Ludwig-Maximilian-Universität München
Institut für Informatik

Fortgeschrittenen-Praktikum
Entwurf und Implementierung von Managementfunktionen für HP-Dolphin zum Management des LRZ-Systemmanagementagenten

Hans Schlenker

Aufgabensteller: Univ.-Prof. Dr. Heinz-Gerd Hegering
Betreuer: Dipl.-Inform. Stephen Heilbronner

August 1996

Zusammenfassung

Ziel dieses Fortgeschrittenenpraktikums war es, Objekte der *LRZ-Unix-MIB* für den System-Management-Plattform-Prototypen *Dolphin* zu modellieren, um damit via SNMP entsprechende Systeme managen zu können. Dabei stellte sich heraus, daß einerseits Dolphin manches noch nicht kann und andererseits die LRZ-MIB nicht SNMP-konform formuliert ist. Also wurden nur die Teile der MIB modelliert, die trotz der Einschränkungen möglich waren. Außerdem werden hier Verbesserungen vorgeschlagen, um aktuelle Unzulänglichkeiten von Dolphin zu beheben.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dies ist die Dokumentation zum Fortgeschrittenen-Praktikum *Entwurf und Implementierung von Managementfunktionen für HP-Dolphin zum Management des LRZ-Systemmanagementagenten.*

Kapitel 2 beschreibt den Aufbau und die grundlegenden Konzepte von *Dolphin*, einem Prototypen einer System-Management-Plattform[1] von *Hewlett Packard (HP)*. Es ist zum Verständnis dieser Arbeit wichtig und soll dem Leser eine einführende Bedienungsanleitung zu Dolphin geben.

Das nächste Kapitel 3 beschreibt die *Modellierungssprache* von Dolphin. Diese heißt schlicht *Dolphin Model Language*, im folgenden kurz *DML*. Deren Syntax — auch dazu gibt es wichtige Anmerkungen — ist zwar original dokumentiert, die Semantik aber erschließt sich nur indirekt. Deshalb ist das ein wichtiger Teil dieser Arbeit.

In Teil 4 wird kurz die LRZ-Unix-MIB[2] von Uwe Krieger (s. [Kri94]) vorgestellt.

Den *MIB-Compiler*, den Übersetzer einer MIB in ein *Dolphin-Modell*, beschreibt Kapitel 5. Er wird erst dort behandelt, weil erst dann die Grundlagen zu seinem Verständnis vorhanden sind.

Kapitel 6 beschreibt dann die konkrete Implementierung der LRZ-Unix-MIB in Dolphin.

Teil 7 enthält die Zusammenfassung und Ausblick. Außerdem wird da eine Bewertung dieser Arbeit versucht in Bezug auf Schwierigkeiten die dabei entstanden und wie sie gelöst wurden bzw. werden hätten können.

2 Dolphin

In diesem Kapitel sind der Aufbau und die wesentlichen Konzepte von Dolphin beschrieben. Gleichzeitig ist dem Leser hier eine einleitende Bedienungsanleitung gegeben. Keine Berücksichtigung finden Umgebungsbedinungen für Dolphin, vor allem, welche Hard- und Software unterstützt bzw. benötigt werden und wie Dolphin gestartet wird. Hilfreich kann hier [Dzi95] sein.

Dolphin ist ein Prototyp einer System-Management-Plattform. Als solche dient es der Programmierung von Management-Anwendungen zur Bildung eines Management-Systems. Dolphin ist ein Forschungsprojekt der Hewlett-Packard-Laboratories zur Entwicklung und Weiterentwicklung von Konzepten für kommerzielle Management-Produkte von HP. Es ist offenbar nicht für die Öffentlichkeit verfügbar und so gibt es vor allem keine offizielle Dokumentation. So stammen die folgenden Kenntnisse im wesentlichen aus [Fed95] und vom Umgang mit Dolphin.

Dolphin ist fakten- und regelbasiert und programmierbar in der dafür ent-

[1] Zu den Begriffen der System-Management-Welt siehe z.B. [GN95].

[2] Uwe Kriegers Management Information Base (MIB) für Unix-Systeme des Leibniz-Rechenzentrums (LRZ), s. [Kri94]

wickelten Sprache DML[3]. Gemäß dem Logik-Jargon heißen im Dolphin-Sprachgebrauch Management-Anwendungen *Modelle*. In Ihnen sind die Eigenschaften von abstrakten *Objekten* und solchen der „realen Welt“ beschrieben.

Dolphins Kontakt zur Außenwelt passiert entweder über Unix-Shell-Skripten oder über SNMP[4], Modelle können auch beides gemischt verwenden. Hier interessiert uns die Möglichkeit des Managements mit SNMP.

2.1 Komponenten von Dolphin

Laut [Fed95] besteht Dolphin intern aus folgenden Einheiten:

- Model-Store
- Inference-Engine
- Fact Base
- Fixer
- Query-System
- Explanation Facility
- User-Interface

Im *Modell-Speicher* sind die Dolphin-*Modelle* gespeichert, das sind Sammlungen von im wesentlichen *Objekt-Definitionen*, *Querys*[5], *Actions* und *Events*. Dem Modell-Speicher können neue Modelle hinzugefügt werden, indem sie im *Modell-Editor* (s.u. 2.5) angelegt werden oder aus Textdateien geladen werden. Entsprechend können geladene Modelle in Textdateien entladen werden. Diese Textdateien enthalten die Modell-Beschreibungen in DML.

Die *Inferenz-Maschine* benutzt die Definitionen des Modell-Speichers und vorhandene Fakten aus der Faktenbasis, um neue Fakten zu generieren. Wie das genau funktioniert, ist hier nicht so wichtig, der interessierte Leser kann das in [Fed95] nachlesen.

Der *Fixer* behandelt die Aktionen und bedient die Inferenz-Maschine mit passenden *Regeln*[6]. Er kann die Umwelt, das sind die Objekte der realen Welt, die verwaltet werden sollen, verändern. Bei Verwendung von SNMP löst er eine solche Veränderung mit einem *Set-Request*[7] aus.

Auch das *Abfrage-* oder *Query-System* verschafft der Inferenzmaschine Regeln. Benötigt etwa die Benutzerschnittstelle Fakten über Objekte, die der Benutzer gerade untersucht, wählt das Abfrage-System Querys aus, die entsprechende Fakten generieren können.

[3] Dolphin Model Language, s. 1

[4] Simple Network Management Protocol, s. [CDFS90, Ros94]

[5] Der Autor bittet, diese Mischung von deutschen und englischen Begriffen zu entschuldigen. Soweit möglich werden deutsche Übersetzungen der englischen Originale benutzt. Die Original-Begriffe werden aber erwähnt, damit der Leser sich in Dolphin leichter zurechtfinden kann.

[6] s. 3.6

[7] s. [CDFS90, Ros94]

Die *Explanation Facility*, die *Erklärungskomponente*, ist zum einen der *Debugger*, mit dem der Benutzer die Programmierung seiner Modelle überprüfen kann, und zum anderen die Darstellung der *Ableitungen*, die die Inferenzmaschine erzeugt hat. Das heißt eine Darstellung zum Beispiel, warum der zu managende Drucker nicht funktioniert, die Inferenzmaschine also eine Ableitung des Fakts „Drucker ist nicht bereit“ erzeugt hat. Das ist das Analysewerkzeug für den Benutzer.

Das User-Interface enthält:

- System-Manager
- Workspace
- Object-Browser
- Model-Editor
- Debugger

Es folgen Erläuterungen der einzelnen Komponenten, soweit sie für unsere Zwecke nötig sind.

2.2 System-Manager

Der *System-Manager* ist für den Benutzer das zentrale Element. Abbildung 1 zeigt den Systemmanager nach Aufruf von Dolphin.

Neben dem Menü zum Starten und Einstellen anderer Komponenten von Dolphin enthält der System-Manager im oberen Teil (unter dem Menü) ein *Workspace* und im unteren Teil einen *Object-Browser*.

Wie wir in Abbildung 1 sehen, ist direkt nach Aufruf von Dolphin eine vorgegebene Konfiguration gespeichert, in der Regel die gröbste Sicht auf die zu verwaltenden Systeme. Die Abbildung zeigt genaugenommen den Zustand, nachdem der Benutzer von Dolphin das Objekt *RND* geöffnet hat.

