH REPLY SYNTAX` Konstrukt zusätzliche Antwort-Informationen definiert werden.

Mittels des `REGISTERED AS` Konstruktes wird die Action unter dem Object Identifier `action-object-identifier` registriert, um sie in Packages eindeutig referenzieren zu können.

2.2.2.8. NOTIFICATION

```
<notification-label> NOTIFICATION
  [ BEHAVIOUR <behaviour-templates> ]
  [ PARAMETERS <parameter-templates> ]
  [ WITH INFORMATION SYNTAX <asn1-data-type>
    [ AND ATTRIBUTE IDS <field-names> <attribute-templates> ] ]
  [ WITH REPLY SYNTAX <asn1-data-type> ]
  REGISTERED AS <notification-object-identifier>
```

Das `NOTIFICATION` Template dient der Definition von Notifications. Dabei können zusätzliche CMIS Fehlermeldungen im `PARAMETERS` Konstrukt definiert werden.

Die Ereignisse, die zur Auslösung der Notification führen, werden im `BEHAVIOUR` Konstrukt beschrieben.

Im `WITH INFORMATION SYNTAX` Konstrukt können zusätzliche Informationen der Notification definiert werden, während im `WITH REPLY SYNTAX` Konstrukt zusätzliche Antwort-Informationen definiert werden.

Mittels des `REGISTERED AS` Konstruktes wird die Notification unter dem Object Identifier `notification-object-identifier` registriert, um sie in Packages eindeutig referenzieren zu können.

2.2.2.9. NAME BINDING

```
<name-binding-label> NAME BINDING
  SUBORDINATE OBJECT CLASS <subordinate-managed-object-class>
  NAMED BY SUPERIOR OBJECT CLASS <superior-managed-object-class>
  WITH ATTRIBUTE <attribute-template>
  [ BEHAVIOUR <behaviour-templates> ]
  [ CREATE [ <create-modifiers> [ <parameter-templates> ] ] ]
  [ DELETE [ <delete-modifiers> [ <parameter-templates> ] ] ]
  REGISTERED AS <name-binding-object-identifier>
```

Durch das `NAME BINDING` Template werden Containment-Relationen zwischen Klassen definiert. Die durch das `SUBORDINATE OBJECT CLASS` Konstrukt bestimmte Klasse wird als Bestandteil der durch das `NAMED BY SUPERIOR OBJECT CLASS` Konstrukt bestimmten Klasse definiert. Beiden Konstrukten kann der Qualifier `AND SUBCLASSES` angehängt werden, um diese Beziehung auch für abgeleitete Klassen geltend zu machen.

Das für den RDN genutzte Attribut in dieser Klassenbeziehung ist durch das `WITH ATTRIBUTE` Konstrukt festgelegt.

Die `CREATE` und `DELETE` Konstrukte dienen der Festlegung von Bedingungen, unter denen Instanzen der `subordinate-managed-object-class` erzeugt und entfernt werden können. So ist erst durch Angabe des `CREATE` Konstruktes das Erzeugen von Instanzen über Management-Operationen möglich. Durch Angabe von `DELETES-CONTAINED-OBJECTS` in den `delete-modifiers` werden beim Löschen einer Objektinstanz implizit alle untergeordneten Objekte gelöscht. Weitere Möglichkeiten sind in [34] spezifiziert.

Durch das `REGISTERED AS` Konstrukt wird das Name Binding unter dem Object Identifier `name-binding-object-identifier` registriert, so daß ein Manager es in *Create* Operationen eindeutig referenzieren kann.

2.2.3. Beispiel

Dieses kleine Beispiel aus [43] zeigt anhand der Definition der `gaugeMonitor` Klasse die Verwendung einiger grundlegender Templates.

```
--------------------------------------------------------------------------------
-- Document : X738

gaugeMonitor MANAGED OBJECT CLASS
    DERIVED FROM
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":top;
    CHARACTERIZED BY
        gaugeMonitorPackage;
    CONDITIONAL PACKAGES
        resetCounterMax PRESENT IF
            "derivedGauge derived from
             resettable counter type",
        wrapCounterDifference PRESENT IF
            "derivedGauge derived
             from wrap counter type",
```

```
        weeklySchedule PRESENT IF
            "both the daily scheduling
             package and external schedular package are
             not present in an instance",
        dailySchedule PRESENT IF
            "both the weekly scheduling
             package and external schedular package are
             not present in an instance",
        externalSchedular PRESENT IF
            "both the daily scheduling
             and weekly scheduling packages are not present
             in an instance";
REGISTERED AS   { x738ObjectClass 1 };

dailySchedule PACKAGE
;

externalSchedular PACKAGE
;

gaugeMonitorPackage PACKAGE
    BEHAVIOUR           gaugeMonitorBehaviour;
    ATTRIBUTES
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":
                                                            administrativeState
            GET-REPLACE,
        gaugeMonitorId
            GET,
        observedObjectClass-X738
            GET,
        observedObjectInstance-X738
            GET,
        observedAttributeId-X738
            GET,
        derivedGauge-X738
            GET,
        granularityPeriod-X738
            GET-REPLACE,
        derivedGaugeThold
            GET-REPLACE;
    NOTIFICATIONS
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":
                                                                   stateChange,
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":
                                                          attributeValueChange,
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":
                                                                objectCreation,
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":
                                                                objectDeletion,
        "Recommendation X.721 | Recommendation X.721 | ISO/IEC 10165-2 : 1992":
                                                         qualityofServiceAlarm;
;

resetCounterMax PACKAGE
;

weeklySchedule PACKAGE
;

wrapCounterDifference PACKAGE
;

derivedGauge-X738 ATTRIBUTE
```

```
    DERIVED FROM            "Recommendation X.721 | Recommendation X.721 |
                                            ISO/IEC 10165-2 : 1992":gauge;
REGISTERED AS   { x738Attribute 1 };

derivedGaugeThold ATTRIBUTE
    DERIVED FROM
                            "Recommendation X.721 | Recommendation X.721 |
                                  ISO/IEC 10165-2 : 1992":gauge-Threshold;

REGISTERED AS   { x738Attribute 2 };

gaugeMonitorId ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.GaugeMonitorId;
REGISTERED AS   { x738Attribute 3 };

granularityPeriod-X738 ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.TimePeriod;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 4 };

numericAttributeIdArray ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.IdList;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 7 };

observedAttributeId-X738 ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.AttributeId;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 5 };

observedObjectClass-X738 ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.ObjectClass;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 6 };

observedObjectInstance-X738 ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.ObjectInstance;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 7 };

onceReportAttributeIdList ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.IdList;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 7 };

scanAttributeIdList ATTRIBUTE
    WITH ATTRIBUTE SYNTAX   X-738-ASN1Module.IdList;
                            MATCHES FOR EQUALITY;
REGISTERED AS   { x738Attribute 7 };

scanReport NOTIFICATION
    BEHAVIOUR               scanReportBehaviour;
    WITH INFORMATION SYNTAX X-738-ASN1Module.AlarmInfo;
REGISTERED AS   { x738Notification 1 };

gaugeMonitorBehaviour BEHAVIOUR
    DEFINED AS
        "From X.739: The gauge monitor metric object monitors a gauge,
resettable or wrap counter type attribute of a related managed object.  If the
observed attribute is a counter, the object will derive a gauge value from the
counter.  For resettable counters, the derived gauge value is the value of teh
counter just prior to being reset.  For a wrap counter, the derived gauge
value is the difference betweeen successive ovservations of the counter.  A
```

```
threadhold is applied o the gauge for the purpose of generating alarm
notifications oftype quality of service.  the relationship attributes for the
gaugeMonitor metric object are observedObjectClass, observedObjectInstnace and
observedAttributeId.";

scanReportBehaviour BEHAVIOUR
    DEFINED AS
        " Current draft document does not contain any defintion for this
      notifications. Added to satisfy latest Q.822 needs ";
```

2.2.4. Bewertung

Das OSI-Referenzmodell konnte sich in vielen Bereichen nur bedingt etablieren. Ein wesentlicher Grund hierfür ist die relativ hohe Komplexität, die das Modell schwerer verständlich macht und Implementierungen verzögert und verteuert. Diese Probleme weist auch das OSI Management Framework auf, was besonders deutlich wird, wenn man es mit dem primitiveren aber sehr viel stärker verbreiteten SNMP Modell vergleicht. Das OSI Management-Modell findet dementsprechend außer im Bereich der *Telecommunications Management Networks* (TMN) kaum Anwendung.

Die Objektorientierung in der Modellierung von Management-Informationen ermöglicht einen strukturierten Aufbau einer Klassenhierarchie. Durch die strikte Klassenvererbung wird das Entfernen von Attributen aus spezialisierten Klassen vermieden, was zu Inkompatibilitäten führen würde. Zusammen mit den in GDMO festgelegten Richtlinien zur Verwendung von Standardklassen, die durch geeignete Spezialisierungen zu den gewünschten Klassen geformt werden, kann hiermit ein hohes Maß an Kompatibilität zwischen Objekten erzielt werden. Dies wird durch das Prinzip der Packages unterstützt, womit Variationen möglich werden, ohne eine unübersichtliche Vielzahl von verwandten Klassendefinitionen zu erhalten.

Trotz der Komplexität läßt GDMO noch einige Wünsche offen. So können Verhaltensbeschreibungen diverser Templates ebenso wie die Bedingungen für das Einbetten von Conditional Packages in Klassen nur informell beschrieben werden. Eine formale Beschreibung, die es ermöglichen könnte, diese Information durch entsprechende Compiler in Agenten- und Manager-Implementierungen zu verwerten, ist bislang nicht vorgesehen. Es gab lediglich Überlegungen in dieser Richtung ([40], [42]).

Die Flexibilität und weitgehende Aufteilung von Modell-Bestandteilen in verschiedene Templates soll die Wiederverwendbarkeit erhöhen. Diese Gliederung macht jedoch das Lesen derartiger Dokumente schwierig: Es müssen zahlreiche Verweise durch verwendete Templates und Klassenableitungen verfolgt werden. Aus demselben Grund ist auch die Wartung von Dokumenten aufwendiger.

Die Tatsache, daß GDMO ähnlich wie die SNMP-SMI lediglich stark an ASN.1 angelehnt ist, jedoch nicht vollständig ASN.1 konform ist, macht die Entwicklung von Werkzeugen, die auf der ASN.1 Basis aufbauen könnten, schwieriger.

2.3. CIM

2.3.1. Entwicklungsprozeß

Ein relativ junger Ansatz eines Modells zur Beschreibung von Objekten in verteilten Systemen ist das *Common Information Model* (CIM) [44] der *Desktop Management Task Force* (DMTF) [45]. Das erste Dokument wurde im April 1997 veröffentlicht. Vorläufer waren einerseits innerhalb der DMTF die Arbeiten am *Desktop Management Interface* (DMI) [46], andererseits die Anfang 1997 an die DMTF übergebenen Entwicklungen der *Web-Based Enterprise Management Initiative* (WBEM) [47].

Die DMTF wurde 1992 aus einem Kreis führender Hersteller der PC-Branche gegründet und ist dementsprechend stark von marktwirtschaftlichen Einflüssen geprägt. Das anfängliche Ziel war eine strukturierte Plattform für das Management von PC-Netzwerken. Inzwischen umfaßt die Liste der Mitglieder über 200 Firmen.

Zentrale Spezifikation der DMTF für die Beschreibung von PCs und ihrer Komponenten ist das Desktop Management Interface (DMI). Die erste Spezifikation DMI 1.0 wurde 1994 veröffentlicht. Anfang 1996 folgte DMI 2.0 und ergänzte im wesentlichen den Zugriff über ein Netzwerk auch zwischen verschiedenen Plattformen. Einige kleinere Modifikationen führten zum heute aktuellen DMI 2.0s [46]. DMI Modelle werden im *Management Information Format* (MIF) formuliert. Aufgabe der Hersteller ist es, diesem Modell entsprechend Informationen über Objekte bereitzustellen.

Mit dem Ziel eines Web-basierten Managements von Hosts, Netzen und Applikationen gründeten Anfang 1996 einige Unternehmen aus dem PC- und Netzwerkbereich die Web-Based Enterprise Management Initiative (WBEM). Es wurden ein grundlegendes Datenmodell, das *HyperMedia Management Schema* (HMMS), und das *HyperMedia Management Protocol* (HMMP) entwickelt. Die Etablierung der WBEM-Entwicklungen innerhalb der *Internet Engineering Task Force* (IETF) scheiterte, da die damals vorgestellte Technologie Fragen nach einer konkreten Integrationsmöglichkeit mit der breiten Basis an SNMP-Technologie nicht beantworten konnte. Schließlich wurden die Arbeiten Anfang 1997 an die DMTF übergeben und gingen nach Gründung des CIM Subcommittee (*Technology Development Committee*, TDC)[3] in einer ersten CIM Spezifikation und weiteren Entwicklungen auf.

Erst mit dem Common Information Model (CIM) kam es zu einer Ausweitung über die typischen Merkmale von PC-Netzen hinweg und der weitgehend objekt-orientierte Ansatz eines Informationsmodells.

2.3.2. Elemente und Eigenschaften

Den Kern von CIM bildet ein Meta-Modell, das überwiegend typische objekt-orientierte Konstrukte bereitstellt:

- Ein *Schema* dient der Gruppierung zusammengehöriger Klassen. Klassen innerhalb eines Schemas müssen eindeutige Namen haben. Die Zugehörigkeit eines Elementes zu einem Schema wird durch ein Schema-Präfix des vollständigen Namens oder durch den SCHEMA-Qualifier ausgedrückt (s.u.).

[3] Charter: To define a Common Information Model (CIM) based on object-oriented technologies for use in Web-based management.

- Eine Klasse (*Class*) repräsentiert im OO-Sinne ein Modell eines Objektes mit einer gewissen Menge von Attributen und Methoden. Klassen können durch Vererbung definiert werden. Mehrfachvererbung ist nicht möglich.

- Eine Instanz (*Instance*) stellt ein konkretes Exemplar einer Klasse dar.

- Ein Attribut (*Property*) stellt eine Variable innerhalb einer Instanz dar und hat stets einen konkreten Wert. Ein vererbtes Attribut kann in der abgeleiteten Klasse überladen werden. Die möglichen Datentypen für Attribute umfassen Integer-Werte mit und ohne Vorzeichen bis zu einer Breite von 64 Bit, IEEE-Fließpunktzahlen, Zeichen und Strings aus Unicode/UCS-2-Zeichen, ein spezielles Stringformat für Datum, Zeit und Zeitintervalle, sowie streng typisierte *Referenzen* (s.u.). Spezielle Eigenschaften eines Attributs, die nicht die binäre Darstellung beeinflussen (beispielsweise `counter` oder `gauge`), erfordern keine zusätzlichen Datentypen, sondern werden durch *Qualifier* (s.u.) ausgedrückt.

- Eine Methode (*Method*) stellt innerhalb einer Klasse eine bestimmte Operation bereit, die auf dieser Klasse arbeitet. Eine Methode ist formal durch eine Signatur (*Signature*) bestehend aus einem Namen, Parametern und einem Rückgabetyp bezeichnet. Eine vererbte Methode kann in der abgeleiteten Klasse überladen werden.

- Ein *Qualifier* dient der näheren Beschreibung einer Eigenschaft eines Elementes (Schema, Klasse, Attribut, etc.) ohne Bestandteil des Elementes selbst zu sein, beispielsweise um die ‘read-only’-Eigenschaft eines Attributs anzuzeigen oder ein Attribut als (Teil-)Schlüssel (*Key*) zur eindeutigen Identifizierung einer Instanz auszuweisen. Es wird unterschieden zwischen *Meta Qualifiers*, *Standard Qualifiers*, *Optional Qualifiers* und *User-defined Qualifiers*. Die einzigen Meta-Qualifier sind ASSOCIATION und INDICATION. Sie dienen der Spezialisierung einer Klasse (s.u.). Die derzeit 27 spezifizierten Standard-Qualifier müssen von jeder konformen CIM-Implementierung unterstützt werden. Einzelne dieser Qualifier dürfen nur auf bestimmte Elemente angewendet werden oder sie schließen sich in gewissen Kombinationen gegenseitig aus. Die Liste der derzeit acht optionalen Qualifier umfaßt solche, die nur für bestimmte Applikationen benötigt werden und somit nicht von jeder konformen CIM-Implementierung unterstützt werden müssen. Darüber hinaus können benutzer-definierte Qualifier eingeführt werden. Hiervon wird aber gewöhnlich abgeraten. Präzise Listen aller Qualifier mit ihren Default-Werten, Anwendungen, Typen, Restriktionen und Bedeutungen sind in [44] nachzulesen.

- Eine Referenz (*Reference*) ist ein spezielles Attribut, welches eine Klasse oder eine Instanz eindeutig identifiziert (s.u.).

- Eine Assoziation (*Associations*) ist eine spezielle Klasse, die mindestens zwei Referenzen enthält. Dadurch lassen sich Relationen zwischen Klassen beschreiben, ohne die Klassen selbst mit zusätzlichen Informationen zu versehen.

- Ein *Trigger* ermöglicht die Erkennung einer Zustandsänderung oder eines Zugriffs einer Klasseninstanz oder eines Attributes.

- Eine *Indication* ist eine Klasse, deren Instanzen durch einen Trigger erzeugt werden.

Alle Konstrukte sind direkt oder indirekt vom *Named Element* Konstrukt abgeleitet. Die Struktur des Meta-Schemas wird in [44] mit Hilfe der *Unified Modeling Language* (UML) modelliert. Diese detailliert definierte grafische Notation erlaubt einen relativ übersichtlichen und schnell verständlichen Überblick der Abhängigkeiten zwischen Elementen eines objektorientierten Modells. Das UML-Diagramm des Meta-Schemas ist in Abbildung 2.4 ersichtlich. Spezifikationen der UML sind unter [48] verfügbar. Die für CIM wichtigen Grundlagen sind ebenfalls in [44] enthalten.

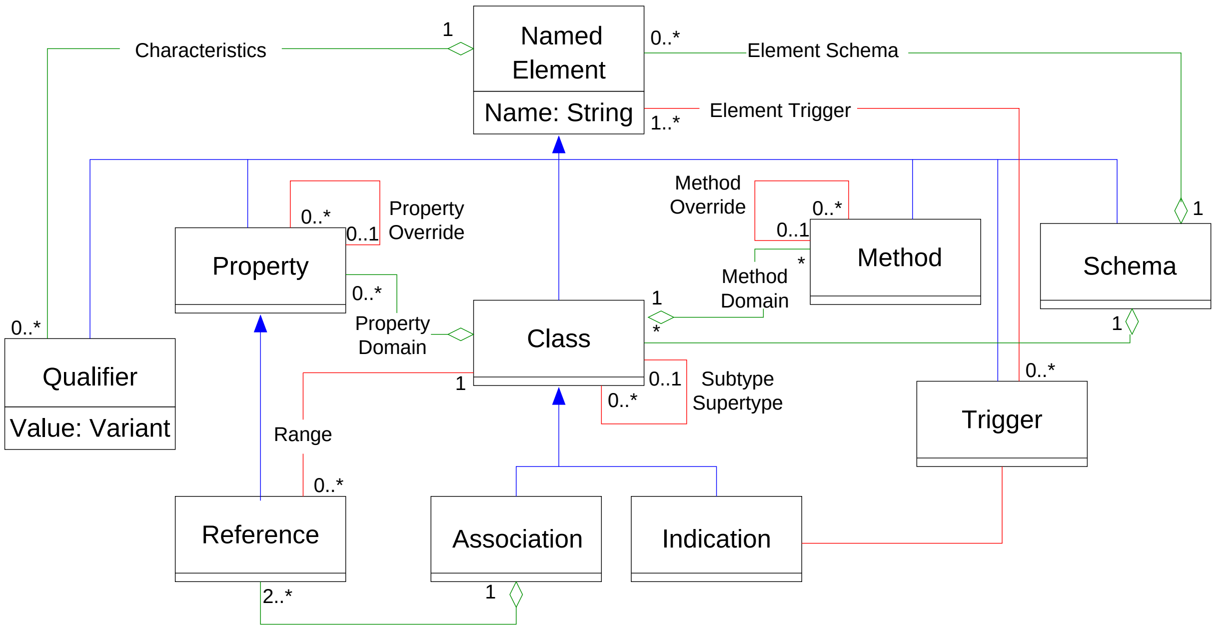

Abbildung 2.4.: UML-Diagramm des CIM Meta Schemas

Die Konstrukte des Meta-Modells genügen, um die drei folgenden aufeinander aufbauenden Schemata zu definieren, die zum Umfang der CIM Spezifikation gehören:

- Das *Core Model* beschreibt eine kleine grundlegende Menge von Klassen, Assoziationen und Qualifiern. Diese sind als eine Basis zu verstehen, deren Definitionen keine Änderungen erfahren sollten. Das UML-Diagramm des Core-Schemas ist in Abbildung 2.5 ersichtlich. Auf die Inhalte des Core-Schemas wird an dieser Stelle nicht näher eingegangen.
- Das *Common Model* beschreibt einige grundlegende Komponenten aus den Bereichen Systeme, Applikationen, Netzwerke und Geräte, ohne dabei auf konkrete Technologien einzugehen. Es dient als Grundgerüst darauf aufbauender Systeme.
- Das *Extension Schema* formuliert Definitionen des Common Model weiter aus, indem es Klassen mit technologiespezifischen Attributen anreichert.

Aufbauend auf diesen drei Modellen können schließlich in spezifischen Schemata die abstrakten Systemkomponenten zu realen Objekten konkretisiert werden.

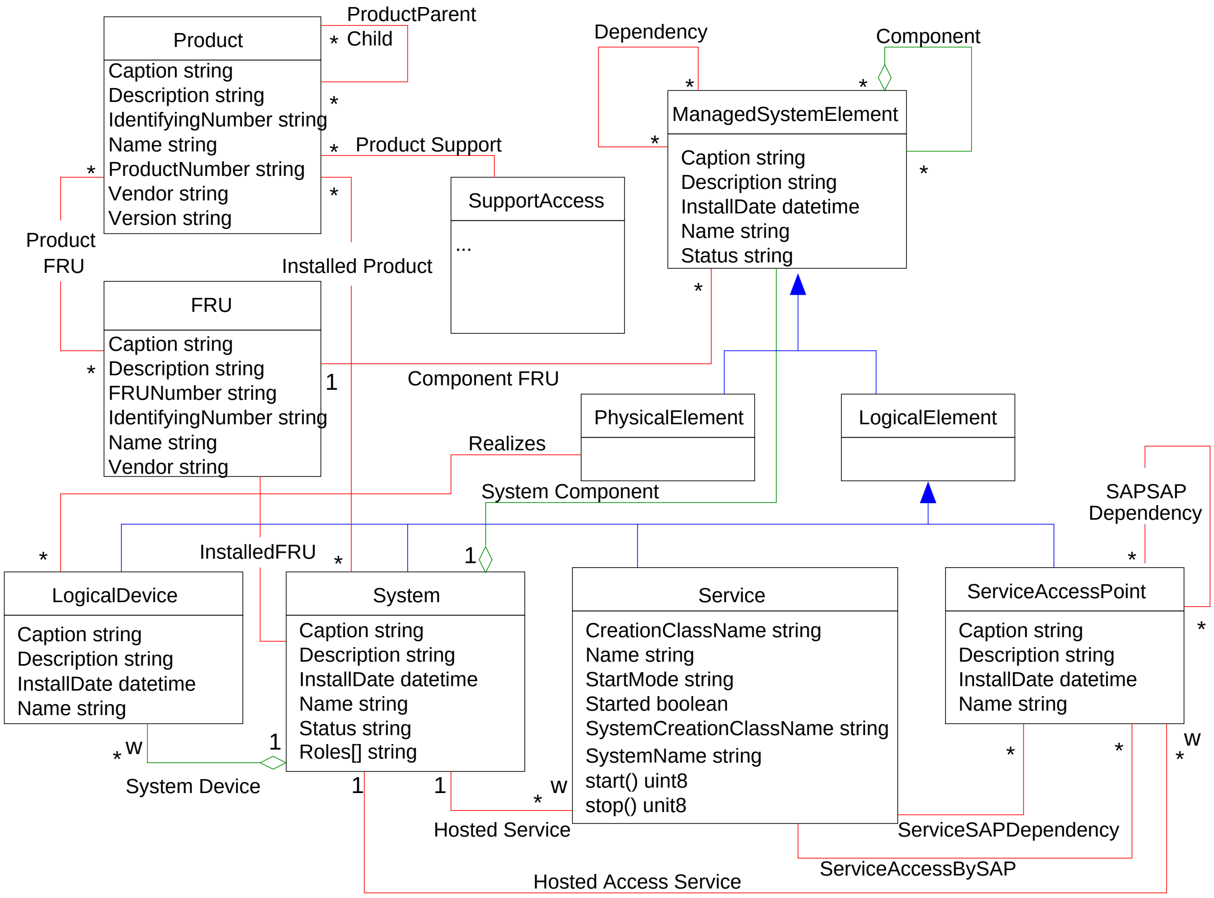

Abbildung 2.5.: UML-Diagramm des CIM Core Schemas

Die Formulierung all dieser Schemata geschieht in Dateien im *Management Object Format* (MOF), einem auf der *Interface Definition Language* (IDL) der Open Group basierenden Dateiformat, das durch eine LL(1)-Grammatik beschrieben werden kann. Eine aus [44] entnommene ABNF Grammatik dieses Formats ist in Anhang C angegeben.

Das Format erlaubt Kommentare durch Einschluß eines '/*'...'*/'-Paares oder durch eine '//'-Einleitung bis zum Zeilenende. Die Groß-/Kleinschreibung von MOF-Schlüsselworten ist nicht relevant. Die Schlüsselworte sind jedoch stets in gleicher Schreibweise zu verwenden, um eine CIM-Einbettung in Umgebungen mit einer Unterscheidung von Groß-/Kleinschreibung zu ermöglichen. Vorwärtsreferenzen zur Ableitung oder Referenzierung von Klassen oder zur Definition von Instanzen werden nicht benötigt und sind nicht gestattet. Die Kodierung erfolgt in Unicode/UTF-8 [49].

Eine Klasse wird durch das Schlüsselwort `class` definiert. Es folgt der Klassenname, optional durch einen Doppelpunkt getrennt eine Elternklasse und in geschweiften Klammern die Liste der Attribute und Methoden. Attribute haben Datentypen und optionale Default-Werte. Methoden haben einen Rückgabetyp und eine in Klammern eingeschlossene typisierte Parameterliste. Klassen, Attributen, Methoden und Parametern können Qualifier vorangestellt werden. Sie werden in eckigen Klammern durch Kommata getrennt notiert. Qualifier können Werte haben, die in runden Klammern auf den Qualifier-Namen folgen. Im Falle von Feldern stehen diese Werte in geschweiften Klammern und werden durch Komma-ta getrennt. Einem Qualifier können durch einen Doppelpunkt getrennt Schlüsselworte folgen, die das Verhalten des Qualifiers bei Vererbung der Klasse beeinflussen.

Durch das Schlüsselwort `qualifier` können Qualifier mit einem bestimmten Typ, Default-Wert, Anwendungsbereich und Vererbungsverhalten definiert werden. Eingeleitet durch die Schlüsselworte `instance of` werden Instanzen von Klassen definiert. Die dabei angegebenen Attribute müssen der Klassendefinition genügen. So müssen beispielsweise alle Attribute mit einem `KEY`-Qualifier explizit definiert sein, da diese zur eindeutigen Identifizierung einer Instanz erforderlich sind. Nicht angegebene Werte werden implizit mit ihren Default-Werten initialisiert. Durch das zusätzliche Schlüsselwort `as` lassen sich Instanz-Aliase erzeugen, die in Referenzen verwendet werden können[4]. Referenz-Attribute werden durch `ref` definiert. Attribute können durch eckige Klammern als Felder fester oder variabler Länge definiert werden. Der Qualifier ARRAYTYPE kann dabei die Art der Verwaltung der Feldelemente beeinflussen. Einige Compiler-Direktiven können das Verhalten von CIM-Parsern und -Compilern beeinflussen, etwa um weitere Schema-Dateien einzubinden (`#pragma include`).

Konstanten aller Basistypen werden für Qualifier-Werte, Default-Werte und Werte in Instanzen benötigt: Strings werden aus UCS-2 Zeichen gebildet und in ein Paar doppelter Anführungstriche eingeschlossen. Einige Sonderzeichen können ähnlich der C-Notation durch Backslash eingeleitet kodiert werden. Einzelne Zeichen werden ebenfalls ähnlich der C-Notation in einfachen Anführungszeichen eingeschlossen. Integer-Konstanten können durch ein '0x'-Präfix und 'B'-Suffix auch hexadezimal oder binär dargestellt werden. Fließpunktzahlen können in Exponentialschreibweise notiert werden. Objekt-Referenzen werden als String notiert, der sich aus einem Namensraumanteil und einem Modellanteil zusammensetzt (s.u.). Der Wert 'NULL' kann für alle Typen benutzt werden, um einen nicht initialisierten Zustand auszudrücken.

Namen von Schemata müssen disjunkt sein, ebenso Namen von Klassen innerhalb eines Schemas und Namen von Methoden und Attributen innerhalb einer Klasse. Vollständige Namen von Klassen (*fully-qualified names*) werden aus dem definierenden Schema und dem Klassennamen, getrennt durch ein '_'-Zeichen, zusammengesetzt.

Durch den Eigentümer (*Owner*) sind nahezu beliebige Änderungen an einem Schema möglich. Viele mögliche Änderungen bedeuten eine Veränderung der Schnittstelle zu Applikationen (Entfernung von Klassen, Properties oder Methoden, Änderung von Property-Typen oder Methoden-Signaturen, etc.), so daß Inkompatibilitäten und syntaktische und semantische Fehler entstehen können. CIM schreibt in diesen Fällen lediglich eine Erhöhung der Hauptversionsnummer des gesamten Schemas (*major version*, VERSION-Qualifier) vor. Andere Änderungen sind hingegen ohne Beeinträchtigung möglich (Einfügen neuer Klassen, Properties, Methoden, etc.).

Modelle mit Klassen-Definitionen werden kompiliert und in den Namensräumen von Implementierungen wie Objekt-Managern oder Datenbanken eingebettet. Hier dienen sie in der Regel der Modellierung konkreter Ausprägungen eines Netzwerkes. Dazu werden Modelle mit Klassen-Instanzen importiert, die gewissermaßen die Modell-Definition mit Leben füllen. Ferner können Modelle aus einem Namensraum exportiert werden und so MOF-Dateien gewonnen werden, welche neben dem Modell auch die konkreten Ausprägungen eines Netzwerkes widerspiegeln.

[4] Aliase können entgegen anderen Elementen auch in Vorwärtsreferenzen auftreten.

Namensräume können genutzt werden, um Sichten auf Teile des gesamten Modells zu unterscheiden. Sie müssen jedoch nicht disjunkt sein. Dies dient der Übersicht für separate Anwendungen und der Performance von Anfragen. Vollständige Namen von Objekten bestehen aus einem Namensraumanteil (*Namespace Path*) und einem innerhalb des Namensraumes gültigen Modellanteil (*Model Path*). Der Namensraumanteil besteht aus einem Typ (*Namespace Type*), der die Art des Zugriffes auf den Namensraum spezifiziert, und einem *Namespace Handle*, das den Namensraum innerhalb der Implementierung eindeutig identifiziert. Die Beschaffenheit des Handles ist implementationsabhängig. So können beispielsweise Implementierungen, die mehrere Namensräume unterstützen, eine hierarchische Struktur von Handles ähnlich der von Pfaden eines Verzeichnisbaumes aufweisen. Der Modellanteil ist durch CIM vollständig beschrieben. Er besteht aus dem vollständigen Klassennamen und bei Instanzen zusätzlich aus der Liste aller Attribut-Wert-Paare aller KEY-Attribute.

CIM verfolgt als neues Modell neben älteren und etablierten Management-Modellen auch das Ziel der Integration. So gibt es drei Ansätze einer Abbildung (*Mapping*) etwa von SNMP MIBs, DMI MIF Modellen und CMIS GDMO MIBs auf CIM: *Technique Mapping*, *Recast Mapping* und *Domain Mapping*. Auch die ursprünglichen Ziele der WBEM eines Web-basierten Managements scheinen durch die jüngsten Entwicklungen einer XML-Kodierung (*Extended Markup Language*) für CIM Schemata aufzugehen.

Die hier beschriebenen Aspekte von CIM sind in [44] vollständig spezifiziert. Das Dokument enthält eine MOF-Grammatik in ABNF-Notation nach [50], UML-Diagramme von wichtigen Teilen des Meta-Schemas, ein MOF-Listing des CIM Meta-Schemas, Beispiele zu den meisten Elementen des Meta-Schemas und einige Richtlinien für Autoren von MOF-Dateien.

2.3.3. Beispiele

Die folgenden Fragmente stammen aus den durch die DMTF spezifizierten MOF-Modulen `CIM_Device21.mof` und `CIM_System21.mof`. Diese werden neben anderen in `CIM_Schema21.mof` eingebunden.

Das erste Beispiel zeigt die Definition der Klasse `CIM_NetworkAdapter`, die als Spezialisierung der Klasse `CIM_LogicalDevice` des Core-Modells gebildet wird. Hiervon wird wiederum die Klasse `CIM_TokenRingAdapter` abgeleitet. Die abgeleitete Klasse erfährt dabei eine Veränderung eines `Description` Qualifiers und eine Ergänzung eines Attributes `MaxDataSize`.

```
// ==================================================================
// Title:     Devices MOF specification 2.1.1
// Filename:  CIM_Device21.mof
// Author:    DMTF System and Devices Working Group
// Description: The object classes below are listed in an order that
//              avoids forward references. Required objects, defined
//                  by other working groups, are omitted.

// ...

// ==================================================================
// NetworkAdapter
// ==================================================================
   [Abstract,
    Description (
```

```
       "NetworkAdapter is a superclass for grouping the numerous types "
       "of network adapters in use.")
   ]
class CIM_NetworkAdapter : CIM_LogicalDevice
{
      [MaxLen (64), Description (
          "PermanentAddress defines the network address hardcoded into "
          "an adapter.  This 'hardcoded' address may be changed via "
          "firmware upgrade or software configuration. If so, this field "
          "should be updated when the change is made.  PermanentAddress "
          "should be left blank if no 'hardcoded' address exists for the "
          "NetworkAdapter."),
       MappingStrings {"MIF.DMTF|Network Adapter 802 Port|001.2"}
        ]
    string PermanentAddress;
        [MaxLen (64),
         Description (
          "An array of strings indicating the network addresses for an "
          "adapter."),
         ArrayType ("Indexed"),
         MappingStrings {"MIF.DMTF|Network Adapter 802 Port|001.3"}
      ]
    string NetworkAddresses[];
        [Description(
          "An estimate of the current bandwidth in Bits per Second. "
          "For endpoints which vary in bandwidth or for those where "
          "no accurate estimation can be made, this property should "
          "contain the nominal bandwidth."),
         Units ("Bits per Second"),
         MappingStrings {"MIB.IETF|RFC1213-MIB.ifSpeed",
               "MIF.DMTF|Network Adapter 802 Port|001.5"}
        ]
    uint64 Speed;
        [Description (
          "The maximum speed, in Bits per Second, for the Network"
          "Adapter."),
         Units ("Bits per Second")
        ]
    uint64 MaxSpeed;
        [Description (
          "A boolean indicating whether the NetworkAdapter is capable "
          "of automatically determining the speed of the attached "
          "network media.")
        ]
    boolean AutoSense;
};

// ...

// ===================================================================
// TokenRingAdapter
// ===================================================================
   [Description ("Capabilities and management of a TokenRingAdapter.")]
class CIM_TokenRingAdapter : CIM_NetworkAdapter
{
        [Override ("NetworkAddresses"),
         Description (
          "Token Ring/802.5 MAC addresses formatted as twelve "
          "hexadecimal digits (e.g. \"010203040506\"), with each pair "
          "representing one of the six octets of the MAC address in "
          "\"canonical\" bit order.  (Thus, the Group address bit is "
          "found in the low order bit of the first character of the "
          "string.)"),
```

```
         ArrayType ("Indexed")
        ]
    string NetworkAddresses[];
        [Description (
          "The maximum size of the INFO (non-MAC) field that will be "
          "received or transmitted."),
         Mappingstrings {"MIB.IETF|BRIDGE-MIB.dot1dTpPortMaxInfo"}
        ]
    uint32 MaxDataSize;
};

// ...

// ====================================================================
// end of file
// ====================================================================
```

Eine Instanziierung eines `CIM_TokenRingAdapter` Objektes könnte nun wie folgt aussehen:

```
instance of CIM_TokenRingAdapter
{
    PermanentAddress = "" ;
    NetworkAddresses = { "010203040506" };
    Speed            = 4000000;
    MaxSpeed         = 4000000;
    AutoSense        = False;
    MaxDataSize      = 42;
}
```

Das zweite Beispiel zeigt die etwas umfangreichere Klasse `CIM_OperatingSystem`. Die anschließende Assoziation `CIM_InstalledOS` stellt als Ableitung von `CIM_SystemComponent` eine ‘enthalten-in’ Beziehung zwischen einem `CIM_OperatingSystem` Objekt und einem (nicht abgedruckten) `CIM_ComputerSystem` Objekt dar.

```
// ====================================================================
// Title:    System MOF specification 2.1
// Filename:  CIM_System21.mof
// Author:    DMTF System and Devices Working Group
// Description: The object classes below are listed in an order that
//              avoids forward references. Required objects, defined
//     by other working groups, are omitted.

// ...

// ====================================================================
// OperatingSystem
// ====================================================================
   [Description (
       "An OperatingSystem is software/firmware that makes a "
       "ComputerSystem's hardware usable, and implements and/or "
       "manages the resources, file systems, processes, user "
       "interfaces, services, ... available on the ComputerSystem.")
   ]
class CIM_OperatingSystem : CIM_LogicalElement
{

// ...

   string Name;
```

```
      [Description (
           "A integer indicating the type of OperatingSystem."),
       Values {"Unknown", "Other", "MACOS", "ATTUNIX", "DGUX",
             "DECNT", "Digital Unix", "OpenVMS", "HPUX", "AIX",
             "MVS", "OS400", "OS/2", "JavaVM", "MSDOS", "WIN3x",
             "WIN95", "WIN98", "WINNT", "WINCE", "NCR3000",
             "NetWare", "OSF", "DC/OS", "Reliant UNIX",
             "SCO UnixWare", "SCO OpenServer", "Sequent",
             "IRIX", "Solaris", "SunOS", "U6000", "ASERIES",
             "TandemNSK", "TandemNT", "BS2000", "LINUX",
             "Lynx", "XENIX", "VM/ESA", "Interactive UNIX",
             "BSDUNIX", "FreeBSD", "NetBSD", "GNU Hurd", "OS9",
             "MACH Kernel", "Inferno", "QNX", "EPOC", "IxWorks",
             "VxWorks", "MiNT", "BeOS", "HP MPE", "NextStep",
             "PalmPilot", "Rhapsody"},
       ModelCorrespondence {"CIM_OperatingSystem.OtherTypeDescription"}
      ]
   uint16 OSType;
      [Description (
  "A string describing the manufacturer and OperatingSystem "
  "type - used when the OperatingSystem property, OSType, is "
  "set to 1 (\"Other\"). The format of the string inserted "
  "in OtherTypeDescription should be similar in format to the "
  "Values strings defined for OSType.  OtherTypeDescription "
  "should be set to NULL when OSType is any value other than 1."),
 MaxLen (64),
 ModelCorrespondence {"CIM_OperatingSystem.OSType"}
]
   string OtherTypeDescription;
      [Description(
  "A string describing the Operating System's version "
  "number. The format of the version information is as follows: "
  "<Major Number>.<Minor Number>.<Revision> or "
  "<Major Number>.<Minor Number>.<Revision Letter>."),
       MappingStrings {"MIF.DMTF|Operating System|001.3"}
      ]
   string Version;
      [Description ("Time when the OperatingSystem was last booted."),
       MappingStrings {"MIF.DMTF|General Information|001.5"}
      ]
   datetime LastBootUpTime;

// ...

      [Description (
  "Requests a shutdown of the OperatingSystem. The return "
  "value should be 0 if the request was successfully executed, "
  "1 if the request is not supported and some other value if "
  "an error occurred. It is up to the implementation or subclass "
  "of OperatingSystem to establish dependencies between "
         "the Shutdown and Reboot methods, and for example, to provide "
         "more sophisticated capabilities such as scheduled shutdown/"
         "reboot, etc.")
      ]
   uint32 Shutdown();
};

// ==================================================================
// InstalledOS
// ==================================================================
   [Association,
    Aggregation,
    Description (
```

```
       "A link between the ComputerSystem and the OperatingSystem(s) "
       "installed or loaded on it. An OperatingSystem is 'installed' "
       "on a ComputerSystem, when placed on one of its StorageExtents "
       "- for example, copied to a disk drive or downloaded to "
       "Memory. Using this definition, an OperatingSystem is "
       "'installed' on a NetPC when downloaded via the network.")
   ]
class CIM_InstalledOS : CIM_SystemComponent
{
      [Override ("GroupComponent"),
       Aggregate,
       Description ("The ComputerSystem."),
       Min(1), Max (1)
      ]
   CIM_ComputerSystem REF GroupComponent;
      [Override ("PartComponent"),
       Description (
          "The OperatingSystem installed on the ComputerSystem."),
       Weak
      ]
   CIM_OperatingSystem REF PartComponent;
      [Description (
          "Boolean indicating that the OperatingSystem is the default "
          "OS for the ComputerSystem."),
       MappingStrings {"MIF.DMTF|Operating System|001.4"}
      ]
   boolean PrimaryOS;
};

// ...

// ==================================================================
// end of file
// ==================================================================
```

Instanzen dieser zwei Klassen könnten wie folgt aussehen:

```
instance of CIM_OperatingSystem
{

// ...

    Name                 = "Solaris-2.5.1" ;
    OSType               = "Solaris" ;
    OtherTypeDescription = "" ;
    Version               = "2.5.1" ;
    LastBootUpTime        = "19981111100000.000000+060" ;

// ...

}

instance of CIM_InstalledOS
{
    GroupComponent        = "//root:CIM_ComputerSystem.Name=\"mumm.ibr.cs.tu-bs.de\"" ;
    PartComponent         = "//root:CIM_OperatingSystem.Name=\"Solaris-2.5.1\"" ;
    PrimaryOS             = True ;
}
```

2.3.4. Bewertung

Viele Aspekte lassen CIM als ein modernes und nach aktuellen Kenntnissen durchdachtes Informationsmodell erscheinen: Die durchgehende Objektorientierung läßt es auch für einen großen Umfang an Objekten hierarchisch übersichtlich erscheinen. Die Verwendung von Unicode gewährleistet die Internationalisierung. Die Basistypen umfassen im Gegensatz zur SMI auch Fließpunktzahlen. Durch die Qualifier-Notation besteht eine deutliche Trennung zwischen der reinen Modellierung der Datenstrukturen, die an die standardisierte IDL angeleht ist, und zusätzlichen Meta-Informationen. Die Anlehnung an die IDL erleichtert die Entwicklung von Compilern und erhöht bei vielen Entwicklern und Lesern das intuitive Verständnis von Definitionen.

Da CIM die grundlegenden Klassen in drei Schemata bereits definiert und einige Richtlinien für MOF-Autoren festlegt, ist ein Weg für die Entwicklung vorgezeichnet, um einem Wildwuchs an Klassendefinitionen vorzubeugen.

Diese Trennung ist aber bereits in den durch die DMTF spezifizierten MOF-Modulen nicht realisiert. Beispielsweise enthalten die *Application MOF Specification 2.1* und die *Devices MOF Specification 2.1.1* zahlreiche Informationen, die konkrete Technologien, Hersteller und Produkte beinhalten, beispielsweise statische Listen von bekannten Betriebssystemen, Prozessortypen, Druckbeschreibungssprachen und dergleichen mehr. Hier wurden zu viele Informationen, die sich laufend ändern werden, in sehr wenigen Modulen zusammengefaßt. Eine Etablierung einer Version eines solchen Dokumentes über eine längere Zeit ist somit praktisch ausgeschlossen.

CIM ermöglicht einerseits durch Vorschriften zu Schreibweisen die Einbettung in verschiedene Umgebungen wie die Programmiersprachen C/C++, Java und in verschiedene Skriptsprachen. Andererseits berücksichtigt CIM mit seinen Mapping-Möglichkeiten die Einbettung von existierenden Management-Modellen.

Betrachtet man die bisherigen Entwürfe von MOF-Files der DMTF, so fallen einige zu bemängelnde Details auf: Manche Typen von Properties in CIM-Spezifikationen sind nicht kompatibel zu entsprechenden SMI-Definitionen, obwohl ein SNMP-Mapping definiert ist. Beispielsweise ist in Abschnitt 2.3.3 `CIM_NetworkAdapter.Speed` als `uint64` definiert. Der aus dem `MappingString` hervorgehende Objekt-Typ `ifSpeed` der `RFC1213-MIB` ist jedoch vom Typ `Gauge`. An anderen Stellen sind Formate von String-Properties nicht dokumentiert, etwa im Falle von `CIM_NetworkAdapter.PermanentAddress`. Ein drittes in Abschnitt 2.3.3 ersichtliches Beispiel ist die Methode `CIM_OperatingSystem.Shutdown()`: hier ist fraglich, wie diese Methode wie im `Description`-Qualifier dokumentiert im Falle eines erfolgreichen Herunterfahrens einen Rückgabewert von Null liefern soll.

Fragwürdig ist, ob die Vielfalt an Qualifiern und deren Wechselwirkungen und sogar ihre Erweiterung durch den Benutzer nicht eher den Überblick und die Interoperabilität von Implementierungen behindert. Denkt man an SNMP, so war es zum Teil sicherlich gerade die Einfachheit, die es zum Erfolg geführt hat.

Da Veränderungen an Modellen unausweichlich scheinen, ist zudem das Fehlen strenger Regeln bei Veränderungen von Objekt-Spezifikationen zu bemängeln. Inkompatibilitäten zwischen Versionen zuzulassen und lediglich zu fordern, eine Versionsnummer zu erhöhen, scheint hier nicht ausreichend, um Konflikten bei verschiedenen kommunizierenden Versionen und beim Mapping sich ggf. ändernder Modelle zu begegnen.

Die konkrete Integration von SNMP-Technologie konnte bis heute nicht eindeutig geklärt werden, obwohl dies aufgrund der breiten SNMP-Basis für den Erfolg von CIM sehr wichtig sein dürfte. An diesem Mangel, der spätestens seit dem Scheitern der Zusammenarbeit mit der IETF bekannt ist, hat sich seit Anfang 1997 nichts geändert.

Weitere kritisierte Punkte an der CIM-Technologie nennt ein Artikel von Amy K. Larsen [51] mit Stimmen von Jeff Case, Steve Waldbusser u.a.: Es fehlten Spezifikationen für die Datenbank-Strukturen von Management-Daten und für festgeschriebene APIs, um die Interaktion zwischen Management-Applikationen zu ermöglichen. Ohne solche Standards drohe eine vermeidbare parallele Entwicklung proprietärer Methoden. Da die Integration von SNMP-Technologie zwingend notwendig erscheint, stelle die CIM-Technologie eine zusätzliche Schicht über dem SNMP-Modell dar, die zusätzliche Probleme schaffe. Außerdem steige durch das teilweise mehrschichtige Mapping die Gefahr des Verlustes von Informationen.

2.4. OMG IDL

2.4.1. Die Object Management Group und die IDL

Die *Object Management Group* (OMG, [53]) wurde 1989 von acht Firmen gegründet. Ihr Ziel ist die Erarbeitung und Etablierung herstellerunabhängiger Standards für die Entwicklung verteilter objektorientierter Systeme. Ein zentrales Element der *Object Management Architecture* (OMA) stellt der *Object Request Broker* (ORB) dar. Daher wird das gesamte Modell meist als *CORBA* (Common Object Request Broker Architecture) bezeichnet. CORBA 1.1 wurde 1991 verabschiedet. 1994 folgte CORBA 2.0. Die aktuellen Spezifikationen für CORBA 2.2 stammen von 1998.

Derzeit umfaßt die OMG weltweit über 800 Mitglieder. Eine Übersicht der OMA und Details einzelner Komponenten und Implementierungen sind der entsprechenden Literatur zu entnehmen, etwa [54], [55] oder anderen Dokumenten unter [53]. Eine sehr umfangreiche Literaturübersicht bietet die OMG selbst in [56].

An dieser Stelle wird lediglich der Teil der OMA betrachtet, der für die Interoperabilität von Objekt-Implementierungen und Applikationen grundlegend ist: die *Interface Definition Language* (IDL). Diese Sprache dient der Definition der Schnittstellen von Objekten. Client-seitige *Stubs* und server-seitige *Skeletons* einer Interface-Definition können auf diverse Programmiersprachen abgebildet werden. Dadurch ist die Möglichkeit des Zusammenarbeitens von Objekt-Implementierungen und Clients auf verschiedenen Architekturen und in verschiedenen Programmiersprachen gegeben.

Obwohl die OMA nicht primär für den Bereich des Netzwerk-Managements entwickelt wurde, sind dennoch zumindest Teile der Architektur flexibel genug, um auch für Management-Aufgaben geeignet zu sein. Beispielsweise befaßt sich das *Joint Inter-Domain Management* Projekt (JIDM, [57], [58], [59]) mit der Einbettung existierender Netzwerk-Management Modelle wie dem OSI Management und SNMP in CORBA. Im Rahmen der folgenden Abschnitte wird die IDL näher beschrieben, um ihre Eignung zur Beschreibung von Management-Informationen zu untersuchen. Im Konzept von JIDM stellt bei der Benutzung von SMI-Definitionen in CORBA-Umgebungen die IDL lediglich einen Zwischenschritt dar. SMI MIBs werden nach JIDM-Regeln in IDL konvertiert, und diese werden wiederum von CORBA-IDL-Compilern gelesen. In diesem Fall werden Objekte nicht in der IDL modelliert, wenngleich dies in anderen Fällen möglich wäre.

2.4.2. Objektmodell

Das Informationsmodell der OMA basiert auf dem Konzept der Objektorientierung mit seinen typischen Merkmalen der Kapselung durch Klassenbildung, Vererbung (auch Mehrfachvererbung) und der dadurch gewonnenen Strukturierbarkeit. Ererbte Elemente dürfen nicht verändert werden. So wird gewährleistet, daß Clients, die über eine Basis-Interface-Definition mit einem Objekt kommunizieren, auch unverändert mit Spezialisierungen dieses Objektes kommunizieren können.

Da das Modell nicht speziell auf Managementinformationen zugeschnitten ist, sind entsprechende spezialisierte Elemente nicht im Modell vorhanden, etwa verfeinerte Zugriffsmöglichkeiten wie `accessible-for-notify` in der SMI oder Elemente zur Modellierung von Notifications. Derartige Eigenschaften müssen mit anderen IDL-Mitteln erst modelliert werden.

Alle Definitionen gelten zunächst nur in den sie umgebenden Sichtbarkeitsbereichen (*Scoping*). Solche Bereiche werden durch *Modules*, *Interfaces*, *Structs*, *Unions*, *Operations* und *Exceptions* gebildet (s.u.). Außerhalb des umgebenden Bereiches sind Definitionen nur durch den *Scoping Operator* (‘`::`’) sichtbar zu machen.

2.4.3. Sprachelemente

Die IDL ist zwar als programmiersprachen-unabhängig anzusehen, viele Eigenschaften sind jedoch an C++ angelehnt:

- Kommentare können durch ein ‘/*’...‘*/’-Paar eingeschlossen werden oder in einer einzelnen Zeile durch ‘//’ eingeleitet werden und bis zum Zeilenende reichen.
- Ein weitreichend an C++ angelehntes Preprocessing kann benutzt werden: Die Anweisungen `#define`, `#ifdef`, `#ifndef`, `#else`, `#endif`, `#include`, `#pragma` sind in gewohnter Weise anwendbar.
- Lexikalische Elemente und syntaktische Grundregeln sind weitestgehend identisch: Blöcke werden in geschweiften Klammern eingeschlossen. Anweisungen werden durch Semikolon getrennt. Die Menge von Schlüsselworten für Definitionen und Typen ist in großem Umfang mit C++ identisch.

Da es sich bei der IDL um eine reine Datenmodellierungssprache handelt, sind Variablen und algorithmische Konstrukte nicht vorhanden. Um die Abbildung von IDL-Objekten auch auf Programmiersprachen ohne Unterscheidung von Groß-/Kleinschreibung zu ermöglichen, dürfen sich Identifier nicht lediglich in der Groß-/Kleinschreibung unterscheiden.

Die gesamte Grammatik der IDL ist in Anhang D angegeben. Die folgenden Erläuterungen verweisen auf einzelne Regeln dieser Grammatik.

2.4.3.1. IDL-Struktur

Äußerstes Strukturierungselement der IDL ist das Modul (`module`)[5]. Ein Modul faßt zusammengehörige Definitionen zusammen und bildet einen durch einen Namen referenzierbaren Sichtbarkeitsbereich (*Scoping*). Die Syntax einer Moduldefinition zeigt Anhang D ab Regel 3.

Durch das `interface` Konstrukt werden Schnittstellen von Objekten modelliert und mit einem Namen versehen. Eine solche Schnittstelle besteht aus den für das Objekt spezifischen Definitionen von Typen, Konstanten, Exceptions, Attributen und Operationen (s.u.). Schnittstellen können von anderen Schnittstellen ererbt werden. Auch Mehrfachvererbung ist möglich. Eine Interface-Definition bildet einen Sichtbarkeitsbereich. Unter Umständen können Vorwärtsdeklarationen von Schnittstellen erforderlich sein. Die Syntax geht aus Anhang D ab Regel 5 hervor.

[5] Außerhalb von Modulen können jedoch auch globale Definitionen vorgenommen werden.

Wie in C/C++ können durch `const` Konstanten und durch `typedef` neue Typen definiert werden (siehe Anhang D, Regeln 12 und 27). Zur Bildung konstanter Ausdrücke existiert eine Vielzahl arithmetischer Operatoren, die der Grammatik zu entnehmen sind.

Die Definition einer Operation besteht aus einem Rückgabetyp, dem Namen der Operation und einer durch Klammern eingeschlossene Liste von Parametern. Der Rückgabetyp muß vorhanden sein. Gibt es keinen Rückgabewert, ist die Operation als `void` zu definieren. Die einzelnen durch Kommata getrennten Parameter bestehen aus einem Attribut, das die 'Richtung' des Parameters anzeigt (`in`, `out` oder `inout`), einem Parametertyp und dem Namen des Parameters. Die Parameterliste kann auch leer sein. Die Syntax von Operationen ist in Anhang D ab Regel 67 zu sehen.

Der gesamten Operation kann optional das Schlüsselwort `oneway` vorangestellt werden. Eine so als asynchron definierte Operation darf weder einen Rückgabewert liefern noch `out`- oder `inout`-Parameter aufweisen.

Der Definition einer Operation kann sich optional das Schlüsselwort `raises` gefolgt von einer in Klammern stehenden und durch Kommata getrennten Liste von zuvor definierten *Exceptions* anschließen. Dies zeigt an, daß die jeweilige Operation diese Exceptions auslösen kann. Name und Umfang einer Exception sind zuvor durch das `exception` Konstrukt zu definieren (siehe Anhang D, Regeln 73 und 66). Einige Exceptions sind bereits durch die OMG definiert.

Nach dem optionalen `raises` Konstrukt kann sich ebenfalls optional ein `context` Konstrukt anschließen. Hierdurch kann ähnlich einem Environment eines Standard-Betriebssystems eine Menge von Variablen in Form von Name-Wert-Paaren der Operation zugänglich gemacht werden. Für Unterstützung von Kontexten ist maßgeblich der ORB verantwortlich. Dieser Mechanismus ist in dieser Betrachtung des Informationsmodells nicht von Bedeutung.

Instanzen von Objekten sind durch Zustände ausgeprägt. Diese Zustände spiegeln sich in Variablen wider, auf die durch lesende und schreibende Operationen zugegriffen wird. Dies kann anstatt durch explizite Formulierung zweier Operationen auch vereinfacht durch das `attribute` Konstrukt notiert werden: Die Definition eines Attributs A vom Typ T entspricht der Definition einer `_get_`A Operation ohne Parameter und vom Rückgabetyp T und einer `_set_`A void-Operation mit einem `in`-Parameter vom Typ T. Ist keine `_set_` Operation gefordert, kann der Attributdefinition das Schlüsselwort `readonly` vorangestellt werden. Die genaue Syntax kann Anhang D ab Regel 65 entnommen werden.

2.4.3.2. Typen

Die IDL-Basistypen sind teilweise an C/C++ angelehnt. Im einzelnen gehen sie aus Tabelle 2.1 hervor.

Darüber hinaus gibt es komplexe Datenstrukturen, die ebenfalls an C/C++ angelehnt sind:

Durch `struct` können zusammengesetzte Datenstrukturen modelliert und unter einem neuen Typnamen referenziert werden (Anhang D, Regel 50).

Das `union` Konstrukt weicht etwas von der gleichnamigen C-Anweisung ab. Kombiniert mit einem `switch`-Anteil enthält es neben den Alternativen eine Variable, die zur Auswahl der jeweiligen Alternative dient. Diese muß von einem der ersten sechs Typen der obigen Tabelle 2.1 oder ein `enum` Typ (s.u.) sein. Die genaue Syntax geht aus Anhang D ab Regel 53 hervor.

short	Integer-Wertebereich: -2^{15} bis $2^{15} - 1$
unsigned short	Integer-Wertebereich: 0 bis $2^{16} - 1$
long	Integer-Wertebereich: -2^{31} bis $2^{31} - 1$
long long	Integer-Wertebereich: -2^{63} bis $2^{63} - 1$
unsigned long	Integer-Wertebereich: 0 bis $2^{32} - 1$
unsigned long long	Integer-Wertebereich: 0 bis $2^{64} - 1$
char	ISO Latin-1 (8859.1) / ASCII (ISO 646) Zeichen
wchar	UCS-kodierte *wide characters*
boolean	Wertemenge: TRUE, FALSE
float	IEEE single-precision Fließpunktzahl [60]
double	IEEE double-precision Fließpunktzahl [60]
long double	IEEE quadruple-precision Fließpunktzahl [60]
octet	8-Bit-Wert ohne jede Interpretation
any	kann jeden IDL-Typ annehmen
Object	Objekt-Referenzen

Tabelle 2.1.: IDL Basistypen

Durch `enum` können Aufzählungstypen mit bis zu 2^{32} Werten modelliert werden. Durch die Reihenfolge der definierten Werte ist eine Ordnung auf diesen Werten festgelegt. Diese Ordnung kann in Sprachen, die einen Vergleich von Werten eines Aufzählungstyps erlauben, benutzt werden. Eine konkrete Zuordnung von Aufzählungswerten zu Integer-Werten wie z.B. in der SMI ist nicht möglich. Die genaue Syntax des `enum` Konstruktes geht aus Anhang D ab Regel 59 hervor.

Durch `sequence` Konstrukte können Folgen von Werten eines Typs definiert werden. Ein solcher Typ ist von fester Größe. Die maximale Länge der Folge kann entweder durch Angabe eines optionalen konstanten Wertes begrenzt sein (*Bounded Sequence*) oder unbegrenzt sein und erst zur Laufzeit etwa bei Parameterübergaben ermittelt werden (*Unbounded Sequence*). Eine Schachtelung von Folgen ist möglich (Folgen von Folgen). Die Syntax ist Anhang D ab Regel 61 zu entnehmen.

`strings` und `wstrings` ähneln einer Folge (`sequence`) von Zeichen (`char` bzw. `wchar`). Da viele Systeme optimierte Funktionen zur Manipulation von Strings bieten, wurden Strings auch in der IDL separat definiert. Die Syntax von Stringdefinitionen geht aus Anhang D ab Regel 62 hervor.

Felder beliebiger Dimension können durch das `array` Konstrukt modelliert werden. Die Feldgrößen jeder Dimension sind konstant und durch je einen Integer-Wert anzugeben. Beliebige Wertebereiche zur Indizierung sind nicht möglich. Da die Implementierung der Feldindizierung von der Abbildung auf die jeweilige Programmiersprache abhängt, ist die Kompatibilität von Indizes als übergebene Parameter nicht gewährleistet. Die Syntax von Felddefinitionen geht aus Anhang D ab Regel 63 hervor.

2.4.4. Beispiele

Das erste Beispiel zeigt ein File mit einer IDL-Spezifikation, wie sie nach den JIDM-Übersetzungsregeln aus der bereits aus Abschnitt 2.1.4 bekannten Interface-Tabelle (`ifTable`) der IF-MIB hervorgeht. Als

Compiler diente `smi2idl` [61]. Das Beispiel ist stark gekürzt, da nur einige markante Strukturen an dieser Stelle interessant sind.

```
/* ********************************************************
 * This IDL file is generated from ifmib2-00.txt on Dec  9 1998
 *
 * SMI to IDL compiler developed at Lucent Technologies Inc.
 * smi2idl version 1.1.0 built on: Mar 20 1998
 * Please send bug reports to hsarin@lucent.com
 *********************************************************/

#ifndef _IF_MIB_IDL_
#define _IF_MIB_IDL_

#include <ASN1Types.idl>
#include <SNMPMgmt.idl>
#include <SNMPv2_SMI.idl>
#include <SNMPv2_TC.idl>
#include <SNMPv2_CONF.idl>
#include <SNMPv2_MIB.idl>
#include <IANAifType_MIB.idl>

module IF_MIB {

    typedef SNMPv2_SMI::Counter32Type Counter32Type;
    typedef SNMPv2_SMI::Gauge32Type Gauge32Type;
    typedef SNMPv2_SMI::Integer32Type Integer32Type;
    typedef SNMPv2_TC::DisplayStringType DisplayStringType;
    typedef SNMPv2_TC::PhysAddressType PhysAddressType;
    typedef SNMPv2_TC::TimeStampType TimeStampType;

    // ...

    typedef Integer32Type InterfaceIndexType;

    // ...

    interface interfaces : SNMPMgmt::SmiEntry {
        /*
        DESCRIPTION:"The number of network interfaces (regardless of their
    current state) present on this system."
        */
        readonly attribute Integer32Type ifNumber;

    };

    // ...

    /*
    DESCRIPTION:"A list of interface entries.  The number of entries is
    given by the value of ifNumber."
    */
    const string ifTable= "::IF_MIB::ifTable";

    /*
    DESCRIPTION:"An entry containing management information applicable to a
    particular interface."
    */
    interface ifEntry : SNMPMgmt::SmiEntry {
        /*
        INDEX : { ifIndex}
        */
```

```
    const string EntryIndexVarList = "ifIndex";
    /*
    DESCRIPTION:"A unique value, greater than zero,for each interface.  It
is recommended that values are assigned contiguously
starting from 1.  The value for each interface sub-layer
must remain constant at least from one re-initialization of
the entity's network management system to the next re-
initialization."
    */
    readonly attribute InterfaceIndexType ifIndex;

    /*
    DESCRIPTION:"A textual string containing information about the
interface. This string should include the name of the
manufacturer, the product name and the version of the
interface hardware/software."
    */
    readonly attribute DisplayStringType ifDescr;

    /*
    DESCRIPTION:"The type of interface.  Additional values for ifType are
assigned by the Internet Assigned Numbers Authority (IANA),
through updating the syntax of the IANAifType textual
convention."
    */
    readonly attribute Integer32Type ifType;

    /*
    DESCRIPTION:"The size of the largest packet which can be sent/received
on the interface, specified in octets.  For interfaces that
are used for transmitting network datagrams, this is the
size of the largest network datagram that can be sent on the
interface."
    */
    readonly attribute Integer32Type ifMtu;

    /*
    DESCRIPTION:"An estimate of the interface's current bandwidth in bits
per second.  For interfaces which do not vary in bandwidth
or for those where no accurate estimation can be made, this
object should contain the nominal bandwidth.  If the
bandwidth of the interface is greater than the maximum value
reportable by this object then this object should report its
maximum value (4,294,967,295) and ifHighSpeed must be used
to report the interace's speed.  For a sub-layer which has
no concept of bandwidth, this object should be zero."
    */
    readonly attribute Gauge32Type ifSpeed;

    /*
    DESCRIPTION:"The interface's address at its protocol sub-layer.  For
example, for an 802.x interface, this object normally
contains a MAC address.  The interface's media-specific MIB
must define the bit and byte ordering and the format of the
value of this object.  For interfaces which do not have such
an address (e.g., a serial line), this object should contain
an octet string of zero length."
    */
    readonly attribute PhysAddressType ifPhysAddress;

    typedef ASN1_Integer IfAdminStatusType;
    const IfAdminStatusType up = 1;
    const IfAdminStatusType down = 2;
```

```
    const IfAdminStatusType testing = 3;

    /*
    DESCRIPTION:"The desired state of the interface.  The testing(3) state
indicates that no operational packets can be passed.  When a
managed system initializes,all interfaces start with
ifAdminStatus in the down(2) state.  As a result of either
explicit management action or per configuration information
retained by the managed system, ifAdminStatus is then
changed to either the up(1) or testing(3) states (or remains
in the down(2) state)."
    */
    attribute IfAdminStatusType ifAdminStatus;

    typedef ASN1_Integer IfOperStatusType;
    const IfOperStatusType up_1 = 1;
    const IfOperStatusType down_1 = 2;
    const IfOperStatusType testing_1 = 3;
    const IfOperStatusType unknown = 4;
    const IfOperStatusType dormant = 5;
    const IfOperStatusType notPresent = 6;
    const IfOperStatusType lowerLayerDown = 7;

    // ...

};

// ...

    struct IfIndexType {
     ASN1_ObjectIdentifier var_name;
     ASN1_ObjectIdentifier var_index;
     InterfaceIndexType var_value;
    };
    struct IfAdminStatusType {
     ASN1_ObjectIdentifier var_name;
     ASN1_ObjectIdentifier var_index;
     IF_MIB::ifEntry::IfAdminStatusType var_value;
    };
    struct IfOperStatusType {
     ASN1_ObjectIdentifier var_name;
     ASN1_ObjectIdentifier var_index;
     IF_MIB::ifEntry::IfOperStatusType var_value;
    };
    struct LinkDownType {
     IfIndexType ifIndex;
     IfAdminStatusType ifAdminStatus;
     IfOperStatusType ifOperStatus;
    };

    // ...

interface SnmpNotifications { // for typed-push event communication
    /*
    DESCRIPTION:"A linkDown trap signifies that the SNMPv2 entity, acting in
an agent role, has detected that the ifOperStatus object for
one of its communication links is about to enter the down
state from some other state (but not from the notPresent
state).  This other state is indicated by the included value
of ifOperStatus."
    */
    void linkDown(
     in ASN1_ObjectIdentifier src_party,
```

```
         in ASN1_ObjectIdentifier snmp_context,
         in SNMPv2_TC::TimeStampType event_time,
         in LinkDownType notification_info
        );

        // ...

    };

    interface PullSnmpNotifications { // for typed-pull event communication
        void try_linkDown(
         out ASN1_ObjectIdentifier src_party,
         out ASN1_ObjectIdentifier snmp_context,
         out SNMPv2_TC::TimeStampType event_time,
         out LinkDownType notification_info
        );
        void pull_linkDown(
         out ASN1_ObjectIdentifier src_party,
         out ASN1_ObjectIdentifier snmp_context,
         out SNMPv2_TC::TimeStampType event_time,
         out LinkDownType notification_info
        );

        // ...

    };
};

#endif /* !_IF_MIB_IDL_ */
```

Das zweite Beispiel zeigt ein (fiktives) Modell einer Routing Tabelle. Die Stärken des Objektmodells werden hier in zwei Punkten deutlich: Zum einen stellen hier Methoden Operationen zur Verfügung, die sonst nur seitens eines Managers und mit höherem Kommunikationsaufwand an der Objekt- bzw. Agent/Manager-Schnittstelle zu erfüllen wären. Zum anderen ist durch Vererbung eine Spezialisierung von Objekten möglich.

```
#include <Net.idl>
#include <IP.idl>

module Routing {

    interface RoutingTableEntry;    // forward reference
    interface AdvRoutingTableEntry; // forward reference

    interface RoutingTable {

        RoutingTableEntry    findRoute(in IP::IpAddress destination);
        AdvRoutingTableEntry findAdvRoute(in IP::IpAddress destination,
                                          in IP::Port destinationPort,
                                          in IP::IpAddress source,
                                          in IP::Port sourcePort);

    };

    interface RoutingTableEntry {

        attribute IP::IpAddress   destination;
```

```
        attribute IP::IpAddress   genmask;
        attribute IP::IpAddress   gateway;
        attribute Net::Interface  iface;
        readonly unsigned long    ref;
        readonly unsigned long    use;

    };

    interface AdvRoutingTableEntry : RoutingTableEntry {

        attribute IP::Port        destinationPort;
        attribute IP::IpAddress   source;
        attribute IP::Port        sourcePort;
        attribute unsigned short  tos;

    };

};
```

2.4.5. Bewertung

Die OMG IDL hat wie die anderen betrachteten objektorientierten Informationsmodelle die typischen Vorteile der Strukturierbarkeit durch Kapselung und Vererbung. Durch mehrschichtige Scoping-Bereiche werden Namenskonflikte vermieden.

Da in Objekt-Schnittstellen Operationen definiert werden können, lassen sich komfortabel Funktionen beschreiben, die sich in der SMI nur durch aufwendige Verhaltensbeschreibungen mehrerer voneinander abhängiger MIB-Variablen modellieren lassen.

Seit der CORBA 2.2 IDL Spezifikation umfaßt die Menge der Basistypen alle erforderlichen Typen einschließlich Fließpunktzahlen verschiedener Auflösung sowie 64 Bit breite Integer-Typen. Auch die aktuellen JIDM Spezifikationen sind hierfür gerüstet.

Die Notation, die sich stark an C/C++ anlehnt, jedoch programmiersprachen-unabhängig bleibt, erleichtert einerseits das intuitive Verständnis von IDL-Spezifikationen und ermöglicht andererseits das Mapping auf verschiedene (sowohl objektorientierte als auch klassische imperative) Programmiersprachen, wie jene für C, C++, Java und Ada belegen.

Die IDL sieht keine Regelungen im Falle von Veränderungen von Spezifikationen vor. Durch Spezialisierungen bei der Vererbung von Schnittstellen zusammen mit der Maßgabe, daß Eigenschaften ererbter Elemente nicht verändert werden dürfen, ist aber eine Kompatibilität älterer Clients zu spezialisierten Objekt-Implementierungen gewährleistet.

In anderen Informationsmodellen gibt es Konstrukte, die der Verhaltensbeschreibung von Objekten dienen oder Hilfen für Management-Applikationen darstellen (z.B. `DESCRIPTION` und `DISPLAY-HINT` in SMIv2). Derartige Informationen können in der IDL nicht formuliert werden. Folglich übersetzen JIDM-Regeln diese Informationen lediglich in Kommentare. Aufgrund dieser eingeschränkten Möglichkeit wird man die IDL auch im allgemeinen nicht direkt nutzen wollen, um Management-Informationen zu spezifizieren.

Die IDL bietet keine spezielle Notation für asynchron zu versendende Notifications wie etwa im Falle der SMI. Diese Funktionalität muß durch Operationen modelliert werden, die entweder per *Polling* von

Clients durch `try_`- und `pull_`-Operationen genutzt werden oder vom Objekt selbst mittels `push_`-Operationen über den CORBA Event-Service vermittelt werden.

3. SMIng Entwurf

In diesem Kapitel werden zunächst Erkenntnisse über die in Kapitel 2 betrachteten Informationsmodelle klassifiziert und im einzelnen erörtert. Hinzu kommen einige in [6], [1] und [5] kritisierte Eigenschaften der SMIv1/v2 sowie weitere, die während der Arbeit aufgedeckt wurden. In jedem der folgenden Abschnitte werden einzelne Gesichtspunkte diskutiert und schließlich zu Entwurfsentscheidungen für SMIng verfeinert. Abschließend hervorgehobene Sätze nennen knapp die Vorteile der jeweiligen Entwurfsentscheidung.

Im Laufe der Entwicklung von SMIng gab es neben den hier aufgeführten und umgesetzten Aspekten einige weitere Überlegungen, die jedoch wegen inakzeptablen Inkompatibilitäten zur SMIv2 nicht in SMIng Einzug gehalten haben. Die wichtigsten nennt der letzte Abschnitt dieses Kapitels. Die insgesamt 54 diskutierten Punkte sind in [62] knapp dokumentiert.

Das Ergebnis sämtlicher SMIng Entwurfsaspekte ist die vollständige Spezifikation einer neuen Sprache zur Definition von Management-Informationen. Um diese Spezifikation nach den Regeln der *Internet Engineering Task Force* (IETF) bzw. der *Internet Research Task Force* (IRTF) publizieren und ggf. weiterentwickeln und standardisieren zu können [63], wurde sie als eigenständiges Dokument und nach entsprechenden formalen Maßgaben für RFCs (*Request for Comments*) der IETF [64] verfaßt. Es ist geplant, diese Spezifikation nach Abschluß der vorliegenden Arbeit im Rahmen der im März 1999 gegründeten IRTF *Network Management Research Group* (NMRG) [65] als öffentliches Arbeitsdokument (*Internet-Draft*) weiterzuentwickeln. Es trägt daher bereits jetzt den Titel „SMIng – A new Structure of Management Information" (`draft-irtf-nmrg-sming-00`).

Aufgrund des eigenständigen Formates und der Abgeschlossenheit des Dokumentes wurde es von diesem Kapitel in den Anhang E verlagert, obwohl es das zentrale Ergebnis dieser Arbeit darstellt.

Bei dem Entwurf einer formalen Sprache sind einige Kenntnisse aus dem Bereich des Compilerbaus von grundlegender Bedeutung. Entsprechende Kenntnisse und Begriffe werden in diesem Kapitel vorausgesetzt. Als einführende Literatur seien die ersten vier Kapitel von [66] empfohlen. Ferner setzen die folgenden Abschnitte detaillierte Kenntnisse über SMIv2 voraus, die diese Arbeit nur eingeschränkt in Abschnitt 2.1 vermitteln kann. Die SMIv2 Spezifikation ist auf drei Standard RFCs verteilt [2], [3], [4], als umfassend dokumentierende Lektüre sei [6] empfohlen.

3.1. Konsistente Struktur der Grammatik

Die grammatikalischen Grundstrukturen von ASN.1, die in SMIv1 und SMIv2 unverändert Anwendung finden, sehen anders als CORBA IDL oder CIM kein explizites Token vor, das einzelne Anweisungen bzw. Definitionen ohne Kenntnis des Kontextes eindeutig voneinander trennt. Dies macht es dem Leser ebenso wie dem Autor von SMIv1/v2 Parsern schwerer, die Strukturen von Definitionen zu erkennen und im Falle von syntaktischen Fehlern an geeigneter Stelle die Analyse fortzusetzen.

Zudem existieren atomare Sprachelemente, die aus mehreren Worten bestehen (z.B. `OBJECT IDENTIFIER`, `OCTET STRING`, `SEQUENCE OF`), so daß sich vermeidbare Fragen bei dem Entwurf von Parsern ergeben: Soll die lexikalische Analyse bereits solche Wortgruppen zu einem Token zusammenfassen oder soll die syntaktische Analyse entsprechende Folgen von Tokens erwarten? Auch für den Autor oder Leser tragen derartige Konstrukte nicht zum intuitiven Verständnis bei.

SMIng weist daher eine durchgängige und einfache syntaktische Struktur auf allen Ebenen von Definitionen auf. Die Basis aller Definitionen bildet eine Anweisung (*Statement*). Jede Anweisung wird durch ein klein geschriebenes Schlüsselwort eingeleitet. Es folgt eine Anzahl von Argumenten, gefolgt von einem abschließenden Semikolon. Das Semikolon dient nur diesem Zweck und kann somit als eindeutiges Trennzeichen zwischen Anweisungen interpretiert werden. Sämtliche SMIng Schlüsselworte (Anweisungen, Typen sowie spezielle Werte und Argumente) bestehen aus einfachen Worten.

Eine besondere Form von Argumenten sind Anweisungsblöcke, die eine Schachtelung von Anweisungen erlauben. Beispielsweise hat eine `module`-Anweisung zur Definition eines Moduls unter anderem ein Anweisungsblock-Argument, welches mehrere `revision`-Anweisungen enthalten kann. Diese wiederum tragen ein Anweisungsblock-Argument, das Anweisungen zur Angabe von Datum und Beschreibung der Revision enthält. Weitere Beispiele zeigen die Abbildungen 3.1 bis 3.3. Diese einfache Struktur wird konsistent in der gesamten SMIng Grammatik beibehalten.

Die so erreichten Ziele sind eine **leichtere Erlernbarkeit** durch eine **kleine Basis-Grammatik** und eine leichtere sog. **Error Recovery** bei der Entwicklung von SMIng Parsern.

Die Spezifikation (Anhang E) dokumentiert die Basis-Grammatik in Abschnitt 4.2.

3.2. Vermeidung von Vorwärtsreferenzen

Im Gegensatz zu CIM treffen die SMIv1/v2 Spezifikationen keine Aussage über die Rechtmäßigkeit der Verwendung von Vorwärtsreferenzen, also die Benutzung von Bezeichnern, die im selben Modul erst im folgenden definiert werden.

Während CIM gänzlich auf Vorwärtsreferenzen verzichten kann, muß im SMI Informationsmodell differenziert werden: Hier können Tabellen zur Indizierung Spaltenobjekte anderer Tabellen verwenden. Da die Definition von Spaltenobjekten an die der umgebenden Tabelle gebunden ist, können „Ringe“ von Tabellen somit nicht ausnahmslos ohne Vorwärtsreferenzen von Spaltenobjekten in Index-Angaben definiert werden. Den einfachsten Fall bildet ein Paar von Tabellen, die jeweils eine Spalte der anderen Tabelle zur Indizierung benutzen, wie das Beispiel in Abbildung 3.1 zeigt.

Auch die Indizierung durch Spaltenobjekte der eigenen Tabelle stellt in der Notation des Beispiels eine Vorwärtsreferenz dar. Um dies zu vermeiden, hätten die Index-Angaben im Anschluß an die Spaltendefinitionen positioniert werden können. Es wurde jedoch die zu sehende Anordnung gewählt, um die Index-Angaben nicht unübersichtlich weit vom Rest der Zeilendefinitionen zu entfernen.

Um Autoren von Management-Applikationen Informationen über die für zeilenerzeugende Operationen nötigen Spaltenobjekte zu geben, führt SMIng die `create`-Anweisung ein. Die darin aufgeführten

```
table table1 {
    ...
    row row1 {
        ...
        index (..., column2, ...);
        ...
        column column1 {
            ...
        };
        ...
    };
};

table table2 {
    ...
    row row2 {
        ...
        index (..., column1, ...);
        ...
        column column2 {
            ...
        };
        ...
    };
};
```

Abbildung 3.1.: Tabellen mit Vorwärtsreferenzen

Spaltenobjekte können wie bei der `index`-Anweisung ebenfalls Vorwärtsreferenzen enthalten.

> Abgesehen von zwei unvermeidbaren Ausnahmen, die sich auf die Definition von Tabellenzeilen beschränken, sind in SMIng **Vorwärtsreferenzen verboten**. Diese klare Anforderung **erleichtert die Implementierung** von SMIng Parsern.

Die Spezifikation (Anhang E) dokumentiert die Verwendung von Bezeichnern und ihr Auftreten in Vorwärtsreferenzen in Abschnitt 2.1.

3.3. Präzise Grammatik

Ein gravierender Nachteil von SMIv1/v2 ist das Fehlen einer vollständigen formalen Grammatik. Bei CIM und der CORBA IDL sind Grammatiken hingegen zentrale Bestandteile der Spezifikation (siehe Anhänge C und D).

Ein großer Teil der grammatikalischen Grundregeln von SMIv1/v2 entstammt der ASN.1 Spezifikation (*Abstract Syntax Notation One*) der ISO (*International Organization for Standardization*) aus dem Jahre 1987. Diese Regeln werden in den SMIv1/v2 Spezifikationen durch insgesamt zehn ASN.1 Makros und einen Pseudotyp[1] ergänzt und mit zahlreichen Einschränkungen und Ausnahmeregeln versehen. Dies erschwert die präzise Auslegung der gesamten Grammatik. Überdies ist die zugrunde liegende ASN.1 Spezifikation bei der ISO nicht mehr offiziell verfügbar. Eine Verwendung von ASN.1 Werkzeugen ist

[1] Der Pseudotyp `BITS` wird zur Übertragung in SNMP auf den ASN.1 Typ `OCTET STRING` abgebildet.

aufgrund der zahlreichen Ausnahmen nicht möglich. Infolge dieser Umstände existieren zahlreiche fehlerhafte MIB Module, auch unter den Standard MIBs der IETF. Da zudem die meisten existierenden SMIv1/v2 Parser aus demselben Grund viele Fehler nicht erkennen, fällt das Auffinden von Fehlern in MIB Modulen schwer.

Um vergleichbare Probleme bei SMIng zu vermeiden, enthält die Spezifikation eine vollständige formale Grammatik, die in der *Augmented Backus-Naur Form* (ABNF) [50] angegeben ist. Diese Notation hat zwei entscheidende Vorteile: Sie ist als *Standards Track* RFC in der IETF etabliert und somit uneingeschränkt verfügbar und im Umfeld der IETF als „bekannte“ Notation für Grammatiken anzusehen. Außerdem werden Grammatiken in ABNF Notation typischweise bis auf Zeichenebene angegeben, was eine sehr präzise und somit im Gegensatz zur SMIv1/v2 unmißverständliche Grammatik erlaubt. Beispielsweise können Konventionen für Kommentare, Zeilenende-Kodierungen sowie Leerzeichen und Tabulatoren für optionale und vorgeschriebene Zwischenräume genau beschrieben werden.

Die konsequente Verwendung der ABNF Notation führt zu einer **eindeutigen Grammatik** und ermöglicht ein **schematisches Erzeugen einer Grammatik für Parser-Generatoren**.

Die Spezifikation (Anhang E) enthält die SMIng ABNF Grammatik in Anhang A.

3.4. Unabhängigkeit von externen Standards

Die im vorangegangenen Abschnitt beschriebene Abhängigkeit der SMIv1/v2 von der ASN.1 Spezifikation, die nicht der IETF unterliegt, birgt Gefahren einer schlechteren Verfügbarkeit und einer Inkompatibilität durch unkontrollierbare Veränderungen der Spezifikation im Laufe der Zeit.

Diese Gefahren vermeidet SMIng durch eine weitgehende Unabhängigkeit von externen Dokumenten. Bei allen Referenzen, die für die Auslegung von Bedeutung sind, handelt es sich um Dokumente, die durch die IETF als RFCs publiziert sind. Die einzige Ausnahme bildet der IEEE-Standard für Fließpunktzahlen [60], der aber innerhalb der IETF „akzeptiert“ ist. Die für SMIng entscheidenden Informationen hierzu sind beispielsweise auch RFC 1832 [67], Abschnitt 3.6 bis 3.8, zu entnehmen.

An dieser Stelle sei erwähnt, daß heutzutage viele Dateiformate basierend auf XML (*Extensible Markup Language*, ein vereinfachter Ableger von SGML, der *Standard Generalized Markup Language*) als DTD (*Document Type Definition*) definiert werden. Ob dies für die Definition von Management-Informationen, wie es auch in der DMTF für CIM angestrebt wird, eine geeignete Methode ist, darf bezweifelt werden. Einerseits ist XML bislang keineswegs als abgeschlossener Standard anzusehen. Es ist in den nächsten Jahren durchaus mit Veränderungen zu rechnen, die entweder Änderungen in den auf XML basierenden Spezifikationen erfordern könnten oder aber die Gefahr vergrößern, daß Spezifikationen auf veralteten XML Versionen beruhen. Eine derartige Situation wäre vergleichbar mit der jetzigen Abhängigkeit der SMIv1/v2 von einem überholten ASN.1 Standard. Andererseits sind Definitionen von Management-Informationen nicht nur von Programmen sondern auch von Menschen zu lesen und zu schreiben. Kürze und Lesbarkeit sind Eigenschaften, die eine XML DTD für Management-Informationen nicht bieten kann.

Der durch die Unabhängigkeit von externen Dokumenten erzielte Vorteil ist eine langfristig **eindeutige und vollständige Spezifikation**.

3.5. Trennung von Informationsmodell und Management Protokoll

Teile der ASN.1 Definitionen der SMIv1/v2 werden in den Spezifikationen des Management Protokolls SNMP wiederverwendet (z.B. sog. *tagged types* aus dem SMI-Modul SNMPv2-SMI [2]). Ferner enthält das Informationsmodell einige Konstrukte, die die Kodierung von Daten im Protokoll beeinflussen (z.B. die kompaktere Kodierung von Indexelementen variabler Länge durch das `IMPLIED`-Schlüsselwort). Dies bietet sich zwar zum Teil an, da beide Spezifikationen auf ASN.1 basieren, es erschwert aber zugleich den Austausch einer Komponente im Gesamtmodell. Eine strikte Trennung von Informationsmodell und Protokoll kann hingegen leichter ermöglichen, den Transport von Daten gemäß des Informationsmodells über neue Protokolle abzuwickeln oder andererseits ein bestehendes Protokoll um Fähigkeiten für den Transport neuer Datentypen zu erweitern.

Diese Trennung wird in SMIng weitestgehend erzielt.

Durch die weitgehende Trennung von Informationsmodell und Protokoll wird eine **separate Entwicklung** beider Komponenten und eine **Anwendung weiterer Protokolle** erleichtert.

Die Spezifikation (Anhang E) dokumentiert in Abschnitt 1.2 die verbleibenden Abhängigkeiten vom Management Protokoll SNMP.

3.6. Erweiterbarer Sprachumfang

Eine Sprache eines Informationsmodells mit einer festen Menge von Anweisungen stößt früher oder später an Grenzen. Bei der Definition von SMIv1/v2 MIB Modulen kam in den letzten Jahren mehrfach der Wunsch nach Möglichkeiten auf, zusätzliche Informationen formal angeben zu können. Beispiele sind:

- Dringlichkeitsgrade von *Notifications*[2],
- *Protocol Identifier*, wie sie für RMON-2 *Probes* in [68] definiert sind,
- Verhaltensbeschreibungen für *Managed Objects*,
- *Discontinuity Indicators* für Zählerobjekte [69], [6].

Die Einsatzgebiete derartiger Informationen sind beschränkt und teilweise umstritten. Daher gehören entsprechende Anweisungen nicht zum SMIng Sprachumfang. Um dennoch diese und weitere Informationen formal und nicht nur informell in `description`-Anweisungen angeben zu können, gestattet SMIng die Definition zusätzlicher Anweisungen. Bei Verwendung derartiger Spracherweiterungen müssen diese im selben Modul definiert oder importiert werden. Stößt ein SMIng Parser auf völlig unbekannte Anweisungen, so muß er sie mitsamt Argumenten bis zum dazugehörigen Semikolon ignorieren.

[2] Derartige *Severity Levels* sind umstritten, da ihre Einstufung in der Regel nicht statisch vorgenommen werden kann, sondern von der Netzwerkumgebung abhängt.

Inwiefern ein Parser eine Spracherweiterung kennt und unterstützt, kann variieren. Die Definition einer Anweisung sollte genaue Hinweise zur Syntax und Semantik sowie ggf. Hinweise zur Implementierung geben.

Neben den o.g. Beispielen schlägt die SMIng Spezifikation (Anhang E) in Abschnitt 14.3 die Definition einer `agentcaps` Spracherweiterung vor, die dem SMIv2 `AGENT-CAPABILITIES`-Makro entspricht. Dies ist das einzige SMIv2-Konstrukt, das in SMIng kein Pendant hat, da es trotz seiner Definition in [22] vor sechs Jahren bis heute kaum praktische Anwendung findet [70].

Ein weiteres Beispiel einer Spracherweiterung zeigt Abbildung 3.2. Die darin definierte `annotation`-Anweisung dient beliebigen Ergänzungen zu lokalen oder externen Definitionen. Eine solche Ergänzung kann die ebenfalls in Abbildung 3.2 definierte `severity`-Anweisung sein, die die Angabe von Dringlichkeitsgraden von Notifications vorsieht. Abbildung 3.3 zeigt das Beispiel einer entsprechenden Anwendung beider Spracherweiterungen.

Vergleichbare flexible Möglichkeiten für ergänzende strukturierte Informationen sind in anderen Informationsmodellen nicht vorhanden. Lediglich in CIM können durch *User-defined Qualifiers* benutzerdefinierte zusätzliche Angaben gemacht werden, die jedoch nicht mit einem formalen Konstrukt definiert und dokumentiert werden können und keine komplexen Strukturen wie Schachtelungen erlauben.

> Spracherweiterungen ermöglichen die **Ergänzung beliebiger formaler Informationen**. Annotationen als ein mächtiges Beispiel einer Spracherweiterung erlauben dabei den **Bezug auf externe Definitionen**. Parsern bleibt die **Implementierung freigestellt**.

Die Spezifikation (Anhang E) dokumentiert die `extension`-Anweisung in Abschnitt 6 und ihre Anwendung in Abschnitt 16.

3.7. Vollständiger Umfang an Basisdatentypen

Das Ziel, SNMP und die SMI möglichst einfach zu gestalten, führte zu einem sehr eingeschränkten Umfang an Basisdatentypen. Die SMIv1 weist keinerlei Fließpunkttypen oder 64 Bit breite Integer-Typen auf. Diesem Defizit begegnete die SMIv2 nur begrenzt: Man führte einen Typ für 64 Bit breite Zähler ein (`Counter64`). Andere Basisdatentypen, die etwa in CIM oder IDL vorhanden sind, fehlen jedoch weiterhin.

SMIng besitzt hingegen einen nach Erkenntnissen anderer Informationsmodelle und Programmiersprachen vollständigen Umfang von Basistypen: vorzeichenlose und vorzeichenbehaftete 32- und 64-Bit Integer-Typen, IEEE Fließpunkttypen in drei verschiedenen Auflösungen, Aufzählungstypen, Bitmengentypen und einen `OctetString`-Typ, sowie den für das zugrunde liegende Informationsmodell notwendigen `ObjectIdentifier`-Typ. Auf einige weitere Typen kann ohne Einschränkungen verzichtet werden: Boolean-Werte werden im Informationsmodell konform zur SMIv1/v2 als `Integer32`-Werte repräsentiert. Einzelne Zeichen können als `OctetStrings` der Länge 1 dargestellt werden. Die aus der IDL bekannten Typen `any` und `Object` haben im SMIng Informationsmodell kein Pendant.

> SMIng verfügt über einen **vollständigen Umfang an Basisdatentypen**.