In einem Workspace sind Objektdefinitonen der Modelle mit realen Objekten instantiiert. Diese Objekte besitzen Eigenschaften: *Attributes*. Wie wir noch genauer sehen werden, sind Attribute nichts anderes als Relationen. Es sind nicht nur Relationen zwischen einem Objekt und einfachen Werten möglich, sondern auch zwischen Objekten. Also kann ein Objekt durch Attribute zu anderen Objekten in Relation stehen.

Durch Doppelklicken[8] auf ein Objekt wird es geöffnet: es erscheint ein neuer Object-Browser (ein neues Fenster) mit dem Inhalt des Objekts.

Im Menü des System-Managers erreichen wir unter *Model Writers* die Funktionen *Workspace* und *Model-Editor*.

[8] Achtung: Dolphin ist oft umständlich zu bedienen! Hier genügt es nicht, um ein Objekt zu öffnen, es einfach doppelzuklicken. Es muß vorher markiert (also einmal angeklickt gewesen) sein.

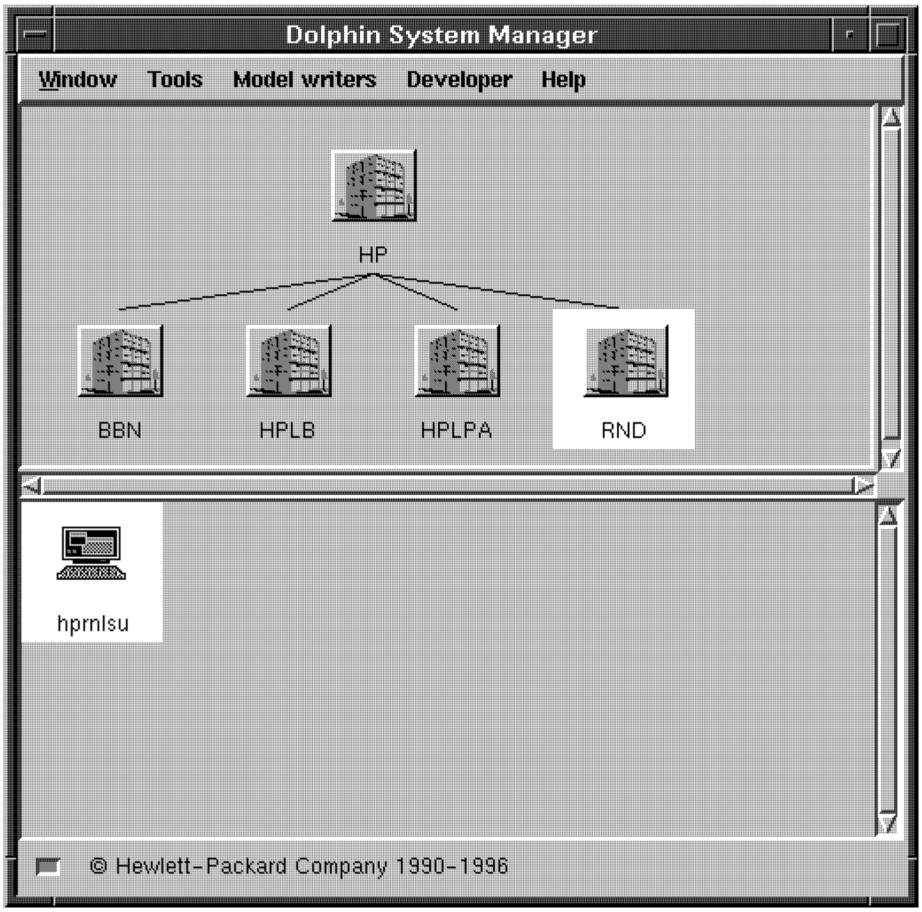

Abbildung 1: System-Manager

2.3 Workspace

Wie gesagt enthält auch der System-Manager ein Workspace.

Mit dem Workspace, das der Benutzer aufrufen kann, kann er neue Instanzen von Objekten erzeugen: im Menü unter *Create*. Hier sehen wir alle Objekte der Art *Manageable Object* und *Manageable Device* der geladenen Modelle. Erzeugen wir ein solches, geben wir ihm einen Namen (in der Regel den DNS-Namen). Im neuen Workspace steht es dann zur Verfügung.

Achtung: Manchmal erscheint ein so instantiiertes Objekt nicht im Workspace. Meist liegt das daran, daß ein Objekt mit demselben Namen schon existiert (z.B. in der Startkonfiguration) oder früher in derselben Sitzung existiert hat (auch, wenn es einen anderen Objekt-Typ hatte). Hier hilft nur (vgl. 2.5), Dolphin zu beenden und neu zu starten.

2.4 Object-Browser

Im Systemmanager haben wir bereits einen Object-Browser gesehen. Immer wenn der Benutzer ein Objekt öffnet, erscheint in einem neuen Fenster ein neuer Object-Browser. Abbildung 2 zeigt einen typischen.

Wir sehen, daß Object-Browser in zwei unterschiedlichen Erscheinungen auftreten:

- Ist das Objekt eine Liste, dann sind die Elemente der Liste der Inhalt des Objekts. Nach Doppelklicken auf ein Objekt *User-List* bekommen wir also die Liste der Einträge. In Abbildung 1 ist das Objekt *RND* eine Liste mit dem einzigen Eintrag *hprnlsu*.

- Alle anderen Objekte haben als Inhalt ihre Attribute mit den Werten. In der Regel sind die Attribute gruppiert. Im Objektbrowser erscheint jede Gruppe als Blatt mit „Reiter": in Abbildung 2 die Blätter *default*, *policies*, *System* und *Table*. In unserem Beispiel sehen wir das Blatt *System* mit seinen Attributen *sysClockTicks* (mit dem Wert *60*), *sysContact* (mit dem Wert *Markus Gutschmidt*) usw.

Im Menü eines Object-Browsers kann die Darstellung erneuert werden: *Update facts* passt die Darstellung den vorhandenen Fakten an und *Forget and update facts* erneuert die Faktenbasis durch Aufruf von Abfragen (Querys) und dann die Darstellung. Durch *Attribute - inspect* wird ein Attribut geöffnet. Das ist dasselbe wie Doppelklicken eines Attributs. Ist das Attribut wieder ein Objekt, öffnet sich ein neuer Object-Browser.

Einfache Attribute werden in der Regel durch Kombinationen dargestellt[9]: *Name Inhalt*. Objekten können *Icons* zugewiesen werden. Wird ein Attribut rot hinterlegt dargestellt, heißt das: „Fehler, kein Fakt vorhanden". Wird ein solches geöffnet, öffnet sich die Erklärungskomponente: die mögliche Ableitung für den Fakt des Attributs wird dargestellt, wobei die Stellen, die fehlgeschlagen sind,

[9] Eine Erweiterung sieht DML vor: der Wert eines Attributs kann in einen natürlichsprachlichen Satz eingebettet werden; s. 3.

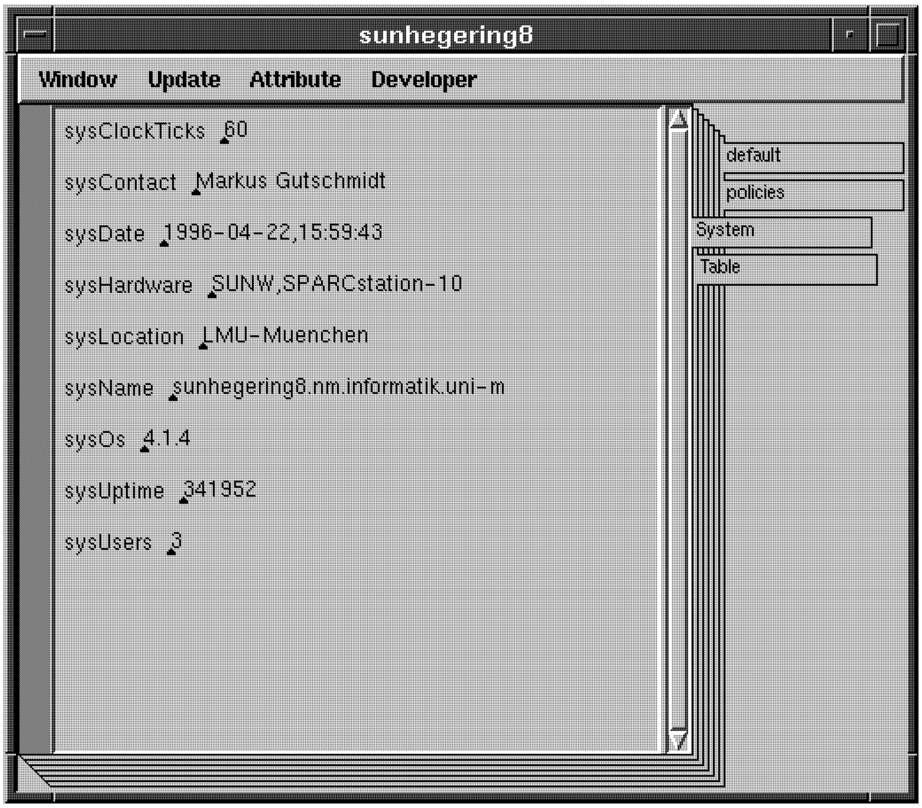

Abbildung 2: Object-Browser

markiert sind. In dem Modell dieser Arbeit wird das aber nicht auftreten. Dieser Mechanismus ist gedacht zur Fehleranalyse: Abwesenheit eines Fakts „Drucker O.K.“ ist ein Fehler im zu verwaltenden System. An der fehlgeschlagenen Ableitung läßt sich dann ablesen, was dem Drucker fehlt. Siehe hierzu 3 und für ausführliche Beispiele [Fed95].