Die Spezifikation (Anhang E) enthält die vollständige Dokumentation aller Basistypen in Abschnitt 3.

```
module IRTF-NMRG-SMING-EXTENSIONS irtfNmrgSmingExtensions {

// ...

    extension annotation {
        status      current;
        description
              "The annotation extension statement is used to describe
               arbitrary annotations to local or external definitions.

               Every annotation gets a name and an object identifier.
               A target statement specifies the referred definition.
               Status, description and reference statement are used
               as usual.

               The annotation extenstion statement should only be used
               in the statement body of a module.

               The following ABNF definitions extend the SMIng
               grammar to include the annotation definition:

               annotationStatement = 'annotation' sep lcIdentifier
                                     optsep '{' stmtsep
                                       oidStatement stmtsep
                                       'target' sep qIdentifier stmtsep
                                       *1(statusStatement stmtsep)
                                       descriptionStatement stmtsep
                                       *1(referenceStatement stmtsep)
                                       *(unknownStatement stmtsep)
                                     '}' optsep ';'
               ";

    extension severity {
        status      current;
        description
              "The severity extension statement is used to describe
               the severity level of a notification. It may be used
               in a notification's statement body or in an annotation.

               A severity statement gets one argument that specifies
               the severity level.

               The following ABNF definitions extend the SMIng
               grammar to include the severity definition:

               severityStatement = 'severity' sep
                                     ('debug' /
                                      'info' /
                                      'notice' /
                                      'warning' /
                                      'error' /
                                      'crit' /
                                      'alert' /
                                      'emerg')
                                   optsep ';'
               ";

};
```

Abbildung 3.2.: Definitionen von Spracherweiterungen: `annotation` und `severity`

```
module TUBS-IBR-IF-MIB-ANNOTATIONS tubsIbrIfMIBAnnotations {

    import IRTF-NMRG-SMING-EXTENSIONS (annotation, severity);

    import IF-MIB (linkDown, linkUp);

// ...

    annotation linkDownAnnotation {
        oid         tubsIbrIfMIBAnnotations.3;
        target      linkDown;
        status      current;
        description "This annotation specifies the default severity
                     level of linkDown notifications to be 'error'.";
        severity    error;
    };

    annotation linkUpAnnotation {
        oid         tubsIbrIfMIBAnnotations.4;
        target      linkUp;
        status      current;
        description "This annotation specifies the default severity
                     level of linkUp notifications to be 'notice'.";
        severity    notice;
    };
};
```

Abbildung 3.3.: Anwendung von Spracherweiterungen

3.8. Einfache und konsistente Notationen

SMIv2 enthält in einigen Konstrukten Notationen, die durch Anlehnung an verbreitete Programmiersprachen wie C/C++ oder Java für viele Menschen intuitiver verständlich sein könnten. Andere Notationen erlauben unnötige Ausnahmen oder führen zu Redundanz. Wiederum andere sind mehrdeutig.

Die folgenden Abschnitte benennen diese SMIv2 Defizite und ihre Lösungen in SMIng.

3.8.1. LAST-UPDATED und REVISION

Für das SMIv2-Makro `MODULE-IDENTITY` ist empfohlen, jede Änderung des Moduls in einem `REVISION`-Konstrukt mit Datum und Beschreibung zu dokumentieren. Zusätzlich muß bei jeder Änderung das Datum im `LAST-UPDATED`-Konstrukt aktualisiert werden.

In SMIng entfällt das `LAST-UPDATED`-Konstrukt. Stattdessen *müssen* Änderungen in `revision`-Anweisungen dokumentiert werden. Somit wird eine konsistente Dokumentation erzwungen und redundante Datumsangaben werden vermieden.

3.8.2. Datumsformat

In Anlehnung an das ISO-Datumsformat [71] besteht das Format für `REVISION`- und `LAST-UPDATED`-Konstrukte aus 10 oder 12 Ziffern gefolgt von einem „`Z`“, um die Angabe nach UTC

(*Universal Time Coordinated*) zu verdeutlichen, obwohl keine anderen Zeitzonen erlaubt sind. Dieses Format ist unnötig schwer zu lesen, z.B. „`199111120233Z`“.

In SMIng entfällt das abschließende „`Z`“. Die Ziffernblöcke für die Angaben von Jahr, Monat, Tag, Stunde und Minute werden durch geeignete Zeichen zur besseren Lesbarkeit voneinander getrennt: „`1991-11-12 02:33`“. Das Jahr muß vierstellig angegeben werden. Die Angabe der meist unwichtigen Uhrzeit kann entfallen („`1991-11-12`“), wodurch die Zeit 00:00 impliziert wird.

3.8.3. Binärstrings und Hexadezimalstrings

Binärstrings werden in SMIv2 nur in den seltensten Fällen benutzt. Ihre Werte können ebenso als Hexadezimalstrings dargestellt werden. Die Notation von Binär- und Hexadezimalstrings in SMIv1/v2 basiert auf ASN.1 und erscheint im Vergleich zu typischen Programmiersprachen ungewohnt (z.B. `'0101'B` bzw. `'f00f'H`).

In SMIng entfallen Binärstrings. Hexadezimalstrings und -werte werden wie in C/C++ oder Java durch ein führendes `0x` angegeben, etwa `0xf00f` (Anhang E, Abschnitte 3.1, 3.3).

3.8.4. Mehrzeilige Zeichenketten

Von mehrzeiligen Zeichenketten wird in SMIv1/v2 praktisch in jedem Modul Gebrauch gemacht. `DESCRIPTION`- und `REFERENCE`-Konstrukte erstrecken sich meist über mehrere Zeilen. Die SMIv1/v2 Spezifikationen machen jedoch keine Aussage darüber, wie die Zeilenanfänge der Folgezeilen zu behandeln sind: Werden bedingungslos alle Zeichen in die Zeichenkette aufgenommen? Werden führende Leerzeichen bis zur Spalte des ersten Zeichens der ersten Zeile übersprungen, um eine vermutlich angemessene Formatierung zu erzielen? Was passiert, wenn diese Zeilenanfänge Tabulatoren enthalten?

SMIng gibt die Empfehlung, derartige Zeilenanfänge zu überspringen (Anhang E, Abschnitt 3.1). Außerdem ist es möglich, jede Zeile einer mehrzeiligen Zeichenkette separat in Anführungsstriche einzuschließen. So kann bei Bedarf jede Zeile explizit ausgerichtet werden.

3.8.5. Reihenfolge von Modulabschnitten und Anweisungen

SMIv1/v2 schreibt keine grundlegende Reihenfolge der Definitionen eines Moduls vor. Eine Ausnahme bildet das `MODULE-IDENTITY`-Makro, das unmittelbar auf die `IMPORT`-Anweisung folgen muß. Dennoch hat sich eine bestimmte Anordnung durchgesetzt. Umso mehr kann eine Abweichung von dieser Anordnung irritieren.

SMIng schreibt eine Abfolge von Abschnitten vor: Auf den Modulkopf mit `import`-Anweisungen und Modul-spezifischen Informationen folgen (ggf. leere) Abschnitte mit den definierenden Anweisungen `extension`, `typedef`, `node/scalar/table`, `notification`, `group` und `compliance`. Im Zusammenhang mit dem Verzicht auf Vorwärtsreferenzen (Abschnitt 3.2) ergibt sich im allgemeinen außerdem im Abschnitt der `node`-, `scalar`- und `table`-Anweisungen implizit eine Pre-Ordnung der definierten Teilbäume.

Die Reihenfolge von Anweisungen in anderen Anweisungsblöcken ist ebenfalls vorgeschrieben. Die dabei in mehreren Fällen auftretenden Anweisungen erscheinen stets in derselben Reihenfolge.

3.8.6. Importieren externer Definitionen

In SMIv2 herrscht in einigen Fällen Unklarheit darüber, welche Bezeichner importiert werden müssen. ASN.1 Typen dürfen nicht importiert werden. Darüber hinausgehende SMIv2 Basistypen müssen hingegen wie andere Bezeichner importiert werden. Bezeichner in `MODULE`-Abschnitten von `MODULE-COMPLIANCE`- und `AGENT-CAPABILITIES`-Makros müssen wiederum nicht importiert werden.

SMIng ist in diesem Punkt konsistent. Basistypen werden nicht importiert. Alle extern definierten Bezeichner müssen für alle Zwecke importiert werden. Parser *dürfen nicht* das Benutzen nicht lokal definierter oder importierter Bezeichner gestatten.

3.8.7. Object Identifier Notationen

In SMIv1/v2 sind mehrere Notationen von Object Identifiern erlaubt:

```
{ 1 3 6 1 }
{ iso org dod 1 }
{ iso org(3) dod(6) internet(1) }
```

Die Definition einstelliger Object Identifier ist hingegen verboten.

Die Notation in SMIng ist der außerhalb von MIB Modulen gebräuchlichen Schreibweise angepaßt: Die geschweiften Klammern entfallen, die einzelnen Elemente werden durch Punkte verbunden. Die definierende Notation (letzte Zeile im Beispiel) entfällt. Außerdem darf nur das erste Element ein Bezeichner sein. Somit werden die Schreibweisen für Object Identifier und ihre Definitionen vereinheitlicht.

3.8.8. Zeilenerzeugung und -löschung

Tabellen, die durch Protokolloperationen ein Erzeugen und Löschen von Zeilen erlauben, werden in SMIv2 durch den Status `read-create` in jedem einzelnen Spaltenobjekten gekennzeichnet, das ansonsten dem Status `read-write` entspricht. Verschiedene Spaltenobjekte mit beiden Statuswerten sind in einer Tabellenzeile nicht zulässig. Dies stellt eine redundante und fehleranfällige Notation dar.

In SMIng wird die Möglichkeit des Erzeugens von Tabellenzeilen durch ein `create`-Schlüsselwort in der `row`-Anweisung angezeigt.

3.8.9. Allgemeines Gruppen-Konstrukt

Die `OBJECT-GROUP`- und `NOTIFICATION-GROUP`-Konstrukte in SMIv2 dienen ausschließlich der Bildung von Gruppen entsprechender Definitionen, um sie in `MODULE-COMPLIANCE`- und

`AGENT-CAPABILITIES`-Konstrukten zu verwenden. Die Bildung von gemischten Gruppen oder das Einbeziehen anderer Definitionen ist nicht möglich. Ferner muß jedes Objekt und jede Notification in mindestens einer Gruppe enthalten sein, was jedoch kaum von Parsern überprüft wird. Selbst IETF Standard MIBs enthalten diesbezüglich Fehler. Externe Definitionen dürfen nicht in einer Gruppe enthalten sein.

SMIng hebt diese Einschränkungen auf. Es gibt nur noch eine `group`-Anweisung, die beliebige Bezeichner von Object Identifiern als Gruppenelemente zuläßt. Somit können Gruppen auch für andere Zwecke definiert werden.

3.8.10. MODULE-IDENTITY

SMIv2 besitzt zur Beschreibung von Modulinformationen das `MODULE-IDENTITY`-Konstrukt. Es definiert einen Object Identifier und fügt dieser Definition Angaben über das Modul an: die verwaltende Organisation, Kontaktadressen, Informationen über die Revisionen des Moduls, sowie eine Beschreibung und optionale Referenzen. Obwohl dies Informationen über das Modul sind, werden sie unter einem Object Identifier registriert.

SMIng bindet diese Informationen hingegen durch Angabe in der `module`-Anweisung an das Modul. Ein dem Modul zugeordneter Object Identifier ist nur noch vorhanden, um die Kompatibilität zu SMIv2 zu bewahren.

3.8.11. Redundante Informationen

Die Konstruktion von Tabellen enthält in SMIv1/v2 redundante Strukturen: Die maßgebliche Zusammensetzung von Objekten zu einer Tabellenzeile geschieht durch die Definition von ASN.1 `SEQUENCE`-Datentypen, die Namen und Typen der Spaltenobjekte aufzählen. Im Anschluß werden diese einzelnen Objekte durch das `OBJECT-TYPE`-Konstrukt definiert. Diese Definitionen enthalten ebenfalls Namen und Typen.

In SMIng wird die Konstruktion von Tabellenzeilen in einer `row`-Anweisung vorgenommen, in der die gesamten Definitionen der Spaltenobjekte als `column`-Anweisungen eingeschlossen sind. Doppelte Angaben von Spaltenobjekten sind nicht mehr erforderlich.

3.8.12. Impliziter Status

In SMIv1/v2 muß jede Definition explizit mit einem Status versehen werden. Dies ist in SMIv2 naturgemäß in den meisten Fällen der Status `current`.

In SMIng darf eine `current` Status-Angabe entfallen.

3.8.13. Kommentare

Die Verwendung zweier Minuszeichen zur Kommentareinleitung in SMIv1/v2 ist problematisch: An vielen Stellen sind Minuszeichen in Bezeichnern gestattet. Da zwei aufeinander folgende Minuszeichen jedoch einen Kommentar einleiten, dürfen — bzw. können — Bezeichner keine aufeinander folgenden Minuszeichen enthalten. Ein anderes Problem entsteht dadurch, daß ein zweites Paar von Minuszeichen in einer Zeile einen Kommentar wieder beendet. Somit kann ein Vielfaches von vier Minuszeichen als Trennlinie verwendet werden. Eine andere Anzahl hingegen stellt einen Fehler dar.

Diese irritierende Verwendung des Minuszeichens für Kommentare ist in SMIng behoben, indem ein anderes Token („`//`“, in Anlehnung an C++ und Java) zur Einleitung von Kommentaren verwendet wird. Außerdem erstrecken sich Kommentare bedingungslos bis zum Zeilenende und können nicht explizit vorzeitig geschlossen werden.

Diese zahlreichen Verfeinerungen von Notationen tragen zum **besseren Verständnis** von SMIng Modulen bei. Die genannten Vereinfachungen erleichtern **fehlerfreie Implementierungen**.

3.9. SMIv2 Kompatibilität und Konvertierbarkeit

Der Entwurf von SMIng bewegt sich auf einem Grat zwischen der vollständigen Kompatibilität zu SMIv2 und dem Bereinigen zahlreicher Probleme, die auch inkompatible Änderungen bedeuten können.

Unter der Kompatibilität zwischen SMIv2 und SMIng ist in diesem Zusammenhang die korrekte und vollständige Widerspiegelung aller formulierbaren Informationen in der jeweils anderen Sprache zu verstehen. Dementsprechend sind schematische und programmierbare Übersetzungen zwischen beiden Sprachen möglich. Implementierungen derartiger Übersetzer sind Gegenstand des folgenden Kapitels.

Tatsächlich ist SMIng so entworfen, daß sämtliche SMIv2 Informationen in SMIng übersetzt werden können. Die einzige Ausnahme bildet das `AGENT-CAPABILITIES`-Konstrukt, für das jedoch eine SMIng Spracherweiterung definiert ist (siehe Abschnitt 3.6).

Über die in SMIv2 formulierbaren Informationen hinaus gibt es in SMIng weitere Konstrukte, Datentypen und flexiblere Formulierungen einiger Anweisungen. Eine verlustfreie Konvertierung von SMIng nach SMIv2 ist nur möglich, wenn von diesen Fähigkeiten kein Gebrauch gemacht wird. Dazu zählen

- die Basisdatentypen `Integer64`, `Unsigned64`, `Float32`, `Float64` und `Float128` (Anhang E, Abschnitt 3),
- Gruppen, die andere Definitionen als Objekte oder Notifications oder beide Arten von Definitionen gemeinsam oder Definitionen aus anderen Modulen enthalten (siehe Abschnitt 3.8.9),
- einige Arten von Bezeichnern, die Minuszeichen enthalten,
- Angaben über Objekte, die in Operationen zur Erzeugung von Tabellenzeilen erforderlich sind (`create`-Anweisung, Anhang E, Abschnitt 10.5.3),
- andere Arten der Index-Spezifizierung als durch `index`- oder `augments`-Anweisungen (Anhang E, Abschnitt 10.5.2),

- `reference`-, `default`-, `format`- und `units`-Angaben, die es in entsprechenden SMIv1/v2-Konstrukten nicht gibt.

Diese ergänzenden Möglichkeiten in SMIng sollten bei einer Übersetzung in SMIv2 ggf. in Kommentaren notiert werden und von Übersetzern als Warnungen angezeigt werden.

Im übrigen wird die in SMIv1/v2 Dokumenten gebräuchliche Terminologie weitestgehend bewahrt. Die SMIng Spezifikation (Anhang E) enthält ein Glossar der wichtigsten Begriffe.

Die Möglichkeit, SMIv2 Module nach SMIng übersetzen zu können, **sichert getätigte Investitionen** beim Übergang zu SMIng. Die umgekehrte (eingeschränkte) Konvertierbarkeit ermöglicht es, **Module für SMIv2 Zielplattformen in SMIng zu entwickeln**.

3.10. Nicht verfolgte Ansätze

Neben den in den vorangegangenen Abschnitten beschriebenen Maßnahmen gibt es einige Konzepte, die andere in Kapitel 2 betrachtete Modelle aufweisen, in SMIng jedoch keine Anwendung finden. Diese werden im folgenden aufgeführt und die Gründe erörtert, warum sie nicht für das Informationsmodell des Internet-Management angemessen sind.

3.10.1. Objektorientierung

GDMO, CIM und IDL basieren auf dem Konzept der Objektorientierung mit den typischen Merkmalen der Kapselung durch Klassenbildung und Vererbung. Dies ist grundsätzlich ein Vorteil, da sich so große Mengen von Definitionen besser strukturieren lassen, redundante Definitionen eher vermieden werden und eine homogene Hierarchie von Objekten erstellt werden kann.

Für die SMIng, die dem Informationsmodell des Internet-Management genügen soll, ist dies jedoch kein geeignetes Konzept. Die Bildung einer Hierarchie von Klassenableitungen würde zu einem hohen Grad an Abhängigkeiten zwischen Definitionen führen, was in der Natur der Objektorientierung liegt. Da Änderungen an Definitionen nicht auszuschließen sind, würden derartige Abhängigkeiten den Standardisierungsprozeß von MIB Modulen behindern. In der SMIv1/v2 und SMIng sind die Abhängigkeiten hingegen eher gering. An den SMIv2 `IMPORT`-Anweisungen ist zu erkennen, daß die meisten Abhängigkeiten zu wenigen Standard-Modulen bestehen, an denen sich in den letzten Jahren keine Veränderungen mehr ergeben haben.

Außerdem ist es erforderlich, SMIv2 Module nach SMIng übersetzen zu können (siehe Abschnitt 3.9). Hierbei besteht ohne zusätzliche Informationen keine Möglichkeit, eine angemessene Klassenhierarchie aufzubauen. Es ließe sich nur eine flache Struktur von Klassen erzeugen. Der Unterschied zur vorliegenden SMIng wäre marginal.

3.10.2. Modulkonzept

Die IDL bedient sich zur Einbindung externer Definitionen des Präprozessors. Durch `#include`-Anweisungen werden hier ganze Dateien bei Bedarf eingebunden.

Bei einer Übertragung dieses Konzeptes auf SMIng könnte es zu Namenskollisionen kommen. Das explizite Importieren benötigter Bezeichner vermeidet dies und bewahrt deutlich die Erkennbarkeit der Abhängigkeiten zwischen Modulen.

3.10.3. Assoziationen

Wie in Abschnitt 3.2 beschrieben, gibt es in SMIng für den Verzicht auf Vorwärtsreferenzen zwei Ausnahmen. Diese Ausnahmen entstehen bei der Beschreibung von Beziehungen zwischen Definitionen, die innerhalb einer der beteiligten Definitionen notiert wird. In CIM werden Beziehungen in einem separaten Konstrukt beschrieben, das Referenzen auf die betreffenden Definitionen enthält. Eine solche sog. Assoziation muß hinter allen beteiligten Definitionen notiert werden.

Die Index-Angaben für SMIng-Tabellen sind grundlegender Bestandteil einer Zeilendefinition. Hierfür Assoziationen einzuführen, würde die Kompaktheit zusammengehöriger Definitionen zerstören und die Lesbarkeit stark beeinträchtigen.

3.10.4. Beschreibung von Operationen

GDMO, CIM und IDL erlauben die Beschreibung von Operationen auf den Objekten des jeweiligen Modells durch die formale Angabe von Signaturen.

Da das dem Internet-Management zugrunde liegende Protokoll SNMP verändernde Operationen nur durch die Set-Protokolloperation als Seiteneffekt auslösen kann, gab es bislang keine Möglichkeit, mächtigere Operationen formal zu beschreiben.

SMIng sieht dies ebenfalls nicht vor. Dennoch ist es denkbar, Operationen in Form von Algorithmen, die auf einer Menge von Objektinstanzen arbeiten, formal zu beschreiben und zusammen mit einer Signatur in Modulen zu notieren. Weitere Überlegungen hierzu bleiben jedoch künftigen Arbeiten vorbehalten.

4. Implementierung

Dieses Kapitel beschreibt Idee, Konzept und Implementierung einer Software-Bibliothek, die es Management-Applikationen erlaubt, auf Management-Informationen aus SMI MIB Modulen zuzugreifen, ohne deren Sprache, Dateiformat und Lokalität zu kennen. Die Bibliothek bildet Informationen des Managementmodells auf Typen und Strukturen der Programmiersprache ANSI-C ab. Funktionen der Applikationsschnittstelle erlauben den gezielten und iterativen Zugriff auf diese Strukturen. Schnittstellen für andere Programmier- oder Skriptsprachen wurden im Rahmen dieser Arbeit nicht entwickelt, lassen sich aber ergänzen.

Mit Hilfe dieser Bibliothek wurden zwei Applikationen erstellt, die in der zweiten Hälfte dieses Kapitels beschrieben werden. Dies sind ein Programm zum Prüfen der Syntax und Semantik von SMIv1/v2 und SMIng MIB Modulen und ein Programm zur vollständigen Ausgabe von MIB Definitionen eines Moduls gemäß der SMIv2- bzw. SMIng-Spezifikationen, so daß Übersetzungen von SMIv2 nach SMIng und von SMIng nach SMIv2 ermöglicht werden.

4.1. libsmi – Eine Bibliothek zum Zugriff auf Management-Informationen

4.1.1. Idee und Konzept

Zahlreiche Management-Applikationen erfordern den Zugriff auf Informationen aus MIB Modulen, sei es um Datentypen von Objekte zu ermitteln, Benutzern Beschreibungen von Objekten anzuzeigen, Managementdaten angemessen darzustellen oder andere Dinge. Heutige Systeme benutzen verschiedene Möglichkeiten, um an diese Informationen zu gelangen:

- Applikationen können zur Laufzeit MIB Module lesen, parsen und die benötigten Informationen in intern genutzte Datenstrukturen überführen. Dies kann zu großen Verzögerungen führen, da beispielsweise für das Auflösen eines einzigen Object Identifiers zu einem Bezeichner alle in Frage kommenden Module untersucht werden müssen. In den meisten Systemen wird diese Verzögerung auf die Startphase begrenzt, da hier alle voraussichtlich benötigten Module in interne Strukturen überführt werden, auf denen anschließend effiziente Operationen möglich sind. Ein weiterer Nachteil ist die Erfordernis der Kenntnis über die Sprachen des Informationsmodells. Viele Systeme enthalten unzureichende Implementierungen von SMIv1- und SMIv2-Parsern. Sollen künftige Sprachen — etwa SMIng — unterstützt werden, so werden Ergänzungen erforderlich. Schließlich hat dieses Konzept einen hohen Speicherbedarf. Ein Vorteil dieses Vorgehens besteht im Verzicht auf zusätzliche Dateien, wie sie der folgende Punkt beschreibt. So bleibt das System übersichtlich.

- Um die beschriebenen Verzögerungen zu verkürzen setzen viele Systeme Dateien in einem eigenen (binären oder textbasierten) Zwischenformat ein. Ein MIB Compiler übersetzt einmalig die ursprünglichen Module in dieses Zwischenformat, das anschließend von allen Applikationen des Systems genutzt wird. Die Struktur dieses Formates erlaubt einen effizienteren Zugriff, entweder wie im obigen Fall beim Einlesen der gesamten Dateien in interne Datenstrukturen oder bei gezielten Zugriffen zur Laufzeit. Im letzteren Fall ist zudem der Speicherbedarf geringer. Ein Nachteil dieses Verfahrens ist die Redundanz und der immer noch hohe Speicherbedarf beim Einlesen der gesamten Dateien bzw. Verzögerungen beim direkten Zugriff auf die Dateien. Außerdem nutzen verschiedene Systeme kaum dasselbe Zwischenformat. Eine gemeinsame Nutzung dieser Dateien ist daher oft ausgeschlossen. Das Problem der Abhängigkeit von Parsern der unterstützen Sprachen bleibt ebenfalls bestehen.

Es sind bislang keine Systeme bekannt, die eine konsequente Trennung zwischen Applikationen und der Organisation von Management-Informationen vornehmen. Diesen Ansatz verfolgt das Konzept der vorliegenden Software-Bibliothek `libsmi` mit den folgenden Vorteilen:

- Applikationen werden von der Aufgabe einer effizienten Organisation von Management-Informationen entbunden. Dies ist zentrale Aufgabe der Bibliothek. Der Code von Applikationen wird so schlanker. Parser der unterstützten Sprachen sowie der Entwurf und die Organisation von zahlreichen Datenstrukturen entfallen.

- Durch die Entkopplung von Applikationen und Parsern wird es leicht möglich, neue Sprachen zu ergänzen, ohne die Quelltexte von Applikationen verändern zu müssen. Dynamisch gebundene Programme können sogar unverändert weiterbenutzt werden.

- Gelingt es einer definierten Programmierschnittstellen (API, *Application Programming Interface*), sich zu etablieren, so wird der Code der benutzenden Applikationen allgemein verständlicher.

- In Hinsicht auf den Umgang mit Management-Informationen sind viele unzureichende Implementierungen bekannt. Anhand des Austausches vorhandender Quelltextteile durch die Benutzung der `libsmi`-Schnittstelle besteht im Zusammenhang mit einer zuverlässigen `libsmi`-Implementierung die Möglichkeit, existierende Systeme zu verbessern.

Da die Bibliothek einerseits feste Bestandteile in Form der Applikationsschnittstelle und einiger interner Datenstrukturen aufweisen soll, andererseits aber flexible Möglichkeiten existieren sollen, um an die zu erbringenden Informationen zu gelangen, wurde eine Architektur in zwei Schichten gewählt. Die obere Schicht implementiert die Applikationsschnittstelle, die in einer C-Deklarationsdatei (`smi.h`) spezifiziert ist. Zudem enthält sie interne Datenstrukturen, in denen sämtliche eingelesenen Informationen abgelegt werden. Die untere Schicht besteht aus einem oder mehreren Treibern, die aus verschiedenen Quellen (derzeit aus SMIv1/v2 Dateien und SMIng Dateien, künftig ggf. aus MOSY[1]-Dateien, Datenbanken oder über Netzwerkdienste) Management-Informationen extrahieren und in den Datenstrukturen der oberen Schicht ablegen. Diese Struktur verdeutlicht Abbildung 4.1.

[1]MOSY: ein MIB Compiler, der SMIv1/v2 Module in ein textbasiertes Zwischenformat überführen kann[72].

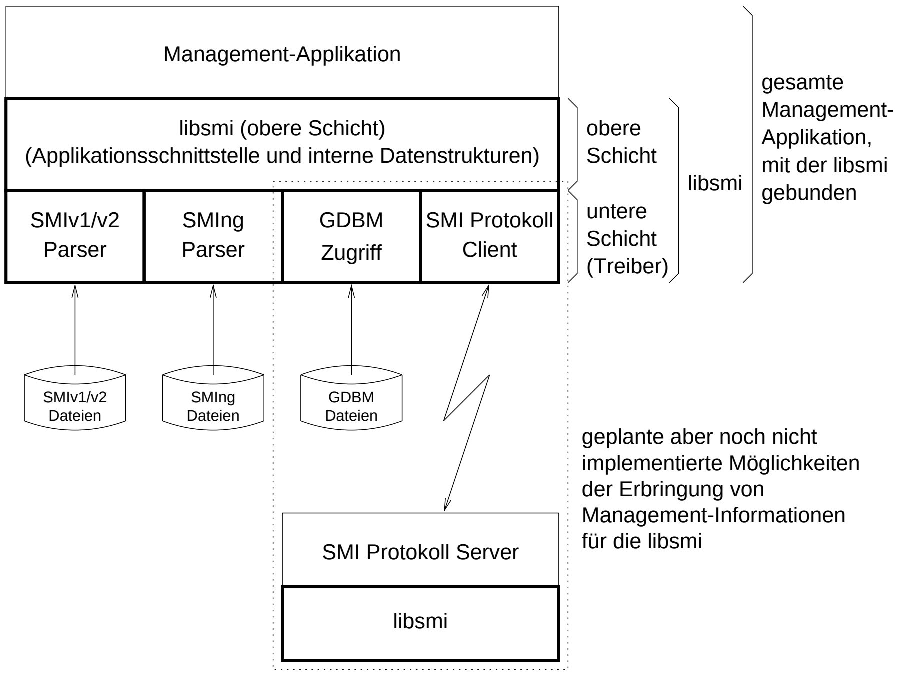

Abbildung 4.1.: `libsmi` Struktur

Nach diesem Schema können beliebige Applikationen mit der Bibliothek gebunden werden und so auf gemeinsame Quellen von Management-Informationen zugreifen. Abbildung 4.2 verdeutlicht dies.

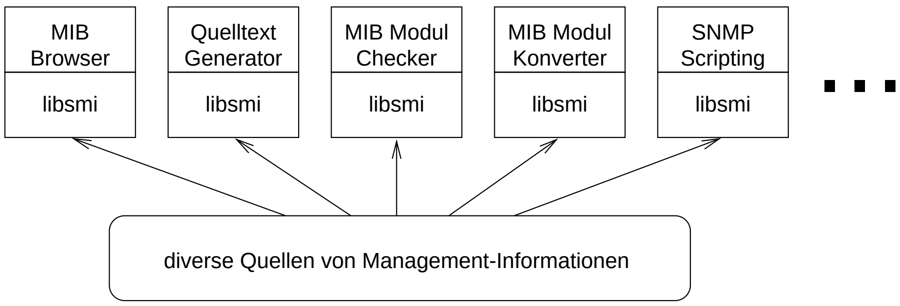

Abbildung 4.2.: libsmi Applikationen

4.1.2. Abbildung der Basisdatentypen

Die SMIv1-, SMIv2- und SMIng-Basisdatentypen werden auf C-Datentypen abgebildet. Dies ist für Default-Werte von Objekten und für Restriktionen von Typen hinsichtlich Größe und Wertebereich von

Bedeutung. Dabei sind die Wertebereiche zu beachten, die ANSI-C für Datentypen garantiert. Für SMI-Fließpunkttypen einfacher Genauigkeit (`Float32`) und 32-Bit-Integer-Typen existieren geeignete C-Datentypen. Für Fließpunkttypen höherer Genauigkeit kann dies nicht garantiert werden. 64 Bit Integer-Typen werden auf „`long long`" Typen abgebildet. Dies setzt geeignete C-Compiler und Laufzeitumgebungen voraus[2]. Eine Liste der Abbildungen aller Basistypen zeigt Tabelle 4.1.

SMIv1/v2 Datentypen	SMIng Datentypen	C Datentypen
`INTEGER`, `Integer32`	`Integer32`	`long`
	`Integer64`	`long long`
`Unsigned32`	`Unsigned32`	`unsigned long`
	`Unsigned64`	`unsigned long long`
	`Float32`	`float`
	`Float64`	`double`
	`Float128`	`long double`
`OCTET STRING`	`OctetString`	`(char *)`[3]
`BITS`	`Bits`	`(struct NamedNumber **)`
`INTEGER` (enum)	`Enumeration`	`(struct NamedNumber **)`
`IpAddress`		abgeleitet: `(char *)` (4 Bytes)
`Counter`, `Counter32`		abgeleitet: `unsigned long`
`Counter64`		abgeleitet: `unsigned long long`
`Gauge`, `Gauge32`		abgeleitet: `unsigned long`
`TimeTicks`		abgeleitet: `unsigned long`
`Opaque`		kommt als Wert nicht vor

Tabelle 4.1.: Abbildung von SMIv1/v2- und SMIng-Basisdatentypen auf C-Datentypen

4.1.3. Abbildung der Datenstrukturen

Die Definitionen, die in MIB Modulen durch SMIv1/v2 Makros und SMIng Anweisungen vorgenommen werden, bildet die Bibliothek auf C Datenstrukturen der oberen Schicht ab[4]. Hierbei sind geeignete Datenstrukturen zu benutzen, die effizient direkte und suchende Zugriffe erlauben. In der vorliegenden Implementierung werden Module in bidirektional verketteten Listen organisiert (`struct Module`). Die Revisionen eines Moduls (`struct Revision`), die in einem Modul importierten Bezeichner (`struct Import`) sowie die definierten Typen (`struct Type`), Makros bzw. Spracherweiterungen (`struct Macro`) und Object Identifier Definitionen jeder Art (`struct Object`) werden ebenfalls in bidirektional verketteten Listen je Modul organisiert (`nextPtr` und `prevPtr` in jeder Struktur). Effizientere Verfahren — etwa durch Hashing — sind künftig denkbar. Darüber hinaus werden die Object

[2] Im Rahmen dieser Arbeit wurde der frei erhältliche GNU C-Compiler auf Sun/SPARC Solaris 2.5.1 und Linux-PC Systemen eingesetzt.

[3] terminiert durch ein 0-Byte, nur für Textdaten. Hexadezimalstrings können 0-Bytes enthalten und müssen daher mit einer zusätzlichen Längenangabe kodiert werden.

[4] Einzig das `AGENT-CAPABILITIES`-Konstrukt wird nicht unterstützt.

Identifier Definitionen aller Module in einer Baumstruktur organisiert (`struct Node` mit Elementen `parentPtr`, `nextPtr`, `prevPtr`, `firstChildPtr`, `lastChildPtr`), so daß ein schnelles Auffinden anhand von numerischen Object Identifier Werten möglich ist. Da es mehrere Definitionen für einen Object Identifier geben kann, werden auch die `Object`-Strukturen je Baumknoten in einer Liste verwaltet. Für *Compliance*-Anweisungen existieren Datenstrukturen, die Informationen über erforderliche und optionale Gruppen sowie Objektverfeinerungen aufnehmen. Ebenso existiert eine Struktur für verschiedene Index-Klauseln von Tabellenzeilen. Diese Strukturen werden in entsprechenden Fällen über Zeiger mit der zugehörigen *Object*-Struktur verknüpft. Die Deklarationen aller internen Datenstrukturen sind in `lib/data.h` der `libsmi`-Quelltexte notiert.

Diese internen Strukturen werden nicht für die Applikationsschnittstelle verwendet. Hierfür existiert ein kleinerer Umfang an Strukturen, die keinerlei Zeigerverkettungen der Relationen zwischen Strukturen und andere für Applikationen unwichtige Informationen enthalten. Instanzen werden zur Laufzeit bei Zugriffen durch die Applikation von der Bibliothek alloziert und an die Applikationen übergeben. Die Applikation hat durch den Aufruf von entsprechenden Freigabefunktionen für die Speicherfreigabe zu sorgen. Die vier Datenstrukturen der Applikationsschnittstelle `SmiModule`, `SmiType`, `SmiNode` und `SmiMacro` sind in der C-Deklarationsdatei `data/smi.h` der `libsmi`-Quelltexte definiert. Die gesamte Datei ist in Anhang F angegeben. Die Bedeutung und Benutzung dieser Strukturen ist in der `libsmi(3)` Manual Page dokumentiert, die in Anhang G enthalten ist.

4.1.4. Funktionen

Der Zugriff auf die im vorangegangenen Abschnitt genannten Strukturen erfolgt durch Funktionsaufrufe der Bibliothek. Hier werden zwei Methoden von Zugriffen unterschieden: direkte Zugriffe, bei denen der Bezeichner oder Object Identifier bekannt ist (`smiGetXXX()`), und iterierende Zugriffe, die ohne Kenntnis von Namen eine Iteration über alle Definitionen eines Moduls erlauben (`smiGetFirstXXX()`, `smiGetNextXXX()`). Letzteres ist beispielsweise für MIB-Compiler oder -Browser erforderlich.

Werden entsprechende Informationen gefunden, so allozieren diese Funktionen die zugehörige Datenstruktur, füllen sie anhand der internen Strukturen mit den entsprechenden Informationen und geben einen Zeiger der Struktur an die aufrufende Applikation zurück. Komplexere Strukturen können Listen enthalten. Beispielsweise enthält eine durch `smiGetNode()` gelieferte `SmiNode` Struktur im Falle einer Tabellenzeile einen Zeiger auf die Liste der Namen von Indexobjekten. Benötigt die Applikation die Daten nicht mehr, so ist die Struktur durch Aufruf einer `SmiFreeXXX()` Funktion wieder freizugeben, womit implizit auch alle enthaltenen Listen freigegeben werden.

Die Funktionen `smiGetParent()` und `smiGetChildren()` erlauben das Navigieren in der Baumstruktur von Object Identifiern.

`smiGetNames()` erlaubt die Auflösung von nicht eindeutigen Namen oder Object Identifiern zu einer Liste von eindeutigen Namen, die sich aus Modulnamen, dem Namensraumoperator „`::`“ und dem Bezeichner der Definition zusammensetzen.

Mit `smiGetImports()` kann die Liste der importierten Bezeichner eines Moduls ermittelt werden.

Darüber hinaus existieren einige Hilfsfunktionen zu folgenden Zwecken:

- Konvertierung eines Datums zwischen der SMIv2-Notation, der SMIng-Notation und dem Typ `time_t`,
- Ermittlung der Modul- und Bezeichnerkomponenten von vollständigen Namen,
- Ermittlung, ob ein Bezeichner in einem Modul importiert wird.

4.1.5. Kontrollfunktionen

Neben den zuvor genannten Funktionen, die Datenstrukturen des Informationsmodells betreffen, gibt es Kontrollfunktionen, um das Verhalten der Bibliothek in einigen Punkten zu beeinflussen und der Umgebung anzupassen.

`smiInit()` initialisiert die Bibliothek. Dabei wird unter anderem eine Konfigurationsdatei gelesen, die Pfade für die Suche nach MIB Modulen enthalten und Grundeinstellungen für Fehler- und Debugging-Ausgaben der Bibliothek vornehmen kann.

Weitere Funktionen erlauben das explizite Anpassen von Fehler- und Debugging-Graden, das Lesen weiterer Konfigurationsdateien, das Angeben weiterer MIB Suchpfade und das Verändern von Flags. Durch Flags läßt sich beeinflussen, ob Fehler von implizit gelesenen Modulen und fehlerhafte Zeilen mit einer Marke angezeigt werden und ob Statistiken über Module ausgegeben werden.

Durch die Funktion `smiLoadModule()` nimmt eine Applikation ein Modul in die Menge der bekannten Module auf. Dies ist erforderlich, um bei Zugriffen, die mehrdeutig sein können, Informationen aus den gewünschten Modulen zu erhalten. Solche Mehrdeutigkeiten gibt es beispielsweise bei der Suche nach der Definition eines Object Identifiers, die in mehreren Versionen desselben Moduls vorliegen kann.

4.1.6. Treiber

In der vorliegenden Implementierung sind bislang zwei Treiber enthalten: Ein Parser für SMIv1- und SMIv2-Module und ein Parser für SMIng-Module. Beide Parser wurden mit Hilfe der GNU-Varianten der Compilerbau-Werkzeuge LEX und YACC[5] erstellt. Für den SMIv1/v2-Parser mußte die YACC-Grammatik aus den ASN.1 Grundlagen, den formal definierten ASN.1 Makros und den informellen Spezifikationen erstellt werden. Im Falle des SMIng-Parser konnte dies sehr viel einfacher schematisch anhand der in Kapitel 3 entworfenen und im Anhang E, Abschnitt A angegebenen ABNF Grammatik geschehen.

Die Parser haben zwei Aufgaben. Sie prüfen eingelesene MIB Module auf syntaktische und begrenzt auf semantische Fehler. Dies stellt für MIB Autoren eine wichtige Hilfe dar. Ferner überführen die Parser die gelesenen Definitionen in die internen Datenstrukturen der oberen `libsmi`-Schicht, um sie Applikationen verfügbar zu machen.

[5] LEX erzeugt aus Regeln, die reguläre Ausdrücke spezifizieren, einen C Quelltext zur lexikalischen Analyse von Eingabetexten. YACC erzeugt aus einer formalen Grammatik den C Quelltext eines entsprechenden Parsers. Die Tokens der YACC Eingabe stammen meist aus einer mit LEX erzeugten Funktion zur lexikalischen Analyse. Die GNU Varianten dieser Werkzeuge (flex und bison) enthalten einige Möglichkeiten, die in der vorliegenden Implementierung erforderlich waren, z.B. die rekursive Verwendung eines Parsers im Falle von Import-Ketten.

Weitere Treiber, wie sie in Abbildung 4.1 angedeutet sind, sind denkbar: Um durch den Verzicht auf aufwendige Parser kürzere Startphasen zu erzielen, könnte ein Treiber auf Dateien mit effizienteren Zugriffsmöglichkeiten oder auf Datenbanken zurückgreifen, ähnlich wie es in Abschnitt 4.1.1 für bisherige Systeme beschrieben ist. Ein anderer Treiber könnte mittels eines Client-/Server-Protokolls auf die Informationen von zentralisierten „MIB-Servern" zugreifen. Ansätze hierzu wurden im Rahmen dieser Arbeit diskutiert, jedoch nicht verfeinert und implementiert. Weiterhin wäre denkbar, noch unbekannte MIB Module über existierende Protokolle wie FTP oder HTTP von zentralisierten Servern zu beziehen.

4.1.7. Anwendungsbeispiel

```
#include <stdio.h>
#include <string.h>
#include <smi.h>

int main(int argc, char *argv[])
{
    char *modulename, *fullname;
    SmiNode *smiNode, *smiStartNode;

    if (argc != 2) {
        fprintf(stderr, "Usage: smisubtree oid\n");
        exit(1);
    }

    smiInit();

    fullname   = argv[1];
    modulename = smiModule(fullname);

    smiLoadModule(modulename);

    for(smiNode = smiStartNode = smiGetNode(fullname, NULL);
        smiNode && strstr(smiNode->oid, smiStartNode->oid) == smiNode->oid;
        smiNode = smiGetNextNode(smiNode->name, modulename)) {

        printf("%s::%-16s   [%s]\n",
               smiNode->module, smiNode->name,
               smiStringBasetype(smiNode->basetype));

    }

    return 0;
}
```

Abbildung 4.3.: Beispiel einer `libsmi`-Applikation

Abbildung 4.3 zeigt das Beispiel einer Anwendung der Bibliothek. Das Programm nimmt als Argument den vollständigen Namen eines Object Identifiers in der Form „`Modul::Bezeichner`" entgegen und gibt dazu den Teilbaum mit der entsprechenden Wurzel in Form von Objektnamen und deren Basistypen aus, wie Abbildung 4.4 zeigt. Bedeutung und Funktionsweise der verwendeten Funktionen gehen aus der `libsmi(3)` Manual Page hervor (Anhang G).

```
$ smisubtree SNMPv2-MIB::sysORTable
SNMPv2-MIB::sysORTable         [SEQUENCE OF]
SNMPv2-MIB::sysOREntry         [SEQUENCE]
SNMPv2-MIB::sysORIndex         [Integer32]
SNMPv2-MIB::sysORID            [ObjectIdentifier]
SNMPv2-MIB::sysORDescr         [OctetString]
SNMPv2-MIB::sysORUpTime        [Integer32]
```

Abbildung 4.4.: Ausgabe der Beispiel-Applikation

4.1.8. Begrenzung und Zukunft der libsmi

Die vorliegende Bibliothek `libsmi` ist als eine erste exemplarische Implementierung aufzufassen. Sie erlaubt es, die Möglichkeiten einer solchen Bibliothek praktisch zu untersuchen und die im folgenden Abschnitt vorgestellten Applikationen zu implementieren. Einige Aspekte können jedoch verbessert und ergänzt werden:

- Die implementierten SMIv1/v2 und SMIng Parser erkennen viele semantische Fehler in MIB Modulen nicht. Gute Parser mit umfangreichen Fehlererkennungen sind eine wichtige Hilfe für MIB Autoren.
- Ähnlich wie die Applikationsschnittstelle definiert sein muß, kann auch die Schnittstelle zwischen den beiden Schichten der Bibliothek vereinheitlicht und dokumentiert werden. Dadurch ist es leichter möglich, Treiber zu ergänzen.
- Weitere Treiber können implementiert werden. Überlegungen bestehen für DBM-Files, Datenbankanbindungen, ein Client/Server-Modell und den Bezug kompletter MIB Module mittels HTTP oder FTP.
- Das Speichermanagement kann optimiert werden. Die bisherige Implementierung liest jedes Modul vollständig ein. Für das bereits angedeutete Client/Server-Modell ist jedoch ein Caching einzelner Definitionen im lokalen Speicher des Client erforderlich.
- Statt der bislang implementierten doppelt verketteten Listen für die verschiedenen Datenstrukturen können effizientere Verfahren gewählt werden, beispielsweise Hashing.
- SMIng Spracherweiterungen können implementiert werden. Insbesondere Annotationen (siehe Abschnitt 3.6) erscheinen vielversprechend, da so beliebige Ergänzungen an MIB Definitionen vorgenommen werden können.

4.2. Applikationen

4.2.1. Analyse von MIB Modulen: smilint

Die Analyse hinsichtlich syntaktischer und semantischer[6] Korrektheit ist eine wichtige Aufgabe beim Entwurf von MIB Modulen. Der Mangel an entsprechenden Werkzeugen und der seltene Einsatz der existierenden Werkzeuge für SMIv1 und SMIv2 hat in der Vergangenheit zu zahlreichen fehlerhaften MIB Modulen — auch unter den IETF Standard MIBs — geführt.

Das vorliegende Werkzeug `smilint` dient der Prüfung von SMIv1-, SMIv2- und SMIng Modulen. Da die Analyse eng mit den entsprechenden Parsern verknüpft ist, ist es die einzige Aufgabe der Applikation, den Grad der Fehlerausgaben einzustellen und den passenden Parser aufzurufen.