Der Object-Browser ist auch der Ort, an dem der Systemverwalter in seine zu verwaltenden Systeme eingreift: er kann u.a. Attribute ändern. Konkret muß er dazu eins anklicken und den dafür angezeigten Wert über die Tastatur ändern (das macht natürlich nur für änderbare Attribute Sinn). Die Änderung löst eine Aktion aus, sofern eine passende vorhanden ist (s. 3.8), die dann die Umwelt beeinflusst.

Obwohl nach Ausführung der Aktion die Umwelt geändert ist, ist es die Darstellung im Object-Browser noch nicht. Sie muß mit *Update facts* oder *Forget and update facts* erneuert werden.

2.5 Model-Editor

Im Modell-Editor hat der Benutzer Zugriff auf geladene Dolphin-Modelle, kann diese Verändern oder neue erstellen oder neue laden. Aufgerufen wird er wie oben erwähnt aus dem Menü des Systemmanagers.

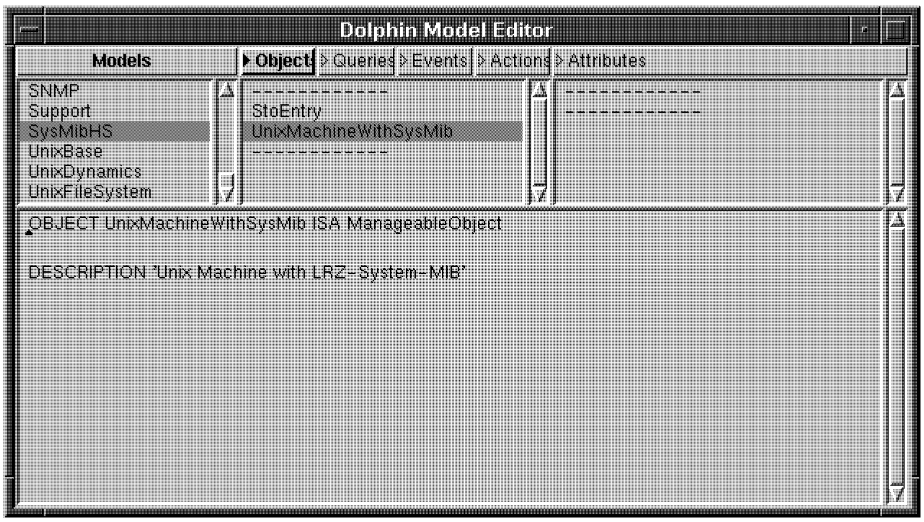

Abbildung 3: Model-Editor

Abbildung 3 zeigt einen typischen Zustand des Modell-Editors. Links oben sehen wir die Liste der geladenen Modelle. Oben in der Mitte die Liste der Objekte des gewählten Modells und rechts seine Attribute. Das Fenster unten zeigt immer die allgemeinste Definition der ausgewählten Modelle, Objekte usw. in DML.

Ist nichts ausgewählt, d.h. kein Modell und kein Objekt/Abfrage/... und kein Attribut, zeigt das untere Fenster ein allgemeines Gerüst[10] für Modelle. Um ein neues zu erzeugen, kann der Benutzer hier beginnen und das Gerüst mit eigenen Definitionen füllen.

In der oberen Leiste des mittleren Fensters wird entschieden, ob dieses Objekte, Abfragen, Ereignise oder Aktionen anzeigen soll. Wie wir schon wissen, besteht ein Modell ja aus diesen.

Wir wählen im linken Fenster ein Objekt aus (durch einfaches Anklicken). Klicken wir in der Auswahlleiste des mittleren Fensters auf *Objects*, erhalten wir die Liste der Objekte des Modells und unten das allgemeine DML-Gerüst für Objekte. Entsprechend nach Wahl von *Querys*, *Events* oder *Actions*.

Ist ein Modell und ein Objekt augewählt und klicken wir auf *Attributes*, sehen wir hier die Liste der Attribute und unten das DML-Gerüst, das direkt editiert werden kann. Hier zeigt das lokale Menü (nach Klicken mit der rechten Maustaste) unter *find entity* die Werte:

- Char
- Integer
- Pathname
- Set
- SortedCollection
- String

Diese Begriffe stehen für die verfügbaren Datentypen, die eigentlich für die Definition von Variablen gedacht ist. Sie können aber nicht einfach in DML-Text eingetragen werden (obwohl der Model-Editor das technisch zuläßt), da Dolphin als Datentyp (syntaktisch: *varSpec*) nur kleingeschriebene (!) zuläßt.

Übrigens: eine gute Idee ist hier wie überhaupt überall in Dolphin, eine Aktion, die keine Reaktion zeigt, einfach nochmal anzustoßen oder Maus-Aktionen mit gleichzeitigem Drücken von Shift oder Ctrl oder ähnlichem zu verbinden. Oft hakt nämlich die Bedienung.

Außerdem ist die einzige wirklich sichere Methode, Änderungen in Dolphin wirksam werden zu lassen, diese zu speichern (hier also das veränderte Modell in eine Datei zu entladen), Dolphin zu beenden, neu zu starten und die Veränderungen wieder zu laden.

Mit der rechten Maustaste ins linke Fenster geklickt, erhält man ein Menü mit Befehlen für's Laden von Modellen, Speichern und ähnliches.

Noch ein Hilfsmittel steht uns im Modell-Editor zur Verfügung: der *Objekt-Hierarchie-Baum*. Den bekommt man, indem man ein Modell auswählt, dann ein Objekt und dann mit der mittleren Maustaste darauf klickt[11].

[10] Dieses Gerüst folgt nicht exakt der Syntax-Definition! Die Syntax, die der Modell-Editor vorgibt, entspricht aber schon genauer der „tatsächlichen“ DML-Syntax (s. 3).

[11] Im Baum selbst ist nie klar, wann beim Drücken der mittleren Maustaste das *lokale Menü flip* und wann *select / spawn* erscheint. Weder eine Kombination mit Tasten (Shift, CTRL) noch eine Position des Mauszeigers garantieren das eine Menü oder das andere.

2.6 Debugger

Mit dem Debugger kann der Entwickler seine Modelle testen. Bei vorliegender Arbeit kam er allerdings kaum zur Anwendung.

Um den Debugger zu aktivieren, muß man ihn im Menü *Developer* des Systemmanagers für die gewünschten Abfragen, Ereignisse und Aktionen einschalten. Er öffnet sich dann bei deren Ausführung und bietet typische Verfahren wie schrittweises Ausführen oder Anzeigen der generierten Fakten an.

3 Die Modellierungssprache

Von Adrian Pell, einem Entwickler von Dolphin, gibt es eine Definition der Syntax seiner Dolphin Model Language (DML[12]): [Pel]. Anhang A gibt diese komplett wieder.

Tatsächlich existieren aber drei verschiedene Syntaxen: die der Definition, die des Modell-Editors und die tatsächliche.

Die Syntax des Modell-Editors ist die, die er für die automatisch angezeigten Gerüste benutzt.

Die Sprache, in der die originalen (Beispiel-)Modelle[13] Dolphins geschrieben sind, nennen wir die „tatsächliche“ Syntax. Sie ist scheinbar identisch der Syntax, die Dolphin verwendet, um geladene Modelle in Textdateien zu entladen[14].

Wie gesagt ist die Semantik von DML undokumentiert. Allerdings gibt es zu Dolphin schon einige fertig formulierte Modelle. Daraus und vor allem aus [Fed95] erschließt sich erst die Bedeutung der Konstrukte von DML. Damit wenigstens einige davon beschrieben sind, folgen einige Anmerkungen. Sie legen aber keinen Wert auf Vollständigkeit.

3.1 `OBJECT Objekt`

Dieses Kontrukt definiert die Objekte. Das sind die Bausteine, aus denen ein Modell zusammengesetzt ist. Verglichen mit SNMP entsprechen diese Objekte ziemlich genau den SNMP-Objekten.

Objekte lassen sich vererben (`OBJECT ObjektA ISA ObjektB`) und ineinander einbetten (`OBJECT ObjektA INA ObjektB`).

[12]Um es nochmal zu sagen (s. 1): die Abkürzung *DML* ist keine offizielle; sie stammt vom Autor.

[13]Diese Modelle sind als Textdateien und nach Aufruf von Dolphin intern geladen vorhanden.

[14]Trotzdem gibt es Probleme, entladene Modelle wieder zu laden: nach Entladen aller originalen Modelle (im Modell-Editor *file-out-all*) und neuem Laden (durch Laden des Modells allmodels.m) gibt es Warnungen der Art: `Model ManageableObject is VERY DODGY because Object DomainPolicy is VERY DODGY because Basic Attribute [ domainPolicy ]name[ String _ ]inDomainNamed[ String string ] is VERY DODGY because Argument Underscore use in [ String _ ] is deprecated`. Diese können aber einfach übergangen werden.

3.2 `ASPECT OF OBJECT Objekt`

Will man in einem Modell ein Objekt eines anderen Modells erweitern, kann man das mit dieser Konstruktion. Sie ist syntaktisch identisch der normalen Objektdefinition `OBJECT Objekt`, nur muß `Objekt` ein bereits vorhandenes Objekt sein, in der Regel aus einem anderen Modell.

Außerdem sind hier die Hierarchiebeziehungen `ISA` und `INA` nicht erlaubt.

3.3 `LOCATION` bzw. `INA`

Die Dokumentation nennt die zweite Vererbungsbeziehung neben `ISA`: `LOCATION`. In Dolphin selbst aber heißt diese `INA`. Beide meinen dasselbe, nämlich die Einbettung eines Objekts in ein anderes.

3.4 Attribute

Attribute sind Relationen von Objekten zu Werten einfachen Datentyps oder zu anderen Objekten. Es sind ein- und mehrstellige Relationen möglich. Für jedes mehrstellige Attribut wird festgelegt, ob es sich um eine 1:1-, 1:N- oder N:1-Beziehung handelt.

`BASIC` und `DERIVED` sind die einzigen Attribut-Arten, die der Modell-Editor kennt. Die Spezifikation gibt dazu noch `METHOD`, `COMMAND` und `VIRTUAL` an, die Dolphin allerdings versteht, also tatsächliche Syntax sind. Umgekehrt schlägt der Model-Editor vor, Attribute mit den Konstrukten `DEFAULT`, `POLICY` und `CATEGORY` genauer zu beschreiben. Diese Möglichkeiten fehlen in der Spezifikation.

3.5 `CATEGORY`

Das Konstrukt `CATEGORY` als Zusatz zu einer Attribut-Definition benennt zu dem Attribut die Kategorie, unter der es im Object-Browser geführt wird.

3.6 `DERIVED`

Attribute der Art `DERIVED` sind von anderen Attributen abgeleitete und als solche Regeln. Eine Definition

`DERIVED` Attribut `IF` A1 `&` A2

erzeugt ein neues Attribut, das dann „gilt“, für das also ein Fakt erzeugt wird, falls A1 und A2 gelten. In der Regel sind A1 und A2 Attribute und das heißt dann, daß A1 und A2 vorhanden sein müssen.

Das sind die oben erwähnten Regeln, die die Inferenzmaschine benutzt, um abstrakte Fakten (wie „Der Rechner A reagiert nicht.“) abzuleiten.

3.7 Abfragen (`QUERY`)

Die Abfragen dienen Dolphin der Erzeugung von Fakten auf die sich alles stützt. Sie stellen den lesenden Zugriff auf die Umwelt dar.

Die *Scanning Directives* (das nicht-terminale Symbol *scanningDirective* der DML-Definition) dienen in *Productions*, das sind Erzeuger von Fakten aus anderen Daten, neben anderen dazu, die Ursprungsdaten zu zerlegen. Diese recht wichtigen Konstrukte sind leider nirgends definiert. In der Syntax-Definition heißt es nur: `scanningDirective ::= uppercaseWord`.

3.8 Aktionen (`ACTION`)

Aktionen werden vor allem durch eine Kombination von Vor-Bedingung und Nach-Bedingung und durch die auszuführende Aktion definiert. Kennzeichen dafür sind die terminalen Symbole `PRE` bzw. `POST` (s. Anhang A).

Eine Aktion wird ausgeführt, wenn die Vorbedingung zutrifft. Eine Aktion ohne Vorbedingung (diese ist nicht notwendig) wird ausgeführt, wenn die Nachbedingung nicht zutrifft. Siehe auch 2.4.

3.9 Kommentare

Die Syntax-Definition gibt keine Möglichkeit der Kommentierung an. Originale Beispiel-Modelle aber zeigen: Kommentare kann man in Dolphin-Modelle eintragen, indem man sie an (fast) beliebiger Position im Modell-Text in doppelte Hochkommas einschließt: `"Kommentar"`. Auch über mehrere Zeilen hinweg!

Aber: ganz am Ende eines Modells scheinen sie nicht erlaubt.

4 Die LRZ-Unix-MIB

Die LRZ-Unix-MIB, die Uwe Krieger in seiner Diplomarbeit [Kri94] entworfen hat, ist wie gesagt das, was hier für Dolphin implementiert wurde. Die Motivationen, die zu der Diplomarbeit führten, und die dort verwirklichten Konzepte werden in [Kri94] dargestellt.

Uwe Krieger entwirft für seine System-MIB[15] folgendes „Klassifikationsschema“, wie er es nennt. Es unterteilt die Informationen der MIB in folgende Gruppen[16]:

- System
- Storage
- Device
  mit Untergruppen, v.a.

[15] *System-MIB*, *LRZ-Unix-MIB* und auch *Unix-LRZ-MIB* sind alles geläufige Begriffe für dasselbe. Je nachdem, in welchem Kontext von der MIB gesprochen wird, wird einer der Begriffe benutzt, in dieser Arbeit die ersten beiden.

[16] In der LRZ-Unix-MIB heißen Objekte eines hohen Abstraktionsniveaus Gruppen. Der Autor Uwe Krieger folgt damit wohl einem üblichen Sprachgebrauch.

  - Processor
  - Printer
  - Disk
  - Partition
  - Filesystem
  - Network
- Process
- User

Die System-MIB (oder vielmehr die Gruppe) hat die SNMP-Adresse: .iso.org.dod.internet.experimental.lrz.lrz-unix (.1.3.6.1.3.100.2). Darunter liegen die Gruppen *system 1* bis *user 5*.

Leider ist die Definition der MIB nicht SNMP-konform insofern als sie Tabellen in Tabellen enthält. So existieren in der Definition der System-MIB Konstruktionen wie z.B.:

```
... SYNTAX SEQUENCE OF DeviceEntry ...
DeviceEntry ::= SEQUENCE ...
```

Eine Idee zur Lösung dieses Problems war, diese Tabellen zu denormalisieren[17]. Allerdings implementiert der Agent die System-MIB korrekt: mit Werkzeugen wie *snmpwalk* lässt sie sich problemlos auslesen.

Genau die Tabellen mit Untertabellen waren aber in Dolphin nicht zu implementieren. Deshalb musste sich diese Arbeit auf die beschränken, die im folgenden beschrieben sind.

4.1 System Group

Die Gruppe *System* enthält folgende Attribute: *sysName*, *sysContact*, *sysLocation*, *sysOs*, *sysHardware*, *sysUptime*, *sysDate*, *sysUsers* und *sysClockTicks*. Das sind grundlegende Informationen zum System.

Diese Gruppe enthält auch die — für diese Arbeit einzigen — schreibbaren Attribute: *sysContact,sysLocation* und *sysDate*. Alle anderen sind nur lesbar.