Die Manual Page zu `smilint` ist in Anhang G angegeben. Abbildung 4.5 zeigt ein Beispiel der Benutzung von `smilint`.

```
$ smilint -v -l9 DNS-SERVER-MIB
DNS-SERVER-MIB:20: The MODULE-IDENTITY clause must be the first declaration in a module
   dnsServMIB MODULE-IDENTITY...
                             ^
DNS-SERVER-MIB:548: Object Identifier name ‘dnsServOptCounterSelfAuthNoDataResps’
                    longer than 32 characters, which is not recommended
   dnsServOptCounterSelfAuthNoDataResps OBJECT-TYPE...
                                                ^
DNS-SERVER-MIB:559: Object Identifier name ‘dnsServOptCounterSelfNonAuthDatas’
                    longer than 32 characters, which is not recommended
   dnsServOptCounterSelfNonAuthDatas OBJECT-TYPE...
                                             ^
[...]
```

Abbildung 4.5.: Ausgabe von `smilint`

4.2.2. Übersetzung von MIB Modulen: smidump

Mehrere tausend Objekte sind derzeit in über 100 Standard MIB Modulen definiert. Darüber hinaus existiert eine noch größere Zahl an herstellerspezifischen MIB Modulen. All diese Module stellen eine große Investition in die SMIv1/v2 Technologie dar. SMIng berücksichtigt diesen Umstand durch eine strikte Kompatibilität zu allen SMIv1/v2 Konstrukten. Folglich ist eine schematische Übersetzung von SMIv2 zu SMIng möglich. Umgekehrt sind Übersetzungen von SMIng zu SMIv2 begrenzt möglich (siehe Abschnitt 3.9).

Mit `smidump` liegt die Implementierung eines solchen Übersetzers vor. Dieses Werkzeug liest alle Definitionen eines Moduls mittels der `libsmi` ein und gibt sie in SMIv2- oder SMIng-Notation aus. Liegt das gelesene Modul in der jeweils anderen SMI-Version vor, stellt dieser Vorgang somit eine Übersetzung dar. SMIv1 kann von `smidump` nicht erzeugt werden. Die Ausgabe von Modulen, die in SMIv1-Notation eingelesen wurden, ist ebenfalls nicht möglich.

[6] Die Prüfung syntaktisch korrekter Konstrukte auf ihre im Kontext sinnvolle Verwendung wird meist als semantische Analyse bezeichnet.

Die Ausgabe jedes Formates geschieht in einer entsprechenden eigenständigen Quelltext-Datei. Zwischen den Formaten wird durch eine `smidump`-Option ausgewählt. Neben SMIv2 und SMIng lassen sich weitere Formate somit leicht ergänzen, beispielsweise für das MOSY-Format [72] oder für Quelltext-Stubs zur Implementierung von Agenten.

Die Manual Page zu `smidump` ist in Anhang G angegeben. Abbildung 4.6 zeigt ein Beispiel der Benutzung von `smilint`. Hierbei wird das SMIv2 Modul IF-MIB, dessen Datei im Arbeitsverzeichnis liegt, nach SMIng konvertiert. Der Anfang der erzeugten Datei wird anschließend angezeigt.

```
$ smidump -Lsmi:. -Dsming IF-MIB > IF-MIB.sming
$ head -35 IF-MIB.sming
//
// This module has been generated by smidump. Do not edit.
//
module IF-MIB ifMIB {

//
// IDENTIFIER IMPORTS
//

    import IRTF-NMRG-SMING-TYPES (Counter32, Gauge32, Counter64,
                        TimeTicks);

    import IRTF-NMRG-SMING (mib-2);

    import SNMPv2-TC (DisplayString, PhysAddress, TruthValue,
                        RowStatus, TimeStamp, AutonomousType,
                        TestAndIncr);

    import SNMPv2-MIB (snmpTraps);

    import IANAifType-MIB (IANAifType);

//
// MODULE META INFORMATION
//

    oid             mib-2.31;

    organization
        "IETF Interfaces MIB Working Group";

    contact
        "   Keith McCloghrie
            Cisco Systems, Inc.
            170 West Tasman Drive

[...]
```

Abbildung 4.6.: Benutzung von `smidump`

5. Zusammenfassung und Ausblick

Die im Internet-Management bislang eingesetzte Sprache zur Definition von Management-Informationen weist Defizite auf, die hauptsächlich auf ihre Entstehungsgeschichte zurückzuführen sind. Ihre mittlerweile erlangte weite Verbreitung und der zeitaufwendige Standardisierungsprozeß von Internet-Technologien erschweren inkompatible Veränderungen, die nötig wären, um einen Großteil dieser Defizite zu beseitigen.

Der vorliegende Entwurf einer neuen Sprache zur Definition von Management-Informationen vereint die Vorteile der bekanntesten Informationsmodelle sofern die vollständige semantische Kompatibilität zur bisherigen SMIv2 dies zuläßt. Die wichtigsten durch SMIng erreichten Ziele sind eine eindeutige und einheitliche Grammatik, eine leichtere Implementierung und Erlernbarkeit, eine hohe Flexibilität durch Erweiterbarkeit und eine vollständige Konvertierbarkeit alter MIB Module.

Diese Vorteile geben SMIng das Potential zu einer mittelfristigen Ablösung der bisherigen SMIv2. Die vollständige und als Internet-Draft verfaßte Spezifikation erlaubt das Einbringen dieses Entwurfes in die entsprechende IETF- bzw. IRTF-Arbeitsgruppe, um ihn in einem größeren Kreis zur Diskussion zu stellen und weiterzuentwickeln. Tatsächlich ist geplant, SMIng im Rahmen der IRTF Network Management Research Group (NMRG) [65] weiter voranzutreiben.

Ferner wurde ein Beitrag zur DSOM '99 Konferenz eingereicht [73], der die Motivation und den Entwurf von SMIng vorstellt und erläutert.

Neben der Ausarbeitung der SMIng Spezifikation ist es hilfreich, die praktische Umsetzung zu untersuchen und zu demonstrieren. Die vorgelegte `libsmi` Implementierung stellt künftigen Management-Applikationen Funktionen zur Verfügung, um ohne Kenntnis der zugrunde liegenden Sprache auf Management-Informationen aus MIB Modulen zugreifen zu können. Treiber für SMIv1/v2 Module einerseits und SMIng Module andererseits belegen die Funktionsweise dieses Konzeptes. Auf der `libsmi` aufsetzende Applikationen, um MIB Module auf syntaktische und semantische Korrektheit zu überprüfen und um zwischen SMIv2 und SMIng zu konvertieren, stellen wichtige Funktionalitäten bereit, die für die Akzeptanz von SMIng von Bedeutung sind: Die Analyse von MIB Modulen stellt eine wichtige Hilfe für Autoren von MIB Modulen dar. Die Übersetzung zahlreicher SMIv2 MIB Module nach SMIng demonstriert die Vorteile der SMIng Notationen und sichert in SMIv2 getätigte Investitionen bei einem möglichen Übergang zu SMIng. Umgekehrt kann beim Entwurf von MIB Modulen bereits SMIng benutzt werden, auch wenn noch SMIv2 Module ausgeliefert werden.

Um darüber hinaus den praktischen Einsatz der `libsmi` zu demonstrieren, ist eine Integration in existierende frei verfügbare Managementsysteme geplant. Eine Einbindung der `libsmi` in die MIB Funktionen des `scotty` Systems [74] ist vorgesehen. Ferner wird eine Integration in das `ucd-snmp` System [75] in Erwägung gezogen.

Die Applikation zum Übersetzen von MIB Modulen kann um weitere Ausgabeformate ergänzt werden. In Planung sind Möglichkeiten zur Code-Generierung von Agent-Stubs, eine MOSY konforme Ausgabe [72] sowie graphische Darstellungsmöglichkeiten von Definitionen und Beziehungen innerhalb von Modulen und über Modulgrenzen hinweg.

Der Umfang der semantischen Fehler, die die implementierten SMIv1/v2 und SMIng Parser erkennen können, ist beschränkt. Hier sind Verbesserungen möglich.

Um die Implementierung weiterer Treiber der `libsmi` zu ermöglichen, die etwa die Verwaltung von MIB Definitionen in Datenbanken oder mittels eines Client/Server-Modells auf einem RPC-Server erlauben, sind Veränderungen am Speichermanagement der Bibliothek erforderlich, so daß neben dem Einlesen kompletter Module auch ein Caching einzelner Definitionen möglich ist. Ferner ist für die Implementierung weiterer Treiber eine einheitliche Schnittstelle zwischen den zwei Schichten der Bibliothek vorteilhaft. Auch die Definition der Applikationsschnittstelle ist in der vorliegenden Form noch in einigen Punkten inkonsistent und für einige Applikationen vermutlich noch unzureichend.

Das Konzept der SMIng Spracherweiterung bietet flexible Möglichkeiten, die einerseits zu verfeinern und genau zu spezifizieren sind und andererseits in der Implementierung umzusetzen sind. Inbesondere die Möglichkeit, existierende Definitionen mit Annotationen versehen zu können verspricht einen mächtigen Ansatz, um Herstellern zu erlauben, Standard MIB Module mit eigenen Informationen zu ergänzen. Hierzu wurden bislang separate Module mit einem großen Anteil redundanter Informationen genutzt.

Insgesamt stellt SMIng einen vielversprechenden Ansatz zur Definition von Management-Informationen dar. Die vorgelegte Implementierung belegt einerseits die Tauglichkeit und veranschaulicht die Vorteile von SMIng, bietet aber andererseits noch Raum für zukünftige Ergänzungen und Integrationen in etablierten Implementierungen.

A. Abkürzungen

ABNF	Augmented Backus-Naur Form
ANSI	American National Standards Institute
API	Application Program Interface
ASN.1	Abstract Syntax Notation One
AVA	Attribute Value Assertion
CIM	Common Information Model
CMIP	Common Management Information Protocol
CMIS	Common Management Information Services
CMOT	Common Management Information Services and Protocol over TCP/IP
CORBA	Common Object Request Broker Architecture
CRS	Communicating Rule Systems
DMI	Desktop Management Interface
DMTF	Desktop Management Task Force
DN	Distinguished Name
DTD	Document Type Definition
FTP	File Transfer Protocol
GDMO	Guidelines for the Definition of Managed Objects
GNU	GNU's Not Unix!
HEMP	High-Level Entity Management Protocol
HEMS	High-Level Entity Management System
HMMP	HyperMedia Management Protocol
HMMS	HyperMedia Management Schema
HTTP	Hypertext Transfer Protocol
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IP	Internet Protocol
IRTF	Internet Research Task Force
ISO	International Organization for Standardization
JIDM	Joint Inter-Domain Management
LDAP	Lightweight Directory Access Protocol
LEX	Lexical analyser

MIB	Management Information Base
MIF	Management Information Format
MIM	Management Information Model
MO	Managed Object
MOF	Management Object Format
MOSY	Managed Object Syntax-compiler (yacc-based)
NMRG	Network Management Research Group
OID	Object Identifier
OMA	Object Management Architecture
OMG	Object Management Group
OO	Objektorientierung
ORB	Object Request Broker
OSI	Open Systems Interconnect
RDN	Relative Distinguished Name
RFC	Request For Comments
RMON	Remote Monitoring
SGML	Standard Generalized Markup Language
SGMP	Simple Gateway Monitoring Protocol
SMI	Structure of Management Information
SMIng	Structure of Management Information next generation
SMIv1	Structure of Management Information version 1
SMIv2	Structure of Management Information version 2
SMS	Systems Management Software
SNMP	Simple Network Management Protocol
TC	Textual Convention
TCP	Transmission Control Protocol
TDC	Technology Development Committee
TLA	Three-Letter Acronym
TMN	Telecommunications Management Networks
UCS	Universal Character Set
UML	Unified Modeling Language
UTC	Universal Time Coordinated
UTF	UCS Transformation Format
WBEM	Web-Based Enterprise Management
XML	Extended Markup Language
YACC	Yet Another Compiler-Compiler

B. SMI Grammatik

Dieser Abschnitt enthält die YACC- bzw. bison[1]-Grammatik der SMIv1/v2. Sie ist Teil eines in der vorliegenden Arbeit entwickelten SMI-Parsers. Anweisungsblöcke des Compilers wurden aus dieser Grammatik entfernt.

```
/*
 * One mibFile may contain multiple MIB modules.
 * It's also possible that there's no module in a file.
 */
mibFile:                modules
        |               /* empty */
        ;

modules:                module
        |               modules module
        ;

/*
 * The general structure of a module is described at REF:RFC1902,3. .
 * An example is given at REF:RFC1902,5.7. .
 */
module:                 moduleName
                        DEFINITIONS COLON_COLON_EQUAL BEGIN_
                        exportsClause
                        linkagePart
                        declarationPart
                        END
        ;

/*
 * REF:RFC1902,3.2.
 */
linkagePart:            linkageClause
        |               /* empty */
        ;

linkageClause:          IMPORTS importPart ';'
        ;

exportsClause:          /* empty */
        |               EXPORTS
                        /* the scanner skips until... */
                        ';'
        ;

importPart:             imports
```

[1] YACC (*Yet Another Compiler-Compiler*) ist ein Code-Generator, der anhand einer Eingabe-Grammatik den C-Quelltext eines Parsers erzeugt. Bison ist eine Variante von YACC, die um einige Fähigkeiten ergänzt wurde.

```
        |               /* empty */
        ;

imports:                import
        |               imports import
        ;

import:                 importIdentifiers FROM moduleName
        ;

importIdentifiers:      importIdentifier
        |               importIdentifiers ',' importIdentifier
        ;

/*
 * Note that some named types must not be imported, REF:RFC1902,590 .
 */
importIdentifier:       LOWERCASE_IDENTIFIER
        |               UPPERCASE_IDENTIFIER
        |               importedKeyword
        ;

/*
 * These keywords are no real keywords. They have to be imported
 * from the SMI, TC, CONF MIBs.
 */
importedKeyword:        ACCESS
        |               AGENT_CAPABILITIES
        |               AUGMENTS
        |               BITS
        |               CONTACT_INFO
        |               CREATION_REQUIRES
        |               COUNTER32
        |               COUNTER64
        |               DEFVAL
        |               DESCRIPTION
        |               DISPLAY_HINT
        |               GROUP
        |               GAUGE32
        |               IMPLIED
        |               INDEX
        |               INTEGER32
        |               IPADDRESS
        |               LAST_UPDATED
        |               MANDATORY_GROUPS
        |               MAX_ACCESS
        |               MIN_ACCESS
        |               MODULE
        |               MODULE_COMPLIANCE
        |               MODULE_IDENTITY
        |               NOTIFICATIONS
        |               NOTIFICATION_GROUP
        |               NOTIFICATION_TYPE
        |               OBJECT_GROUP
        |               OBJECT_IDENTITY
        |               OBJECT_TYPE
        |               OBJECTS
        |               ORGANIZATION
        |               OPAQUE
        |               PRODUCT_RELEASE
        |               REFERENCE
        |               REVISION
        |               STATUS
```

```
        |               SUPPORTS
        |               SYNTAX
        |               TEXTUAL_CONVENTION
        |               TIMETICKS
        |               TRAP_TYPE
        |               UNITS
        |               UNSIGNED32
        |               VARIATION
        |               WRITE_SYNTAX
        ;

moduleName:             UPPERCASE_IDENTIFIER
        ;

/*
 * The paragraph at REF:RFC1902,490 lists roughly what's allowed
 * in the body of an information module.
 */
declarationPart:        declarations
        |               /* empty */
        ;

declarations:           declaration
        |               declarations declaration
        ;

declaration:            typeDeclaration
        |               valueDeclaration
        |               objectIdentityClause
        |               objectTypeClause
        |               trapTypeClause
        |               notificationTypeClause
        |               moduleIdentityClause
        |               moduleComplianceClause
        |               objectGroupClause
        |               notificationGroupClause
        |               macroClause
        |               error '}'
        ;

/*
 * Macro clauses. Its contents are not really parsed, but skipped by
 * the scanner until 'END' is read. This is just to make the SMI
 * documents readable.
 */
macroClause:            macroName
                        MACRO
                        /* the scanner skips until... */
                        END
        ;

macroName:              MODULE_IDENTITY
        |               OBJECT_TYPE
        |               TRAP_TYPE
        |               NOTIFICATION_TYPE
        |               OBJECT_IDENTITY
        |               TEXTUAL_CONVENTION
        |               OBJECT_GROUP
        |               NOTIFICATION_GROUP
        |               MODULE_COMPLIANCE
        |               AGENT_CAPABILITIES
        ;
```

```
choiceClause:           CHOICE
                        /* the scanner skips until... */
                        '}'
        ;

/*
 * The only ASN.1 value declarations are for OIDs, REF:RFC1902,491 .
 */
valueDeclaration:       LOWERCASE_IDENTIFIER
                        OBJECT IDENTIFIER
                        COLON_COLON_EQUAL '{' objectIdentifier '}'
        ;

/*
 * This is for simple ASN.1 style type assignments and textual conventions.
 */
typeDeclaration:        typeName
                        COLON_COLON_EQUAL typeDeclarationRHS

        ;

typeName:               UPPERCASE_IDENTIFIER
        |               typeSMI
        ;

typeSMI:                INTEGER32
        |               IPADDRESS
        |               COUNTER32
        |               GAUGE32
        |               UNSIGNED32
        |               TIMETICKS
        |               OPAQUE
        |               COUNTER64
        ;

typeDeclarationRHS:     Syntax
        |               TEXTUAL_CONVENTION
                        DisplayPart
                        STATUS Status
                        DESCRIPTION Text
                        ReferPart
                        SYNTAX Syntax
        |               choiceClause
        ;

/* REF:RFC1902,7.1.12. */
conceptualTable:        SEQUENCE OF row
        ;

row:                    UPPERCASE_IDENTIFIER
                        /*
                         * In this case, we do NOT allow 'Module.Type'.
                         * The identifier must be defined in the local
                         * module.
                         */
        ;

/* REF:RFC1902,7.1.12. */
entryType:              SEQUENCE '{' sequenceItems '}'
;

sequenceItems:          sequenceItem
        |               sequenceItems ',' sequenceItem
```

```
        ;

/*
 * In a SEQUENCE { ... } there are no sub-types, enumerations or
 * named bits. REF: draft, p.29
 * NOTE: REF:RFC1902,7.1.12. was less clear, it said:
 * `normally omitting the sub-typing information'
 */
sequenceItem:           LOWERCASE_IDENTIFIER sequenceSyntax
        ;

Syntax:                 ObjectSyntax
        |               BITS '{' NamedBits '}'
        ;

sequenceSyntax:         /* ObjectSyntax */
                        sequenceObjectSyntax
        |               BITS
        |               UPPERCASE_IDENTIFIER anySubType
        ;

NamedBits:              NamedBit
        |               NamedBits ',' NamedBit
        ;

NamedBit:               identifier
                        '(' number ')'
        ;

identifier:             LOWERCASE_IDENTIFIER
        ;

objectIdentityClause:   LOWERCASE_IDENTIFIER
                        OBJECT_IDENTITY
                        STATUS Status
                        DESCRIPTION Text
                        ReferPart
                        COLON_COLON_EQUAL
                        '{' objectIdentifier '}'
        ;

objectTypeClause:       LOWERCASE_IDENTIFIER
                        OBJECT_TYPE
                        SYNTAX Syntax
                        UnitsPart
                        MaxAccessPart
                        STATUS Status
                        descriptionClause
                        ReferPart
                        IndexPart
                        DefValPart
                        COLON_COLON_EQUAL '{' ObjectName '}'
        ;

descriptionClause:      /* empty */
        |               DESCRIPTION Text
        ;

trapTypeClause:         LOWERCASE_IDENTIFIER
                        TRAP_TYPE
                        ENTERPRISE objectIdentifier
                        VarPart
                        DescrPart
```

```
                        ReferPart
                        COLON_COLON_EQUAL number
        ;

VarPart:                VARIABLES '{' VarTypes '}'
        |               /* empty */
        ;

VarTypes:               VarType
        |               VarTypes ',' VarType
        ;

VarType:                ObjectName
        ;

DescrPart:              DESCRIPTION Text
        |               /* empty */
        ;

MaxAccessPart:          MAX_ACCESS
                        Access
        |               ACCESS
                        Access
        ;

notificationTypeClause: LOWERCASE_IDENTIFIER
                        NOTIFICATION_TYPE
                        ObjectsPart
                        STATUS Status
                        DESCRIPTION Text
                        ReferPart
                        COLON_COLON_EQUAL
                        '{' NotificationName '}'
        ;

moduleIdentityClause:   LOWERCASE_IDENTIFIER
                        MODULE_IDENTITY
                        LAST_UPDATED ExtUTCTime
                        ORGANIZATION Text
                        CONTACT_INFO Text
                        DESCRIPTION Text
                        RevisionPart
                        COLON_COLON_EQUAL
                        '{' objectIdentifier '}'
        ;

ObjectSyntax:           SimpleSyntax
        |               typeTag SimpleSyntax
        |               conceptualTable
        |               row                     /* the uppercase name of a row  */
        |               entryType               /* SEQUENCE { ... } phrase */
        |               ApplicationSyntax
        ;

typeTag:                '[' APPLICATION number ']' IMPLICIT
        |               '[' UNIVERSAL number ']' IMPLICIT
        ;

/*
 * In a SEQUENCE { ... } there are no sub-types, enumerations or
 * named bits. REF: draft, p.29
 */
sequenceObjectSyntax:   sequenceSimpleSyntax
```

```
        |               sequenceApplicationSyntax
        ;

valueofObjectSyntax:    valueofSimpleSyntax
        ;

SimpleSyntax:           INTEGER                 /* (-2147483648..2147483647) */
        |               INTEGER integerSubType
        |               INTEGER enumSpec
        |               INTEGER32               /* (-2147483648..2147483647) */
        |               INTEGER32 integerSubType
        |               UPPERCASE_IDENTIFIER enumSpec
        |               moduleName '.' UPPERCASE_IDENTIFIER enumSpec
        |               UPPERCASE_IDENTIFIER integerSubType
        |               moduleName '.' UPPERCASE_IDENTIFIER integerSubType
        |               OCTET STRING            /* (SIZE (0..65535))         */
        |               OCTET STRING octetStringSubType
        |               UPPERCASE_IDENTIFIER octetStringSubType
        |               moduleName '.' UPPERCASE_IDENTIFIER octetStringSubType
        |               OBJECT IDENTIFIER
        ;

valueofSimpleSyntax:    number                  /* 0..2147483647 */
                        /* NOTE: Counter64 must not have a DEFVAL */
        |               '-' number              /* -2147483648..0 */
        |               BIN_STRING              /* number or OCTET STRING */
        |               HEX_STRING              /* number or OCTET STRING */
        |               LOWERCASE_IDENTIFIER    /* enumeration or named oid */
        |               QUOTED_STRING           /* an OCTET STRING */
                        /* NOTE: If the value is an OBJECT IDENTIFIER, then
                         *       it must be expressed as a single ASN.1
                         *       identifier, and not as a collection of
                         *       of sub-identifiers.
                         *       REF: draft,p.34
                         *       Anyway, we try to accept it. But it's only
                         *       possible for numbered sub-identifiers, since
                         *       other identifiers would make something like
                         *       { gaga } indistiguishable from a BitsValue.
                         */
        |               '{' objectIdentifier_defval '}'
                        /*
                         * This is only for some MIBs with invalid numerical
                         * OID notation for DEFVALs. We DO NOT parse them
                         * correctly. We just don't want to produce a
                         * parser error.
                         */
        ;

/*
 * In a SEQUENCE { ... } there are no sub-types, enumerations or
 * named bits. REF: draft, p.29
 */
sequenceSimpleSyntax:   INTEGER anySubType      /* (-2147483648..2147483647) */
        |               INTEGER32 anySubType    /* (-2147483648..2147483647) */
        |               OCTET STRING anySubType
        |               OBJECT IDENTIFIER
        ;

ApplicationSyntax:      IPADDRESS
        |               COUNTER32               /* (0..4294967295)           */
        |               GAUGE32                 /* (0..4294967295)           */
        |               GAUGE32 integerSubType
        |               UNSIGNED32              /* (0..4294967295)           */
```

```
        |               UNSIGNED32 integerSubType
        |               TIMETICKS               /* (0..4294967295)           */
        |               OPAQUE                  /* IMPLICIT OCTET STRING     */
        |               COUNTER64               /* (0..18446744073709551615) */
        ;

/*
 * In a SEQUENCE { ... } there are no sub-types, enumerations or
 * named bits. REF: draft, p.29
 */
sequenceApplicationSyntax: IPADDRESS
        |               COUNTER32               /* (0..4294967295)           */
        |               GAUGE32 anySubType      /* (0..4294967295)           */
        |               UNSIGNED32 anySubType /* (0..4294967295)             */
        |               TIMETICKS               /* (0..4294967295)           */
        |               OPAQUE                  /* IMPLICIT OCTET STRING     */
        |               COUNTER64               /* (0..18446744073709551615) */
        ;

anySubType:             integerSubType
        |               octetStringSubType
        |               enumSpec
        |               /* empty */
        ;

/* REF: draft,p.46 */
integerSubType:         '(' ranges ')'          /* at least one range        */
                        /*
                         * the specification mentions an alternative of an
                         * empty RHS here. this would lead to reduce/reduce
                         * conflicts. instead, we differentiate the parent
                         * rule(s) (SimpleSyntax).
                         */
        ;

octetStringSubType:     '(' SIZE '(' ranges ')' ')'
                        /*
                         * the specification mentions an alternative of an
                         * empty RHS here. this would lead to reduce/reduce
                         * conflicts. instead, we differentiate the parent
                         * rule(s) (SimpleSyntax).
                         */
        ;

ranges:                 range
        |               ranges '|' range
        ;

range:                  value
        |               value DOT_DOT value
        ;

value:                  '-' number
        |               number
        |               HEX_STRING
        |               BIN_STRING
        ;

enumSpec:               '{' enumItems '}'
        ;

enumItems:              enumItem
        |               enumItems ',' enumItem
```

```
        ;

enumItem:               LOWERCASE_IDENTIFIER
                        '(' enumNumber ')'
        ;

enumNumber:             number
        |               '-' number
        ;

Status:                 LOWERCASE_IDENTIFIER
        ;

Status_Capabilities:    LOWERCASE_IDENTIFIER
        ;

DisplayPart:            DISPLAY_HINT Text
        |               /* empty */
        ;

UnitsPart:              UNITS Text
        |               /* empty */
        ;

Access:                 LOWERCASE_IDENTIFIER
        ;

IndexPart:              INDEX
                        '{' IndexTypes '}'
        |               AUGMENTS '{' Entry '}'
        |               /* empty */
        ;

IndexTypes:             IndexType
        |               IndexTypes ',' IndexType
        ;

IndexType:              IMPLIED Index
        |               Index
        ;

Index:                  ObjectName
        ;

Entry:                  ObjectName
        ;

DefValPart:             DEFVAL '{' Value '}'
        |               /* empty */
        ;

Value:                  valueofObjectSyntax
        |               '{' BitsValue '}'
        ;

BitsValue:              BitNames
        |               /* empty */
        ;

BitNames:               BitName
        |               BitNames ',' BitName
        ;
```

```
BitName:                identifier
        ;

ObjectName:             objectIdentifier
        ;

NotificationName:       objectIdentifier
        ;

ReferPart:              REFERENCE Text
        |               /* empty */
        ;

RevisionPart:           Revisions
        |               /* empty */
        ;

Revisions:              Revision
        |               Revisions Revision
        ;

Revision:               REVISION ExtUTCTime
                        DESCRIPTION Text
        ;

ObjectsPart:            OBJECTS '{' Objects '}'
        |               /* empty */
        ;

Objects:                Object
        |               Objects ',' Object
        ;

Object:                 ObjectName
        ;

NotificationsPart:      NOTIFICATIONS '{' Notifications '}'
        ;

Notifications:          Notification
        |               Notifications ',' Notification
        ;

Notification:           NotificationName
        ;

Text:                   QUOTED_STRING
        ;

ExtUTCTime:             QUOTED_STRING
        ;

objectIdentifier:
                        subidentifiers
        ;

subidentifiers:
                        subidentifier
        |               subidentifiers
                        subidentifier
        ;

subidentifier:
```

```
                        LOWERCASE_IDENTIFIER
        |               moduleName '.' LOWERCASE_IDENTIFIER
        |               number
        |               LOWERCASE_IDENTIFIER '(' number ')'
        |               moduleName '.' LOWERCASE_IDENTIFIER '(' number ')'
        ;

objectIdentifier_defval: subidentifiers_defval
        ;

subidentifiers_defval:  subidentifier_defval
        |               subidentifiers_defval subidentifier_defval
        ;

subidentifier_defval:   LOWERCASE_IDENTIFIER '(' number ')'
        |               number
        ;

objectGroupClause:      LOWERCASE_IDENTIFIER
                        OBJECT_GROUP
                        ObjectsPart
                        STATUS Status
                        DESCRIPTION Text
                        ReferPart
                        COLON_COLON_EQUAL '{' objectIdentifier '}'
        ;

notificationGroupClause: LOWERCASE_IDENTIFIER
                        NOTIFICATION_GROUP
                        NotificationsPart
                        STATUS Status
                        DESCRIPTION Text
                        ReferPart
                        COLON_COLON_EQUAL '{' objectIdentifier '}'
        ;

moduleComplianceClause: LOWERCASE_IDENTIFIER
                        MODULE_COMPLIANCE
                        STATUS Status
                        DESCRIPTION Text
                        ReferPart
                        ComplianceModulePart
                        COLON_COLON_EQUAL '{' objectIdentifier '}'
        ;

ComplianceModulePart:   ComplianceModules
        ;

ComplianceModules:      ComplianceModule
        |               ComplianceModules ComplianceModule
        ;

ComplianceModule:       MODULE ComplianceModuleName
                        MandatoryPart
                        CompliancePart
        ;

ComplianceModuleName:   UPPERCASE_IDENTIFIER objectIdentifier
        |               UPPERCASE_IDENTIFIER
        |               /* empty, only if contained in MIB module */
        ;

MandatoryPart:          MANDATORY_GROUPS '{' MandatoryGroups '}'
```

```
        |               /* empty */
        ;

MandatoryGroups:        MandatoryGroup
        |               MandatoryGroups ',' MandatoryGroup
        ;

MandatoryGroup:         objectIdentifier
        ;

CompliancePart:         Compliances
        |               /* empty */
        ;

Compliances:            Compliance
        |               Compliances Compliance
        ;

Compliance:             ComplianceGroup
        |               ComplianceObject
        ;

ComplianceGroup:        GROUP objectIdentifier
                        DESCRIPTION Text
        ;

ComplianceObject:       OBJECT ObjectName
                        SyntaxPart
                        WriteSyntaxPart
                        AccessPart
                        DESCRIPTION Text
        ;

SyntaxPart:             SYNTAX Syntax
        |               /* empty */
        ;

WriteSyntaxPart:        WRITE_SYNTAX WriteSyntax
        |               /* empty */
        ;

WriteSyntax:            Syntax
        ;

AccessPart:             MIN_ACCESS Access
        |               /* empty */
        ;

agentCapabilitiesClause: LOWERCASE_IDENTIFIER
                        AGENT_CAPABILITIES
                        PRODUCT_RELEASE Text
                        STATUS Status_Capabilities
                        DESCRIPTION Text
                        ReferPart
                        ModulePart_Capabilities
                        COLON_COLON_EQUAL '{' objectIdentifier '}'
        ;

ModulePart_Capabilities: Modules_Capabilities
        |               /* empty */
        ;

Modules_Capabilities:   Module_Capabilities
```

```
        |               Modules_Capabilities Module_Capabilities
        ;

Module_Capabilities:    SUPPORTS ModuleName_Capabilities
                        INCLUDES '{' CapabilitiesGroups '}'
                        VariationPart
        ;

CapabilitiesGroups:     CapabilitiesGroup
        |               CapabilitiesGroups ',' CapabilitiesGroup
        ;

CapabilitiesGroup:      objectIdentifier
        ;

ModuleName_Capabilities: objectIdentifier
        |               UPPERCASE_IDENTIFIER
        |               /* empty */
                        /* empty, only if contained in MIB module */
        ;

VariationPart:          Variations
        |               /* empty */
        ;

Variations:             Variation
        |               Variations Variation
        ;

Variation:              ObjectVariation
        |               NotificationVariation
        ;

NotificationVariation:  VARIATION ObjectName
                        AccessPart
                        DESCRIPTION Text
        ;

ObjectVariation:        VARIATION ObjectName
                        SyntaxPart
                        WriteSyntaxPart
                        AccessPart
                        CreationPart
                        DefValPart
                        DESCRIPTION Text
        ;

CreationPart:           CREATION_REQUIRES '{' Cells '}'
        |               /* empty */
        ;

Cells:                  Cell
        |               Cells ',' Cell
        ;

Cell:                   ObjectName
        ;

number:                 NUMBER
        ;
```

C. MOF Grammatik

Dieser Abschnitt enthält die ABNF Grammatik [50] der in 2.3 beschriebenen CIM MOF-Syntax, wie sie auch in [44] enthalten ist.

```
mofSpecification = *mofProduction

mofProduction = compilerDirective |
  classDeclaration |
  assocDeclaration |
  qualifierDeclaration |
  instanceDeclaration

compilerDirective = PRAGMA pragmaName "(" pragmaParameter ")"

pragmaName = IDENTIFIER

pragmaParameter = stringValue

classDeclaration = [ qualifierList ]
  CLASS className [ alias ] [ superClass ]
  "{" *classFeature "}" ";"

assocDeclaration = "[" ASSOCIATION *( "," qualifier ) "]"
  CLASS className [ alias ] [ superClass ]
  "{" *associationFeature "}" ";"

  // Context:
  // The remaining qualifier list must not include
  // the ASSOCIATION qualifier again. If the
  // association has no super association, then at
  // least two references must be specified! The
  // ASSOCIATION qualifier may be omitted in
  // sub associations.

className = schemaName "_" IDENTIFIER // NO whitespace !

  // Context:
  // Schema name must not include "_" !

alias = AS aliasIdentifer

aliasIdentifer = "$" IDENTIFIER // NO whitespace !

superClass = ":" className

classFeature = propertyDeclaration | methodDeclaration

associationFeature = classFeature | referenceDeclaration

qualifierList = "[" qualifier *( "," qualifier ) "]"
```

```
qualifier = qualifierName [ qualifierParameter ] [ ":" 1*flavor ]

qualifierParameter = "(" constantValue ")" | arrayInitializer

flavor = ENABLEOVERRIDE | DISABLEOVERRIDE | RESTRICTED |
  TOSUBCLASS | TRANSLATABLE

propertyDeclaration = [ qualifierList ] dataType propertyName
  [ array ] [ defaultValue ] ";"

referenceDeclaration = [ qualifierList ] objectRef referenceName
  [ defaultValue ] ";"

methodDeclaration = [ qualifierList ] dataType methodName
  "(" [ parameterList ] ")" ";"

propertyName = IDENTIFIER

referenceName = IDENTIFIER

methodName = IDENTIFIER

dataType = DT_UINT8 | DT_SINT8 | DT_UINT16 | DT_SINT16 |
  DT_UINT32 | DT_SINT32 | DT_UINT64 | DT_SINT64 |
  DT_REAL32 | DT_REAL64 | DT_CHAR16 |
  DT_STR | DT_BOOL | DT_DATETIME

objectRef = className REF

parameterList = parameter *( "," parameter )

parameter = [ qualifierList ] (dataType|objectRef) parameterName
  [ array ]

  // Context:
  // The qualifier list for "parameter" is restricted
  // to IN, OUT, or both.

parameterName = IDENTIFIER

array = "[" [positiveDecimalValue] "]"

positiveDecimalValue = positiveDecimalDigit *decimalDigit

defaultValue = "=" initializer

initializer = simpleInitializer | arrayInitializer | objectPath

simpleInitializer = constantValue | aliasIdentifier

arrayInitializer = "{" simpleInitializer *( "," simpleInitializer )"}"

constantValue = integerValue | realValue | charValue | stringValue |
  booleanValue | nullValue | objectPath

integerValue = binaryValue | octalValue | decimalValue | hexValue

qualifierDeclaration = QUALIFIER qualifierName qualifierType scope
  [ defaultFlavor ] ";"

qualifierName = IDENTIFIER
```

```
qualifierType = ":" dataType [ array ] [ defaultValue ]

scope = "," SCOPE "(" metaElement *( "," metaElement ) ")"

metaElement = SCHEMA | CLASS | ASSOCIATION | INDICATION | QUALIFIER
  PROPERTY | REFERENCE | METHOD | PARAMETER | ANY

defaultFlavor = "," FLAVOR "(" flavor *( "," flavor ) ")"

instanceDeclaration = [ qualifierList ] INSTANCE OF className [ alias ]
  "{" 1*propertyInit "}" ";"

propertyInit = [ qualifierList ] propertyName "=" initializer ";"

  // Context:
  // Note that associations may be instantiated also,
  // and thus references may be initialized. The
  // assigned value must be compatible with the
  // feature type.  For a reference, the assigned
  // value is either a string that holds an object
  // key or an alias that indirectly holds it.
```

These productions do not allow whitespace between the terms:

```
namespacePath = """ [ host ] localNsPath """

host = ( "\\" | "//" ) ( "." | hostName | address )

hostName = 1*ucs2Character

  // Context:
  // An unquoted string that specifies a host name
  // that can be resolved to a network address.
  // May not contain forward slash, backslash, or
  // colon.

address = 1*ucs2Character

  // Context:
  // An unquoted string that specifies a network
  // address in a standard format (for example,
  // an IPaddress in dotted decimal notation).

localNsPath = 1*( ( "/" | "\" ) namespaceName )

namespaceName = IDENTIFIER

schemaName = IDENTIFIER

  // Context:
  // Schema name must not include "_" !

objectPath = stringValue

  // Context:
  // A special string that serves as reference to
  // an object instance. It consists of an optional
  // namespace path followed by the object keys,
  // including the scoping hierarchy. See related
  // sections on object keys and naming for more
  // information.
```

```
fileName = stringValue

binaryValue = [ "+" | "-" ] 1*binaryDigit ( "b" | "B" )

binaryDigit = "0" | "1"

octalValue = [ "+" | "-" ] "0" 1*octalDigit

octalDigit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"

decimalValue = [ "+" | "-" ] ( positiveDecimalDigit *decimalDigit | "0" )

decimalDigit = "0" | positiveDecimalDigit

positiveDecimalDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

hexValue = [ "+" | "-" ] ( "0x" | "0X" ) 1*hexDigit

hexDigit = decimalDigit | "a" | "A" | "b" | "B" | "c" | "C" |
  "d" | "D" | "e" | "E" | "f" | "F"

realValue = [ "+" | "-" ] *decimalDigit "." 1*decimalDigit
  [ ( "e" | "E" ) [ "+" | "-" ] 1*decimalDigit ]

charValue = // any single-quoted Unicode-character, except
  // single quotes

stringValue = 1*( """ *ucs2Character """ )

ucs2Character = // any valid UCS-2-character

booleanValue = TRUE | FALSE

nullValue = NULL
```

The remaining productions are case-insensitive keywords:

```
ANY = "any"

AS = "as"

ASSOCIATION = "association"

CLASS = "class"

DISABLEOVERRIDE = "disableOverride"

DT_BOOL = "boolean"

DT_CHAR16 = "char16"

DT_DATETIME = "datetime"

DT_REAL32 = "real32"

DT_REAL64 = "real64"

DT_SINT16 = "sint16"

DT_SINT32 = "sint32"
```

```
DT_SINT64 = "sint64"

DT_SINT8 = "sint8"

DT_STR = "string"

DT_UINT16 = "uint16"

DT_UINT32 = "uint32"

DT_UINT64 = "uint64"

DT_UINT8 = "uint8"

ENABLEOVERRIDE = "enableoverride"

FALSE = "false"

FLAVOR = "flavor"

INDICATION = "indication"

INSTANCE = "instance"

METHOD = "method"

NULL = "null"

OF = "of"

PARAMETER = "parameter"

PRAGMA = "#pragma"

PROPERTY = "property"

QUALIFIER = "qualifier"

REF = "ref"

REFERENCE = "reference"

RESTRICTED = "restricted"

SCHEMA = "schema"

SCOPE = "scope"

TOSUBCLASS = "tosubclass"

TRANSLATABLE = "translatable"

TRUE = "true"
```

D. IDL Grammatik

Dieser Abschnitt enthält die Original-Grammatik der OMG CORBA 2.2 IDL, entnommen aus [76].

```
(1)  <specification>       ::=  <definition>+
(2)  <definition>          ::=  <type_dcl> ";"
                           |    <const_dcl> ";"
                           |    <except_dcl> ";"
                           |    <interface> ";"
                           |    <module> ";"
(3)  <module>              ::=  "module" <identifier> "{" <definition>+ "}"
(4)  <interface>           ::=  <interface_dcl>
                           |    <forward_dcl>
(5)  <interface_dcl>       ::=  <interface_header> "{" <interface_body> "}"
(6)  <forward_dcl>         ::=  "interface" <identifier>
(7)  <interface_header>    ::=  "interface" <identifier> [ <inheritance_spec> ]
(8)  <interface_body>      ::=  <export>*
(9)  <export>              ::=  <type_dcl> ";"
                           |    <const_dcl> ";"
                           |    <except_dcl> ";"
                           |    <attr_dcl> ";"
                           |    <op_dcl> ";"
(10)  <inheritance_spec>   ::=  ":" <scoped_name> { "," <scoped_name> }*
(11)  <scoped_name>        ::=  <identifier>
                           |    "::" <identifier>
                           |    <scoped_name> "::" <identifier>
(12)  <const_dcl>          ::=  "const" <const_type> <identifier> "=" <const_exp>
(13)  <const_type>         ::=  <integer_type>
                           |    <char_type>
                           |    <wide_char_type>
                           |    <boolean_type>
                           |    <floating_pt_type>
                           |    <string_type>
                           |    <wide_string_type>
                           |    <fixed_pt_const_type>
                           |    <scoped_name>
(14)  <const_exp>          ::=  <or_expr>
(15)  <or_expr>            ::=  <xor_expr>
                           |    <or_expr> "|" <xor_expr>
(16)  <xor_expr>           ::=  <and_expr>
                           |    <xor_expr> "^" <and_expr>
(17)  <and_expr>           ::=  <shift_expr>
                           |    <and_expr> "&" <shift_expr>
(18)  <shift_expr>         ::=  <add_expr>
                           |    <shift_expr> ">>" <add_expr>
                           |    <shift_expr> "<<" <add_expr>
(19)  <add_expr>           ::=  <mult_expr>
                           |    <add_expr> "+" <mult_expr>
                           |    <add_expr> "-" <mult_expr>
(20)  <mult_expr>          ::=  <unary_expr>
                           |    <mult_expr> "*" <unary_expr>
                           |    <mult_expr> "/" <unary_expr>
```

```
                                |       <mult_expr> "%" <unary_expr>
(21)  <unary_expr>              ::=     <unary_operator> <primary_expr>
                                |       <primary_expr>
(22)  <unary_operator>          ::=     "-"
                                |       "+"
                                |       "~"
(23)  <primary_expr>            ::=     <scoped_name>
                                |       <literal>
                                |       "(" <const_exp> ")"
(24)  <literal>                 ::=     <integer_literal>
                                |       <string_literal>
                                |       <wide_string_literal>
                                |       <character_literal>
                                |       <wide_character_literal>
                                |       <fixed_pt_literal>
                                |       <floating_pt_literal>
                                |       <boolean_literal>
(25)  <boolean_literal>         ::=     "TRUE"
                                |       "FALSE"
(26)  <positive_int_const>      ::=     <const_exp>
(27)  <type_dcl>                ::=     "typedef" <type_declarator>
                                |       <struct_type>
                                |       <union_type>
                                |       <enum_type>
(28)  <type_declarator>         ::=     <type_spec> <declarators>
(29)  <type_spec>               ::=     <simple_type_spec>
                                |       <constr_type_spec>
(30)  <simple_type_spec>        ::=     <base_type_spec>
                                |       <template_type_spec>
                                |       <scoped_name>
(31)  <base_type_spec>          ::=     <floating_pt_type>
                                |       <integer_type>
                                |       <char_type>
                                |       <wide_char_type>
                                |       <boolean_type>
                                |       <octet_type>
                                |       <any_type>
                                |       <object_type>
(32)  <template_type_spec>      ::=     <sequence_type>
                                |       <string_type>
                                |       <wide_string_type>
                                |       <fixed_pt_type>
(33)  <constr_type_spec>        ::=     <struct_type>
                                |       <union_type>
                                |       <enum_type>
(34)  <declarators>             ::=     <declarator> { "," <declarator> }*
(35)  <declarator>              ::=     <simple_declarator>
                                |       <complex_declarator>
(36)  <simple_declarator>       ::=     <identifier>
(37)  <complex_declarator>      ::=     <array_declarator>
(38)  <floating_pt_type>        ::=     "float"
                                |       "double"
                                |       "long" "double"
(39)  <integer_type>            ::=     <signed_int>
                                |       <unsigned_int>
(40)  <signed_int>              ::=     <signed_short_int>
                                |       <signed_long_int>
                                |       <signed_longlong_int>
(41)  <signed_short_int>        ::=     "short"
(42)  <signed_long_int>         ::=     "long"
(43   <signed_longlong_int>     ::=     "long" "long"
(44)  <unsigned_int>            ::=     <unsigned_short_int>
                                |       <unsigned_long_int>
```

```
                                    |       <unsigned_longlong_int>
(45)  <unsigned_short_int>          ::=     "unsigned" "short"
(46)  <unsigned_long_int>           ::=     "unsigned" "long"
(47)  <unsigned_longlong_int>       ::=     "unsigned" "long" "long"
(48)  <char_type>                   ::=     "char"
(49)  <wide_char_type>              ::=     "wchar"
(50)  <boolean_type>                ::=     "boolean"
(51)  <octet_type>                  ::=     "octet"
(52)  <any_type>                    ::=     "any"
(53)  <object_type>                 ::=     "Object"
(54)  <struct_type>                 ::=     "struct" <identifier> "{" <member_list> "}"
(55)  <member_list>                 ::=     <member>+
(56)  <member>                      ::=     <type_spec> <declarators> ";"
(57)  <union_type>                  ::=     "union" <identifier> "switch" "("
                                    |        <switch_type_spec> ")" "{" <switch_body> "}"
(58)  <switch_type_spec>            ::=     <integer_type>
                                    |       <char_type>
                                    |       <boolean_type>
                                    |       <enum_type>
                                    |       <scoped_name>
(59)  <switch_body>                 ::=     <case>+
(60)  <case>                        ::=     <case_label>+ <element_spec> ";"
(61)  <case_label>                  ::=     "case" <const_exp> ":"
                                    |       "default" ":"
(62)  <element_spec>                ::=     <type_spec> <declarator>
(63)  <enum_type>                   ::=     "enum" <identifier> "{" <enumerator> {
                                            "," <enumerator> }* "}"
(64)  <enumerator>                  ::=     <identifier>
(65)  <sequence_type>               ::=     "sequence" "<" <simple_type_spec>
                                            "," <positive_int_const> ">"
                                    |       "sequence" "<" <simple_type_spec> ">"
(66)  <string_type>                 ::=     "string" "<" <positive_int_const> ">"
                                    |       "string"
(67)  <wide_string_type>            ::=     "wstring" "<" <positive_int_const> ">"
                                    |       "wstring"
(68)  <array_declarator>            ::=     <identifier> <fixed_array_size>+
(69)  <fixed_array_size>            ::=     "[" <positive_int_const> "]"
(70)  <attr_dcl>                    ::=     [ "readonly" ] "attribute"  <param_type_spec>
                                            <simple_declarator> {  "," <simple_declarator> }*
(71)  <except_dcl>                  ::=     "exception" <identifier> "{" <member>* "}"
(72)  <op_dcl>                      ::=     [ <op_attribute> ] <op_type_spec> <identifier>
                                            <parameter_cdls> [ <raises_expr> ] [ <context_expr> ]
(73)  <op_attribute>                ::=     "oneway"
(74)  <op_type_spec>                ::=     <param_type_spec>
                                    |       "void"
(75)  <parameter_dcls>              ::=     "(" <param_dcl> { "," <param_dcl> }* ")"
                                    |       "(" ")"
(76)  <param_dcl>                   ::=     <param_attribute> <param_type_spec>
                                            <simple_declarator>
(77)  <param_attribute>             ::=     "in"
                                    |       "out"
                                    |       "inout"
(78)  <raises_expr>                 ::=     "raises" "(" <scoped_name> { ","  <scoped_name> }* ")"
(79)  <context_expr>                ::=     "context" "(" <string_literal>
                                            { "," <string_literal> }* ")"
(80)  <param_type_spec>             ::=     <base_type_spec>
                                    |       <string_type>
                                    |       <wide_string_type>
                                    |       <fixed_pt_type>
                                    |       <scoped_name>
(81)  <fixed_pt_type>               ::=     "fixed" "<" <positive_int_const>
                                            "," <integer_literal> ">"
(82)  <fixed_pt_const_type>         ::=     "fixed"
```

E. SMIng Spezifikation: draft-irtf-nmrg-sming-00

Dieser Abschnitt enthält die im Rahmen von Kapitel 3 erarbeitete SMIng Spezifikation. Sie wurde als eigenständiges und abgeschlossenes Dokument nach den Richtlinien für RFCs (*Request for Comments*) der IETF (*Internet Engineering Task Force*) verfaßt [63], [64], um sie an die IRTF (*Internet Research Task Force*) NMRG (*Network Management Research Group*) übergeben und in ihrem Rahmen öffentlich publizieren und weiterentwickeln zu können.

```
Network Working Group                                         F. Strauss
Internet-Draft                                           TU Braunschweig
Expires: September 30, 1999                                   April 1999


           SMIng - A new Structure of Management Information
                       draft-irtf-nmrg-sming-00

Status of this Memo

   This document is an Internet-Draft and is in full conformance with
   all provisions of Section 10 of RFC2026.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF), its areas, and its working groups. Note that
   other groups may also distribute working documents as
   Internet-Drafts.

   Internet-Drafts are draft documents valid for a maximum of six
   months and may be updated, replaced, or obsoleted by other documents
   at any time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   To view the entire list of Internet-Draft Shadow Directories, see
   http://www.ietf.org/shadow.html.

   This Internet-Draft will expire on September 30, 1999.

Abstract

   This memo presents a language for management information
   specifications. It eliminates known problems present in the current
   SMIv2 as specified in [2], [3], and [4]. Language extensibility
   features and a more efficient and programmer-friendqly notation are
   introduced. Conversions from SMIv2 to SMIng and vice versa are also
   considered by this document.

Copyright Notice

   Copyright (C) The Internet Society (1999). All Rights Reserved.










Strauss                 Expires September 30, 1999              [Page 1]
```

Table of Contents

1. Introduction

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules. These modules are written conforming to a specification language, the Structure of Management Information (SMI). There are different versions of the SMI. SMIv1 [5], [6], [7] and SMIv2 [2], [3], [4] are based on adapted subsets of OSI's Abstract Syntax Notation One, ASN.1 [9]. It is the purpose of this document to define a successor of SMIv1 and SMIv2, the SMIng, in a self-contained way independent from other standardization bodies' documents.