4.2 Storage Group

Die *Storage*-Gruppe ist eine Tabelle mit folgenden Einträgen: *stoType*, *stoDescr*, *stoAllocationUnits*, *stoSize*, *stoUsed* und *stoState*.

Diese Gruppe beschreibt die Speicher eines Systems. Alle Attribute sind nur lesbar.

[17] Ein Begriff aus der Datenbankwelt: eine Tabelle normalisieren heißt dort sie in mehrere Tabellen aufzuteilen um u.a. Datenredundanzen zu beseitigen.

4.3 Disk Group

Die Gruppe *Disk*, eine Untergruppe von *Device*, hat folgende Attribute: *diskAccess*, *diskMedia*, *diskRemoveable*, *diskCapacity*, *diskState*, *diskLabel*, *diskBusy*, *diskReads*, *diskXfer*, *diskXferRate* und *diskWords*.

Die Gruppe beschreibt die Massenspeicher des Systems. Alle Attribute sind nur lesbar.

5 MIB-Compiler

Seit einiger Zeit existiert ein Übersetzer für MIBs in Dolphin-Modelle. Hierfür gibt es eine kurze Dokumentation [Riv].

5.1 Aufruf

Aufgerufen wird der Compiler im *System-Manager* im Menü *Model Writers* - *MIB Compiler*. Danach ist die MIB-Datei anzugeben. Sollte beim Import kein Fehler auftreten[18], wird der Inhalt der MIB in einem neuen Fenster *MIB File Structure* gezeigt, und zwar als Baum: Abbildung 4.

```
SNMP MIB FILE --+-- IDENTIFIER --+-- NAME: SNMPv1-LRZ
                |                |
                |                +-- TREE STRUCTURE
                |
                +-- DEFINITION --+-- IMPORTS
                                 |
                                 +-- GROUPS & ATTRIBUTES
```

Abbildung 4: MIB File Structure

So sieht dieser Baum bis auf *NAME* immer aus. Ein Klick mit der mittleren Maustaste auf einen Knoten[19] bringt ein[20] Menü mit einem Eintrag: *explain*. Damit wird eine Erklärung des Knotens angefordert. Wichtig fuer den Import der MIB nach Dolphin ist der Knoten *Groups & Attributes*.

Ruft man für *Groups & Attributes* den Befehl *explain* auf (im „Kontextmenü“, dem Menü, das die mittlere Maustaste bringt), öffnet sich ein weiteres Fenster mit nur diesem Objekt. Dieses wird wieder im Kontextmenü durch den Befehl *expand all groups* komplett dargestellt. Das heißt, daß damit die komplette Hierarchie der MIB sichtbar wird. Mit dem Befehl *select* für die Wurzel des Baums oder irgend einen anderen Knoten kann man Teilbäume für den

[18] Tritt ein Fehler auf, gibts als Meldung leider nur ein schlichtes „Error“.

[19] Ein Klick auf den Hintergrund bringt den Befehl *flip* mit dem man den Inhalt des Fensters kippen kann.

[20] Leider nur mit etwas Glück: die Bedienung ist hier wie im *Objekt-Hierarchie-Baum*, siehe oben 2.5, nicht ganz einfach

tatsächlichen Import auswählen. Bis hierher wurde also noch nichts in ein Modell übersetzt oder gar nach Dolphin importiert[21].

Nachdem also ein (Teil-)Baum ausgewählt wurde, kann dieser mit *import* — dieser Befehl ist aus dem Kontextmenü des Fensterhintergrunds also nicht eines Knotens! — importiert werden.

Existiert in dem laufenden Dolphin-Image[22] ein Modell gleichen Namens, fragt Dolphin nach, ob dieses überschrieben werden soll. Ansonsten wird importiert.

5.2 Beim Übersetzen der LRZ-Sysmib

gibt's das Problem daß der Compiler einige Datentypen nicht kennt und nach entsprechenden Typen aus seinem Repertoire fragt. Das waren im einzelnen *DateAndTime*, *KBytes* und *Bytes*.

Wenn man für *KBytes* und *Bytes* den Dolphin-Typ *Integer* und für *DateAndTime* den Typ *String* verwendet, läßt sich die MIB importieren.

Eigenartigerweise werden nur die Untergruppen zu *lrz-unix* importiert: *device, system, user* usw. *lrz* und *lrz-unix* tauchen in dem ganzen Modell nicht auf, obwohl sie mit ausgewählt sind. Es scheint auch egal zu sein, in welcher Höhe ein Teilbaum selektiert und importiert wird. SNMP-Gruppen werden scheinbar erst (im Sinne von oben nach unten im Baum) importiert, wenn sie Attribute besitzen. Aus einer Gruppe wird dann ein *ManageableDevice*!

Oben unter 4 wurde das Problem der verschachtelten Tabellen erwähnt. Der Compiler hat damit zunächst keine Probleme. Will man ein solches Objekt aber dann im Workspace instantiieren, gibt es Fehlermeldungen wie hier beim Objekt *user*:

```
the preconditions for query Domain::subDomains are non-functional!
the offending expression is:
          [Domain_] name [parentName]
the undeterminable variable is: Domain_

the preconditions for query ComputerIP::traceroute are non-functional!
the offending expression is:
          [m] name [name]
the undeterminable variable is: m
```

5.3 Fehler der Übersetzung

Das vom Compiler generierte Modell war so leider nicht benutzbar. Zum Teil mag das daran liegen, daß eine Übersetzung einer MIB in eine Management-

[21] Genauer gesagt scheint der Compiler schon mindestens zwei Prüfungen der Quelldatei durchgeführt zu haben: unmittelbar nach dem Start (hier kann die schlichte „Error"-Meldung erscheinen, die nicht auf so Triviales wie *falscher Dateiname* oder ähnliches zurückzuführen ist) und nach dem *select* eines Knotens: siehe hierzu auch weiter unten.

[22] Das Image ist der aktuelle Zustand des Programms. Dieser kann komplett gesichert werden.

Anwendung prinzipiell nicht automatisch gehen kann. Stephen Heilbronner ist da der wohlbegründeten Ansicht, daß eine Management-Anwendung Bedeutungen benötigt, die sich nicht alleine aus der MIB ablesen lassen.

Doch scheitert die automatische Übersetzung in unserem Fall schon früher. Der Compiler erzeugt sogar DML-Code, der syntaktisch inkorrekt ist. Aktuell macht er dabei mindestens folgende Fehler:

1. In der MIB hat jede Tabelle ein Index-Feld (z.B. stoIndex). Diese sind aber durchweg mit `ACCESS not-accessible` gekennzeichnet und der Agent erzeugt dazu keine Werte. Der MIB-Compiler aber legt ein entsprechendes Attribut an und die dazugehörende Query versucht das Feld zu lesen.

2. In Abfragen für Tabellen wird die Reihenfolge der Attribute festgelegt, wie sie der Agent liefert (s. B). Bei den automatisch erzeugten Querys ist die Reihenfolge genau verkehrt.

3. Das nichtterminale Symbol *productionRHS* der DML-Definition ist wie folgt festgelegt:

   ```
   productionRHS ::=
       andProduction [ ('|' andProduction)* ]
   ```

   Zum Teil aber generiert der Übersetzer Konstrukte der Form *productionRHS* mit einem zusätzlichen Zeichen `|` am Schluß.

Außerdem könnte der Compiler der MIB-Definition genauer genügen. Wie oben bemerkt, wird die Struktur der MIB für sehr abstrakte Objekte recht vage wiedergegeben: einige Objekte tauchen gar nicht auf und die erzeugten Objekte sind kaum verschachtelt. Dazu müsste aber wohl zuerst die Einschränkung zur unten (6) nochmal erwähnten Beziehung `INA` aufgehoben sein.

6 Implementierung

Nachdem das automatisch erzeugte Modell keine sofort brauchbaren Resultate lieferte (die Objekte liessen sich entweder gar nicht in einem Workspace benutzen oder die Einträge waren unbrauchbar), musste von Hand implementiert werden. Tatsächlich genügte es aber, das übersetzte Modell zu reduzieren und umzuschreiben.

Eine direkte Umsetzung des vom Compiler erzeugten Codes wäre (für die Gruppe *Storage*)[23]:

```
OBJECT UnixMachineWithSysMib
...
OBJECT Storage INA UnixMachineWithSysMib
```

[23] Tatsächlich erzeugt der MIB-Compiler kein Objekt *UnixMachineWithSysMib*. Das wurde von Hand aus dem automatisch generierten Objekt *System* gemacht.

```
...
OBJECT StoEntry INA Storage
...
```

Das aber verbietet die Modellierungssprache Dolphins: für die Zukunft zwar angekündigt, bis heute aber noch unerlaubt, sind verschachtelte `INA`-Beziehungen.

Deshalb muss eine Ebene weggelassen werden:

```
OBJECT UnixMachineWithSysMib
...
OBJECT StoEntry INA UnixMachineWithSysMib
...
```

Eine Beziehung `ObjA INA ObjB` fügt dem ObjB das ObjA aber noch nicht als Attribut hinzu. Wir erreichen die obige Hierarchie mit folgender Konstruktion:

```
OBJECT UnixMachineWithSysMib

...

[unixMachineWithSysMib] Storage [unixMachineWithSysMib : stoEntry] (0:M)
.

...
```

Der Leser kann die komplette Implementierung in Anhang B sehen.

Für die Gruppe *disk* muss noch eine Ebene eliminiert werden. Statt

```
OBJECT UnixMachineWithSysMib
...
OBJECT Device INA UnixMachineWithSysMib
...
OBJECT Disk INA Device
...
OBJECT DiskEntry INA Disk
...
```

also

```
OBJECT UnixMachineWithSysMib
...
OBJECT DiskEntry INA UnixMachineWithSysMib
...
```

Die Hierarchie der MIB kann aber für den Benutzer mit dem oben erwähnten `CATEGORY`-Konstrukt nachgebildet werden: das Attribut *Strorage* bekommt die Kategorie *Storage* und das Attribut *Disk* die Kategorie *Device*. Unter diesen Kategorien sind sie dann im Object-Browser sichtbar.

Außerdem benennen wir die Objekte leicht um: aus dem Objekt *Storage* machen wir eine *StorageTable*, aus dem Objekt *Disk* eine *DiskTable*. Diese Namen werden ja im Object-Browser angezeigt und entsprechen der Bedeutung der Objekte in diesem Kontext genauer.

Die Abfragen wurden den automatisch erzeugten nachgebildet und entsprechend den obigen Bemerkungen (5.3) angepasst.

Die Modellierung aller anderen Objekte erwies sich leider als unmöglich. Zum Teil liegt das wohl daran, daß Dolphins SNMP-System damit Probleme hat und zum anderen an fehlender Dokumentation.

7 Zusammenfassung und Ausblick

Im Vordergrund dieses Fortgeschrittenenpraktikums standen die Probleme. Diese nochmal zusammengefasst:

1. Die Managementplattform Dolphin ist leider im wesentlichen undokumentiert. Folglich sind Erfolge nur durch Versuch und Irrtum zu erreichen.

2. Die Bedienung des Systems ist enorm zeitraubend. Wie gesehen reagiert Dolphin einerseits oft unbestimmt[24] und andererseits ist es im Prinzip notwendig, nach Änderungen an Implementierungen oder auch nur an Instantiierungen Dolphin neu zu starten.

3. Manches, was Dolphin vorgibt zu können (das scheint vor allem im Zusammenhang mit SNMP zu gelten), kann es nicht oder ist noch nicht implementiert. Auch das wurde wohl deutlich.

Diese drei Punkte hängen auf eine fatale Art stark zusammen. Dadurch kann die Verwirklichung eines Dolphin-Projekts leicht zur Katastrophe werden.

Anfangs wurde diese Arbeit von den Entwicklern unterstützt. Dafür möchte ich ihnen an dieser Stelle danken. Die Unterstützung brachte mir meist wertvolle Kenntnisse.

Leider brach diese später ab. Ich muß daraus schließen, daß durch die Probleme, die bei dieser Arbeit auftauchten, Schwächen von Dolphin deutlich wurden, die nicht zu beheben waren. Jedenfalls konnte ohne diese Unterstützung die Arbeit nicht im ursprünglich geplanten Umfang vollendet werden.

Insgesamt aber ist Dolphin durchaus vielversprechend. Sieht man sich die vorhandenen Implementierungen an[25], die fast sämtlich aus den Dolphin-Labors

[24] Anmerkungen im Text wie *meistens* oder *wenn man Glück hat* beziehen sich auf die Erfahrungen des Autors mit dem System, d.h. den Fehlern des Systems. Er möchte das nicht als Kritik verstanden wissen, schließlich ist Dolphin aktuell ein Forschungsprojekt, sondern vielmehr klar machen, daß Probleme mit der Bedienung nicht unbedingt am Benutzer liegen sondern oft am System. Natürlich kann es sein, daß der Leser mit einer neueren Dolphin-Version arbeitet und die hier beschriebenen Probleme nicht hat. Die Entwickler von Dolphin wissen von den Unzulänglichkeiten.

[25] Die vorhandenen Implementierungen kommen aber fast ohne SNMP aus.

stammen[26], bekommt man einen erstaunlichen Eindruck von den Fähigkeiten Dolphins.

Nachdem also davon ausgegangen werden muß, daß Dolphin, was System-Management mit SNMP anbelangt, nicht mehr kann, als hier benutzt wurde, ist leider keine vollständige Implementierung der LRZ-Unix-MIB in Aussicht. Wie das bei Forschungsprojekten üblich ist, kann sich das aber sehr schnell ändern.

Abschließend möchte ich noch eine persönliche Bemerkung machen, vor allem für die, die in Zukunft Fortgeschrittenpraktika oder Diplomarbeiten machen. Der größte Fehler, den ich bei dieser Arbeit gemacht habe, war, daß ich Anfangs versucht habe, alles alleine zu machen. Erst der Kontakt mit dem Betreuer und den Entwicklern brachte einen Fortschritt. Deshalb denke ich, daß bei solchen Arbeiten es sehr wichtig ist, zu möglichst vielen kompetenten Leuten Kontakt zu haben. Und jeder, der schon irgendwas in Richtung der Arbeit gemacht hat, ist kompetenter als man selbst.

Schließlich möchte ich noch meinem Betreuer, Herrn Stephen Heilbronner vom Lehrstuhl Hegering der Ludwig-Maximilian-Universtät München, für seine Unterstützung danken. Die zu wenigen Male, die ich sie in Anspruch nahm, war sie immer eine Hilfe.

[26]Eine Ausnahme bildet da das schon erwähnte Fortgeschrittenenpraktikum von Micheal Dzik: [Dzi95].

A Syntax von DML

Der Autor hat für diese Arbeit die Definition von A. Pell in eine HTML-Version übersetzt. Die Definition ist in einer erweiterten BNF-Notation formuliert. Jedes terminale Symbol ist am Schluß aufgelistet und mit einem Verweis auf die Regel, in der es erscheint, versehen. Damit kann man (in einem geeigneten HTML-Browser) mit einem Mausklick vom terminalen Symbol zur Regel springen. Genauso ist jedes nicht-terminale Symbol mit seiner Definition verbunden. So kann ein gegebener DML-Text leichter analysiert werden.

Die HTML-version ist das Ergebnis einer LaTeX2html[27]-Datei die wiederum automatisch von einem Prolog-Programm erzeugt wurde, das die erweiterte BNF-Notation lesen kann. Das Prolog-Programm stammt von mir. Wer Interesse hat, kann es jederzeit gerne bekommen.