Section 2 gives an overview of the information model without respect to the SMIng language while the subsequent sections present the concepts of SMIng: the base types, the SMIng file structure, and all SMIng statements in detail. Section 14 contains two core modules that are part of the SMIng specification. Section 15 lists rules to follow when changes are applied to a module. One flexible feature of SMIng is its extensibility which is described in Section 16. Finally, Section 17 discusses conversions of SMIv2 modules to SMIng and vice versa. Appendix A contains the grammar of SMIng in ABNF [8] notation.

1.1 Terminology

Many terms in SMIv1 and SMIv2 are derived from ASN.1. In some cases, SMIv1/v2 uses different terms for equal or similar language items without precise definitions. Appendix B presents a glossary of terms used throughout this document and in related SMIng documents. Also refer to the glossary and Appendix A to get detailed information on the syntax of single language items where not explained in place.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].

1.2 Relation to SNMP

Although SMIng is primarily intended to be applied to the the SNMP management framework [10], the information model represented by SMIng is designed to have as few relations to a specific management protocol as possible. However, there are some implications of SNMP on SMIv2 that have to be retained in SMIng:

o In SNMP the last item on a list of objects used to identify a table row can be encoded without an explicit length encoding. This has to be expressed by the SMIng 'implied' keyword. See

Section 10.5 for details.

- o SNMP allows to distinguish several related types "on the wire" while SMIng has a minimal but complete set of base types. To keep SMIng compatible with the SNMP framework, additional types derived from SMIng base types are defined in the standard SMIng module IRTF-NMRG-SMING-TYPES (Section 14.2).

- o The encoding of values of type 'Bits' in SNMP is dependent on their named number values. It is RECOMMENDED that they start at 0 and be numbered contiguously to allow a compact encoding for use with SNMP.

- o SNMP implies some restrictions on indexing columnar objects: Those columns that are used for indexing have to be defined as non-accessible auxiliary objects, with two exceptions. Hence, the access level 'noaccess' has to be retained in SMIng.

- o In order to achieve compatibility with SNMPv1 traps, the next to last sub-identifier of notification object identifiers SHOULD have the value zero.

2. The Information Model

Internet management is based on the model of "managed objects". A managed object represents a class of any real or synthesized variable of systems that are to be managed. Those objects are organized hierarchically in an "object identifier tree", where only leaf nodes may represent objects.

Nodes may also identify tables, row definitions of tables, notifications, groups of objects and/or notifications, compliance statements, modules or other information. Each node is identified by an unique "object identifier" value which is an ordered list of non-negative numbers, named "sub-identifiers", where the left-most sub-identifier refers to the node next to the root of the tree and the right-most sub-identifier refers to the node that is identified by the complete object identifier.

The SMI in general, and SMIng in particular, is the information model used to define and organize the structure of managed objects and related information. Thus, SMIng describes a language designed to specify management information in a way readable to computer programs, named MIB compilers, as well as to human readers.

Related management information is defined in MIB modules. A module may refer to definitions from other modules by importing the required identifiers from those modules. Each module may serve one or multiple purposes:

o the definition of inter-related managed objects, notifications and groups of them,

o the definition of (restricted) types for use in the local module or in other modules,

o the definition of compliance statements for groups of objects and notifications to guide agent implementors in what they have to serve or implementors of management applications in what they can expect,

o the definition of SMIng extensions to allow the local module or other modules to specify information beyond the scope of the base SMIng in a machine readable notation,

o the definition of information beyond the scope of the base SMIng statements, based on locally defined or imported SMIng extensions, like capability statements for agent implementations.

This classification scheme does not imply a rigid taxonomy. For example, a "standard" module will normally include definitions of managed objects and a compliance statement. Similarly, an "enterprise-specific" module might include definitions of managed objects and a capability statement based on an extension. Of course, a "standard" module may not contain capability statements.

Each module is identified by an upper-case identifier. The names of all standard modules must be unique (but different versions of the same module should have the same name). Developers of enterprise modules are encouraged to choose names for their modules that will have a low probability of colliding with standard or other enterprise modules, e.g. by using the enterprise or organization name as a prefix.

2.1 Identifiers

Identifiers are used to identify different kinds of SMIng items by name. These names are valid in a namespace which is dependent on the SMIng item. Those items are

- o modules (Section 5), whose namespace is the global range of all SMIng definitions,
- o defined data types (Section 7), whose namespace is the local module where the type is defined,
- o elements of enumeration and bitset types (Section 3.10, Section 3.11), whose namespace is the related type definition construct and all constructs using that type,
- o object identifier tree nodes of different kinds (Section 5, and Section 8 through Section 13), whose namespace is the local module where the node is defined, and
- o SMIng extension statements (Section 6), whose namespace is the local module where the extension is defined.

Each identifier starts with an upper-case or lower-case character, dependent on the kind of SMIng item, followed by zero or more letters, digits, and hyphens.

For all identifiers of a single kind of item defined in a namespace, the identifier MUST be unique and SHOULD be mnemonic. Identifiers MUST NOT exceed 64 characters in length. Furthermore, the set of all identifiers defined in all modules of a single standardization body or organization SHOULD be unique. This promotes a common language for humans to use when discussing a module.

To reference an item that is defined in the local module, its definition MUST sequentially precede the reference. Thus, there MUST

NOT be any forward references, except in two cases:

o The specification of table indexing objects in a table row may include objects that cannot be defined without the definition of the current table row. Hence table indexing statements are allowed to include forward references.

o Similarly, the specification of columns needed in row creation operations is allowed to include forward references.

To reference an item, that is defined in an external module it MUST be imported into the local module's namespace (Section 5.1). Identifiers that are neither defined nor imported MUST NOT be visible in the local module. On the other hand, all items defined in a module are implicitly exported.

Note when identifiers from external modules are referenced, there is the possibility of name collisions. As such, if different items with the same identifier are imported or if imported identifiers collide with identifiers of locally defined items, then this ambiguity is resolved by prefixing those identifiers with the names of their modules and the namespace operator '::', i.e. 'Module::item'. Of course, this notation can be used to refer to identifiers even when there is no name collision.

Note that SMIng language keywords MUST NOT be imported. The main set of keywords are SMIng statements and base types. See the '...Keyword' rules of the SMIng ABNF grammar in Appendix A for a list of those keywords.

Finally, by convention, if the identifier refers to an object with a counter type (Counter32 or Counter64, derived from Unsigned32 and Unsigned64) then the identifier used for the object SHOULD denote plurality.

2.2 Object Identifier Hierarchy

The layers of the object identifier tree near the root are well defined and organized by standardization bodies. The first level next to the root has three nodes:

0: ccitt

1: iso

2: joint-iso-ccitt

Note that the renaming of the Commite Consultatif International de Telegraphique et Telephonique (CCITT) to International

Telecommunications Union (ITU) had no consequence on the names used in the object identifier tree.

The root of the subtree administered by the Internet Assigned Numbers Authority (IANA) for the Internet is '1.3.6.1' which is assigned with the identifier 'internet'. That is, the Internet subtree of object identifiers starts with the prefix '1.3.6.1.'.

Several branches underneath this subtree are used for network management:

The 'mgmt' (internet.2) subtree is used to identify "standard" information.

The 'experimental' (internet.3) subtree is used to identify information being designed by working groups of the IETF or IRTF. If a module produced by a working group becomes a "standard" module then at the very beginning of its entry onto the Internet standards track, the information is moved under the mgmt subtree.

The 'private' (internet.4) subtree is used to identify information defined unilaterally. The 'enterprises' (private.1) subtree beneath private is used, among other things, to permit providers of networking subsystems to register models of their products.

These and some other nodes are defined in the SMIng standard module IRTF-NMRG-SMING (Section 14.1).

2.3 Kinds of Nodes

Each node in the object identifier tree may be of a single kind which may represent management information or not:

o simple nodes, that do not represent management information, but may be used for grouping nodes in a subtree,

o nodes representing the identity of a module to allow references to a module in other objects of type 'ObjectIdentifier',

o scalar objects, which have zero or one object instance. See Section 2.4 for scalar objects' instances,

o tables, which represent the root node of a collection of information structured in table rows,

o rows, which belong to a table (that is, the table's full object identifier is a prefix of the row's object identifier) and represent a sequence of one or more columnar objects. A row's object identifier MUST be constructed by appending a '1' sub-identifier to the table's object identifier,

- columnar objects, which belong to a row (that is, the row's full object identifier is a prefix of the columnar objects' object identifier) and have zero or more object instances. Each columnar object's object identifier MUST be constructed by appending a single sub-identifier to the row's object identifier. See Section 2.4 for columnar objects' instances,

- notifications, which represent information that is sent by agents within unsolicited transmissions,

- groups of objects and notifications, which may be used for compliance statements or other purposes,

- compliance statements which define requirements for MIB module implementations,

- other nodes, that may be used to identify arbitrary information.

2.4 Scalar and Columnar Objects' Instances

Instances of managed objects are identified by appending an instance-identifier to the object's object identifier. Scalar objects and columnar objects use different ways to construct the instance-identifier.

Scalar objects have zero or one object instance. If it exists, it is identified by appending a single '0' sub-identifier to the object identifier of the scalar object.

Within tables, different instances of the same columnar object are identified by appending a sequence of one or more sub-identifiers to the object identifier of the columnar object which consists of the values of object instances that unambiguously distinguish a table row. These indexing objects can be columnar objects of the same and/or another table, but MUST NOT be scalar objects. Multiple applications of the same object in a single table indexing specification are strongly discouraged.

The base types (Section 3) of the indexing objects indicate how to form the instance-identifier:

- integer-valued or enumeration-valued: a single sub-identifier taking the integer value (this works only for non-negative integers),

- string-valued, fixed-length strings (or variable-length with compact encoding): 'n' sub-identifiers, where 'n' is the length of the string (each octet of the string is encoded in a separate sub-identifier),

o string-valued, variable-length strings or bits-valued: 'n+1' sub-identifiers, where 'n' is the length of the string or bits encoding (the first sub-identifier is 'n' itself, following this, each octet of the string or bits is encoded in a separate sub-identifier),

o object identifier-valued (with compact encoding): 'n' sub-identifiers, where 'n' is the number of sub-identifiers in the value (each sub-identifier of the value is copied into a separate sub-identifier),

o object identifier-valued: 'n+1' sub-identifiers, where 'n' is the number of sub-identifiers in the value (the first sub-identifier is 'n' itself, following this, each sub-identifier in the value is copied),

Note that compact encoding can only be applied to an object having a variable-length syntax (e.g., variable-length strings, bits objects or object identifier-valued objects). Further, compact encoding can only be associated with the last object in a list of indexing objects. Finally, compact encoding may not be used on a variable-length string object if that string might have a value of zero-length.

Instances identified by use of integer-valued or enumeration-valued objects are RECOMMENDED to be numbered starting from one (i.e., not from zero).

Objects which are both specified for indexing in a row and also columnar objects of the same row are termed auxiliary objects. Auxiliary objects SHOULD be non-accessible, except in the following circumstances:

o within a MIB module originally written to conform to SMIv1, or

o a row must contain at least one columnar object which is not an auxiliary object. In the event that all of a row's columnar objects are also specified to be indexing objects then one of them MUST be accessible. (Note that this situation does not arise for a row allowing create access, since such a row will have a status column which will not be an auxiliary object.)

3. Base Types and Derived Types

SMIng has a minimal but complete set of base types, similar to those of many programming languages, but with some differences due to special requirements from the management information model described in Section 2.

Additional types may be defined, derived from those base types and even from other derived types. Derived types may use subtyping to formally restrict the possible values. A set of derived types commonly used in the SNMP framework is defined in the SMIng standard module IRTF-NMRG-SMING-TYPES (Section 14.2).

Note that types can also be restricted "inline" in object definitions (Section 9, Section 10.6) or in refinements of compliance statements (Section 13) without referring an explicitly defined type.

The different base types and its derived types allow different kinds of subtyping, namely size restrictions and range restrictions. See the following sections on base types (Section 3.1 through Section 3.11) for details.

3.1 OctetString

The OctetString base type represents arbitrary binary or textual data. Although SMIng has a theoretical size limitation of 2^32-1 (4294967295) octets for this base type, MIB designers should realize that there may be implementation and interoperability limitations for sizes in excess of 255 octets.

Values of octet strings may be denoted as textual data enclosed in double quotes or as arbitrary binary data denoted as a '0x'-prefixed hexadecimal value of arbitrary but even length, where each pair of hexadecimal digits represents a single octet. Letters in hexadecimal values MAY be upper-case but lower-case characters are RECOMMENDED. Textual data may contain any number (possibly zero) of any 7-bit displayable ASCII characters except double quote '"', including tab characters, spaces and line terminator characters (nl or cr & nl). Textual data may span multiple lines, where each subsequent line prefix containing only white space up to the column where the first line's data starts SHOULD be skipped by parsers for a better text formatting. Note that textual data is not only used in object (default) values, but also in arguments of many SMIng statements.

When defining a type derived (directly or indirectly) from the OctetString base type, the size in octets may be restricted by appending a list of size ranges or explicit size values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A size range consists of a lower bound, two consecutive dots '..' and an upper bound. Each value can be given in decimal or '0x'-prefixed hexadecimal notation. Of course, size restricting values MUST NOT be negative. If multiple values or ranges are given, they all MUST be disjunct and SHOULD be in ascending order. If a size restriction is applied to an already size restricted octet string the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative size choices, or splitting ranges into multiple ranges with intermediate gaps.

Value Examples:

```
"This is a multiline
 textual data example."         // legal
"This is "illegally" quoted."   // illegal quotes
"But this is 'ok'."             // legal apostrophe quoting
""                              // legal zero length
0x123                           // illegal odd hex length
0x534d 496e 670a                // legal octet string
```

Restriction Examples:

```
OctetString (0 | 4..255)        // legal size spec
OctetString (4)                 // legal exact size
OctetString (-1 | 1)            // illegal negative size
OctetString (1 | 1..10)         // illegal overlapping
```

3.2 ObjectIdentifier

The ObjectIdentifier base type represents any object identifier value, which is a list of up to 128 numbers, named sub-identifiers. Each sub-identifier has a value between 0 and 2^32-1 (4294967295).

Values of object identifiers may be denoted as a sequence of numerical sub-identifier values (decimal or '0x'-prefixed hexadecimal) separated by single dots and without any intermediate white space. Alternatively (and preferred in most cases), the first element may be a previously defined or imported lower-case identifier, representing a static object identifier prefix while the total number of sub-identifiers MUST NOT exceed 128 including the expanded identifier.

Object identifier derived types cannot be restricted in any way.

Value Examples:

```
  0                           // legal single subid
  1.3.6.1                     // legal numerical oid
  mib-2.1                     // legal oid with identifier prefix
  internet.4.1.0x0627.0x01    // legal oid with hex subids
  iso.-1                      // illegal negative subid
  iso.org.6                   // illegal non-heading identifier
```

3.3 Integer32

The Integer32 base type represents integer values between -2^31 (-2147483648) and 2^31-1 (2147483647).

Values of type Integer32 may be denoted as decimal or hexadecimal numbers, where only decimal numbers can be negative. Other decimal numbers than zero MUST NOT have leading zero digits. Hexadecimal numbers are prefixed by '0x' and MUST have an even number of hexadecimal digits, where letters MAY be upper-case but lower-case characters are RECOMMENDED.

When defining a type derived (directly or indirectly) from the Integer32 base type, the set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots '..' and an upper bound. Each value can be given in decimal or '0x'-prefixed hexadecimal notation. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps.

Value Examples:

```
  015                         // illegal leading zero
  -123                        // legal negative value
  - 1                         // illegal intermediate space
  0xabc                       // illegal hexadecimal value length
  -0xff                       // illegal sign on hex value
  0xf00f                      // legal hexadecimal value
```

Restriction Examples:

```
  Integer32 (0 | 5..10)       // legal range spec
  Integer32 (4..8 | 5..10)    // illegal overlapping
```

3.4 Integer64

The Integer64 base type represents integer values between -2^63 (-9223372036854775808) and 2^63-1 (9223372036854775807).

Values of type Integer64 may be denoted as decimal or hexadecimal numbers, where only decimal numbers can be negative. Other decimal numbers than zero MUST NOT have leading zero digits. Hexadecimal numbers are prefixed by '0x' and MUST have an even number of hexadecimal digits, where letters MAY be upper-case but lower-case characters are RECOMMENDED.

When defining a type derived (directly or indirectly) from the Integer64 base type, the set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots '..' and an upper bound. Each value can be given in decimal or '0x'-prefixed hexadecimal notation. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps.

3.5 Unsigned32

The Unsigned32 base type represents positive integer values between 0 and 2^32-1 (4294967295).

Values of type Unsigned32 may be denoted as decimal or hexadecimal numbers. Other decimal numbers than zero MUST NOT have leading zero digits. Hexadecimal numbers are prefixed by '0x' and MUST have an even number of hexadecimal digits, where letters MAY be upper-case but lower-case characters are RECOMMENDED.

When defining a type derived (directly or indirectly) from the Unsigned32 base type, the set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots '..' and an upper bound. Each value can be given in decimal or '0x'-prefixed hexadecimal notation. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps.

3.6 Unsigned64

The Unsigned64 base type represents positive integer values between 0 and 2^64-1 (18446744073709551615).

Values of type Unsigned64 may be denoted as decimal or hexadecimal numbers. Other decimal numbers than zero MUST NOT have leading zero digits. Hexadecimal numbers are prefixed by '0x' and MUST have an even number of hexadecimal digits, where letters MAY be upper-case but lower-case characters are RECOMMENDED.

When defining a type derived (directly or indirectly) from the Unsigned64 base type, the set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots '..' and an upper bound. Each value can be given in decimal or '0x'-prefixed hexadecimal notation. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps.

3.7 Float32

The Float32 base type represents floating point values of single precision as described by [11].

Values of type Float32 may be denoted as a decimal fraction with an optional exponent as known from many programming languages. See the grammar rule 'floatValue' of Appendix A for the detailed syntax. Special values are 'snan' (signaling Not-a-Number), 'qnan' (quiet Not-a-Number), 'neginf' (negative infinity), and 'posinf' (positive infinity). Note that -0.0 and +0.0 are different floating point values. 0.0 is equal to +0.0.

When defining a type derived (directly or indirectly) from the Float32 base type the, set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots '..' and an upper bound. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps. The special values 'snan', 'qnan', 'neginf', and 'posinf' must be explicitly listed in restrictions if they shall be included, where 'snan' and 'qnan' cannot be used in ranges.

Note that encoding is not subject to this specification. It has to be described by protocols that transport objects of type Float32. Note also that most floating point encodings disallow the representation of many values that can be written as decimal fractions as used in SMIng for human readability. Therefore, explicit values in floating point type restrictions should be handled with care.

Value Examples:

```
  00.1                          // illegal leading zero
  3.1415                        // legal value
  -2.5E+3                       // legal negative exponential value
```

Restriction Examples:

```
  Float32 (-1.0..1.0)         // legal range spec
  Float32 (1 | 3.3 | 5)       // legal, probably unrepresentable 3.3
  Float32 (-10.0..10.0 | 0)   // illegal overlapping
```

3.8 Float64

The Float64 base type represents floating point values of double precision as described by [11].

Values of type Float64 may be denoted as a decimal fraction with an optional exponent as known from many programming languages. See the grammar rule 'floatValue' of Appendix A for the detailed syntax. Special values are 'snan' (signaling Not-a-Number), 'qnan' (quiet Not-a-Number), 'neginf' (negative infinity), and 'posinf' (positive infinity). Note that -0.0 and +0.0 are different floating point values. 0.0 is equal to +0.0.

When defining a type derived (directly or indirectly) from the Float64 base type the, set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe '|' characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots '..' and an upper bound. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps. The special values 'snan', 'qnan', 'neginf', and 'posinf' must be explicitly listed in restrictions if they shall be included, where 'snan' and 'qnan' cannot be used in ranges. If 0.0 is included, -0.0 will also be.

Note that encoding is not subject to this specification. It has to be described by protocols that transport objects of type Float64. Note also that most floating point encodings disallow the representation of many values that can be written as decimal fractions as used in SMIng for human readability. Therefore, explicit values in floating point type restrictions should be handled with care.

Value Examples:

```
00.1                        // illegal leading zero
3.1415                      // legal value
-2.5E+3                     // legal negative exponential value
```

Restriction Examples:

```
Float64 (-1.0..1.0)         // legal range spec
Float64 (1 | 3.3 | 5)       // legal, probably unrepresentable 3.3
Float64 (-10.0..10.0 | 0)   // illegal overlapping
```

3.9 Float128

The Float128 base type represents floating point values of quadruple precision as described by [11].

Values of type Float128 may be denoted as a decimal fraction with an optional exponent as known from many programming languages. See the grammar rule ‘floatValue’ of Appendix A for the detailed syntax. Special values are ‘snan’ (signaling Not-a-Number), ‘qnan’ (quiet Not-a-Number), ‘neginf’ (negative infinity), and ‘posinf’ (positive infinity). Note that -0.0 and +0.0 are different floating point values. 0.0 is equal to +0.0.

When defining a type derived (directly or indirectly) from the Float128 base type the, set of possible values may be restricted by appending a list of ranges or explicit values, separated by pipe ‘|’ characters and the whole list enclosed in parenthesis. A range consists of a lower bound, two consecutive dots ‘..’ and an upper bound. If multiple values or ranges are given they all MUST be disjunct and SHOULD be in ascending order. If a value restriction is applied to an already restricted type the new restriction MUST be more limiting, that is raising the lower bounds, reducing the upper bounds, reducing the alternative choices, or splitting ranges into multiple ranges with intermediate gaps. The special values ‘snan’, ‘qnan’, ‘neginf’, and ‘posinf’ must be explicitly listed in restrictions if they shall be included, where ‘snan’ and ‘qnan’ cannot be used in ranges. If 0.0 is included, -0.0 will also be.

Note that encoding is not subject to this specification. It has to be described by protocols that transport objects of type Float128. Note also that most floating point encodings disallow the representation of many values that can be written as decimal fractions as used in SMIng for human readability. Therefore, explicit values in floating point type restrictions should be handled with care.

Value Examples:

```
00.1                       // illegal leading zero
3.1415                     // legal value
-2.5E+3                    // legal negative exponential value
```

Restriction Examples:

```
Float128 (-1.0..1.0)       // legal range spec
Float128 (1 | 3.3 | 5)     // legal, probably unrepresentable 3.3
Float128 (-10.0..10.0 | 0) // illegal overlapping
```

3.10 Enumeration

The Enumeration base type represents values from a set of integers in the range between -2^31 (-2147483648) and 2^31-1 (2147483647), where each value has an assigned name. The list of those named numbers has to be comma-separated, enclosed in parenthesis and appended to the ‘Enumeration’ keyword. Each named number is denoted by its lower-case identifier followed by the assigned integer value, denoted as a decimal or ‘0x’-prefixed hexadecimal number, enclosed in parenthesis. Every name and every number in an enumeration type MUST be unique. It is RECOMMENDED that values start at 1 and be numbered contiguously.

Values of enumeration types may be denoted as decimal or ‘0x’-prefixed hexadecimal numbers or preferably as their assigned names.

When defining a type derived (directly or indirectly) from an enumeration type the set, of named numbers may be restricted by removing one or more named numbers. But no named numbers may be added or changed regarding its name, value, or both.

Type and Value Examples:

```
Enumeration (up(1), down(2), testing(3))

0                           // illegal, value 0 out of range
up                          // legal value given by name
2                           // legal value given by number
```

3.11 Bits

The Bits base type represents bit sets. That is, a Bits value is a set of flags identified by small integer numbers starting at 0. Each number has an assigned name. The list of those named numbers has to be comma-separated, enclosed in parenthesis and appended to the 'Bits' keyword. Each named number is denoted by its lower-case identifier followed by the assigned integer value, denoted as a decimal or '0x'-prefixed hexadecimal number, enclosed in parenthesis. Every name and every number in a bits type MUST be unique. It is RECOMMENDED that numbers start at 0 and be numbered contiguously.

Values of bits types may be denoted as a comma-separated list of decimal or '0x'-prefixed hexadecimal numbers or preferably their assigned names enclosed in parenthesis. There MUST NOT be any element (by name or number) listed more than once. It is RECOMMENDED to list elements in ascending order, although the order is semantically irrelevant.

When defining a type derived (directly or indirectly) from a bits type, the set of named numbers may be restricted by removing one or more named numbers. But no named numbers may be added or changed regarding its name, value, or both.

Type and Value Examples:

```
Bits (readable(0), writeable(1), executable(2))

 ()                            // legal empty value
 (readable, writeable, 2)      // legal value
 (0, readable, executable)     // illegal, readable(0) appears twice
 (writeable, 4)                // illegal, element 4 out of range
```

3.12 Display Formats

Some SMIng definitions, namely scalar and columnar object definitions and type definitions, allow the specification of a format to be used, when a value of that object or an object of that type is displayed. Format specifications are represented as textual data.

When the object or type has an underlying base type of Integer32, Integer64, Unsigned32, or Unsigned64, the format consists of an integer-format specification, containing two parts. The first part is a single character suggesting a display format, either: 'x' for hexadecimal, or 'd' for decimal, or 'o' for octal, or 'b' for binary. For all types, when rendering the value, leading zeros are omitted, and for negative values, a minus sign is rendered immediately before the digits. The second part is always omitted for 'x', 'o' and 'b', and need not be present for 'd'. If present, the second part starts with a hyphen and is followed by a decimal number, which defines the implied decimal point when rendering the value. For example 'd-2' suggests that a value of 1234 be rendered as '12.34'.

When the object or type has an underlying base type of OctetString, the hint consists of one or more octet-format specifications. Each specification consists of five parts, with each part using and removing zero or more of the next octets from the value and producing the next zero or more characters to be displayed. The octets within the value are processed in order of significance, most significant first.

The five parts of a octet-format specification are:

1. the (optional) repeat indicator; if present, this part is a '*', and indicates that the current octet of the value is to be used as the repeat count. The repeat count is an unsigned integer (which may be zero) which specifies how many times the remainder of this octet-format specification should be successively applied. If the repeat indicator is not present, the repeat count is one.

2. the octet length: one or more decimal digits specifying the number of octets of the value to be used and formatted by this octet-specification. Note that the octet length can be zero. If less than this number of octets remain in the value, then the lesser number of octets are used.

3. the display format, either: 'x' for hexadecimal, 'd' for decimal, 'o' for octal, 'a' for ASCII, or 't' for UTF-8 [12]. If the octet length part is greater than one, and the display format part refers to a numeric format, then network byte-ordering (big-endian encoding) is used interpreting the octets in the value. The octets processed by the 't' display format do not necessarily form an integral number of UTF-8 characters. Trailing octets which do not form a valid UTF-8 encoded character are discarded.

4. the (optional) display separator character; if present, this part is a single character which is produced for display after each application of this octet-specification; however, this character is not produced for display if it would be immediately followed by the display of the repeat terminator character for this octet specification. This character can be any character other than a decimal digit and a '*'.

5. the (optional) repeat terminator character, which can be present only if the display separator character is present and this octet specification begins with a repeat indicator; if present, this part is a single character which is produced after all the zero or more repeated applications (as given by the repeat count) of this octet specification. This character can be any character other than a decimal digit and a '*'.

Output of a display separator character or a repeat terminator character is suppressed if it would occur as the last character of the display.

If the octets of the value are exhausted before all the octet format specification have been used, then the excess specifications are ignored. If additional octets remain in the value after interpreting all the octet format specifications, then the last octet format specification is re-interpreted to process the additional octets, until no octets remain in the value.

Note that for some types (e.g. ObjectIdentifier) no format specifications are defined and SHOULD be omitted. Implementations MUST ignore format specifications they cannot interpret. Also note that the SMIng grammar (Appendix A) does not specify the syntax of format specifications.

4. The SMIng File Structure

The topmost container of SMIng information is a file. An SMIng file may contain zero, one or more modules. It is RECOMMENDED to separate modules into files named by their modules, where possible. Though, for dedicated purposes it may be reasonable to collect several modules in a single file.

The top level SMIng construct is the ‘module’ statement (Section 5) that defines a single MIB module. A module contains a sequence of sections in an obligatory order with different kinds of definitions. Whether these sections contain statements or remain empty mainly depends on the purpose of the module (see Section 2).

4.1 Comments

Comments can be included at any position in an SMIng file, except in between the characters of a single token like those of a quoted strings. However, it is RECOMMENDED that all substantive descriptions be placed within an appropriate description clause.

Comments commence with a pair of adjacent slashes ‘//’ and end at the end of the line.

4.2 Statements and Arguments

SMIng has a very small set of basic grammar rules based on the concept of statements. Each statement starts with a lower-case keyword identifying the statement followed by a number (possibly zero) of arguments. An argument may be quoted text, an identifier, a value of any base type, a list of identifiers enclosed in parenthesis ‘( )’ or a statement block enclosed in curly braces ‘{ }’. Since statement blocks are valid arguments, it is possible to nest statement sequences. Each statement is terminated by a semicolon ‘;’.

The core set of statements may be extended using the SMIng ‘extension’ statement. See Section 6 and Section 16 for details.

At places where a statement is expected, but an unknown lower-case word is read, those statements MUST be skipped up to the proper semicolon, including nested statement blocks.

5. The module Statement

The ‘module’ statement is used as a container of all definitions of a single SMIng MIB module. It gets three arguments: an upper-case module name, a lower-case module identifier associated with the module’s object identifier and a statement block that contains mandatory and optional statements and sections of statements in an obligatory order:

```
module <MODULE-NAME> <moduleIdentifier> {

    <optional import statements>
    <oid statement>
    <organization statement>
    <contact statement>
    <description statement>
    <optional reference statement>
    <at least one revision statement>

    <optional extension statements>

    <optional typedef statements>

    <optional node/scalar/table statements>

    <optional notification statements>

    <optional group statements>

    <optional compliance statements>

};
```

The optional ‘import’ statements are followed by the mandatory ‘oid’, ‘organization’, ‘contact’, and ‘description’ statements and the optional ‘reference’ statement, which in turn are followed by the mandatory ‘revision’ statements. This part defines the module’s meta information while the following sections contain its main definitions.

See the ‘moduleStatement’ rule of the SMIng grammar (Appendix A) for the formal syntax of the ‘module’ statement.

5.1 The module’s import Statement

The optional module’s ‘import’ statement is used to import descriptors from external modules into the local module’s namespace. It gets two arguments: the name of the external module and a comma-separated list of one or more identifiers to be imported enclosed in parenthesis.

Multiple 'import' statements for the same module but with disjunct lists of identifiers are allowed, though NOT RECOMMENDED. Anyhow, the same identifier from the same module MUST NOT be imported multiple times. To import identifiers with the same name from different modules might be necessary and is allowed. To distinguish them in the local module, they have to be referred by qualified names. It is NOT RECOMMENDED to import identifiers not used in the local module.

See the 'importStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'import' statement.

5.2 The module's oid Statement

The module's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to the local module.

5.3 The module's organization Statement

The module's 'organization' statement, which must be present, gets one argument which is used to specify a textual description of the organization(s) under whose auspices this module was developed.

5.4 The module's contact Statement

The module's 'contact' statement, which must be present, gets one argument which is used to specify the name, postal address, telephone number, and electronic mail address of the person to whom technical queries concerning this revision of this module should be sent.

5.5 The module's description Statement

The module's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of the contents of this module.

5.6 The module's reference Statement

The module's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related management information, or some other document which provides additional information relevant to this module.

5.7 The module's revision Statement

The module's 'revision' statement is repeatedly used to specify the editorial revisions of the module, including the initial revision. It gets one argument which is a statement block that holds detailed information in an obligatory order. A module MUST have at least one initial 'revision' statement. For every editorial change, a new one MUST be added in front of the revisions sequence, so that all revisions are in reverse chronological order.

See the 'revisionStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'revision' statement.

5.7.1 The revision's date Statement

The revision's 'date' statement, which must be present, gets one argument which is used to specify the date and time of the revision in the format 'YYYY-MM-DD HH:MM' or 'YYYY-MM-DD' which implies the time '00:00'. The time is always given in UTC.

See the 'date' rule of the SMIng grammar (Appendix A) for the formal syntax of the revision's 'date' statement.

5.7.2 The revision's description Statement

The revision's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of the revision.

5.8 Usage Example

Consider how a skeletal MIB module might be constructed: e.g.,

```
module FIZBIN-MIB fizbin {

  import IRTF-NMRG-SMING (experimental);

  oid                  experimental.xx;
                       // contact IANA for actual number

  organization
            "IETF SNMPv2 Working Group,
             IRTF Network Management Research Group (NMRG)";

  contact
            "        Juergen Schoenwaelder

             Postal: TU Braunschweig
                     Bueltenweg 74/75
```

```
                        38106 Braunschweig
                        DE

                 Phone: +49 531 391-3283
                 EMail: schoenw@ibr.cs.tu-bs.de";

     description
               "The MIB module for entities implementing
                the xxxx protocol.";

     reference
               "RFC 2578, Section 5.7.";

     revision {
       date            "1999-04-19";
       description
               "Conversion from SMIv2 to SMIng.";
     };
     revision {
       date            "1995-05-24 18:11";
       description
               "Revision for RFC 1902.";
     };
     revision {
       date            "1992-10-07 04:33";
       description
               "The initial version of this MIB module,
                published in RFC 1442.";
     };

     // ... further definitions ...

   }; // end of module FIZBIN-MIB.
```

6. The extension Statement

The ‘extension’ statement is used to define new statements to be used in the local module or in external modules. It gets two arguments: a lower-case extension statement identifier and a statement block that holds detailed extension information in an obligatory order.

Extension statement identifiers SHOULD NOT contain any upper-case characters or hyphens.

Note that the SMIng extension feature does not allow to formally specify the context, argument syntax and semantics of an extension. Its only purpose is to declare the existence of an extension and to allow a unique reference to an extension. See Section 16 for detailed information on extensions and Section 14.3 for a common example, the ‘agentcaps’ extension.

See the ‘extensionStatement’ rule of the SMIng grammar (Appendix A) for the formal syntax of the ‘extension’ statement.

6.1 The extension’s status Statement

The extension’s ‘status’ statement, which need not be present, gets one argument which is used to specify whether this extension definition is current or historic. The value ‘current’ means that the definition is current and valid. The value ‘obsolete’ means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value ‘deprecated’ also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

If the ‘status’ statement is omitted, the status value ‘current’ is implied.

6.2 The extension’s description Statement

The extension’s ‘description’ statement, which must be present, gets one argument which is used to specify a high-level textual description of the extension statement.

It is RECOMMENDED to include information on the extension’s context, its syntax (preferably in ABNF notation), and its semantics.

6.3 The extension’s reference Statement

The extension’s ‘reference’ statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related extension definitions, or some other document which provides additional information relevant to this extension.

7. The typedef Statement

The 'typedef' statement is used to define new data types to be used in the local module or in external modules. It gets two arguments: an upper-case type identifier and a statement block that holds detailed type information in an obligatory order.

Type identifiers SHOULD NOT consist of all upper-case characters and SHOULD NOT contain hyphens.

See the 'typedefStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'typedef' statement.

7.1 The typedef's type Statement

The typedef's 'type' statement, which must be present, gets one argument which is used to specify the type from which this type is derived. Optionally, type restrictions may be applied to the new type by appending subtyping information according to the rules of the base type. See Section 3 for SMIng base types and their type restrictions.

7.2 The typedef's default Statement

The typedef's 'default' statement, which need not be present, gets one argument which is used to specify an acceptable default value for objects of this type. A default value may be used at the discretion of an agent when an object instance is created. That is, the value is a "hint" to implementors.

The value of the 'default' statement must, of course, correspond to the (probably restricted) type specified in the typedef's 'type' statement.

The default value of a type may be overwritten by a default value of an object of this type.

Note that for some types, default values make no sense, e.g. IRTF-NMRG-SMING-TYPES::Counter32.

7.3 The typedef's format Statement

The typedef's 'format' statement, which need not be present, gets one argument which is used to give a hint as to how the value of an instance of an object of this type might be displayed. See Section 3.12 for a description of format specifications.

The format specification of a type may be overwritten by a format specification of an object of this type.

7.4 The typedef's units Statement

The typedef's `units' statement, which need not be present, gets one argument which is used to specify a textual definition of the units associated with objects of this type.

The units specification of a type may be overwritten by a units specification of an object of this type.

The units specification has to be appropriate for values displayed according to the typedef's format specification, if present. E.g., if the type defines frequency values of type Unsigned64 measured in thousands of Hertz, the format specification should be `d-3' and the units specification should be `Hertz' or `Hz'. If the format specification would be omitted, the units specification should be `Milli-Hertz' or `mHz'. MIB Authors should pay attention to keep format and units specifications of type and object definitions synced. Application implementors MUST NOT implement units specifications without implementing format specifications.

7.5 The typedef's status Statement

The typedef's `status' statement, which need not be present, gets one argument which is used to specify whether this type definition is current or historic. The value `current' means that the definition is current and valid. The value `obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value `deprecated' also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

Derived types SHOULD NOT be defined as `current' if their underlying type is `deprecated' or `obsolete'. Similarly, they SHOULD NOT be defined as `deprecated' if their underlying type is `obsolete'. Nevertheless, subsequent revisions of the underlying type cannot be avoided, but SHOULD be taken into account in subsequent revisions of the local module.

If the `status' statement is omitted, the status value `current' is implied.

7.6 The typedef's description Statement

The typedef's `description' statement, which must be present, gets one argument which is used to specify a high-level textual description of the newly defined type.

It is RECOMMENDED to include all semantic definitions necessary for implementation, and to embody any information which would otherwise be communicated in any commentary annotations associated with this type definition.

7.7 The typedef's reference Statement

The typedef's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related type definitions, or some other document which provides additional information relevant to this type definition.

7.8 Usage Examples

```
typedef RptrOperStatus {
  type            Enumeration (other(1), ok(2), rptrFailure(3),
                               groupFailure(4), portFailure(5),
                               generalFailure(6));
  default         other;       // undefined by default.
  status          deprecated;
  description
          "A type to indicate the operational state
           of a repeater.";
  reference
          "[IEEE 802.3 Mgt], 30.4.1.1.5, aRepeaterHealthState.";
};

typedef DateAndTime {
  type            OctetString (8 | 11);
  format          "2d-1d-1d,1d:1d:1d.1d,1a1d:1d";
  status          current;     // could be omitted
  description
          "A date-time specification.
           ...
           Note that if only local time is known, then timezone
           information (fields 8-10) is not present.";
  reference
          "RFC 2579, SNMPv2-TC.DateAndTime.";
};

typedef Frequency {
  type            Unsigned64;
  format          "d-3"
  units           "Hertz";
  description
          "A wide-range frequency specification measured
           in thousands of Hertz.";
};
```

8. The node Statement

The ‘node’ statement is used to name and describe a node in the object identifier tree, without associating object information with this node. This may be useful to group definitions in a subtree of related management information, or to uniquely define an identity of arbitrary kind to be referenced in objects of type ObjectIdentifier. The ‘node’ statement gets two arguments: a lower-case node identifier and a statement block that holds detailed node information in an obligatory order.

See the ‘nodeStatement’ rule of the SMIng grammar (Appendix A) for the formal syntax of the ‘node’ statement.

8.1 The node’s oid Statement

The node’s ‘oid’ statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this node.

8.2 The node’s status Statement

The node’s ‘status’ statement, which need not be present, gets one argument which is used to specify whether this node definition is current or historic. The value ‘current’ means that the definition is current and valid. The value ‘obsolete’ means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value ‘deprecated’ also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

If the ‘status’ statement is omitted, the status value ‘current’ is implied.

8.3 The node’s description Statement

The node’s ‘description’ statement, which must be present, gets one argument which is used to specify a high-level textual description of this node.

It is RECOMMENDED to include all semantics and purposes of this node.

8.4 The node’s reference Statement

The node’s ‘reference’ statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related nodes, or some other document which provides additional information relevant to this node.

8.5 Usage Examples

```
node iso                                { oid 1;                };
node   org                              { oid iso.3;            };
node     dod                            { oid org.6;            };
node       internet                     { oid dod.1;            };

node   zeroDotZero {
    oid         0.0;
    description "A value used for null identifiers.";
    reference   "RFC 2578, 2. Definitions.";
};
```

9. The scalar Statement

The 'scalar' statement is used to define a scalar managed object. The 'scalar' statement gets two arguments: a lower-case scalar identifier and a statement block that holds detailed object information in an obligatory order.

See the 'scalarStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'scalar' statement.

9.1 The scalar's oid Statement

The scalar's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this scalar object.

9.2 The scalar's type Statement

The scalar's 'type' statement, which must be present, gets one argument which is used to specify the data type of this scalar object. Optionally, type restrictions may be applied to the type by appending subtyping information according to the rules of the base type. See Section 3 for SMIng base types and their type restrictions.

9.3 The scalar's access Statement

The scalar's 'access' statement, which must be present, gets one argument which is used to specify whether it makes sense to read and/or write an instance of the object, or to include its value in a notification. This is the maximal level of access for the object. This maximal level of access is independent of any administrative authorization policy.

The value 'readwrite' indicates that read and write access makes sense. The value 'readonly' indicates that read access makes sense, but write access is never possible. The value 'notifyonly' indicates an object which is accessible only via a notification.

These values are ordered, from least to greatest access level: 'notifyonly', 'readonly', 'readwrite'.

9.4 The scalar's default Statement

The scalar's 'default' statement, which need not be present, gets one argument which is used to specify an acceptable default value for this scalar object. A default value may be used at the discretion of an agent when an object instance is created. That is, the value is a "hint" to implementors.

The value of the 'default' statement must, of course, correspond to the (probably restricted) type specified in the scalar's 'type' statement.

The scalar's default value overrides the default value of the underlying type definition, if both are present.

Note that for objects of some types, default values make no sense, e.g. IRTF-NMRG-SMING-TYPES::Counter32.

9.5 The scalar's format Statement

The scalar's 'format' statement, which need not be present, gets one argument which is used to give a hint as to how the value of an instance of this object might be displayed. See Section 3.12 for a description of format specifications.

The scalar's format specification overrides the format specification of the underlying type definition if both are present.

9.6 The scalar's units Statement

The scalar's 'units' statement, which need not be present, gets one argument which is used to specify a textual definition of the units associated with this scalar object.

The scalar's units specification overrides the units specification of the underlying type definition if both are present.

The units specification has to be appropriate for values displayed according to the scalar's format specification if present. E.g., if the scalar object represents a frequency value of type Unsigned64 measured in thousands of Hertz, the format specification should be 'd-3' and the units specification should be 'Hertz' or 'Hz'. If the format specification would be omitted, the units specification should be 'Milli-Hertz' or 'mHz'. MIB Authors should pay attention to keep format and units specifications of type and object definitions synced. Application implementors MUST NOT implement units specifications without implementing format specifications.

9.7 The scalar's status Statement

The scalar's 'status' statement, which need not be present, gets one argument which is used to specify whether this scalar object definition is current or historic. The value 'current' means that the definition is current and valid. The value 'obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value 'deprecated' also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

Scalar objects SHOULD NOT be defined as 'current' if their type is 'deprecated' or 'obsolete'. Similarly, they SHOULD NOT be defined as 'deprecated' if their type is 'obsolete'. Nevertheless, subsequent revisions of used type definition cannot be avoided, but SHOULD be taken into account in subsequent revisions of the local module.

If the 'status' statement is omitted the status value 'current' is implied.

9.8 The scalar's description Statement

The scalar's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of this scalar object.

It is RECOMMENDED to include all semantic definitions necessary for the implementation of this scalar object.

9.9 The scalar's reference Statement

The scalar's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related scalar objects, or some other document which provides additional information relevant to this scalar object.

9.10 Usage Examples

```
scalar dialCtlTrapEnable {
  oid             dialCtlConfiguration.2;
  type            Enumeration (enabled(1), disabled(2));
  access          readwrite;
  default         disabled;
  description
          "This object indicates whether dialCtlPeerCallInformation
           and dialCtlPeerCallSetup traps should be generated for
           all peers. If the value of this object is enabled(1),
           traps will be generated for all peers. If the value
           of this object is disabled(2), traps will be generated
           only for peers having dialCtlPeerCfgTrapEnable set
           to enabled(1).";
};
```

10. The table Statement

The 'table' statement is used to define the root node of a table. The 'table' statement gets two arguments: a lower-case table identifier and a statement block that holds detailed table information in an obligatory order.

See the 'tableStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'table' statement.

10.1 The table's oid Statement

The table's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this table node.

10.2 The table's status Statement

The table's 'status' statement, which need not be present, gets one argument which is used to specify whether this table definition is current or historic. The value 'current' means that the definition is current and valid. The value 'obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value 'deprecated' also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

If the 'status' statement is omitted the status value 'current' is implied.

10.3 The table's description Statement

The table's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of this table.

It is RECOMMENDED to include all semantic definitions necessary for the implementation of this table.

10.4 The table's reference Statement

The table's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related tables, or some other document which provides additional information relevant to this table.

10.5 The table's row Statement

The table's 'row' statement, which must be present, gets two arguments: a lower-case row identifier and a statement block that holds detailed row information in an obligatory order.

See the 'rowStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'row' statement.

10.5.1 The row's oid Statement

The row's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this table row.

10.5.2 Table Indexing Statements

SMIng offers five methods to supply table indexing information: ordinary tables, table augmentations, sparse table augmentations, table expansions, and reordered tables use different statements to denote their indexing information. Each table row definition must contain exactly one of the following indexing statements.

10.5.2.1 The row's index Statement for Table Indexing

The row's 'index' statement, which is used to supply table indexing information of base table rows, gets at least one argument that specifies a comma-separated list of object identifiers, that are used for table indexing, enclosed in parenthesis.

Under some circumstances, an optional 'implied' keyword may be added in front of the list to indicate a compact encoding of the last object in the list. See Section 2.4 for details.

10.5.2.2 The row's augments Statement for Table Indexing

The row's 'augments' statement, which is used to supply table indexing information of table rows that augment a base table row, gets one argument that specifies the identifier of the table row to be augmented. Note that a row augmentation cannot itself be augmented. Anyhow, a base row may be augmented by multiple row augmentations.

A row augmentation makes instances of subordinate columnar objects identified according to the index specification of the base row corresponding to the row named in the 'augments' statement. Further, instances of subordinate columnar objects of a row augmentation exist according to the same semantics as instances of subordinate columnar objects of the base row being augmented. As such, note that creation of a base row implies the correspondent creation of any row augmentations. Row augmentations MUST NOT be used in row creation and deletion operations.

10.5.2.3 The row's sparse Statement for Table Indexing

The row's 'sparse' statement, which is used to supply table indexing information of table rows that sparsely augment a base table row, gets one argument that specifies the identifier of the table row to be sparsely augmented. Note that a sparse row augmentation cannot itself be augmented. Anyhow, a base row may be augmented by multiple row augmentations, sparsely or not.

A sparse row augmentation makes instances of subordinate columnar objects identified, if present, according to the index specification of the base row corresponding to the row named in the 'sparse' statement. Further, instances of subordinate columnar objects of a sparse row augmentation exist according to the semantics as instances of subordinate columnar objects of the base row and the (non-formal) rules that confine the sparse relationship. As such, note that creation of a base row implies the correspondent creation of any sparse row augmentations if the sparse relationship is matched. Sparse row augmentations MUST NOT be used in row creation and deletion operations.

10.5.2.4 The row's reorders Statement for Table Indexing

The row's 'reorders' statement is used to supply table indexing information of table rows, that contain exactly the same index objects of a base table row, except in another order. It gets at least two arguments. The first one specifies the identifier of the base table row. The second one specifies a comma-separated list of exactly those object identifiers of the base row's 'index' statement, but in the order to be used in this table. Note that a reordered table cannot itself be reordered. Anyhow, a base row may be used for multiple reordered tables.

Under some circumstances, an optional 'implied' keyword may be added in front of the list to indicate a compact encoding of the last object in the list. See Section 2.4 for details.

Instances of subordinate columnar objects of a reordered table exist according to the same semantics as instances of subordinate columnar objects of the base row. As such, note that creation of a base row implies the correspondent creation of any related reordered table. Rows or reordered tables MUST NOT be used in row creation and deletion operations.

10.5.2.5 The row's expands Statement for Table Indexing

The row's 'expands' statement is used to supply table indexing information of table row expansions. Table row expansions use exactly the same index objects of another row together with additional indexing objects. Thus, the 'expands' statement gets at least two arguments. The first one specifies the identifier of the related table row. The second one specifies a comma-separated list of the additional object identifiers used for indexing. Note that an expanded table may itself be expanded, and related rows may be used for multiple table expansions.

Under some circumstances, an optional 'implied' keyword may be added in front of the list to indicate a compact encoding of the last object in the list. See Section 2.4 for details.

10.5.3 The row's create Statement

The row's 'create' statement, which need not be present, gets one argument which specifies whether the table allows row creation and deletion and which columns are required to be supplied in row creation operations.

Only if the 'create' statement is present, row creation and deletion is possible. A comma-separated list of column identifiers enclosed in parenthesis may be supplied as an optional argument. This denotes, that the named columns must be supplied in row creation operations. If this list is not present, there are no explicitly required columns to be supplied in row creation operations.

10.5.4 The row's status Statement

The row's 'status' statement, which need not be present, gets one argument which is used to specify whether this row definition is current or historic. The value 'current' means that the definition is current and valid. The value 'obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value 'deprecated' also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

If the 'status' statement is omitted the status value 'current' is implied.

10.5.5 The row's description Statement

The row's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of this table row.

It is RECOMMENDED to include all semantic definitions necessary for the implementation of this table row. Especially, if the row's indexing statement contains objects that are not columnar objects of the row, it SHOULD be described how these objects are used in uniquely identifying instances of the row's columnar objects.

10.5.6 The row's reference Statement

The row's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related table rows, or some other document which provides additional information relevant to this table row.

10.6 The table row's column Statement

The row's 'column' statement is used to define a columnar managed object. The 'column' statement gets two arguments: a lower-case column identifier and a statement block that holds detailed object information in an obligatory order.

See the 'columnStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'column' statement.

10.6.1 The column's oid Statement

The column's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this columnar object.

10.6.2 The column's type Statement

The column's 'type' statement, which must be present, gets at least one argument which is used to specify the data type of this columnar object. Optionally, type restrictions may be applied to the type by appending subtyping information according to the rules of the base type. See Section 3 for SMIng base types and their type restrictions.

10.6.3 The column's access Statement

The column's 'access' statement, which must be present, gets one argument which is used to specify whether it makes sense to read and/or write an instance of the object, or to include its value in a notification. This is the maximal level of access for the object. This maximal level of access is independent of any administrative authorization policy.

The value 'readwrite' indicates that read and write access makes sense. The value 'readonly' indicates that read access makes sense, but write access is never possible. The value 'noaccess' indicates an auxiliary object (see Section 2.4). The value 'notifyonly' indicates an object which is accessible only via a notification.

These values are ordered, from least to greatest access level: 'noaccess', 'notifyonly', 'readonly', 'readwrite'.

10.6.4 The column's default Statement

The column's 'default' statement, which need not be present, gets one argument which is used to specify an acceptable default value for this columnar object. A default value may be used at the discretion of an agent when an object instance is created. That is, the value is a "hint" to implementors.

The value of the 'default' statement must, of course, correspond to the (probably restricted) type specified in the column's 'type' statement.

The column's default value overrides the default value of the underlying type definition if both are present.

Note that for objects of some types, default values make no sense, e.g. IRTF-NMRG-SMING-TYPES::Counter32.

10.6.5 The column's format Statement

The column's 'format' statement, which need not be present, gets one argument which is used to give a hint as to how the value of an instance of this object might be displayed. See Section 3.12 for a description of format specifications.

The column's format specification overrides the format specification of the underlying type definition if both are present.

10.6.6 The column's units Statement

The column's 'units' statement, which need not be present, gets one argument which is used to specify a textual definition of the units associated with this columnar object.

The column's units specification overrides the units specification of the underlying type definition if both are present.

The units specification has to be appropriate for values displayed according to the column's format specification if present. E.g., if the columnar object represents a frequency value of type Unsigned64 measured in thousands of Hertz, the format specification should be 'd-3' and the units specification should be 'Hertz' or 'Hz'. If the format specification would be omitted the units specification should be 'Milli-Hertz' or 'mHz'. MIB Authors should pay attention to keep format and units specifications of type and object definitions synced. Application implementors MUST NOT implement units specifications without implementing format specifications.

10.6.7 The column's status Statement

The column's 'status' statement, which need not be present, gets one argument which is used to specify whether this columnar object definition is current or historic. The value 'current' means that the definition is current and valid. The value 'obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value 'deprecated' also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

Columnar objects SHOULD NOT be defined as 'current' if their type is 'deprecated' or 'obsolete'. Similarly, they SHOULD NOT be defined as 'deprecated' if their type is 'obsolete'. Nevertheless, subsequent revisions of used type definition cannot be avoided, but SHOULD be taken into account in subsequent revisions of the local module.

If the 'status' statement is omitted the status value 'current' is implied.

10.6.8 The column's description Statement

The column's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of this columnar object.

It is RECOMMENDED to include all semantic definitions necessary for the implementation of this columnar object.

10.6.9 The column's reference Statement

The column's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related columnar objects, or some other document which provides additional information relevant to this columnar object.

10.7 Usage Example

Consider how one might define a table and its subordinates. (This example uses the RowStatus type defined in Section 14.2.)

```
     scalar evalSlot {
       oid             eval.1;
       type            Integer32 (0..2147483647);
       access          readonly;
       description
               "The index number of the first unassigned entry in the
                evaluation table, or the value of zero indicating that
                all entries are assigned.

                A management station should create new entries in the
                evaluation table using this algorithm:  first, issue a
                management protocol retrieval operation to determine the
                value of evalSlot; and, second, issue a management
                protocol set operation to create an instance of the
                evalStatus object setting its value to createAndGo(4) or
                createAndWait(5).  If this latter operation succeeds,
                then the management station may continue modifying the
                instances corresponding to the newly created conceptual
                row, without fear of collision with other management
                stations.";
     };

     table evalTable {
       oid             eval.2;
       description
               "The (conceptual) evaluation table.";
       reference
               "RFC 2578, 7.11.";

       row evalEntry {
         oid           evalTable.1;
         index         (evalIndex);
         create        (evalStatus);
         description
               "An entry (conceptual row) in the evaluation table.";

         column evalIndex {
           oid         evalEntry.1;
           type        Integer32 (1..2147483647);
           access      noaccess;
           description
               "The auxiliary variable used for identifying instances of
                the columnar objects in the evaluation table.";
         };

         column evalString {
           oid         evalEntry.2;
```

```
           type        DisplayString;
           access      readwrite;
           description
               "The string to evaluate.";
         };

         column evalValue {
           oid         evalEntry.3;
           type        Integer32;
           access      readonly;
           default     0;
           description
               "The value when evalString was last evaluated, or zero if
                no such value is available.";
         };

         column evalStatus {
           oid         evalEntry.4;
           type        RowStatus;
           access      readwrite;
           default     active;
           description
               "The status column used for creating, modifying, and
                deleting instances of the columnar objects in the
                evaluation table.";
         };
       };
     };
```

11. The notification Statement

The 'notification' statement is used to define a notification. It gets two arguments: a lower-case notification identifier and a statement block that holds detailed information in an obligatory order.

See the 'notificationStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'notification' statement.

11.1 The notification's oid Statement

The notification's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this notification.

11.2 The notification's objects Statement

The notification's 'objects' statement, which need not be present, gets one argument which specifies an ordered sequence of objects by their identifiers to be emitted within this notification. The list of objects has to be comma-separated and enclosed in parenthesis.

One and only one object instance for each occurrence of each object must be present, and in the specified order, in every instance of the notification. If the same object occurs multiple times in a notification's 'objects' statement, then an object instance is present for each of them. An object specified in this sequence must not have an 'access' statement argument of 'noaccess'.

Note that an agent is allowed, at its own discretion, to append as many additional objects as it considers useful to the end of the notification (i.e., after the objects defined by the notification's 'objects' statement if present).

11.3 The notification's status Statement

The notification's 'status' statement, which need not be present, gets one argument which is used to specify whether this notification definition is current or historic. The value 'current' means that the definition is current and valid. The value 'obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value 'deprecated' also indicates an obsolete definition, it permits new/continued implementation in order to foster interoperability with older/existing implementations.

If the 'status' statement is omitted the status value 'current' is implied.

11.4 The notification's description Statement

The notification's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of this notification.

It is RECOMMENDED to include all semantic definitions necessary for the implementation of this notification. In particular, it SHOULD be documented which instances of the objects mentioned in the 'objects' statement should be contained within notifications of this type.

11.5 The notification's reference Statement

The notification's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related notifications, or some other document which provides additional information relevant to this notification.

11.6 Usage Example

Consider how a configuration change notification might be described:

```
node entityMIBTraps      { oid             entityMIB.2;      };

node entityMIBTrapPrefix { oid             entityMIBTraps.0; };

notification entConfigChange {
  oid             entityMIBTrapPrefix.1;
  description
          "An entConfigChange trap is sent when the value of
           entLastChangeTime changes. It can be utilized by an NMS
           to trigger logical/physical entity table maintenance
           polls.  An agent must not generate more than one
           entConfigChange 'trap-event' in a five second period,
           where a 'trap-event' is the transmission of a single
           trap PDU to a list of trap destinations.  If additional
           configuration changes occur within the five second
           'throttling' period then these trap-events should be
           suppressed by the agent. An NMS should periodically
           check the value of entLastChangeTime to detect any
           missed entConfigChange trap-events, e.g. due to
           throttling or transmission loss.";
};
```

12. The group Statement

The 'group' statement is used to define a group of arbitrary nodes in the object identifier tree. It gets two arguments: a lower-case group identifier and a statement block that holds detailed group information in an obligatory order.

Note that the primary application of groups are compliance statements, although they might be referred in other formal or informal documents.

See the 'groupStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'group' statement.

12.1 The group's oid Statement

The group's 'oid' statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this group.

12.2 The group's members Statement

The group's 'members' statement, which must be present, gets one argument which specifies the list of nodes by their identifiers to be contained in this group. The list of nodes has to be comma-separated and enclosed in parenthesis.

12.3 The group's status Statement

The group's 'status' statement, which need not be present, gets one argument which is used to specify whether this group definition is current or historic. The value 'current' means that the definition is current and valid. The value 'obsolete' means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value 'deprecated' also indicates an obsolete definition, it permits new/continued use of this group.

If the 'status' statement is omitted the status value 'current' is implied.

12.4 The group's description Statement

The group's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of this group. It is RECOMMENDED to include any relation to other groups.

12.5 The group's reference Statement

The group's 'reference' statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related groups, or some other document which provides additional information relevant to this group.

12.6 Usage Example

The snmpGroup, originally defined in [13], may be described as follows:

```
group snmpGroup {
  oid             snmpMIBGroups.8;
  objects         (snmpInPkts, snmpInBadVersions,
                   snmpInASNParseErrs,
                   snmpSilentDrops, snmpProxyDrops,
                   snmpEnableAuthenTraps);
  description
          "A collection of objects providing basic
           instrumentation and control of an SNMPv2
           entity.";
};
```

13. The compliance Statement

The ‘compliance’ statement is used to define a set of compliance requirements, named a ‘compliance statement’. It gets two arguments: a lower-case compliance identifier and a statement block that holds detailed compliance information in an obligatory order.

See the ‘complianceStatement’ rule of the SMIng grammar (Appendix A) for the formal syntax of the ‘compliance’ statement.

13.1 The compliance’s oid Statement

The compliance’s ‘oid’ statement, which must be present, gets one argument which specifies the object identifier value that is assigned to this compliance statement.

13.2 The compliance’s status Statement

The compliance’s ‘status’ statement, which need not be present, gets one argument which is used to specify whether this compliance statement is current or historic. The value ‘current’ means that the definition is current and valid. The value ‘obsolete’ means the definition is obsolete and should not be implemented and/or can be removed if previously implemented. While the value ‘deprecated’ also indicates an obsolete definition, it permits new/continued use of the compliance specification.

If the ‘status’ statement is omitted the status value ‘current’ is implied.

13.3 The compliance’s description Statement

The compliance’s ‘description’ statement, which must be present, gets one argument which is used to specify a high-level textual description of this compliance statement.

13.4 The compliance’s reference Statement

The compliance’s ‘reference’ statement, which need not be present, gets one argument which is used to specify a textual cross-reference to some other document, either another module which defines related compliance statements, or some other document which provides additional information relevant to this compliance statement.

13.5 The compliance’s mandatory Statement

The compliance’s ‘mandatory’ statement, which need not be present, gets one argument which is used to specify a comma-separated list of one or more groups (Section 12) of objects and/or notifications enclosed in parenthesis. These groups are unconditionally mandatory for implementation.

If an agent claims compliance to a MIB module then it must implement each and every object and notification within each group listed the 'mandatory' statement(s) of the compliance statement(s) of that module.

13.6 The compliance's optional Statement

The compliance's 'optional' statement, which need not be present, is repeatedly used to name each group which is conditionally mandatory for compliance to the module. It can also be used to name unconditionally optional groups. A group named in an 'optional' statement MUST be absent from the correspondent 'mandatory' statement. The 'optional' statement gets two arguments: a lower-case group identifier and a statement block that holds detailed compliance information on that group.

Conditionally mandatory groups include those which are mandatory only if a particular protocol is implemented, or only if another group is implemented. The 'description' statement specifies the conditions under which the group is conditionally mandatory.

A group which is named in neither a 'mandatory' statement nor an 'optional' statement, is unconditionally optional for compliance to the module.

See the 'optionalStatement' rule of the SMIng grammar (Appendix A) for the formal syntax of the 'optional' statement.

13.6.1 The optional's description Statement

The optional's 'description' statement, which must be present, gets one argument which is used to specify a high-level textual description of the conditions under which this group is conditionally mandatory or unconditionally optional.

13.7 The compliance's refine Statement

The compliance's 'refine' statement, which need not be present, is repeatedly used to specify each object for which compliance has a refined requirement with respect to the module definition. The object must be present in one of the conformance groups named in the correspondent 'mandatory' or 'optional' statements. The 'refine' statement gets two arguments: a lower-case identifier of a scalar or columnar object and a statement block that holds detailed refinement information on that object.

See the ‘refineStatement’ rule of the SMIng grammar (Appendix A) for the formal syntax of the ‘refine’ statement.

13.7.1 The refine’s type Statement

The refine’s ‘type’ statement, which need not be present, gets one argument that is used to provide a refined type for the correspondent object. Type restrictions may be applied by appending subtyping information according to the rules of the base type. See Section 3 for SMIng base types and their type restrictions.

Note that if a ‘type’ and a ‘writetype’ statement are both present then this type only applies when instances of the correspondent object are read.

13.7.2 The refine’s writetype Statement

The refine’s ‘writetype’ statement, which need not be present, gets one argument that is used to provide a refined type for the correspondent object, only when instances of that object are written. Type restrictions may be applied by appending subtyping information according to the rules of the base type. See Section 3 for SMIng base types and their type restrictions.

13.7.3 The refine’s access Statement

The refine’s ‘access’ statement, which need not be present, gets one argument that is used to specify the minimal level of access that the correspondent object must implement in the sense of its original ‘access’ statement. Hence, the refine’s ‘access’ statement MUST NOT specify a greater level of access than is specified in the correspondent object definition.

An implementation is compliant if the level of access it provides is greater or equal to the minimal level in the refine’s ‘access’ statement and less or equal to the maximal level in the object’s ‘access’ statement.

13.7.4 The refine’s description Statement

The refine’s ‘description’ statement, which must be present, gets one argument which is used to specify a high-level textual description of the refined compliance requirement.

13.8 Usage Example

The compliance statement contained in the SNMPv2-MIB, converted to SMIng:

```
     compliance snmpBasicCompliance {
       oid             snmpMIBCompliances.2;
       description
               "The compliance statement for SNMPv2 entities which
                implement the SNMPv2 MIB.";

       mandatory       (snmpGroup, snmpSetGroup, systemGroup,
                        snmpBasicNotificationsGroup);

       optional snmpCommunityGroup {
         description
               "This group is mandatory for SNMPv2 entities which
                support community-based authentication.";
       };
     };
```

14. SMIng Core Modules

MIB modules, either SMIv1, SMIv2 or SMIng, are built on top of some core definitions. These core definitions are imported from some "well-defined" core modules. In case of SMIng, there are two core modules:

1. IRTF-NMRG-SMING, which defines a special node, named 'zeroDotZero' that may be used by objects of type 'ObjectIdentifier' as a null value, and a core set of 18 nodes near the root of the object identifier tree. Only some of those core nodes are significant for the Internet management and have been described in Section 2.2.

2. IRTF-NMRG-SMING-TYPES, which defines SMIng data types. Regarding to their origin, these types can be divided into two groups: those being ASN.1 application types defined as SMIv2 core types [2] and those being textual conventions in SMIv2 [3]. For use in SMIng, there is no need to care about these differences, though it might be useful in SNMP applications, to know that some types are distinguishable "on the wire" and others are not.

3. IRTF-NMRG-SMING-EXTENSIONS, which defines proposed SMIng extensions. At this moment, there is only one extension, the 'agentcaps' statement. See Section 16 for details on extensions.

14.1 IRTF-NMRG-SMING

```
module IRTF-NMRG-SMING irtfNmrgSming {

//
// $RCSfile: IRTF-NMRG-SMING,v $
// $Revision: 1.9 $
// $Author: strauss $
// $Date: 1999/05/07 15:57:26 $
//

    oid             1.3.6.1.3.1575.1.8;
                    // experimental.1575.1.8

    organization    "IRTF Network Management Research Group (NMRG),
                     Network Management Group, TU Braunschweig";

    contact         "        Frank Strauss

                     Postal: TU Braunschweig
                             Bueltenweg 74/75
                             38106 Braunschweig
                             Germany
```

```
                         Phone: +49 531 391-3283
                         EMail: strauss@ibr.cs.tu-bs.de
                          http://www.ibr.cs.tu-bs.de/~strauss/sming/";

       description     "Core node definitions for SMIng.";

       revision {
           date        "1999-05-07";
           description "Initial Revision.";
       };


       node ccitt                          { oid 0;                };

       node   zeroDotZero {
           oid         0.0;
           description "A value used for null identifiers.";
       };

       node iso                            { oid 1;                };
       node   org                          { oid iso.3;            };
       node     dod                        { oid org.6;            };
       node       internet                 { oid dod.1;            };
       node         directory              { oid internet.1;       };
       node         mgmt                   { oid internet.2;       };
       node           mib-2                { oid mgmt.1;           };
       node             transmission       { oid mib-2.10;         };
       node         experimental           { oid internet.3;       };
       node         private                { oid internet.4;       };
       node           enterprises          { oid private.1;        };
       node         security               { oid internet.5;       };
       node         snmpV2                 { oid internet.6;       };
       node           snmpDomains          { oid snmpV2.1;         };
       node           snmpProxys           { oid snmpV2.2;         };
       node           snmpModules          { oid snmpV2.3;         };

       node joint-iso-ccitt                { oid 2;                };

   };
```

14.2 IRTF-NMRG-SMING-TYPES

```
   module IRTF-NMRG-SMING-TYPES irtfNmrgSmingTypes {

   //
   // $RCSfile: IRTF-NMRG-SMING-TYPES,v $
   // $Revision: 1.14 $
   // $Author: strauss $
   // $Date: 1999/05/07 15:57:28 $
```

```
   //

       oid             1.3.6.1.3.1575.1.9;
                       // experimental.1575.1.9

       organization    "IRTF Network Management Research Group (NMRG),
                        Network Management Group, TU Braunschweig";

       contact         "        Frank Strauss

                        Postal: TU Braunschweig
                                Bueltenweg 74/75
                                38106 Braunschweig
                                Germany

                         Phone: +49 531 391-3283
                         EMail: strauss@ibr.cs.tu-bs.de
                          http://www.ibr.cs.tu-bs.de/~strauss/sming/";

       description     "Core type definitions for SMIng.";

       revision {
           date        "1999-05-07";
           description "Initial Revision.";
       };


       typedef Gauge32 {
           type        Unsigned32;
           description
              "The Gauge32 type represents a non-negative integer,
               which may increase or decrease, but shall never
               exceed a maximum value, nor fall below a minimum
               value.  The maximum value can not be greater than
               2^32-1 (4294967295 decimal), and the minimum value
               can not be smaller than 0.  The value of a Gauge32
               has its maximum value whenever the information
               being modeled is greater than or equal to its
               maximum value, and has its minimum value whenever
               the information being modeled is smaller than or
               equal to its minimum value.  If the information
               being modeled subsequently decreases below
               (increases above) the maximum (minimum) value, the
               Gauge32 also decreases (increases).  (Note that
               despite of the use of the term `latched' in the
               original definition of this type, it does not
               become `stuck' at its maximum or minimum value.)";
           reference
              "RFC 2578, Sections 2. and 7.1.7.";
```

```
       };

       typedef Counter32 {
           type        Unsigned32;
           description
              "The Counter32 type represents a non-negative integer
               which monotonically increases until it reaches a
               maximum value of 2^32-1 (4294967295 decimal), when it
               wraps around and starts increasing again from zero.

               Counters have no defined `initial' value, and thus, a
               single value of a Counter has (in general) no
               information content.  Discontinuities in the
               monotonically increasing value normally occur at
               re-initialization of the management system, and at
               other times as specified in the description of an
               object using this type.  If such other times can
               occur, for example, the creation of an object
               instance at times other than re-initialization, then
               a corresponding object should be defined, with an
               appropriate type, to indicate the last discontinuity.
               Examples of appropriate types include: TimeStamp,
               DateAndTime or TimeTicks (other types defined in this
               module).

               The value of the access statement for objects with a
               type value of Counter32 should be either `readonly'
               or `notifyonly'.

               A default statement should not be used for objects
               with a type value of Counter32.";
           reference
              "RFC 2578, Sections 2. and 7.1.6.";
       };

       typedef Gauge64 {
           type        Unsigned64;
           description
              "The Gauge64 type represents a non-negative integer,
               which may increase or decrease, but shall never
               exceed a maximum value, nor fall below a minimum
               value.  The maximum value can not be greater than
               2^64-1 (18446744073709551615), and the minimum value
               can not be smaller than 0.  The value of a Gauge64
               has its maximum value whenever the information
               being modeled is greater than or equal to its
               maximum value, and has its minimum value whenever
               the information being modeled is smaller than or
               equal to its minimum value.  If the information
```

```
             being modeled subsequently decreases below
             (increases above) the maximum (minimum) value, the
             Gauge64 also decreases (increases).  (Note that
             despite of the use of the term `latched' in the
             original definition of this type, it does not
             become `stuck' at its maximum or minimum value.)";
     };

     typedef Counter64 {
         type        Unsigned64;
         description
            "The Counter64 type represents a non-negative integer
             which monotonically increases until it reaches a
             maximum value of 2^64-1 (18446744073709551615), when
             it wraps around and starts increasing again from zero.

             Counters have no defined `initial' value, and thus, a
             single value of a Counter has (in general) no
             information content.  Discontinuities in the
             monotonically increasing value normally occur at
             re-initialization of the management system, and at
             other times as specified in the description of an
             object using this type.  If such other times can
             occur, for example, the creation of an object
             instance at times other than re-initialization, then
             a corresponding object should be defined, with an
             appropriate type, to indicate the last discontinuity.
             Examples of appropriate types include: TimeStamp,
             DateAndTime or TimeTicks (other types defined in this
             module).

             The value of the access statement for objects with a
             type value of Counter64 should be either `readonly'
             or `notifyonly'.

             A default statement should not be used for objects
             with a type value of Counter64.";
         reference
            "RFC 2578, Sections 2. and 7.1.10.";
     };

     typedef Opaque {
         type        OctetString;
         description
            "The Opaque type is provided solely for
             backward-compatibility, and shall not be used for
             newly-defined object types.

             The Opaque type supports the capability to pass
```

```
                arbitrary ASN.1 syntax.  A value is encoded using
                the ASN.1 Basic Encoding Rules into a string of
                octets.  This, in turn, is encoded as an
                OctetString, in effect `double-wrapping' the
                original ASN.1 value.

                Note that a conforming implementation need only be
                able to accept and recognize opaquely-encoded data.
                It need not be able to unwrap the data and then
                interpret its contents.

                A requirement on `standard' MIB modules is that no
                object may have a type value of Opaque.";
            reference
               "RFC 2578, Sections 2. and 7.1.9.";
        };

        typedef IpAddress {
            type        OctetString (4);
            status      deprecated;
            description
               "******* THIS TYPE DEFINITION IS DEPRECATED *******

                The IpAddress type represents a 32-bit internet
                IPv4 address.  It is represented as an OctetString
                of length 4, in network byte-order.

                Note that the IpAddress type is present for
                historical reasons. IPv4 and IPv6 addresses should
                be represented using the IpAddr type. Generic
                Network addresses should be represented using a
                pair of TDomain and TAddress types (all defined in
                this module).";
            reference
               "RFC 2578, Sections 2. and 7.1.5.";
        };

        typedef TimeTicks {
            type        Unsigned32;
            description
               "The TimeTicks type represents a non-negative
                integer which represents the time, modulo 2^32
                (4294967296 decimal), in hundredths of a second
                between two epochs.  When objects are defined which
                use this type, the description of the object
                identifies both of the reference epochs.

                For example, the TimeStamp type (defined in this
                module) is based on the TimeTicks type.
```

```
            With a TimeStamp, the first reference epoch is
            defined as the time when SNMPv2-MIB::sysUpTime was
            zero, and the second reference epoch is defined as
            the current value of sysUpTime.

            The TimeTicks type should not be sub-typed.";
        reference
           "RFC 2578, Sections 2. and 7.1.8.";
    };

    //
    //  The following type definitions are plain
    //  conversions of the textual conventions from
    //  the SNMPv2-TC module (RFC 2579).
    //

    typedef DisplayString {
        type        OctetString (0..255);
        format      "255a";
        description
        "Represents textual information taken from the NVT ASCII
         character set, as defined in pages 4, 10-11 of RFC 854.

         To summarize RFC 854, the NVT ASCII repertoire specifies:

          - the use of character codes 0-127 (decimal)

          - the graphics characters (32-126) are interpreted as
            US ASCII

          - NUL, LF, CR, BEL, BS, HT, VT and FF have the special
            meanings specified in RFC 854

          - the other 25 codes have no standard interpretation

          - the sequence 'CR LF' means newline

          - the sequence 'CR NUL' means carriage-return

          - an 'LF' not preceded by a 'CR' means moving to the
            same column on the next line.

          - the sequence 'CR x' for any x other than LF or NUL is
            illegal.  (Note that this also means that a string may
            end with either 'CR LF' or 'CR NUL', but not with CR.)

         Any object defined using this syntax may not exceed 255
         characters in length.";
    };
```

```
       typedef PhysAddress {
           type        OctetString;
           format      "1x:";
           description
           "Represents media- or physical-level addresses.";
       };

       typedef MacAddress {
           type        OctetString (6);
           format      "1x:";
           description
           "Represents an 802 MAC address represented in the
            `canonical' order defined by IEEE 802.1a, i.e., as if it
            were transmitted least significant bit first, even though
            802.5 (in contrast to other 802.x protocols) requires MAC
            addresses to be transmitted most significant bit first.";
       };

       typedef TruthValue {
           type        Enumeration (true(1), false(2));
           description
           "Represents a boolean value.";
       };

       typedef TestAndIncr {
           type        Integer32 (0..2147483647);
           description
           "Represents integer-valued information used for atomic
            operations.  When the management protocol is used to specify
            that an object instance having this syntax is to be
            modified, the new value supplied via the management protocol
            must precisely match the value presently held by the
            instance.  If not, the management protocol set operation
            fails with an error of `inconsistentValue'.  Otherwise, if
            the current value is the maximum value of 2^31-1 (2147483647
            decimal), then the value held by the instance is wrapped to
            zero; otherwise, the value held by the instance is
            incremented by one.  (Note that regardless of whether the
            management protocol set operation succeeds, the variable-
            binding in the request and response PDUs are identical.)

            The value of the ACCESS clause for objects having this
            syntax is either `read-write' or `read-create'.  When an
            instance of a columnar object having this syntax is created,
            any value may be supplied via the management protocol.

            When the network management portion of the system is re-
            initialized, the value of every object instance having this
            syntax must either be incremented from its value prior to
```

```
          the re-initialization, or (if the value prior to the re-
          initialization is unknown) be set to a pseudo-randomly
          generated value.";
    };

    typedef AutonomousType {
        type        ObjectIdentifier;
        description
        "Represents an independently extensible type identification
         value.  It may, for example, indicate a particular sub-tree
         with further MIB definitions, or define a particular type of
         protocol or hardware.";
    };

    typedef InstancePointer {
        type        ObjectIdentifier;
        status      obsolete;
        description
        "A pointer to either a specific instance of a MIB object or
         a conceptual row of a MIB table in the managed device.  In
         the latter case, by convention, it is the name of the
         particular instance of the first accessible columnar object
         in the conceptual row.

         The two uses of this textual convention are replaced by
         VariablePointer and RowPointer, respectively.";
    };

    typedef VariablePointer {
        type        ObjectIdentifier;
        description
        "A pointer to a specific object instance.  For example,
         sysContact.0 or ifInOctets.3.";
    };

    typedef RowPointer {
        type        ObjectIdentifier;
        description
        "Represents a pointer to a conceptual row.  The value is the
         name of the instance of the first accessible columnar object
         in the conceptual row.

         For example, ifIndex.3 would point to the 3rd row in the
         ifTable (note that if ifIndex were not-accessible, then
         ifDescr.3 would be used instead).";
    };

    typedef RowStatus {
        type        Enumeration (active(1), notInService(2),
```

```
                           notReady(3), createAndGo(4),
                           createAndWait(5), destroy(6));
           description
           "The RowStatus textual convention is used to manage the
            creation and deletion of conceptual rows, and is used as the
            value of the SYNTAX clause for the status column of a
            conceptual row (as described in Section 7.7.1 of [2].)

            The status column has six defined values:

                - `active', which indicates that the conceptual row is
                available for use by the managed device;

                - `notInService', which indicates that the conceptual
                row exists in the agent, but is unavailable for use by
                the managed device (see NOTE below);

                - `notReady', which indicates that the conceptual row
                exists in the agent, but is missing information
                necessary in order to be available for use by the
                managed device;

                - `createAndGo', which is supplied by a management
                station wishing to create a new instance of a
                conceptual row and to have its status automatically set
                to active, making it available for use by the managed
                device;

                - `createAndWait', which is supplied by a management
                station wishing to create a new instance of a
                conceptual row (but not make it available for use by
                the managed device); and,

                - `destroy', which is supplied by a management station
                wishing to delete all of the instances associated with
                an existing conceptual row.

            Whereas five of the six values (all except `notReady') may
            be specified in a management protocol set operation, only
            three values will be returned in response to a management
            protocol retrieval operation: `notReady', `notInService' or
            `active'.  That is, when queried, an existing conceptual row
            has only three states: it is either available for use by the
            managed device (the status column has value `active'); it is
            not available for use by the managed device, though the

            agent has sufficient information to make it so (the status
            column has value `notInService'); or, it is not available
```

for use by the managed device, and an attempt to make it so would fail because the agent has insufficient information (the state column has value ‘notReady’).

NOTE WELL

This textual convention may be used for a MIB table, irrespective of whether the values of that table’s conceptual rows are able to be modified while it is active, or whether its conceptual rows must be taken out of service in order to be modified. That is, it is the responsibility of the DESCRIPTION clause of the status column to specify whether the status column must not be ‘active’ in order for the value of some other column of the same conceptual row to be modified. If such a specification is made, affected columns may be changed by an SNMP set PDU if the RowStatus would not be equal to ‘active’ either immediately before or after processing the PDU. In other words, if the PDU also contained a varbind that would change the RowStatus value, the column in question may be changed if the RowStatus was not equal to ‘active’ as the PDU was received, or if the varbind sets the status to a value other than ’active’.

Also note that whenever any elements of a row exist, the RowStatus column must also exist.

To summarize the effect of having a conceptual row with a status column having a SYNTAX clause value of RowStatus, consider the following state diagram:

```
                                      STATE
              +--------------+-----------+-------------+-------------
              |      A       |     B     |      C      |      D
              |              |status col.|status column|
              |status column |    is     |      is     |status column
    ACTION    |does not exist| notReady  | notInService|  is active
--------------+--------------+-----------+-------------+-------------
set status    |noError    ->D|inconsist- |inconsistent-|inconsistent-
column to     |       or     |   entValue|        Value|        Value
createAndGo   |inconsistent- |           |             |
              |         Value|           |             |
--------------+--------------+-----------+-------------+-------------
set status    |noError  see 1|inconsist- |inconsistent-|inconsistent-
column to     |       or     |   entValue|        Value|        Value
createAndWait |wrongValue    |           |             |
--------------+--------------+-----------+-------------+-------------
```

```
   set status     |inconsistent- |inconsist- |noError      |noError
   column to      |         Value|  entValue |             |
   active         |              |           |             |
                  |              |    or     |             |
                  |              |           |             |
                  |              |see 2   ->D|see 8     ->D|          ->D
   --------------+--------------+-----------+-------------+-------------
   set status     |inconsistent- |inconsist- |noError      |noError   ->C
   column to      |         Value|  entValue |             |
   notInService   |              |           |             |
                  |              |    or     |             |      or
                  |              |           |             |
                  |              |see 3   ->C|          ->C|see 6
   --------------+--------------+-----------+-------------+-------------
   set status     |noError       |noError    |noError      |noError   ->A
   column to      |              |           |             |      or
   destroy        |           ->A|        ->A|          ->A|see 7
   --------------+--------------+-----------+-------------+-------------
   set any other  |see 4         |noError    |noError      |see 5
   column to some |              |           |             |
   value          |              |      see 1|          ->C|          ->D
   --------------+--------------+-----------+-------------+-------------
```

(1) goto B or C, depending on information available to the

agent.

(2) if other variable bindings included in the same PDU, provide values for all columns which are missing but required, then return noError and goto D.

(3) if other variable bindings included in the same PDU, provide values for all columns which are missing but required, then return noError and goto C.

(4) at the discretion of the agent, the return value may be either:

inconsistentName: because the agent does not choose to create such an instance when the corresponding RowStatus instance does not exist, or

inconsistentValue: if the supplied value is inconsistent with the state of some other MIB object's value, or

noError: because the agent chooses to create the instance.

If noError is returned, then the instance of the status column must also be created, and the new state is B or C, depending on the information available to the agent. If inconsistentName or inconsistentValue is returned, the row remains in state A.

(5) depending on the MIB definition for the column/table, either noError or inconsistentValue may be returned.

(6) the return value can indicate one of the following errors:

wrongValue: because the agent does not support createAndWait, or

inconsistentValue: because the agent is unable to take the row out of service at this time, perhaps because it is in use and cannot be de-activated.

(7) the return value can indicate the following error:

inconsistentValue: because the agent is unable to remove the row at this time, perhaps because it is in use and cannot be de-activated.

NOTE: Other processing of the set request may result in a response other than noError being returned, e.g., wrongValue, noCreation, etc.

Conceptual Row Creation

There are four potential interactions when creating a conceptual row: selecting an instance-identifier which is not in use; creating the conceptual row; initializing any objects for which the agent does not supply a default; and, making the conceptual row available for use by the managed device.

Interaction 1: Selecting an Instance-Identifier

The algorithm used to select an instance-identifier varies for each conceptual row. In some cases, the instance-identifier is semantically significant, e.g., the destination address of a route, and a management station selects the instance-identifier according to the semantics.

In other cases, the instance-identifier is used solely to distinguish conceptual rows, and a management station without specific knowledge of the conceptual row might examine the instances present in order to determine an unused instance-identifier. (This approach may be used, but it is often highly sub-optimal; however, it is also a questionable practice for a naive management station to attempt conceptual row creation.)

Alternately, the MIB module which defines the conceptual row might provide one or more objects which provide assistance in determining an unused instance-identifier. For example, if the conceptual row is indexed by an integer-value, then an object having an integer-valued SYNTAX clause might be defined for such a purpose, allowing a management station to issue a management protocol retrieval operation. In order to avoid unnecessary collisions between competing management stations, ‘adjacent’ retrievals of this object should be different.

Finally, the management station could select a pseudo-random number to use as the index. In the event that this index was already in use and an inconsistentValue was returned in response to the management protocol set operation, the management station should simply select a new pseudo-random number and retry the operation.

A MIB designer should choose between the two latter algorithms based on the size of the table (and therefore the efficiency of each algorithm). For tables in which a large number of entries are expected, it is recommended that a MIB object be defined that returns an acceptable index for creation. For tables with small numbers of entries, it is recommended that the latter pseudo-random index mechanism be used.

Interaction 2: Creating the Conceptual Row

Once an unused instance-identifier has been selected, the management station determines if it wishes to create and activate the conceptual row in one transaction or in a negotiated set of interactions.

Interaction 2a: Creating and Activating the Conceptual Row

The management station must first determine the column requirements, i.e., it must determine those columns for which it must or must not provide values. Depending on the complexity of the table and the management station’s knowledge of the agent’s capabilities, this determination can be made locally by the management station. Alternately, the management station issues a management protocol get operation to examine all columns in the conceptual row that it wishes to create. In response, for each column, there are three possible outcomes:

- a value is returned, indicating that some other management station has already created this conceptual row. We return to interaction 1.

- the exception 'noSuchInstance' is returned, indicating that the agent implements the object-type associated with this column, and that this column in at least one conceptual row would be accessible in the MIB view used by the retrieval were it to exist. For those columns to which the agent provides read-create access, the 'noSuchInstance' exception tells the management station that it should supply a value for this column when the conceptual row is to be created.

- the exception 'noSuchObject' is returned, indicating that the agent does not implement the object-type associated with this column or that there is no conceptual row for which this column would be accessible in the MIB view used by the retrieval. As such, the management station can not issue any management protocol set operations to create an instance of this column.

Once the column requirements have been determined, a management protocol set operation is accordingly issued. This operation also sets the new instance of the status column to 'createAndGo'.

When the agent processes the set operation, it verifies that it has sufficient information to make the conceptual row available for use by the managed device. The information available to the agent is provided by two sources: the management protocol set operation which creates the conceptual row, and, implementation-specific defaults supplied by the agent (note that an agent must provide implementation-specific defaults for at least those objects which it implements as read-only). If there is sufficient information available, then the conceptual row is created, a 'noError' response is returned, the status column is set to 'active', and no further interactions are necessary (i.e., interactions 3 and 4 are skipped). If there is insufficient information, then the conceptual row is not created, and the set operation fails with an error of ‘inconsistentValue’. On this error, the management station can issue a management protocol retrieval operation to determine if this was because it failed to specify a value for a required column, or, because the selected instance of the status column already existed. In the latter case, we return to interaction 1. In the former case, the management station can re-issue the set operation with the additional information, or begin interaction 2 again using ‘createAndWait’ in order to negotiate creation of the conceptual row.