Es folgt das Ergebnis der Übersetzung. Die Regeln sind in der originalen Reihenfolge. Die letzte Regel ist gewissermaßen die Wurzel: sie beschreibt, den Aufbau eines ganzen Modells (wholeModel).

modelName ::=
uppercaseWord

objectName ::=
uppercaseWord

attributeNamePart ::=
lowercaseWord

variableName ::=
lowercaseWord

variableWildCard ::=
'`_`'

attributeUse ::=
variableSpecifier (attributeNamePart variableSpecifier) * [attributeNamePart]

qualifiedObjectName ::=
modelName '`.`' objectName

modelSpec ::=
'`MODEL`' modelName [imports] [modelDescription]

imports ::=
'`IMPORT`' modelName +

[27] LaTeX2html ist der Generator von Nikos Drakos (s. [Dra]), der aus LaTeX-Dokumenten HTML-Dokumente compiliert.

modelDescription ::=
'`DESCRIPTION`' string

objectSpec ::=
coreObject |
nonCoreObject |
aspectObject

coreObject ::=
'`CORE`' '`OBJECT`' objectName '`BUILT`' '`FROM`' smalltalkClassName

nonCoreObject ::=
'`OBJECT`' objectName [objectHierarchy] [objectLocation]

aspectObject ::=
'`ASPECT`' '`OF`' '`OBJECT`' objectName

smalltalkClassName ::=
uppercaseWord

objectHierarchy ::=
'`ISA`' objectName

objectLocation ::=
'`LOCATION`' objectName

attributeSpec ::=
(basicAttribute |
derivedAttribute |
methodAttribute |
commandAttribute |
virtualAttribute) [attributeExplanation]

attributeExplanation ::=
'`EXPLANATION`' (string |
variableDeclaration |
trueFalsePart) +

trueFalsePart ::=
'`(`' string '`|`' string '`)`'

attributeHead ::=
variableDeclaration (attributeNamePart variableDeclaration) * [attributeNamePart]

attributeCardinality ::=
'`(`' attributeCardinalValue ('`:`' attributeCardinalValue) * '`)`'

attributeCardinalValue ::=

'O' |
'M'

attributeModes ::=
'(' attributeModeValue + ')'

attributeModeValue ::=
'+' |
'?'

basicAttribute ::=
'BASIC' attributeHead attributeCardinality

derivedAttribute ::=
'DERIVED' attributeHead attributeCardinality [attributeModes] 'IF' attribute-Body

attributeBody ::=
ruleExpression

virtualAttribute ::=
'VIRTUAL' attributeHead attributeCardinality [attributeModes]

commandAttribute ::=
'UI' attributeHead attributeCardinality [attributeModes] 'IF' attributeBody

methodAttribute ::=
'METHOD' attributeHead (attributeModes smalltalkMethodName) +

smalltalkMethodName ::=
string

ruleExpression ::=
ruleAndExpression ('|' ruleAndExpression) *

ruleAndExpression ::=
ruleExpressionElement ('&' ruleExpressionElement) *

ruleExpressionElement ::=
'(' ruleExpression ')' |
'~' ruleExpressionElement |
attributeUse

variableDeclaration ::=
'[' (qualifiedObjectName variableIdentifier) |
variableIdentifierNoWildCard ']'

variableIdentifier ::=
[variableLocation] (variableName |

variableWildCard)

variableIdentifierNoWildCard ::=
[variableLocation] variableName

variableLocation ::=
((variableName |
variableWildCard) ':') +

variableSpecifier ::=
'[' ((qualifiedObjectName (variableIdentifier |
variableValue)) |
variableIdentifierNoWildCard) ']'

variableValue ::=
string |
signedNumber

facilitatorHead ::=
facilitatorName facilitatorParams

facilitatorName ::=
lowercaseWord

facilitatorParams ::=
variableDeclaration *

facilitatorCommand ::=
commandSelector '::=' commandDef

commandSelector ::=
uppercaseWord

commandDef ::=
(string |
variableSpecifier) +

facilitatorTarget ::=
'TARGET' targetSpec

targetSpec ::=
'LOCAL' |
variableSpecifier |
string

facilitatorDeclaration ::=
'DECLARE' variableDeclaration +

facilitatorPreConditions ::=

'PRE' simpleRuleExpression

simpleRuleExpression ::=
simpleRuleExpressionElement ('&' simpleRuleExpressionElement) *

simpleRuleExpressionElement ::=
(attributeUse |
'~' attributeUse)

initialProduction ::=
'RETURN' '::=' productionRHS '.'

otherProductions ::=
anotherProduction *

anotherProduction ::=
productionRef '::=' productionRHS '.'

productionRef ::=
lowercaseWord

productionRHS ::=
andProduction [('|' andProduction) *]

andProduction ::=
productionPart +

productionPart ::=
'(' productionRHS ')' |
'[' productionRHS ']' |
productionItem

productionItem ::=
newObject |
retypeObject |
fixedItem |
variableItem |
productionRef ['*' |
'+'] |
newFact |
completionFact |
scanningDirective

newObject ::=
'NEW' variableSpecifier

retypeObject ::=
'RETYPE' variableSpecifier

fixedItem ::=
number |
string

variableItem ::=
scannerToken [temporaryVariableSpecifier]

scannerToken ::=
lowercaseWord

temporaryVariableSpecifier ::=
'<' [qualifiedObjectName] variableIdentifier '>'

scanningDirective ::=
uppercaseWord

newFact ::=
'{' facts '}'

completionFact ::=
'{{' facts '}}'

facts ::=
fact (',' fact) *

fact ::=
attributeUse |
'~' attributeUse

parseConfiguration ::=
'CONFIG' configItem +

configItem ::=
lowercaseWord ['(' configParams ')']

configParams ::=
configParam (',' configParam) *

configParam ::=
string |
number

actionSpec ::=
'ACTION' actionHead actionExecutionInfo

actionHead ::=
facilitatorHead

actionExecutionInfo ::=

facilitatorCommand facilitatorTarget [facilitatorDeclaration] [facilitatorPreConditions] actionTrigger actionPostCondition

actionTrigger ::=
'TRIGGER' simpleRuleExpressionElement (',' simpleRuleExpressionElement) *

actionPostCondition ::=
'POST' simpleRuleExpression

querySpec ::=
'QUERY' queryHead queryExecutionInfo queryParserDef

queryHead ::=
facilitatorHead

queryExecutionInfo ::=
facilitatorCommand facilitatorTarget [queryValidity] [facilitatorDeclaration]
[facilitatorPreConditions]

queryValidity ::=
'VALID' 'FOR' duration

duration ::=
number [timeUnit]

timeUnit ::=
'SEC' |
'MIN' |
'HOUR' |
'DAY'

queryParserDef ::=
initialProduction otherProductions [parseConfiguration]

wholeModel ::=
modelSpec (objectSpec attributeSpec *) + querySpec * actionSpec *

Undefined:

uppercaseWord
lowercaseWord
string
signedNumber
number

Terminal Symbols:

'&' ruleAndExpression
'&' simpleRuleExpression
'(' attributeCardinality
'(' attributeModes
'(' configItem
'(' productionPart
'(' ruleExpressionElement
'(' trueFalsePart
')' attributeCardinality
')' attributeModes
')' configItem
')' productionPart
')' ruleExpressionElement
')' trueFalsePart
'*' productionItem
'+' attributeModeValue
'+' productionItem
',' actionTrigger
',' configParams
',' facts
'.' anotherProduction
'.' initialProduction
'.' qualifiedObjectName
':' attributeCardinality
':' variableLocation
'::=' anotherProduction
'::=' facilitatorCommand
'::=' initialProduction
'<' temporaryVariableSpecifier
'>' temporaryVariableSpecifier
'?' attributeModeValue
'ACTION' actionSpec
'ASPECT' aspectObject
'BASIC' basicAttribute
'BUILT' coreObject
'CONFIG' parseConfiguration
'CORE' coreObject
'DAY' timeUnit
'DECLARE' facilitatorDeclaration
'DERIVED' derivedAttribute
'DESCRIPTION' modelDescription
'EXPLANATION' attributeExplanation
'FOR' queryValidity
'FROM' coreObject
'HOUR' timeUnit
'IF' commandAttribute

'IF' derivedAttribute
'IMPORT' imports
'ISA' objectHierarchy
'LOCAL' targetSpec
'LOCATION' objectLocation
'M' attributeCardinalValue
'METHOD' methodAttribute
'MIN' timeUnit
'MODEL' modelSpec
'NEW' newObject
'O' attributeCardinalValue
'OBJECT' aspectObject
'OBJECT' coreObject
'OBJECT' nonCoreObject
'OF' aspectObject
'POST' actionPostCondition
'PRE' facilitatorPreConditions
'QUERY' querySpec
'RETURN' initialProduction
'RETYPE' retypeObject
'SEC' timeUnit
'TARGET' facilitatorTarget
'TRIGGER' actionTrigger
'UI' commandAttribute
'VALID' queryValidity
'VIRTUAL' virtualAttribute
'[' productionPart
'[' variableDeclaration
'[' variableSpecifier
']' productionPart
']' variableDeclaration
']' variableSpecifier
'_' variableWildCard
'{' newFact
'{{' completionFact
'|' productionRHS
'|' ruleExpression
'|' trueFalsePart
'}' newFact
'}}' completionFact
'~' fact
'~' ruleExpressionElement
'~' simpleRuleExpressionElement