NOTE WELL

Regardless of the method used to determine the column requirements, it is possible that the management station might deem a column necessary when, in fact, the agent will not allow that particular columnar instance to be created or written. In this case, the management protocol set operation will fail with an error such as ‘noCreation’ or ‘notWritable’. In this case, the management station decides whether it needs to be able to set a value for that particular columnar instance. If not, the management station re-issues the management protocol set operation, but without setting a value for that particular columnar instance; otherwise, the management station aborts the row creation algorithm.

Interaction 2b: Negotiating the Creation of the Conceptual Row

The management station issues a management protocol set operation which sets the desired instance of the status column to ‘createAndWait’. If the agent is unwilling to process a request of this sort, the set operation fails with an error of ‘wrongValue’. (As a consequence, such an agent must be prepared to accept a single management protocol set operation, i.e., interaction 2a above, containing all of the columns indicated by its column requirements.) Otherwise, the conceptual row is created, a ‘noError’ response is returned, and the status column is immediately set to either ‘notInService’ or ‘notReady’, depending on whether it has sufficient information to make the conceptual row available for use by the managed device. If there is sufficient information available, then the status column is set to 'notInService'; otherwise, if there is insufficient information, then the status column is set to 'notReady'. Regardless, we proceed to interaction 3.

Interaction 3: Initializing non-defaulted Objects

The management station must now determine the column requirements. It issues a management protocol get operation to examine all columns in the created conceptual row. In the response, for each column, there are three possible outcomes:

- a value is returned, indicating that the agent implements the object-type associated with this column and had sufficient information to provide a value. For those columns to which the agent provides read-create access (and for which the agent allows their values to be changed after their creation), a value return tells the management station that it may issue additional management protocol set operations, if it desires, in order to change the value associated with this column.

- the exception 'noSuchInstance' is returned, indicating that the agent implements the object-type associated with this column, and that this column in at least one conceptual row would be accessible in the MIB view used by the retrieval were it to exist. However, the agent does not have sufficient information to provide a value, and until a value is provided, the conceptual row may not be made available for use by the managed device. For those columns to which the agent provides read-create access, the 'noSuchInstance' exception tells the management station that it must issue additional management protocol set operations, in order to provide a value associated with this column.

- the exception 'noSuchObject' is returned, indicating that the agent does not implement the object-type associated with this column or that there is no conceptual row for which this column would be accessible in the MIB view used by the retrieval. As such, the management station can not issue any management protocol set operations to create an instance of this column.

If the value associated with the status column is 'notReady', then the management station must first deal with all 'noSuchInstance' columns, if any. Having done so, the value of the status column becomes ‘notInService’, and we proceed to interaction 4.

Interaction 4: Making the Conceptual Row Available

Once the management station is satisfied with the values associated with the columns of the conceptual row, it issues a management protocol set operation to set the status column to ‘active’. If the agent has sufficient information to make the conceptual row available for use by the managed device, the management protocol set operation succeeds (a ‘noError’ response is returned). Otherwise, the management protocol set operation fails with an error of ‘inconsistentValue’.

NOTE WELL

A conceptual row having a status column with value ‘notInService’ or ‘notReady’ is unavailable to the managed device. As such, it is possible for the managed device to create its own instances during the time between the management protocol set operation which sets the status column to ‘createAndWait’ and the management protocol set operation which sets the status column to ‘active’. In this case, when the management protocol set operation is issued to set the status column to ‘active’, the values held in the agent supersede those used by the managed device.

If the management station is prevented from setting the status column to ‘active’ (e.g., due to management station or network failure) the conceptual row will be left in the ‘notInService’ or ‘notReady’ state, consuming resources indefinitely. The agent must detect conceptual rows that have been in either state for an abnormally long period of time and remove them. It is the responsibility of the DESCRIPTION clause of the status column to indicate what an abnormally long period of time would be. This period of time should be long enough to allow for human response time (including ‘think time’) between the creation of the conceptual row and the setting of the status to ‘active’. In the absence of such information in the DESCRIPTION clause,
it is suggested that this period be approximately 5 minutes in length. This removal action applies not only to newly-created rows, but also to previously active rows which are set to, and left in, the notInService state for a prolonged period exceeding that which is considered normal for such a

```
            conceptual row.


                            Conceptual Row Suspension

            When a conceptual row is `active', the management station
            may issue a management protocol set operation which sets the
            instance of the status column to `notInService'.  If the
            agent is unwilling to do so, the set operation fails with an
            error of `wrongValue' or `inconsistentValue'.
            Otherwise, the conceptual row is taken out of service, and a
            `noError' response is returned.  It is the responsibility of
            the DESCRIPTION clause of the status column to indicate
            under what circumstances the status column should be taken
            out of service (e.g., in order for the value of some other
            column of the same conceptual row to be modified).

                             Conceptual Row Deletion

            For deletion of conceptual rows, a management protocol set
            operation is issued which sets the instance of the status
            column to `destroy'.  This request may be made regardless of
            the current value of the status column (e.g., it is possible
            to delete conceptual rows which are either `notReady',
            `notInService' or `active'.) If the operation succeeds, then
            all instances associated with the conceptual row are
            immediately removed.";
       };

       typedef TimeStamp {
           type        TimeTicks;
           description
           "The value of the sysUpTime object at which a specific
            occurrence happened.  The specific occurrence must be
            defined in the description of any object defined using this
            type.  When the specific occurrence occurred prior to the
            last time sysUpTime was zero, then the TimeStamp value is
            zero.  Note that this requires all TimeStamp values to be
            reset to zero when the value of sysUpTime reaches 497+ days
            and wraps around to zero.";
       };

       typedef TimeInterval {
           type        Integer32 (0..2147483647);
           description
           "A period of time, measured in units of 0.01 seconds.";
       };

       typedef DateAndTime {
```

```
           type        OctetString (8 | 11);
           format      "2d-1d-1d,1d:1d:1d.1d,1a1d:1d";
           description
           "A date-time specification.

            field  octets  contents                  range
            -----  ------  --------                  -----
             1      1-2   year*                     0..65536
             2       3    month                     1..12
             3       4    day                       1..31
             4       5    hour                      0..23
             5       6    minutes                   0..59
             6       7    seconds                   0..60
                          (use 60 for leap-second)
             7       8    deci-seconds              0..9
             8       9    direction from UTC        '+' / '-'
             9      10    hours from UTC*           0..13
            10      11    minutes from UTC          0..59

            * Notes:
            - the value of year is in big-endian encoding
            - daylight saving time in New Zealand is +13

            For example, Tuesday May 26, 1992 at 1:30:15 PM EDT would be
            displayed as:

                             1992-5-26,13:30:15.0,-4:0

            Note that if only local time is known, then timezone
            information (fields 8-10) is not present.";
       };

       typedef StorageType {
           type        Enumeration (other(1), volatile(2),
                           nonVolatile(3), permanent(4),
                           readOnly(5));
           description
           "Describes the memory realization of a conceptual row.  A
            row which is volatile(2) is lost upon reboot.  A row which
            is either nonVolatile(3), permanent(4) or readOnly(5), is
            backed up by stable storage.  A row which is permanent(4)
            can be changed but not deleted.  A row which is readOnly(5)
            cannot be changed nor deleted.

            If the value of an object with this syntax is either
            permanent(4) or readOnly(5), it cannot be modified.
            Conversely, if the value is either other(1), volatile(2) or
            nonVolatile(3), it cannot be modified to be permanent(4) or
            readOnly(5).  (All illegal modifications result in a
```

```
            'wrongValue' error.)

            Every usage of this textual convention is required to
            specify the columnar objects which a permanent(4) row must
            at a minimum allow to be writable.";
       };

       typedef TDomain {
           type        ObjectIdentifier;
           description
           "Denotes a kind of transport service.

            Some possible values, such as snmpUDPDomain, are defined in
            'Transport Mappings for Version 2 of the Simple Network
            Management Protocol (SNMPv2)'.";
       };

       typedef TAddress {
           type        OctetString (1..255);
           description
           "Denotes a transport service address.

            For snmpUDPDomain, a TAddress is 6 octets long, the initial
            4 octets containing the IP-address in network-byte order and
            the last 2 containing the UDP port in network-byte order.
            Cosult 'Transport Mappings for Version 2 of the Simple
            Network Management Protocol (SNMPv2)' for further
            information on snmpUDPDomain.";
       };

       typedef IpAddr {
           type        OctetString (4 | 16);
           description
           "The IpAddr type represents a 32-bit internet IPv4 address
            if it is 4 octets long, or a 128-bit internet IPv6 address
            if it is 16 octets long, both in network byte-order."
       };

   };
```

14.3 IRTF-NMRG-SMING-EXTENSIONS

```
   module IRTF-NMRG-SMING-EXTENSIONS irtfNmrgSmingExtensions {

   //
   // $RCSfile: IRTF-NMRG-SMING-EXTENSIONS,v $
   // $Revision: 1.7 $
   // $Author: strauss $
   // $Date: 1999/05/07 15:57:27 $
```

```
   //

       oid             1.3.6.1.3.1575.1.10;
                       // experimental.1575.1.10

       organization    "IRTF Network Management Research Group (NMRG),
                        Network Management Group, TU Braunschweig";

       contact         "        Frank Strauss

                        Postal: TU Braunschweig
                                Bueltenweg 74/75
                                38106 Braunschweig
                                Germany

                         Phone: +49 531 391-3283
                         EMail: strauss@ibr.cs.tu-bs.de
                          http://www.ibr.cs.tu-bs.de/~strauss/sming/";

       description     "Core extension definitions for SMIng.";

       revision {
           date        "1999-05-07";
           description "Initial Revision.";
       };

       extension agentcaps {
           status      current;
           description
                 "The agentcaps extension statement is used to describe
                  an agent's deviation from the compliance statements
                  of the modules it implements. It is designed to be
                  compatible with the SMIv2 AGENT-CAPABILITIES macro.

                  The agentcaps extenstion statement should only be used
                  in the statement body of a module that does not
               contain any other type or node definitions that do not
                  correspond to an agent implementation.

                  The following ABNF definitions extend the SMIng
                  grammar to include the agentcaps definition:

                  agentcapsStatement = `agentcaps' sep lcIdentifier
                                     optsep `{' stmtsep
                                       oidStatement stmtsep
                                       releaseStatement stmtsep
                                       *1(statusStatement stmtsep)
                                       descriptionStatement stmtsep
                                       *1(referenceStatement stmtsep)
```

```
                                        *(includesStatement stmtsep)
                                      `}' optsep `;'

                    includesStatement  = `includes' sep qlcIdentifier
                                      optsep `{' stmtsep
                                        *(variationStatement stmtsep)
                                      `}' optsep `;'

                    variationStatement = `variation' sep qlcIdentifier
                                      optsep `{' stmtsep
                                        typeStatement stmtsep
                                        writetypeStatement stmtsep
                                        accessStatement stmtsep
                                        createStatement stmtsep
                                      `}' optsep `;'
                    ";
            reference
                  "RFC 2580, Section 6 describes the SMIv2
                    compatible AGENT-CAPABILITIES macro.";
        };
    };
```

15. Extending a Module

As experience is gained with a module, it may be desirable to revise that module. However, changes are not allowed if they have any potential to cause interoperability problems between an implementation using an original specification and an implementation using an updated specification(s).

For any change, the module's header MUST be updated to include information about the revision: specifically, a new 'revision' statement (Section 5.7) must be included in front of the 'revision' statements. Furthermore, any necessary changes MUST be applied to other statements, including the 'organization' and 'contact' statements (Section 5.3, Section 5.4).

Note that any definition contained in a module is available to be imported by any other module, and is referenced in an 'import' statement via the module name. Thus, a module name MUST NOT be changed. Specifically, the module name (e.g., 'FIZBIN-MIB' in the example of Section 5.8) MUST NOT be changed when revising a module (except to correct typographical errors), and definitions MUST NOT be moved from one module to another.

Also note, that obsolete definitions MUST NOT be removed from modules since their descriptors may still be referenced by other modules, and the object identifiers used to name them MUST never be re-assigned.

A definition may be revised in any of the following ways:

- In 'typedef', 'scalar' and 'column' statement blocks, a 'type' statement containing an 'Enumeration' or 'Bits' type may have new named numbers added.

- In 'typedef', 'scalar' and 'column' statement blocks, the value of a 'type' statement may be replaced by another type if the new type is derived (directly or indirectly) from the same base type, has the same set of values, and has identical semantics.

- In any statement block, a 'status' statement value of 'current' (or a missing 'status' statement) may be revised as 'deprecated' or 'obsolete'. Similarly, a 'status' statement value of 'deprecated' may be revised as 'obsolete'. When making such a change, the 'description' statement SHOULD be updated to explain the rationale.

- In 'typedef', 'scalar' and 'column' statement blocks, a 'default' statement may be added or updated.

o In any statement block, a 'reference' statement may be added or updated.

o In 'typedef', 'scalar' and 'column' statement blocks, a 'units' statement may be added.

o A row may be augmented by adding new columnar objects ('column' statements) at the end of the row's statement block.

o In any statement block, clarifications and additional information may be included in the 'description' statement.

o Entirely new extensions, types, nodes, scalar objects, tables, groups, or compliance statements may be defined, using previously unassigned identifiers and object identifier values.

Otherwise, if the semantics of any previous definition are changed (i.e., if a non-editorial change is made to any definition other than those specifically allowed above), then the 'oid' statement value associated with that object MUST also be changed.

Note that changing the identifier associated with an existing object is considered a semantic change, as these strings may be used in an 'import' statement.

16. SMIng Language Extensibility

While SMIng has a well defined set of statements (Section 5 through Section 13) that are used to specify those aspects of management information commonly regarded as necessary, there may be further information, people wish to express. To describe additional information informally in description statements has the disadvantage, that this information cannot be parsed by any program.

SMIng allows modules to include statements that are unknown to a parser but fulfill some core grammar rules (Section 4.2). Furthermore, additional statements may be defined by the 'extension' statement (Section 6). Extensions can be used in the local module or in other modules, that import the extension. This has some advantages:

- A parser can differentiate between statements known as extensions and unknown statements. This enables the parser to complain about unknown statements, e.g. due to typos.

- Since, there might be some common need for extensions, there is a relatively high probability of extension name collisions originated by different organizations, as long as there is no standardized extension for that purpose. The requirement to explicitly import extension statements allows to distinguish those extensions.

- The supported extensions of an SMIng implementation, e.g. a MIB compiler, can be clearly expressed.

The only formal effect of an extension statement definition is to declare its existence and its status. All additional aspects SHOULD be described in the 'description' statement:

- The detailed semantics of the new statement SHOULD be described.

- The contexts in which the new statement can be used, SHOULD be described, e.g., a new statement may be designed to be used only in the statement block of a module, but not in other nested statement blocks. Others may be applicable in multiple contexts. In addition, the point in the sequence of an obligatory order of other statements, where the new statement may be inserted, might be prescribed.

- The circumstances that make the new statement mandatory or optional SHOULD be described.

- The syntax of the new statement SHOULD be described, either informally or with formal methods (but within the description statement). It is RECOMMENDED to describe the syntax formally, conforming and extending the SMIng grammar (Appendix A).

o It might be reasonable to give some suggestions under which conditions the implementation of the new statement is adequate and how it could be integrated into existent implementations.

Some possible extension applications might be:

o Agent capabilities statements, that allow to describe an agent implemention's deviation from the compliance statements of the modules they implement. SMIv2 has a regular AGENT-CAPABILITIES construct [4], though it's rarely used. That's why it's not a common SMIng statement. Instead, it is defined as an 'agentcaps' extension statement in the module IRTF-NMRG-SMING-EXTENSIONS (Section 14.3).

o Inlined annotations to definitions. E.g., a vendor may wish to describe additional information to object definitions in private modules. An example are severity levels of notifications in the statement block of a 'notification' statement.

o Arbitrary annotations to external definitions. E.g., a vendor may wish to describe additional information to object definitions in a "standard" module. This allows a vendor to implement "standard" modules as well as additional private features, without redundant MIB definitions, but on top of "standard" MIB definitions.

17. Module Conversions

17.1 Converting SMIv2 to SMIng

Conversions of SMIv2 modules to SMIng are feasible. The only SMIv2 construct that cannot be mapped to an SMIng statement is the AGENT-CAPABILITIES macro. However, this can be mapped to the 'agentcaps' extension. All other constructs and types are directly mappable without loss of information.

Detailed mapping information is subject to a future document.

17.2 Converting SMIng to SMIv2

SMIng is more liberal in some aspects to ensure a more consistent information model and a short grammar with less exceptions. This leads to a limited backwards convertability to SMIv2 of different kinds of identifiers, groups, type definition hierarchies, octet string length limitations, and object identifier values (those with a single sub-identifier). Some other information gets lost when being converted to SMIv2: some occurences of references, units, formats, default values, table indexing information, and table row creation information.

Detailed mapping information is subject to a future document.

17.3 SMIv1

Conversions between SMIv1 and SMIng are not covered by this document. Anyhow, conversions between SMIv1 and SMIv2 are subject to SMIv2 and partly described in [14]. Conversions between SMIv2 and SMIng are covered in the sections above.

In few words, conversion from SMIv1 to SMIng is not possible in some cases and without some additional information. Conversion from SMIng to SMIv1 is possible if 64 bit types and floating point data types are not used and if identifiers conform to the SMIv1 restrictions.

18. Security Considerations

This document defines a language with which to write and read descriptions of management information. The language itself has no security impact on the Internet.

19. Acknowledgements

This document is originated as a central part of a master's thesis at the Technical University of Braunschweig, under the guidance of Juergen Schoenwaelder.

Since SMIng represents a successor of SMIv2, a huge amount of paragraphs and phrases are taken from the SMIv2 specifications [2], [3], [4].

Finally, M.T. Rose's work on an XML framework for RFC authors made the writing of an Internet standards document much more feasible.

Thanks to these people and the authors of these documents.

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[2] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., Waldbusser, S., "Structure of Management Information Version 2 (SMIv2)", RFC 2578, STD 58, April 1999.

[3] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., Waldbusser, S., "Textual Conventions for SMIv2", RFC 2579, STD 59, April 1999.

[4] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M., Waldbusser, S., "Conformance Statements for SMIv2", RFC 2580, STD 60, April 1999.

[5] Rose, M., McCloghrie, K., "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, STD 16, May 1990.

[6] Rose, M., McCloghrie, K., "Concise MIB Definitions", RFC 1212, STD 16, March 1991.

[7] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[8] Crocker, D., Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[9] International Organization for Standardization, "Specification of Abstract Syntax Notation One (ASN.1)", International Standard 8824, December 1987.

[10] Harrington, D., Presuhn, R., Wijnen, B., "An Architecture for Describing SNMP Management Frameworks", RFC 2271, January 1999.

[11] Institute of Electrical and Electronics Engineers, "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, August 1985.

[12] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[13] Case, J., McCloghrie, K., Rose, M., Waldbusser, S., "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907, January 1996.

[14] Wijnen, B., Levi, D., "V2ToV1 - Mapping SNMPv2 onto SNMPv1

within a bi-lingual SNMP agent", RFC 2089, January 1997.

Author's Address

Frank Strauss
TU Braunschweig
Bueltenweg 74/75
38106 Braunschweig
Germany

Phone: +49 531 391-3283
EMail: strauss@ibr.cs.tu-bs.de
URI: http://www.ibr.cs.tu-bs.de/

Appendix A. The SMIng ABNF grammar

The SMIng grammar conforms to the Augmented Backus-Naur Form (ABNF)[8], with one exception: For readability, keywords are represented as quoted strings, although ABNF would declare these strings to be case-insensitive. Anyhow, SMIng keyword are meant to be case-sensitive.

```
;;
;; sming.abnf -- SMIng grammar in ABNF notation (RFC 2234).
;;
;; @(#) $Id: sming.abnf,v 1.22 1999/04/20 18:06:07 strauss Exp $
;;
;; Copyright (C) The Internet Society (1999). All Rights Reserved.
;;

smingFile               = optsep *(moduleStatement optsep)

;;
;; Statement rules.
;;

moduleStatement         = moduleKeyword sep ucIdentifier sep
                              lcIdentifier optsep "{" stmtsep
                              *(importStatement stmtsep)
                              oidStatement stmtsep
                              organizationStatement stmtsep
                              contactStatement stmtsep
                              descriptionStatement stmtsep
                              *1(referenceStatement stmtsep)
                              1*(revisionStatement stmtsep)
                              *(extensionStatement stmtsep)
                              *(typedefStatement stmtsep)
                              *(anyObjectStatement stmtsep)
                              *(notificationStatement stmtsep)
                              *(groupStatement stmtsep)
                              *(complianceStatement stmtsep)
                          "}" optsep ";"

extensionStatement      = extensionKeyword sep lcIdentifier optsep
                              "{" stmtsep
                              *1(statusStatement stmtsep)
                              descriptionStatement stmtsep
                              *1(referenceStatement stmtsep)
                          "}" optsep ";"

typedefStatement        = typedefKeyword sep ucIdentifier optsep
                              "{" stmtsep
                              typedefTypeStatement stmtsep
```

```
                                *1(defaultStatement stmtsep)
                                *1(formatStatement stmtsep)
                                *1(unitsStatement stmtsep)
                                *1(statusStatement stmtsep)
                                descriptionStatement stmtsep
                                *1(referenceStatement stmtsep)
                            "}" optsep ";"

   anyObjectStatement      = nodeStatement /
                             scalarStatement /
                             tableStatement

   nodeStatement           = nodeKeyword sep lcIdentifier optsep
                                "{" stmtsep
                                oidStatement stmtsep
                                *1(statusStatement stmtsep)
                                *1(descriptionStatement stmtsep)
                                *1(referenceStatement stmtsep)
                            "}" optsep ";"

   scalarStatement         = scalarKeyword sep lcIdentifier optsep
                                "{" stmtsep
                                oidStatement stmtsep
                                typeStatement stmtsep
                                accessStatement stmtsep
                                *1(defaultStatement stmtsep)
                                *1(formatStatement stmtsep)
                                *1(unitsStatement stmtsep)
                                *1(statusStatement stmtsep)
                                descriptionStatement stmtsep
                                *1(referenceStatement stmtsep)
                            "}" optsep ";"

   tableStatement          = tableKeyword sep lcIdentifier optsep
                                "{" stmtsep
                                oidStatement stmtsep
                                *1(statusStatement stmtsep)
                                descriptionStatement stmtsep
                                *1(referenceStatement stmtsep)
                                rowStatement stmtsep
                            "}" optsep ";"

   rowStatement            = rowKeyword sep lcIdentifier optsep
                                "{" stmtsep
                                oidStatement stmtsep
                                anyIndexStatement stmtsep
                                *1(createStatement stmtsep)
                                *1(statusStatement stmtsep)
                                descriptionStatement stmtsep
```

```
                                 *1(referenceStatement stmtsep)
                                 1*(columnStatement stmtsep)
                             "}" optsep ";"

   columnStatement         = columnKeyword sep lcIdentifier optsep
                                 "{" stmtsep
                                 oidStatement stmtsep
                                 typeStatement stmtsep
                                 accessStatement stmtsep
                                 *1(defaultStatement stmtsep)
                                 *1(formatStatement stmtsep)
                                 *1(unitsStatement stmtsep)
                                 *1(statusStatement stmtsep)
                                 descriptionStatement stmtsep
                                 *1(referenceStatement stmtsep)
                             "}" optsep ";"

   notificationStatement   = notificationKeyword sep lcIdentifier
                                 optsep "{" stmtsep
                                 oidStatement stmtsep
                                 *1(objectsStatement stmtsep)
                                 *1(statusStatement stmtsep)
                                 descriptionStatement stmtsep
                                 *1(referenceStatement stmtsep)
                             "}" optsep ";"

   groupStatement          = groupKeyword sep lcIdentifier optsep
                                 "{" stmtsep
                                 oidStatement stmtsep
                                 membersStatement stmtsep
                                 *1(statusStatement stmtsep)
                                 descriptionStatement stmtsep
                                 *1(referenceStatement stmtsep)
                             "}" optsep ";"

   complianceStatement     = complianceKeyword sep lcIdentifier optsep
                                 "{" stmtsep
                                 oidStatement stmtsep
                                 *1(statusStatement stmtsep)
                                 descriptionStatement stmtsep
                                 *1(referenceStatement stmtsep)
                                 *1(mandatoryStatement stmtsep)
                                 *(optionalStatement stmtsep)
                                 *(refineStatement stmtsep)
                             "}" optsep ";"

   importStatement         = importKeyword sep ucIdentifier optsep
                                 "(" optsep
                                 identifierList optsep
```

```
                               ")" optsep ";"

   revisionStatement        = revisionKeyword optsep "{" stmtsep
                                   dateStatement stmtsep
                                   descriptionStatement stmtsep
                               "}" optsep ";"

   typedefTypeStatement     = typeKeyword sep refinedBaseType optsep ";"

   typeStatement            = typeKeyword sep
                                   (refinedBaseType / refinedType)
                                   optsep ";"

   writetypeStatement       = writetypeKeyword sep
                                   (refinedBaseType / refinedType)
                                   optsep ";"

   anyIndexStatement        = indexStatement /
                              augmentsStatement /
                              reordersStatement /
                              sparseStatement /
                              expandsStatement

   indexStatement           = indexKeyword *1(sep impliedKeyword) optsep
                                   "(" optsep qlcIdentifierList
                                   optsep ")" optsep ";"

   augmentsStatement        = augmentsKeyword sep qlcIdentifier
                                   optsep ";"

   reordersStatement        = reordersKeyword sep qlcIdentifier
                                   *1(sep impliedKeyword)
                                   optsep "(" optsep
                                   qlcIdentifierList optsep ")"
                                   optsep ";"

   sparseStatement          = sparseKeyword sep qlcIdentifier optsep ";"

   expandsStatement         = expandsKeyword sep qlcIdentifier
                                   *1(sep impliedKeyword)
                                   optsep "(" optsep
                                   qlcIdentifierList optsep ")"
                                   optsep ";"

   createStatement          = createKeyword *1(optsep createColumns)
                                   optsep ";"

   createColumns            = "(" optsep qlcIdentifierList optsep ")"
```

```
   oidStatement            = oidKeyword sep objectIdentifier optsep ";"

   dateStatement           = dateKeyword sep date optsep ";"

   organizationStatement   = organizationKeyword sep text optsep ";"

   contactStatement        = contactKeyword sep text optsep ";"

   formatStatement         = formatKeyword sep format optsep ";"

   unitsStatement          = unitsKeyword sep units optsep ";"

   statusStatement         = statusKeyword sep status optsep ";"

   accessStatement         = accessKeyword sep access optsep ";"

   defaultStatement        = defaultKeyword sep anyValue optsep ";"

   descriptionStatement    = descriptionKeyword sep text optsep ";"

   referenceStatement      = referenceKeyword sep text optsep ";"

   membersStatement        = membersKeyword optsep "(" optsep
                                 qlcIdentifierList optsep
                                 ")" optsep ";"

   objectsStatement        = objectsKeyword optsep "(" optsep
                                 qlcIdentifierList optsep
                                 ")" optsep ";"

   mandatoryStatement      = mandatoryKeyword optsep "(" optsep
                                 qlcIdentifierList optsep
                                 ")" optsep ";"

   optionalStatement       = optionalKeyword sep qlcIdentifier optsep
                                 "{" descriptionStatement stmtsep
                             "}" optsep ";"

   refineStatement         = refineKeyword sep qlcIdentifier optsep "{"
                                 *1(typeStatement stmtsep)
                                 *1(writetypeStatement stmtsep)
                                 *1(accessStatement stmtsep)
                                 descriptionStatement stmtsep
                             "}" optsep ";"

   ;;
   ;;
   ;;
```

```
   refinedBaseType         = OctetStringKeyword *1(optsep numberSpec) /
                             ObjectIdentifierKeyword /
                             Integer32Keyword *1(optsep numberSpec) /
                             Unsigned32Keyword *1(optsep numberSpec) /
                             Integer64Keyword *1(optsep numberSpec) /
                             Unsigned64Keyword *1(optsep numberSpec) /
                             Float32Keyword *1(optsep floatSpec) /
                             Float64Keyword *1(optsep floatSpec) /
                             Float128Keyword *1(optsep floatSpec) /
                             EnumerationKeyword namedNumberSpec /
                             BitsKeyword namedNumberSpec

   refinedType             = qucIdentifier *1(optsep anySpec)

   anySpec                 = numberSpec / floatSpec

   numberSpec              = "(" optsep numberElement
                                 *furtherNumberElement
                                 optsep ")"

   furtherNumberElement    = optsep "|" optsep numberElement

   numberElement           = signedNumber *1numberUpperLimit

   numberUpperLimit        = optsep ".." optsep signedNumber

   floatSpec               = "(" optsep floatElement
                                 *furtherFloatElement
                                 optsep ")"

   furtherFloatElement     = optsep "|" optsep floatElement

   floatElement            = floatValue *1floatUpperLimit

   floatUpperLimit         = optsep ".." optsep floatValue

   namedNumberSpec         = "(" optsep namedNumberList optsep ")"

   namedNumberList         = namedNumberItem
                                 *(optsep "," optsep namedNumberItem)
                                 *1(optsep ",")

   namedNumberItem         = lcIdentifier optsep "(" optsep number
                                 optsep ")"

   identifierList          = identifier
                                 *(optsep "," optsep identifier)
                                 *1(optsep ",")
```

```
   qIdentifierList         = qIdentifier
                                 *(optsep "," optsep qIdentifier)
                                 *1(optsep ",")

   qlcIdentifierList       = qlcIdentifier
                                 *(optsep "," optsep qlcIdentifier)
                                 *1(optsep ",")

   bitsValue               = "(" optsep bitsList optsep ")"

   bitsList                = *1(lcIdentifier
                                 *(optsep "," optsep lcIdentifier))
                                 *1(optsep ",")

   ;;
   ;; Other basic rules.
   ;;

   identifier              = ucIdentifier / lcIdentifier

   qIdentifier             = qucIdentifier / qlcIdentifier

   ucIdentifier            = ucAlpha *63(ALPHA / DIGIT / "-")

   qucIdentifier           = *1(ucIdentifier "::") ucIdentifier

   lcIdentifier            = lcAlpha *63(ALPHA / DIGIT / "-")

   qlcIdentifier           = *1(ucIdentifier "::") lcIdentifier

   text                    = textSegment *(optsep textSegment)

   textSegment             = DQUOTE *textAtom DQUOTE

   textAtom                = textVChar / HTAB / SP / lineBreak

   date                    = DQUOTE 4DIGIT "-" 2DIGIT "-" 2DIGIT
                                 *1(" " 2DIGIT ":" 2DIGIT)
                                 DQUOTE
                             ; always in UTC

   format                  = textSegment

   units                   = textSegment

   anyValue                = bitsValue /
                             negativeNumber /
                             hexadecimalNumber /
                             floatValue /
```

```
                             text /
                             objectIdentifier
                             ; Note: `objectIdentifer' includes the
                             ; syntax of enumeration labels and postive
                             ; numbers. They are not named literally to
                             ; avoid reduce/reduce conflicts when
                             ; building LR parsers based on this
                             ; grammar.

   status                  = currentKeyword /
                             deprecatedKeyword /
                             obsoleteKeyword

   access                  = noaccessKeyword /
                             notifyonlyKeyword /
                             readonlyKeyword /
                             readwriteKeyword

   objectIdentifier        = (qlcIdentifier / subid) *127("." subid)

   subid                   = decimalNumber

   number                  = hexadecimalNumber / decimalNumber

   negativeNumber          = "-" decimalNumber

   signedNumber            = number / negativeNumber

   decimalNumber           = "0" / (nonZeroDigit *DIGIT)

   zeroDecimalNumber       = 1*DIGIT

   hexadecimalNumber       = "0x" 1*(HEXDIG HEXDIG)

   floatValue              = neginfKeyword /
                             posinfKeyword /
                             snanKeyword /
                             qnanKeyword /
                             signedNumber "." zeroDecimalNumber
                                 *1("E" ("+"/"-") zeroDecimalNumber)

   ;;
   ;; Rules to skip unknown statements
   ;; with arbitrary arguments and blocks.
   ;;

   unknownStatement        = unknownKeyword optsep *unknownArgument
                                 optsep ";"
```

```
   unknownArgument         = ("(" optsep unknownList optsep ")") /
                             ("{" optsep *unknownStatement optsep "}") /
                             qucIdentifier /
                             anyValue /
                             anySpec

   unknownList             = namedNumberList /
                             qIdentifierList

   unknownKeyword          = lcIdentifier

   ;;
   ;; Keyword rules.
   ;;
   ;; Typically, keywords are represented by tokens returned from the
   ;; lexical analyzer.  Note, that the lexer has to be stateful to
   ;; distinguish keywords from identifiers depending on the context
   ;; position in the input stream.
   ;;
   ;; Also note, that these keyword definitions are represented in
   ;; cleartext for readability, while SMIng keywords are meant to be
   ;; case-sensitive, although ABNF makes quoted strings like these to
   ;; be case-insensitive.
   ;;

   ;; Statement keywords.

   moduleKeyword           = "module"
   importKeyword           = "import"
   revisionKeyword         = "revision"
   oidKeyword              = "oid"
   dateKeyword             = "date"
   organizationKeyword     = "organization"
   contactKeyword          = "contact"
   descriptionKeyword      = "description"
   referenceKeyword        = "reference"
   extensionKeyword        = "extension"
   typedefKeyword          = "typedef"
   typeKeyword             = "type"
   writetypeKeyword        = "writetype"
   nodeKeyword             = "node"
   scalarKeyword           = "scalar"
   tableKeyword            = "table"
   columnKeyword           = "column"
   rowKeyword              = "row"
   notificationKeyword     = "notification"
   groupKeyword            = "group"
   complianceKeyword       = "compliance"
   formatKeyword           = "format"
```

```
   unitsKeyword            = "units"
   statusKeyword           = "status"
   accessKeyword           = "access"
   defaultKeyword          = "default"
   impliedKeyword          = "implied"
   indexKeyword            = "index"
   augmentsKeyword         = "augments"
   reordersKeyword         = "reorders"
   sparseKeyword           = "sparse"
   expandsKeyword          = "expands"
   createKeyword           = "create"
   membersKeyword          = "members"
   objectsKeyword          = "objects"
   mandatoryKeyword        = "mandatory"
   optionalKeyword         = "optional"
   refineKeyword           = "refine"

   ;; Base type keywords.

   OctetStringKeyword      = "OctetString"
   ObjectIdentifierKeyword = "ObjectIdentifier"
   Integer32Keyword        = "Integer32"
   Unsigned32Keyword       = "Unsigned32"
   Integer64Keyword        = "Integer64"
   Unsigned64Keyword       = "Unsigned64"
   Float32Keyword          = "Float32"
   Float64Keyword          = "Float64"
   Float128Keyword         = "Float128"
   BitsKeyword             = "Bits"
   EnumerationKeyword      = "Enumeration"

   ;; Status keyword.

   currentKeyword          = "current"
   deprecatedKeyword       = "deprecated"
   obsoleteKeyword         = "obsolete"

   ;; Access keywords.

   noaccessKeyword         = "noaccess"
   notifyonlyKeyword       = "notifyonly"
   readonlyKeyword         = "readonly"
   readwriteKeyword        = "readwrite"

   ;; Special floating point values' keywords.

   neginfKeyword           = "neginf"
   posinfKeyword           = "posinf"
   snanKeyword             = "snan"
```

```
   qnanKeyword             = "qnan"

   ;;
   ;; Some low level rules.
   ;; These tokens are typically skipped by the lexical analyzer.
   ;;

   sep                     = 1*(comment / lineBreak / WSP)
                               ; unconditional separator

   optsep                  = *(comment / lineBreak / WSP)

   stmtsep                 = *(comment /
                               lineBreak /
                               WSP /
                               unknownStatement)

   comment                 = "//" *(WSP / VCHAR) lineBreak

   lineBreak               = CRLF / LF

   ;;
   ;; Encoding specific rules.
   ;;

   textVChar               = %x21 / %x23-7E
                               ; any VCHAR except DQUOTE

   ucAlpha                 = %x41-5A

   lcAlpha                 = %x61-7A

   nonZeroDigit            = %x31-39

   ;;
   ;; RFC 2234 core rules.
   ;;

   ALPHA          =  %x41-5A / %x61-7A
                          ; A-Z / a-z

   CR             =  %x0D
                          ; carriage return

   CRLF           =  CR LF
                          ; Internet standard newline

   DIGIT          =  %x30-39
                          ; 0-9
```

```
   DQUOTE          =  %x22
                           ; " (Double Quote)

   HEXDIG          =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

   HTAB            =  %x09
                           ; horizontal tab

   LF              =  %x0A
                           ; linefeed

   SP              =  %x20
                           ; space

   VCHAR           =  %x21-7E
                           ; visible (printing) characters

   WSP             =  SP / HTAB
                           ; white space

   ;;
   ;; EOF
   ;;
```

Appendix B. Glossary

columnar object: An object in a table row that may have zero, one or more object instances. Instances are identified by using the values of ‘indexing objects’ as ‘instance identifier’.

extension: The SMIng ‘extension’ statement can be used to define new statements. Extensions can express annotations to existent management information, agent capabilities known from SMIv2, or arbitrary other information. See Section 16 for details.

identifier: The name of any definition, either a module, type, node, scalar object, table, row, columnar object, notification, group, compliance, a named number of an enumeration or bits type or any construct defined by an SMIng extension. Every identifier starts with an upper-case or lower-case character, followed by letters, digits and hyphens, but without consecutive or trailing hyphens. The length of an identifier MUST NOT exceed 64 characters. Note that SMIng keywords may be used as identifiers, though it’s NOT RECOMMENDED. See also Section 2.1.

indexing objects: A Table may contain multiple instances of single columnar objects. That is, there may be multiple rows. The table’s ‘indexing objects’ are used to unambiguously distinguish the rows of a table. A special encoding of their values represents the columns’ instance-identifier, and thus identifies the row.

instance-identifier: That part of an object identifier value, that is used to unambiguously identify an instance of an object. For scalar objects, the instance-identifier is ‘0’. For columnar objects, the instance-identifier is built from the values of the ‘indexing objects’. See also Section 2.4.

module: A module is the container of inter-related management information, either managed objects or other definitions like type definitions or annotations. A module has to conform the SMIng grammar and semantics described by this document. A module represents a namespace in which local definitions are available and external definitions have to be imported.

named number: Values of ‘Enumeration’ types (Section 3.10) and single elements of ‘Bits’ (Section 3.11) types are integer numbers, each associated with an identifers. Those number-identifier pairs are called ‘named numbers’.

object: A leaf definition in the object identifier tree, that represents a class of object instances. Objects are exactly those definitions declared by the SMIng keywords ‘scalar’ or ‘column’.

object identifier: Management information is organized in a tree of nodes. Each node is unambiguously identified by an 'object identifier', that consists of a sequence of integer numbers ('sub-identifiers') which represent the path of nodes from the root to the addressed node in the tree. See also Section 2.

row: The kind of node used to group columnar objects of a table. Some significant information on table indexing and information used for row creation and deletion is associated with a table's row definition.

scalar object: An object that has zero or one object instance.

sub-identifier: A single component of an object identifier. There are at most 128 sub-identifiers in an object identifier value and each sub-identifier has a maximum value of 2^32-1 (4294967295).

table: The kind of node used to group management information that is organized in a sequence of rows.

Full Copyright Statement

F. libsmi Applikationsschnittstelle: smi.h

Dieser Abschnitt enthält die Datei `smi.h`, in der die Applikationsschnittstelle der `libsmi` definiert ist.

```
/*
 * smi.h --
 *
 *      Interface Implementation of libsmi.
 *
 * Copyright (c) 1999 Technical University of Braunschweig.
 *
 * See the file "license.terms" for information on usage and redistribution
 * of this file, and for a DISCLAIMER OF ALL WARRANTIES.
 *
 * @(#) $Id: smi.h,v 1.18 1999/05/10 14:26:47 strauss Exp $
 */

#ifndef _SMI_H
#define _SMI_H

#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>



#define SMI_CONFIG_FILE         "/usr/local/etc/smi.conf"

#define SMI_NAMESPACE_OPERATOR  "::"

#define SMI_VIEWALL     0x0800  /* all modules are `known', need no views.   */
#define SMI_ERRORS      0x1000  /* print parser errors.                      */
#define SMI_ERRORLINES  0x2000  /* print error line contents of modules.     */
#define SMI_STATS       0x4000  /* print statistics after parsing a module.  */
#define SMI_RECURSIVE   0x8000  /* recursively parse imported modules.       */
#define SMI_FLAGMASK \
                (SMI_VIEWALL|SMI_STATS|SMI_RECURSIVE|SMI_ERRORS|SMI_ERRORLINES)

/* limits of string lengths                                                  */
#define SMI_MAX_DESCRIPTOR      64              /*                           */
#define SMI_MAX_OID             1407            /* 128 * 10 digits + 127 dots */
#define SMI_MAX_STRING          65535           /* limit only valid in v1/v2  */
#define SMI_MAX_FULLNAME        130             /* 64 + 2 separator + 64      */



/* misc mappings of SMI types to C types                                     */
typedef char                    *SmiQIdentifier;
typedef char                    *SmiIdentifier;
typedef unsigned long           SmiUnsigned32;
typedef long                    SmiInteger32;
typedef unsigned long long      SmiUnsigned64;
```

```
typedef long long               SmiInteger64;
typedef char                    *SmiObjectIdentifier;
typedef unsigned int            SmiSubid;
typedef float                   SmiFloat32;
typedef double                  SmiFloat64;
typedef double                  SmiFloat128;   /* currently, no 64/128 diffs */



/* SmiBasetype -- base types of all languages                               */
typedef enum SmiBasetype {
    SMI_BASETYPE_UNKNOWN                = 0,  /* should not occur           */
    SMI_BASETYPE_INTEGER32              = 1,  /* also SMIv1/v2 INTEGER      */
    SMI_BASETYPE_OCTETSTRING            = 2,
    SMI_BASETYPE_OBJECTIDENTIFIER       = 3,
    SMI_BASETYPE_UNSIGNED32             = 4,
    SMI_BASETYPE_INTEGER64              = 5,  /* only SMIng                 */
    SMI_BASETYPE_UNSIGNED64             = 6,  /* SMIv2 and SMIng            */
    SMI_BASETYPE_FLOAT32                = 7,  /* only SMIng                 */
    SMI_BASETYPE_FLOAT64                = 8,  /* only SMIng                 */
    SMI_BASETYPE_FLOAT128               = 9,  /* only SMIng                 */
    SMI_BASETYPE_ENUM                   = 10,
    SMI_BASETYPE_BITS                   = 11, /* only SMIv2 and SMIng       */
    SMI_BASETYPE_HEXSTRING              = 12, /* only for values            */
    SMI_BASETYPE_BINSTRING              = 13, /* only for values, SMIv1/v2  */
    SMI_BASETYPE_LABEL                  = 14, /* labels as values           */
    SMI_BASETYPE_CHOICE                 = 15, /* only for parsing SMI specs */
    SMI_BASETYPE_SEQUENCE               = 16, /* only for parsing SMI specs */
    SMI_BASETYPE_SEQUENCEOF             = 17  /* only for parsing SMI specs */
} SmiBasetype;

/* SmiStatus -- values of status levels                                     */
typedef enum SmiStatus {
    SMI_STATUS_UNKNOWN          = 0, /* should not occur                    */
    SMI_STATUS_CURRENT          = 1, /* only SMIv2 and SMIng                */
    SMI_STATUS_DEPRECATED       = 2, /* SMIv1, SMIv2 and SMIng              */
    SMI_STATUS_MANDATORY        = 3, /* only SMIv1                          */
    SMI_STATUS_OPTIONAL         = 4, /* only SMIv1                          */
    SMI_STATUS_OBSOLETE         = 5  /* only SMIv2 and SMIng                */
} SmiStatus;

/* SmiAccess -- values of access levels                                     */
typedef enum SmiAccess {
    SMI_ACCESS_UNKNOWN          = 0, /* should not occur                    */
    SMI_ACCESS_NOT_ACCESSIBLE   = 1,
    SMI_ACCESS_NOTIFY           = 2,
    SMI_ACCESS_READ_ONLY        = 3,
    SMI_ACCESS_READ_WRITE       = 4,
    SMI_ACCESS_READ_CREATE      = 5,
    SMI_ACCESS_WRITE_ONLY       = 6
} SmiAccess;

/* SmiDecl -- type or statement that leads to a definition                  */
typedef enum SmiDecl {
    SMI_DECL_UNKNOWN            = 0,  /* shout not occur                    */
    /* SMIv1/V2 ASN.1 statements and macros */
    SMI_DECL_TYPEASSIGNMENT     = 1,
    SMI_DECL_VALUEASSIGNMENT    = 2,
    SMI_DECL_OBJECTTYPE         = 3,
    SMI_DECL_OBJECTIDENTITY     = 4,
    SMI_DECL_MODULEIDENTITY     = 5,
    SMI_DECL_NOTIFICATIONTYPE   = 6,
    SMI_DECL_TRAPTYPE           = 7,
```

```
    SMI_DECL_OBJECTGROUP        = 8,
    SMI_DECL_NOTIFICATIONGROUP  = 9,
    SMI_DECL_MODULECOMPLIANCE   = 10,
    SMI_DECL_AGENTCAPABILITIES  = 11,
    SMI_DECL_TEXTUALCONVENTION  = 12,
    /* SMIng statements */
    SMI_DECL_MODULE             = 33,
    SMI_DECL_TYPEDEF            = 34,
    SMI_DECL_NODE               = 35,
    SMI_DECL_SCALAR             = 36,
    SMI_DECL_TABLE              = 37,
    SMI_DECL_ROW                = 38,
    SMI_DECL_COLUMN             = 39,
    SMI_DECL_NOTIFICATION       = 40,
    SMI_DECL_GROUP              = 41,
    SMI_DECL_COMPLIANCE         = 42
} SmiDecl;

/* SmiIndexkind -- actual kind of a table row's index method                 */
typedef enum SmiIndexkind {
    SMI_INDEX_UNKNOWN           = 0,
    SMI_INDEX_INDEX             = 1,
    SMI_INDEX_AUGMENT           = 2,
    SMI_INDEX_REORDER           = 3,
    SMI_INDEX_SPARSE            = 4,
    SMI_INDEX_EXPAND            = 5
} SmiIndexkind;

/* SmiRevision -- content of a single module's revision clause               */
typedef struct SmiRevision {
    time_t              date;
    char                *description;
} SmiRevision;

/* SmiValue -- any single value; for use in default values and subtyping     */
typedef struct SmiValue {
    SmiBasetype         basetype;
    union {
        SmiUnsigned64       unsigned64;
        SmiInteger64        integer64;
        SmiUnsigned32       unsigned32;
        SmiInteger32        integer32;
        SmiFloat32          float32;
        SmiFloat64          float64;
        SmiFloat128         float128;
        SmiObjectIdentifier oid;
        char                *ptr;
        char                **bits;      /* array of SmiNamedNumber pointers */
    } value;
} SmiValue;

/* SmiNamedNumber -- a named number; for enumeration and bitset types        */
typedef struct SmiNamedNumber {
    SmiIdentifier       name;
    SmiValue            *valuePtr;
} SmiNamedNumber;

/* SmiRange -- a min-max value range; for subtyping of sizes or numbers      */
typedef struct SmiRange {
    SmiValue            *minValuePtr;
    SmiValue            *maxValuePtr;
} SmiRange;
```

```
/* SmiOption -- an optional group in a compliance statement                  */
typedef struct SmiOption {
    SmiIdentifier       name;
    SmiIdentifier       module;
    char                *description;
} SmiOption;

/* SmiRefinement -- a refined object in a compliance statement               */
typedef struct SmiRefinement {
    SmiIdentifier       name;
    SmiIdentifier       module;
    char                *description;
    SmiQIdentifier      type;
    SmiQIdentifier      writetype;
    SmiAccess           access;
} SmiRefinement;

/* SmiModule -- the main structure of a module                               */
typedef struct SmiModule {
    SmiIdentifier       name;
    SmiQIdentifier      object;
    time_t              lastupdated;   /* for apps with SMIv2 semantics */
    char                *organization;
    char                *contactinfo;
    char                *description;
    char                *reference;
} SmiModule;

/* SmiNode -- the main structure of any clause that defines a node           */
typedef struct SmiNode {
    char                *name;
    char                *module;
    SmiObjectIdentifier oid;
    char                *type;
    char                **list;
    SmiIndexkind        indexkind;
    int                 implied;
    char                **index;
    char                *relatedrow;
    SmiOption           **option;
    SmiRefinement       **refinement;
    SmiDecl             decl;
    SmiBasetype         basetype;
    SmiAccess           access;
    SmiStatus           status;
    char                *format;
    SmiValue            *value;
    char                *units;
    char                *description;
    char                *reference;
} SmiNode;

/* SmiType -- the main structure of a type definition (also base types)      */
typedef struct SmiType {
    char                *name;
    char                *module;
    SmiBasetype         basetype;
    char                *parent;
    void                **list;   /* SmiNamedNumber or SmiRange */
    SmiDecl             decl;
    char                *format;
    SmiValue            *value;
    char                *units;
```

```
    SmiStatus           status;
    char                *description;
    char                *reference;
} SmiType;

/* SmiMacro -- the main structure of a SMIv1/v2 macro or SMIng extension     */
typedef struct SmiMacro {
    char                *name;
    char                *module;
} SmiMacro;


extern char *smingStringStatus(SmiStatus status);

extern char *smiStringStatus(SmiStatus status);

extern char *smingStringAccess(SmiAccess access);

extern char *smiStringAccess(SmiAccess access);

extern char *smiStringDecl(SmiDecl macro);

extern char *smiStringBasetype(SmiBasetype basetype);

extern char *smiModule(char *fullname);

extern char *smiDescriptor(char *fullname);

extern time_t smiMkTime(const char *s);

extern char *smiCTime(time_t t);

extern char *smingCTime(time_t t);


extern void smiInit();

extern void smiSetDebugLevel(int level);

extern void smiSetErrorLevel(int level);

extern void smiSetFlags(int userflags);

extern int smiGetFlags();

extern int smiReadConfig(const char *file);

extern int smiAddLocation(const char *location);

extern int smiLoadModule(char *modulename);


extern SmiModule *smiGetModule(char *spec);

extern void smiFreeModule(SmiModule *smiModulePtr);

extern char **smiGetImports(char *modulename);

extern int smiIsImported(char *modulename,
                         char *fullname);
```

```
extern SmiRevision *smiGetFirstRevision(char *modulename);

extern SmiRevision *smiGetNextRevision(char *modulename,
                                       time_t date);

extern void smiFreeRevision(SmiRevision *smiRevisionPtr);



extern SmiType *smiGetType(char *spec,
                           char *mod);

extern SmiType *smiGetFirstType(char *modulename);

extern SmiType *smiGetNextType(char *modulename,
                               char *name);

extern void smiFreeType(SmiType *smiTypePtr);



extern SmiMacro *smiGetMacro(char *spec,
                             char *mod);

extern void smiFreeMacro(SmiMacro *smiMacroPtr);



extern SmiNode *smiGetNode(char *spec,
                           char *mod);

extern SmiNode *smiGetFirstNode(char *modulename);

extern SmiNode *smiGetNextNode(char *modulename,
                               char *name);

extern void smiFreeNode(SmiNode *smiNodePtr);



extern char **smiGetNames(char *spec,
                          char *mod);

extern char *smiGetParent(char *spec,
                          char *mod);

extern char **smiGetChildren(char *spec,
                             char *mod);


#endif /* _SMI_H */
```

G. Manual Pages

Die folgenden Seiten umfassen die Manual Pages der im Rahmen dieser Arbeiten entstandenen `libsmi`-Implementierung sowie der zwei Applikationen `smilint` und `smidump`.

NAME

libsmi − library to access SMI management information

DESCRIPTION

This library contains functions that allow access to management information stored in repositories that are managed by **libsmi**, currently directories containing SMIv1/v2 and SMIng MIB module files.

The libsmi API is defined in **smi.h**. Functions are documented in **smi_config**(3), **smi_module**(3), **smi_node**(3), **smi_type**(3), **smi_macro**(3), and **smi_string**(3).

The purpose of libsmi is to separate the weird parsing and handling of MIB modules from management applications. MIB modules conforming to SMIv1, SMIv2 and SMIng - a new language for management information, developed at the Technical University of Braunschweig, Germany - can be handled by libsmi.

EXAMPLE

```
#include <stdio.h>
#include <string.h>
#include <smi.h>

int main(int argc, char *argv[])
{
   char *modulename, *fullname;
   SmiNode *smiNode, *smiStartNode;

   if (argc != 2) {
      fprintf(stderr, "Usage: smisubtree oid\n");
      exit(1);
   }

   smiInit();

   fullname   = argv[1];
   modulename = smiModule(fullname);

   smiLoadModule(modulename);

   for(smiNode = smiStartNode = smiGetNode(fullname, NULL);
      smiNode &&
      strstr(smiNode->oid, smiStartNode->oid) == smiNode->oid;
      smiNode = smiGetNextNode(smiNode->name, modulename)) {

      printf("%s::%-16s   [%s]\n",
           smiNode->module, smiNode->name,
           smiStringBasetype(smiNode->basetype));
   }
}
```

FILES

/usr/local/include/smi.h SMI library header file
/usr/local/etc/smi.conf global configuration file
/usr/local/etc/smi*.conf other configuration files

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

RFCs 2578-2580 specify the SMIv2 standard. SMIng is specified as part of a master's thesis by Frank Strauss at the Technical University of Braunschweig, Germany. The specification is a working document of the IRTF Network Management Working Group (NMRG) (draft-irtf-nmrg-sming).

smi.h, **smi_config**(3), **smi_module**(3), **smi_node**(3), **smi_type**(3), **smi_macro**(3), **smi_string**(3)

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smiInit, smiSetDebugLevel, smiSetErrorLevel, smiGetFlags, smiSetFlags, smiReadConfig, smiAddLocation, smiLoadModule – SMI library configuration routines

SYNOPSIS

#include <smi.h>

void smiInit();

void smiSetDebugLevel(int *level***);**

void smiSetErrorLevel(int *level***);**

int smiGetFlags();

void smiSetFlags(int *userflags***);**

int smiReadConfig(const char * *file***);**

int smiAddLocation(const char ****location*);**

int smiLoadModule(char ****modulename*);**

DESCRIPTION

These functions provide some initialization and adjustment operations for the SMI library.

The **smiInit()** function should be the first SMI function called in an application. It initializes its internal structures and reads a global configuration file (**/usr/local/etc/smi.conf**) which may set some default behaviours. See *CONFIGURATION FILES* below.

The **smiSetDebugLevel()** function sets the debugging verbosity (0-9) of the SMI library. This is for library debugging purpose only and might even be turned off completely at library compile time. The higher the level, the louder it shouts. The perfect default level is 0.

The **smiSetErrorLevel()** function sets the pedantic level (0-9) of the SMI parsers of the SMI library, currently SMIv1/v2 and SMIng (which might, but should not be turned off at library compile time). The higher the level, the louder it complains. Values up to 3 should be regarded as errors, higher level could be interpreted as warnings. But note that this classification is some kind of personal taste. The default level is 0, since usually only MIB checkers want to tune a higher level.

The **smiGetFlags()** and **smiSetFlags()** functions allow to fetch, modify, and set some *userflags* that control the SMI library's behaviour. If **SMI_ERRORS** is not set, no error messages are printed at all to keep the SMI library totally quiet, which might be mandatory for some applications. If **SMI_ERRORLINES** is set and an error is printed, it is followed by two lines: the current source module line and a second line that marks the current point in the line above. If **SMI_STATS** is set and the error level is 9, the library prints some module statistics. If **SMI_RECURSIVE** is set, the library also complains about errors in modules that are read due to import statements.

The **smiReadConfig()** function allows to read an additional configuration *file*. See *CONFIGURATION FILES* below.

The **smiAddLocation()** function allows to specify an additional *location* where the SMI library looks for MIB modules to read. See *MODULE LOCATIONS* below.

The **smiLoadModule()** function specifies an additional MIB module by *modulename* that the application claims to know. Only after a module is made known through this function, iterating retrieval functions and retrieval functions without fully qualified descriptors will return results from this module.

RETURN VALUES

The functions **smiReadConfig()**, **smiAddLocation()**, and **smiLoadModule()** return 0 on success, or negative values on error. **smiGetFlags()** returns the current set of flags. The other functions are void.

MODULE LOCATIONS

The SMI library may retrieve MIB modules from different kinds of resources, named locations. A location is specified by the method (currently "smi" or "sming"), a colon ":" and a concrete specification which depends on the method (currently only a directory specification, where modules are search by file name without any extension). Examples are:

```
smi:.
smi:/usr/local/lib/smi/mibs-smi
sming:/usr/local/lib/smi/mibs-sming
```

Locations are searched in the order they are specified.

CONFIGURATION FILES

Configuration files contain line oriented text. Lines starting with a '#' character and blank lines are ignored. Other lines are expected to be valid configuration statements starting with a keyword followed by an optional argument.

location *location*
: specifies an additional *location*, equivalent to a **smiAddLocation()** call.

preload *module*
: (pre)loads a *module*, equivalent to a **smiLoadModule()** call.

loglevel *level*
: sets the level (0-9) of error logging to *level*, equivalent to a **smiErrorLevel()** call.

debuglevel *level*
: sets the level (0-9) of debugging to *level*, equivalent to a **smiDebugLevel()** call.

yydebug *1|0*
: turns the yyerror facility on or off, see bison(1).

statistics *1|0*
: turns the statistics flag on or off, see **smiSetFlags()** above.

importlogging *1|0*
: turns the error logging of imported modules on or off, see **smiSetFlags()** above.

errorlines *1|0*
: turns the output of marked erroneous lines on or off, see **smiSetFlags()** above.

The global configuration file **/usr/local/etc/smi.conf** is read before any other libsmi operation.

FILES

/usr/local/include/smi.h	SMI library header file
/usr/local/etc/smi.conf	global configuration file
/usr/local/etc/smi*.conf	other configuration files

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

libsmi(3), **smi.h**

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smingStringStatus, smiStringStatus, smingStringAccess, smiStringAccess, smiStringDecl, smiStringBasetype, smiModule, smiDescriptor, smiMkTime, smiCTime, smingCTime – SMI string conversion routines

SYNOPSIS

#include <time.h>
#include <smi.h>

char *smingStringStatus(SmiStatus *status***);**

char *smiStringStatus(SmiStatus *status***);**

char *smingStringAccess(SmiAccess *access***);**

char *smiStringAccess(SmiAccess *access***);**

char *smiStringDecl(SmiDecl *macro***);**

char *smiStringBasetype(SmiBasetype *basetype***);**

char *smiModule(char * *fullname***);**

char *smiDescriptor(char * *fullname***);**

time_t smiMkTime(const char ***s*);**

char *smiCTime(time_t *t***);**

char *smingCTime(time_t *t***);**

DESCRIPTION

These functions allow conversions between various types of SMI elements and printable strings.

The **smingStringStatus()** function returns a string that represents the SMIng notation of a *status* value. If *status* is not an SMIng conformant status or does not allow an adequate conversion to SMIng, the string "<unknown>" is returned.

The **smiStringStatus()** function returns a string that represents the SMIv1/v2 notation of a *status* value. If *status* is not an SMIv1 or SMIv2 conformant status or does not allow an adequate conversion to SMIv2, the string "<unknown>" is returned.

The **smingStringAccess()** function returns a string that represents the SMIng notation of a *access* value. If *access* is not an SMIng conformant access level or does not allow an adequate conversion to SMIng, the string "<unknown>" is returned.

The **smiStringAccess()** function returns a string that represents the SMIv1/v2 notation of a *access* value. If *access* is not an SMIv1/v2 conformant access level or does not allow an adequate conversion to SMIv1/v2, the string "<unknown>" is returned.

The **smiStringDecl()** function returns a string that represents the SMIv1/v2 notation of the macro or statement or the SMIng notation of the statement given by the *macro* value. If *macro* is not such a macro or statement, the string "<unknown>" is returned.

The **smiStringBasetype()** function returns a string that represents the SMIng notation of the base type given by the *basetype* value. If *basetype* is not a valid base type, the string "<unknown>" is returned.

The **smiModule()** function returns a string that represents the module part of the fully qualified descriptor *fullname*.

The **smiDescriptor()** function returns a string that represent the local descriptor part of the fully qualified descriptor *fullname*.

The **smiMkTime()** function takes a string argument that represents an 11-char or 13-char SMIv2 style date (**LAST-UPDATED** or **REVISION**), or a 10-char or 16-char SMIng style date (**revision**) and returns its value of type *time_t* or zero if it cannot be converted.

The **smiCTime()** function returns a string that represents the 13-char SMIv2 style date and time given by *t*.

The **smingCTime()** function returns a string that represents the 16-char SMIng style date and time given by *t*.

RETURN VALUE

The returned strings of all functions must not be released by the caller. The strings, returned by **smiModule()**, **smiDescriptor()**, **smiCTime()** and **smingCTime()** are static. Thus, the caller must copy them before the next call to the same function, if required.

FILES

/usr/local/include/smi.h SMI library header file

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

smi.h, **libsmi**(3)

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smiGetModule, smiFreeModule, smiGetImports, smiIsImported, smiGetFirstRevision, smiGetNextRevision, smiFreeRevision – SMI module information routines

SYNOPSIS

#include <time.h>
#include <smi.h>

SmiModule *smiGetModule(char ***_spec_);**

void smiFreeModule(SmiModule ***_smiModulePtr_);**

char **smiGetImports(char ***_spec_, char ***_mod_**);**

int smiIsImported(char ***_modulename_, char ***_fullname_**);**

SmiRevision *smiGetFirstRevision(char ***_modulename_);**

SmiRevision *smiGetNextRevision(char ***_modulename_, time_t** _date_**);**

void smiFreeRevision(SmiRevision ***_smiRevisionPtr_);**

```
typedef struct SmiModule {
   SmiIdentifier    name;
   SmiQIdentifier   object; /* module identity object */
   time_t           lastupdated;
   char             *organization;
   char             *contactinfo;
   char             *description;
   char             *reference;
} SmiModule;

typedef struct SmiRevision {
   time_t           date;
   char             *description;
} SmiRevision;
```

DESCRIPTION

These functions retrieve different meta information on MIB modules. The functions to retrieve the definitions within a module are documented in smi_type(3), smi_node(3), and smi_macro(3).

The **smiGetModule()** function retrieves a **struct SmiModule** that represents the module named **spec**. Elements not available or not present in the module contain NULL.

The **smiFreeModule()** function should be called when a retrieved **struct SmiModule** is no longer needed to release its memory.

The **smiGetImports()** function returns an array of all fully qualified descriptors imported in _modulename_.

The **smiIsImported()** function returns a positive value if the fully qualified descriptor _fullname_ is imported in module _modulename_, or zero otherwise.

The **smiGetFirstRevision()** and **smiGetNextRevision()** functions are used to iteratively retrieve **struct SmiModule**s. **smiGetFirstRevision()** returns the first (in the module that is most recent) revision information on module _modulename_. Subsequent calls to **smiGetNextRevision()** return the revision after (timely before) that one specified by _date_ which is usually the date of the last revision. If there are no more revisions NULL is returned.

The **smiFreeRevision()** function should be called for each retrieved **struct SmiRevision** that is no longer needed to release its memory.

RETURN VALUES

The retrieval functions **smiGetModule()**, **smiGetFirstRevision()**, and **smiGetNextRevision()** return the requested structure on success, or NULL on error. Structures should be released after use by calling **smiFreeModule()**, or **smiFreeRevision()**, respectively.

smiGetImports() returns an array of strings, or NULL on error. **smiIsImported()** returns an appropriate integer truth value. Other functions are void.

FILES

/usr/local/include/smi.h   SMI library header file

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

libsmi(3), **smi.h**

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smiGetNode, smiGetFirstNode, smiGetNextNode, smiFreeNode − SMI type information routines

SYNOPSIS

#include <smi.h>

SmiNode *smiGetNode(char ***spec*, char ***mod***);**

SmiNode *smiGetFirstNode(char ***modulename*);**

SmiNode *smiGetNextNode(char ***modulename*, char ***name***);**

void smiFreeNode(SmiNode ***smiNodePtr*);**

```
typedef struct SmiNode {
    char              *name;
    char              *module;
    SmiObjectIdentifier oid;        /* dotted string */
    char              *type;      /* fully qualified */
    char              **list;     /* depends on kind of node */
    SmiIndexkind      indexkind;
    int               implied;
    char              **index;
    char              *relatedrow;
    SmiOption         **option;    /* only compliance nodes */
    SmiRefinement     **refinement; /* only compliance nodes */
    SmiDecl           decl;
    SmiBasetype       basetype;
    SmiAccess         access;
    SmiStatus         status;
    char              *format;
    SmiValue          *value;
    char              *units;
    char              *description;
    char              *reference;
} SmiNode;
```

DESCRIPTION

These functions retrieve information on any node definition in the object identifier tree, these are ASN.1 object identifier assignments, MODULE-IDENTITYs, OBJECT-IDENTITYs, OBJECT-TYPEs, NOTIFICATION-TYPEs, TRAP-TYPEs, OBJECT-GROUPs, NOTIFICATION-GROUPs, MODULE-COMPLIANCEs, and AGENT-CAPABILITYs in SMIv1/v2 and module, node, scalar, table, row, column, notification, group, and compliance statements in SMIng.

The **smiGetNode()** function retrieves a **struct SmiNode** that represents a node of any kind. Either *spec* must be the fully qualified descriptor leaving *mod* NULL, or the name is split into the module *mod* and descriptor *spec*. Elements not available or not present in the definition contain NULL. See **smi.h** for the structure of contained elements.

The **smiGetFirstNode()** and **smiGetNextNode()** functions are used to iteratively retrieve **struct SmiNodes**. **smiGetFirstNode()** returns the first node defined in module *modulename* with respect to an object identifier tree in pre-order (thus not necessarily the first defined in the MIB module). Subsequent calls to **smiGetNextNode()** return the next node after that one specified by *name* which is usually the **name** element of the last returned **struct SmiNode**. If there are no more node definitions in the module NULL is returned.

The **smiFreeNode()** function should be called when a retrieved **struct SmiNode** is no longer needed to release its memory.

RETURN VALUES

The retrieval functions **smiGetNode()**, **smiGetFirstNode()**, and **smiGetNextNode()** return the requested **struct smiNode** on success, or NULL on error. Structures should be released after use by calling **smiFreeNode()** which is a void function.

FILES

/usr/local/include/smi.h SMI library header file

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

libsmi(3), **smi.h**

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smiGetType, smiGetFirstType, smiGetNextType, smiFreeType − SMI type information routines

SYNOPSIS

#include <smi.h>

SmiType *smiGetType(char ***_spec_, char ***_mod_**);**

SmiType *smiGetFirstType(char ***_modulename_);**

SmiType *smiGetNextType(char ***_modulename_, char ***_name_**);**

void smiFreeType(SmiType ***_smiTypePtr_);**

```
typedef struct SmiType {
    char        *name;
    char        *module;
    SmiBasetype  basetype;
    char        *parent;  /* fully qualified */
    void        **list;   /* SmiNamedNumber or SmiRange */
    SmiDecl      decl;
    char        *format;
    SmiValue     *value;   /* default value */
    char        *units;
    SmiStatus    status;
    char        *description;
    char        *reference;
} SmiType;
```

DESCRIPTION

These functions retrieve information on a type definition (SMIng) or a simple ASN.1 type definition (SMIv1/v2) or a TEXTUAL-CONVENTION definition (SMIv2). Base types may also be retrieved, their **module** element is an empty string.

The **smiGetType()** function retrieves a **struct SmiType** that represents a type or textual convention. Either *spec* must be the fully qualified descriptor leaving *mod* NULL, or the name is split into the module *mod* and descriptor *spec*. Elements not available or not present in the definition contain NULL. See **smi.h** for the structure of contained elements.

The **smiGetFirstType()** and **smiGetNextType()** functions are used to iteratively retrieve **struct SmiType**s. **smiGetFirstType()** returns the first type or textual convention defined in module *modulename*. Subsequent calls to **smiGetNextType()** return the type or textual convention defined after that one specified by *name* which is usually the **name** element of the last returned **struct SmiType**. If there are no more type or textual convention definitions in the module NULL is returned.

The **smiFreeType()** function should be called when a retrieved **struct SmiType** is no longer needed to release its memory.

RETURN VALUES

The retrieval functions **smiGetType()**, **smiGetFirstType()**, and **smiGetNextType()** return the requested **struct smiType** on success, or NULL on error. Structures should be released after use by calling **smiFreeType()** which is a void function.

FILES

/usr/local/include/smi.h   SMI library header file

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

libsmi(3), **smi.h**

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smiGetMacro, smiFreeMacro − SMI statement information routines

SYNOPSIS

#include <smi.h>

SmiMacro *smiGetMacro(char ***_spec_, char ***_mod_**);**

void smiFreeMacro(SmiMacro ***_smiMacroPtr_);**

```
typedef struct SmiMacro {
    char            *name;
    char            *module;
} SmiMacro;
```

DESCRIPTION

These functions retrieve information on an extension statement definition (SMIng) or a macro (SMIv1/v2).

The **smiGetMacro()** function retrieves a **struct SmiMacro** that represents an extension or macro definition. Either _spec_ must be the fully qualified descriptor leaving _mod_ NULL, or the name is split into the module _mod_ and descriptor _spec_.

The **smiFreeMacro()** function should be called when a retrieved **struct SmiMacro** is no longer needed to release its memory.

RETURN VALUES

The retrieval function **smiGetMacro()** returns the requested **struct smiMacro** on success, or NULL on error. Structures should be released after use by calling **smiFreeMacro()** which is a void function.

FILES

/usr/local/include/smi.h   SMI library header file

NOTES

Libsmi is work in progress. Interface inconsistencies and deficiencies will be removed in near future.

SEE ALSO

libsmi(3), **smi.h**

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smilint – syntax and semantic checks of SMIv1/v2/ng modules

SYNOPSIS

smilint [**-sSvV**] [**-l** *level*] [**-d** *level*] [**-c** *config*] [**-L** *location*] *module*

DESCRIPTION

The **smilint** program is used to check MIB modules for syntax errors and semantics at some degree. Both, SMIv1/v2 and SMIng style MIB modules are supported.

The rules that **smilint** is based on are taken from RFC 1155, RFC 1212 and RFC 1215 for SMIv1, RFCs 2578-2580 for SMIv2 and the specifications of SMIng that are part of a master's thesis at the Technical University of Braunschweig, Germany.

Smilint first reads the global libsmi configuration file **/usr/local/etc/smi.conf**, as any libsmi application does. Then the command line options are interpreted and finally, the **/usr/local/etc/smilint.conf** configuration file is read, if present, before **smilint** starts its main operation.

OPTIONS

-s Report some per module statistics to stderr.

-S Don't report statistics (default).

-v On errors, do print the current line of input to stderr after the error message and a marker pointing to the current token in the input stream. Note that the corresponding point of error usually lies before this marker.

-V Don't print error lines (default).

-d *level* Print libsmi debugging messages at the given *level* (default: 0).

-l *level* Report errors and warnings up to the given *level* to stderr. A higher level shows more warnings. The actual levels are:

0 internal error, no recovery possible,
1 major SMI error, recovery somehow possible but
may lead to severe problems,
2 SMI error, probably tolerated by some
implementations,
3 SMI error, tolerated by many implementations,
4 something which is recommended to be changed,
5 something that might be ok, but might not be
recommended under some circumstances.

Higher levels are actually not used and lead to the same effects as level 5 does. Note that errors up to level 3 are errors violating the specification and must be fixed by the responsible author. Level 3 is the default level.

-c *config*

Read the configuration from *file*. See smi_config(3) section CONFIGURATION FILES for its format.

-L *location*

Add *location* to the list of locations to search for MIB modules. See smi_config(3) section MODULE LOCATIONS for location details.

module This is the module to be checked. Note that this module is searched in the list of known locations. If the module shall be taken from an SMIv1/v2 file in the current directory, a "**-L smi:.**" option should be added to the command line.

FILES

/usr/local/etc/smi.conf general configuration file
/usr/local/etc/smilint.conf smilint configuration file

NOTES

Libsmi and this tool are work in progress. Expect changes in near future.

SEE ALSO

The SMIng and **libsmi**(3) project is documented at **http://www.ibr.cs.tu-bs.de/˜strauss/sming/**. Other commonly used MIB checkers are **mosy**(1) and **smicng**(1).

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

NAME

smidump − dump and convert SMIv1/v2/ng modules

SYNOPSIS

smidump [**-sSvV**] [**-l** *level*] [**-d** *level*] [**-c** *config*] [**-L** *location*] [**-D** *format*] *module*

DESCRIPTION

The **smidump** program is used to dump the contents of a single MIB module to stdout in a selectable format. This format may be a simple list of objects or types but also a format fully compliant to SMIv2 or SMIng. This allows to convert modules from SMIv2 to SMIng and from SMIng to SMIv2.

Smidump first reads the global libsmi configuration file **/usr/local/etc/smi.conf**, as any libsmi application does. Then the command line options are interpreted and finally, the **/usr/local/etc/smidump.conf** configuration file is read, if present, before **smidump** starts its main operation.

OPTIONS

-s Report some per module statistics to stderr.

-S Don't report statistics (default).

-v On errors, do print the current line of input to stderr after the error message and a marker pointing to the current token in the input stream. Note that the corresponding point of error usually lies before this marker.

-V Don't print error lines (default).

-d *level* Print libsmi debugging messages at the given *level* (default: 0).

-l *level* Report errors and warnings up to the given *level* to stderr. A higher level shows more warnings. The actual levels are:

```
0   internal error, no recovery possible,
1   major SMI error, recovery somehow possible but
    may lead to severe problems,
2   SMI error, probably tolerated by some
    implementations,
3   SMI error, tolerated by many implementations,
4   something which is recommended to be changed,
5   something that might be ok, but might not be
    recommended under some circumstances.
```

Higher levels are actually not used and lead to the same effects as level 5 does. Note that errors up to level 3 are errors violating the specification and must be fixed by the responsible author. Level 3 is the default level.

-c *config*

Read the configuration from *file*. See smi_config(3) section CONFIGURATION FILES for its format.

-L *location*

Add *location* to the list of locations to search for MIB modules. See smi_config(3) section MODULE LOCATIONS for location details.

-D *format*

Use *format* when dumping a module. Supported formats are **sming** for fully compliant SMIng, **smiv2** for fully compliant SMIv2, **objects** for a simple list of nodes in the object identifer tree known to libsmi (only for libsmi debugging) and **types** for a simple list of all types known to libsmi (also for libsmi debugging only). The default is to dump in SMIng format.

module This is the module to be checked. Note that this module is searched in the list of known locations. If the module shall be taken from an SMIv1/v2 file in the current directory, a "**-L smi:.**" option should be added to the command line.

EXAMPLE

This example converts the SMIv2 module IF-MIB in the current directory with the highest parser error verbosity to IF-MIB.sming in SMIng format.

```
$ smidump -Lsmi:. -l9 -Dsming IF-MIB > IF-MIB.sming
```

FILES

/usr/local/etc/smi.conf general configuration file
/usr/local/etc/smidump.conf smidump configuration file

NOTES

Libsmi and this tool are work in progress. Expect changes in near future.

SEE ALSO

The SMIng and **libsmi**(3) project is documented at **http://www.ibr.cs.tu-bs.de/˜strauss/sming/**.

smilint(1)

AUTHOR

(C) 1999 Frank Strauss, TU Braunschweig, Germany <strauss@ibr.cs.tu-bs.de>

H. Software

Die dieser Arbeit beiliegende Diskette im MSDOS-Format enthält ein mit `gzip` komprimiertes *UNIX tape archive* (`tar`) mit dem Namen `libsmi.tgz`. Dieses Archiv enthält die in Kapitel 3 entwickelte und in Anhang E angegebene SMIng Spezifikation und die in Kapitel 4 beschriebene Implementierung der `libsmi`, einschließlich Manual Pages, Konfigurationsdateien sowie die in Abschnitt 4.2 beschriebenen Werkzeuge `smilint` und `smidump`.

Das komprimierte Archiv ist zunächst auf das Zielsystem zu kopieren. Dies kann mit den *MTools* wie folgt geschehen:

```
mcopy a:libsmi.tgz libsmi.tar.gz
```

Anschließend wird das komprimierte Archiv dekomprimiert:

```
gunzip libsmi.tar.gz
```

Zur Überprüfung, ob es sich um das korrekte Archiv handelt, kann mit der folgenden Anweisung die MD5 Prüfsumme des Archivs berechnet werden:

```
md5sum libsmi.tar
```

Nur wenn hierbei der Wert

```
5c2183c456ae82c7c90cf7a9db37f4f6
```

berechnet wird, handelt es sich exakt um das Archiv, das Bestandteil der vorliegenden Arbeit ist. Ist dies der Fall oder soll bewußt mit einem anderen (neueren) Archiv weitergearbeitet werden, so kann es nun wie folgt ausgepackt werden:

```
tar xvf libsmi.tar
```

Dabei entsteht das Verzeichnis `libsmi`, das folgende Dateien und Verzeichnisse enthält:

`README`	Dokumentation des `libsmi` Archivs
`Makefile`	Makefile
`doc/*.3`	Manual Pages der `libsmi` Bibliothek
`doc/*.1`	Manual Pages der Applikationen
`doc/draft-irtf-nmrg-sming-00.txt`	SMIng Spezifikation
`doc/sming.abnf`	SMIng ABNF Grammatik
`doc/IRTF-NMRG-SMING*`	SMIng Module aus der Spezifikation
`lib/`	Quelltexte der Bibliothek
`tools/`	Quelltexte der Applikationen
`etc/`	Konfigurationsdateien zur Installation
`mibs/sming/`	SMIng MIB Module zur Installation
`mibs/smi/`	SMIv1/v2 MIB Module zur Installation

Das Erzeugen der Bibliothek und der Applikationen erfordert einige GNU Werkzeuge: `make`, `gcc`, `flex` und `bison`. Nach dem Anpassen des Makefiles an die lokalen Gegebenheiten, können die Bibliothek und die Applikationen durch einen Aufruf von

```
make
```

erzeugt werden. Wenn dies fehlerfrei geschehen ist, können mit

```
make install
```

die Bibliothek (`smi.h`, `libsmi.a`), die Applikationen (`smilint`, `smidump`) sowie einige MIB Module und Manual Pages installiert werden.

Beispiele für die Benutzung der Bibliothek und der Applikationen sind in den Abschnitten 4.1.7 und 4.2 zu finden.

Literaturverzeichnis

[1] *SMIv2 Design Team Home Page*. WWW Page, 1998. `http://www.ibr.cs.tu-bs.de/projects/smi-dt/`.

[2] McCloghrie, K., D. Perkins, J. Schoenwaelder, J. Case, M. Rose und S. Waldbusser: *RFC 2578: Structure of Management Information Version 2 (SMIv2)*, April 1999. See also STD0058. Obsoletes RFC1902 [23]. Status: STANDARD.

[3] McCloghrie, K., D. Perkins, J. Schoenwaelder, J. Case, M. Rose und S. Waldbusser: *RFC 2579: Textual Conventions for SMIv2*, April 1999. See also STD0058. Obsoletes RFC1903 [24]. Status: STANDARD.

[4] McCloghrie, K., D. Perkins, J. Schoenwaelder, J. Case, M. Rose und S. Waldbusser: *RFC 2580: Conformance Statements for SMIv2*, April 1999. See also STD0058. Obsoletes RFC1904 [25]. Status: STANDARD.

[5] Perkins, D.: *SMI Clarification Project*. WWW Page, 1998. `http://www.snmpinfo.com/sismiclr.htm`.

[6] Perkins, D. und E. McGinnis: *Understanding SNMP MIBs*. Prentice Hall, 1997.

[7] Partridge, C. und G. Trewitt: *RFC 1021: High-level Entity Management System (HEMS)*, Oktober 1987. Status: HISTORIC.

[8] Partridge, C. und G. Trewitt: *RFC 1022: High-level Entity Management Protocol (HEMP)*, Oktober 1987. Status: UNKNOWN.

[9] Trewitt, G. und C. Partridge: *RFC 1023: HEMS monitoring and control language*, Oktober 1987. Obsoleted by RFC1076. Status: UNKNOWN.

[10] Partridge, C. und G. Trewitt: *RFC 1024: HEMS variable definitions*, Oktober 1987. Status: UNKNOWN.

[11] *CCITT Recommendation X.208 — ISO/IEC 8824, Information Technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*, 1995.

[12] Davin, J., J. D. Case, M. Fedor und M. L. Schoffstall: *RFC 1028: Simple Gateway Monitoring Protocol*, November 1987. Status: HISTORIC.

[13] Cerf, V. G.: *RFC 1052: IAB recommendations for the development of Internet network management standards*, April 1988. Status: UNKNOWN.

[14] CASE, J. D., M. FEDOR, M. L. SCHOFFSTALL und J. DAVIN: *RFC 1067: Simple Network Management Protocol*, August 1988. Obsoleted by RFC1098. Status: UNKNOWN.

[15] MCCLOGHRIE, K. und M. T. ROSE: *RFC 1065: Structure and Identification of Management Information for TCP/IP-Based Internets*, August 1988. Obsoleted by RFC1155 [17]. Status: UNKNOWN.

[16] MCCLOGHRIE, K. und M. T. ROSE: *RFC 1066: Management Information Base for Network Management of TCP/IP-Based Internets*, August 1988. Obsoleted by RFC1156. Status: UNKNOWN.

[17] ROSE, M. T. und K. MCCLOGHRIE: *RFC 1155: Structure and Identification of Management Information for TCP/IP-Based Internets*, Mai 1990. See also STD0016. Obsoletes RFC1065 [15]. Status: STANDARD.

[18] ROSE, M. T. und K. MCCLOGHRIE: *RFC 1212: Concise MIB definitions*, März 1991. Status: STANDARD.

[19] ROSE, M. T.: *RFC 1215: Convention for defining traps for use with the SNMP*, März 1991. Status: INFORMATIONAL.

[20] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1442: Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)*, April 1993. Obsoleted by RFC1902 [23]. Status: PROPOSED STANDARD.

[21] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1443: Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2)*, April 1993. Obsoleted by RFC1903 [24]. Status: PROPOSED STANDARD.

[22] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1444: Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2)*, April 1993. Obsoleted by RFC1904 [25]. Status: PROPOSED STANDARD.

[23] SNMPV2 WORKING GROUP, J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1902: Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, Januar 1996. Obsoleted by RFC2578 [2]. Status: DRAFT STANDARD.

[24] SNMPV2 WORKING GROUP, J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1903: Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)*, Januar 1996. Obsoleted by RFC2579 [3]. Status: DRAFT STANDARD.

[25] SNMPV2 WORKING GROUP, J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *RFC 1904: Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)*, Januar 1996. Obsoleted by RFC2580 [4]. Status: DRAFT STANDARD.

[26] CERF, V. G.: *RFC 1109: Report of the second Ad Hoc Network Management Review Group*, August 1989. Status: UNKNOWN.

[27] SLOMAN, MORRIS: *Network and Distributed Systems Management*. Addison-Wesley, 1994.

[28] HEGERING, HEINZ-GERD und YECHIAM YEMINI: *Proceedings of the Third International Symposium on Integrated Network Management*, 1993.

[29] *ISO/IEC 7498-1, Information Technology - Open Systems Interconnection - Basic Reference Model - Part 1: The Basic Model*, 1994. 2nd edition.

[30] *International Organization for Standardization.* WWW Page, 1999. `http://www.iso.ch/`.

[31] *CCITT Recommendation X.700 — ISO/IEC 7498-4, Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework*, 1989.

[32] TUCKER, JEREMY: *OSI Structure of Management Information*. In: *Sloman* [27], Seiten 131–164.

[33] *CCITT Recommendation X.720 — ISO/IEC 10165-1, Information Technology - Open Systems Interconnection - Management Information Services - Structure of Management Information: Management Information Model*, 1993.

[34] *CCITT Recommendation X.722 — ISO/IEC 10165-4, Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects*, 1992.

[35] *CCITT Recommendation X.721 — ISO/IEC 10165-2, Information Technology - Open Systems Interconnection - Structure of Management Information: Definition of Management Information*, 1992.

[36] *CCITT Recommendation X.723 — ISO/IEC 10165-5, Information Technology - Open Systems Interconnection - Structure of Management Information: Generic Management Information*, 1994.

[37] *CCITT Recommendation X.724 — ISO/IEC 10165-6, Information Technology - Open Systems Interconnection - Structure of Management Information: Requirements and Guidelines for Implementation Conformance Statement Proformas associated with OSI Management*, 1997.

[38] *CCITT Recommendation X.725 — ISO/IEC 10165-7, Information Technology - Open Systems Interconnection - Structure of Management Information: General Relationship Model*, 1996.

[39] JEFFREE, TONY: *Guidelines for the Definition of Managed Objects*. In: *Sloman* [27], Seiten 95–130.

[40] BÖHMER, OLIVER: *Management Information Model und Guidelines for the Definition of Managed Objects (GDMO)*. Seminarausarbeitung, 1996. `http://www.informatik.th-darmstadt.de/VS/Lehre/SS96/MgmtSem/v4/referat4/gdmo.html`.

[41] ZOTTMANN, HARALD: *GDMO - Guidelines for Definition of Managed Objects*. WWW Page, 1998. `http://home.t-online.de/home/0210452427-0001/telecomm.htm`.

[42] FESTER, OLIVIER und GEORG ZÖRNTLEIN: *Formal Description of Managed Object Behavior — A Rule Based Approach*. In: *INM3* [28], Seiten 45–58.

[43] *ISO/IEC 10164-13, Information Technology - Open Systems Interconnection - Systems Management: Summarization Function*, 1995.

[44] *Common Information Model (CIM) Specification*, 1998.

[45] *Desktop Management Task Force*. WWW Page, 1999. `http://www.dmtf.org/`.

[46] *Desktop Management Interface Specifications 2.0s*, 1998.

[47] MOLL, W.: *Netzwerkmanagement - Die Nächste Generation*. PIK, 21(3):180–182, 1998.

[48] GRADY BOOCH, IVAR JACOBSON, JIM RUMBAUGH: *Unified Modeling Language Resource Center*. WWW Page, 1999. `http://www.rational.com/uml/`.

[49] YERGEAU, F.: *RFC 2279: UTF-8, a transformation format of ISO 10646*, Januar 1998. Obsoletes RFC2044. Status: PROPOSED STANDARD.

[50] CROCKER, D. und P. OVERELL: *RFC 2234: Augmented BNF for Syntax Specifications: ABNF*, November 1997. Status: PROPOSED STANDARD.

[51] LARSEN, AMY K.: *CIM's Missing Pieces*. WWW Page, März 1997. `http://www.data.com/tutorials/cim.html`.

[52] AUREL LAZAR, ROBERTO SARACCO, ROLF STADLER: *Proceedings of the Fifth International Symposium on Integrated Network Management*, 1997.

[53] *Object Management Group*. WWW Page, 1999. `http://www.omg.org/`.

[54] SIEGEL, JON: *CORBA – Fundamentals and Programming*. Wiley Computer Publishing, 1996.

[55] *A Discussion of the Object Management Architecture*. WWW Page, 1997. `http://www.omg.org/library/oma2.html`.

[56] *Object Management Group*. WWW Page, 1999. `http://www.omg.org/library/`.

[57] *Joint Inter-Domain Management*. WWW Page, 1999. `http://www.jidm.org/`.

[58] *Inter-domain Management: Specification Translation*. WWW Page, 1999. `http://www.opengroup.org/onlinepubs/8349099/toc.htm/`.

[59] N. SOUKOUTI, U. HOLLBERG: *Joint Inter Domain Management: CORBA, CMIP and SNMP*. In: *INM5* [52], Seiten 153–164.

[60] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*, 1985.

[61] SUBRATA MAZUMDAR, KEN SWANSON und HELENA SARIN: *WEB Based Management: CORBA/SNMP Gateway Approach*. WWW Page, 1999. `http://nsm.research.bell-labs.com/~mazum/CorbaSnmp/`.

[62] STRAUSS, FRANK: *List of Issues for the SMIng Design*. WWW Page, 1999. `http://www.ibr.cs.tu-bs.de/~strauss/sming/issues.html`.

[63] BRADNER, S.: *RFC 2026: The Internet Standards Process — Revision 3*, Oktober 1996. See also BCP0009. Obsoletes RFC1602. Status: BEST CURRENT PRACTICE.

[64] POSTEL, J. und J. REYNOLDS: *RFC 2223: Instructions to RFC Authors*, Oktober 1997. Status: INFORMATIONAL.

[65] *Network Management Research Group (NMRG)*. WWW Page, 1999. `http://www.ibr.cs.tu-bs.de/projects/nmrg/`.

[66] ALFRED V. AHO, RAVI SETHI und JEFFREY D. ULLMAN: *Compilerbau, Teil 1*. Addison-Wesley, 1994.

[67] SRINIVASAN, R.: *RFC 1832: XDR: External Data Representation Standard*, August 1995. Status: DRAFT STANDARD.

[68] BIERMAN, A.: *RFC 2074: Remote Network Monitoring MIB Protocol Identifiers*, Januar 1997. Status: PROPOSED STANDARD.

[69] PERKINS, D.: *Testing Support of Counters*. WWW Page, 1998. `http://www.snmpinfo.com/CounterTest/index.html`.

[70] SCHÖNWÄLDER, JÜRGEN: *SMIv2 Implementation and Usage Report*. WWW Page, 1999. `http://www.ibr.cs.tu-bs.de/projects/smi-dt/report.txt`.

[71] *ISO 8601:1988, Date/Time Representations*, 1988.

[72] SCHÖNWÄLDER, JÜRGEN: *MOSY MIB Compiler*. FTP Software Source Code Archive, 1998. `ftp://ftp.ibr.cs.tu-bs.de/pub/networking/snmp/compiler/mosy-8.0.0.tar.gz`.

[73] SCHÖNWÄLDER, JÜRGEN und FRANK STRAUSS: *Next Generation Structure of Management Information for the Internet*. April 1999. (submitted for publication at DSOM '99 – Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, ETH Zürich, Switzerland, October 11-13, 1999).

[74] SCHÖNWÄLDER, JÜRGEN: *Scotty - Tcl Extensions for Network Management Applications*. WWW Page, 1998. `http://www.ibr.cs.tu-bs.de/projects/scotty/`.

[75] HARDAKER, WES: *The UCD-SNMP Home Page*. WWW Page, 1998. `http://ucd-snmp.ucdavis.edu/`.

[76] *OMG Formal Documentation*. WWW Page, 1999. `http://www.omg.org/library/specindx.html`.

[77] STRAUSS, FRANK: *SMIng Project Online Documentation*. WWW Page, 1999. `http://www.ibr.cs.tu-bs.de/~strauss/sming/`.

Diskette

Diese Diskette enthält die in Anhang H beschriebene Software. Sollte diese Diskette hier nicht enthalten sein, ist die Software (ggf. auch in neueren Versionen) unter [77] oder beim Autor erhältlich.

Technische Universität Braunschweig **Institut für** **Betriebssysteme und Rechnerverbund** **libsmi** **Eine Bibliothek zum Zugriff auf** **Management-Informationen** **Teil der Diplomarbeit** **SMIng - eine neue Sprache zur** **Definition von Management-Informationen** **cand. inform. Frank Strauß, 10. Mai 1999**