B SysMIB.m

```
MODEL SysMibHS


IMPORT ManageableObject





OBJECT StoEntry INA UnixMachineWithSysMib


BASIC

[stoEntry]  stoAllocationUnits [integer]  (M:O)
EXPLANATION  [stoEntry] , 'stoAllocationUnits' ,  (' is ' | ' is not') ,  [integer] . .
[stoEntry]  stoDescr [string]  (M:O)
EXPLANATION  [stoEntry] , 'stoDescr' ,  (' is ' | ' is not') ,  [string] . .
[stoEntry]  stoType [string]  (M:O)
EXPLANATION  [stoEntry] , 'stoType' ,  (' is ' | ' is not') ,  [string] . .
[stoEntry]  stoIndex [integer]  (M:O)
EXPLANATION  [stoEntry] , 'stoIndex' ,  (' is ' | ' is not') ,  [integer] . .  INVISIBLE
[stoEntry]  stoState [integer]  (M:O)
EXPLANATION  [stoEntry] , 'stoState' ,  (' is ' | ' is not') ,  [integer] . .
[stoEntry]  stoEntryIdent [integer]  (O:O)
EXPLANATION  [stoEntry] , 'stoEntryIdent ' ,  (' is ' | ' is not') ,  [integer] . .  INVISIBLE
[stoEntry]  stoSize [integer]  (M:O)
EXPLANATION  [stoEntry] , 'stoSize' ,  (' is ' | ' is not') ,  [integer] . .
[stoEntry]  stoUsed [integer]  (M:O)
EXPLANATION  [stoEntry] , 'stoUsed' ,  (' is ' | ' is not') ,  [integer] . .

UI


DERIVED

MAP

VIRTUAL


IDENT '[]stoEntryIdent[]'
```

```
OBJECT DiskEntry INA UnixMachineWithSysMib

BASIC

[diskEntry]  diskWords [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskWords' ,  (' is ' | ' is not') ,  [integer] . .
[diskEntry]  diskXferRate [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskXferRate' ,  (' is ' | ' is not') ,  [integer] . .
[diskEntry]  diskXfer [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskXfer' ,  (' is ' | ' is not') ,  [integer] . .
[diskEntry]  diskAccess [string]  (M:O)
EXPLANATION  [diskEntry] , 'diskAccess' ,  (' is ' | ' is not') ,  [string] .
 POLICY
 [string] isAssigned ['read-write'] ;
 [string] isAssigned ['read-only'] . .
[diskEntry]  diskLabel [string]  (M:O)
EXPLANATION  [diskEntry] , 'diskLabel' ,  (' is ' | ' is not') ,  [string] . .
[diskEntry]  diskReads [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskReads' ,  (' is ' | ' is not') ,  [integer] . .
[diskEntry]  diskCapacity [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskCapacity' ,  (' is ' | ' is not') ,  [integer] . .
[diskEntry]  diskMedia [string]  (M:O)
EXPLANATION  [diskEntry] , 'diskMedia' ,  (' is ' | ' is not') ,  [string] .
 POLICY
 [string] isAssigned ['unknown'] ;
 [string] isAssigned ['hardDisk'] ;
 [string] isAssigned ['opticalDisks'] ;
 [string] isAssigned ['floppyDisk'] . .
[diskEntry]  diskState [string]  (M:O)
EXPLANATION  [diskEntry] , 'diskState' ,  (' is ' | ' is not') ,  [string] .
 POLICY
 [string] isAssigned ['enabled'] ;
 [string] isAssigned ['disabled'] . .
[diskEntry]  diskRemoveable [string]  (M:O)
EXPLANATION  [diskEntry] , 'diskRemoveable' ,  (' is ' | ' is not') ,  [string] .
 POLICY
 [string] isAssigned ['removeable'] ;
 [string] isAssigned ['not-removeable'] . .
[diskEntry]  diskBusy [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskBusy' ,  (' is ' | ' is not') ,  [integer] . .
[diskEntry]  diskIndex [integer]  (M:O)
EXPLANATION  [diskEntry] , 'diskIndex' ,  (' is ' | ' is not') ,  [integer] . .  INVISIBLE
[diskEntry]  diskEntryIdent [integer]  (O:O)
EXPLANATION  [diskEntry] , 'diskEntryIdent ' ,  (' is ' | ' is not') ,  [integer] . .  INVISIBLE
```

```
UI

DERIVED

MAP

VIRTUAL

IDENT '[]diskEntryIdent[]'



OBJECT UnixMachineWithSysMib ISA ManageableObject

DESCRIPTION 'Unix Machine with LRZ-System-MIB'

BASIC

[unixMachineWithSysMib] sysName [string] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysContact [string] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysLocation [string] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysOs [string] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysHardware [string] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysUptime [integer] (M:O)

CATEGORY 'System'.
```

```
[unixMachineWithSysMib] sysDate [string] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysUsers [integer] (M:O)

CATEGORY 'System'.

[unixMachineWithSysMib] sysClockTicks [integer] (M:O)

CATEGORY 'System'.


[unixMachineWithSysMib] StorageTable [unixMachineWithSysMib : stoEntry] (O:M)

CATEGORY 'Storage'.


[unixMachineWithSysMib] DiskTable [unixMachineWithSysMib : diskEntry]  (O:M)

CATEGORY  'Device' .




UI


DERIVED

MAP

VIRTUAL




QUERY  getSysName
SNMP   ::= '.1.3.6.1.3.100.2.1.1.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= string <s> {[m]sysName[s]}.

QUERY  getSysContact
SNMP   ::= '.1.3.6.1.3.100.2.1.2.0'
```

```
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= string <s> {[m]sysContact[s]}.

QUERY  getSysLocation
SNMP   ::= '.1.3.6.1.3.100.2.1.3.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= string <s> {[m]sysLocation[s]}.

QUERY  getSysOs
SNMP   ::= '.1.3.6.1.3.100.2.1.4.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= string <s> {[m]sysOs[s]}.

QUERY  getSysHardware
SNMP   ::= '.1.3.6.1.3.100.2.1.5.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= string <s> {[m]sysHardware[s]}.

QUERY  getSysUptime
SNMP   ::= '.1.3.6.1.3.100.2.1.6.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= integer <s> {[m]sysUptime[s]}.

QUERY  getSysDate
SNMP   ::= '.1.3.6.1.3.100.2.1.7.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= string <s> {[m]sysDate[s]}.

QUERY  getSysUsers
SNMP   ::= '.1.3.6.1.3.100.2.1.8.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= integer <s> {[m]sysUsers[s]}.

QUERY  getSysClockTicks
SNMP   ::= '.1.3.6.1.3.100.2.1.9.0'
TARGET [UnixMachineWithSysMib m].
VALID FOR 3600.
RETURN ::= integer <s> {[m]sysClockTicks[s]}.
```

```
QUERY stoEntry
SNMPTable  ::= '.1.3.6.1.3.100.2.2.2.1'
TARGET [UnixMachineWithSysMib m].
VALID FOR 240.
DECLARE  [StoEntry i]   [Integer n] .
RETURN ::= chaines  { { [m] stoEntry [m:_] } } .
chaines ::= chaine+.
chaine ::= ( NEW [i] COUNTER [n]
stoType
stoDescr
stoAllocationUnits
stoSize
stoUsed
stoState
 { [m] stoEntry [m:i] , [i] stoEntryIdent [n] } ).
stoState ::= integer <s>  { [m : i] stoState [s] } .
stoUsed ::= integer <s>  { [m : i] stoUsed [s] } .
stoSize ::= integer <s>  { [m : i] stoSize [s] } .
stoAllocationUnits ::= integer <s>  { [m : i] stoAllocationUnits [s] } .
stoDescr ::= string <s>  { [m : i] stoDescr [s] } .
stoType ::= objectID <s> { [m : i] stoType [s] }.
```

```
QUERY diskEntry
SNMPTable  ::= '.1.3.6.1.3.100.2.3.1.3.1.1'
TARGET [UnixMachineWithSysMib m].
VALID FOR 240.
DECLARE  [DiskEntry i]   [Integer n] .
RETURN ::= chaines  { { [m] diskEntry [m:_] } } .
chaines ::= chaine+.
chaine ::= ( NEW [i] COUNTER [n]
diskAccess
diskMedia
diskRemoveable
diskCapacity
diskState
diskLabel
diskBusy
```

```
diskReads
diskXfer
diskXferRate
diskWords
 { [m] diskEntry [m:i] , [i] diskEntryIdent [n] } ).
diskWords ::= integer <s>  { [m : i] diskWords [s] } .
diskXferRate ::= integer <s>  { [m : i] diskXferRate [s] } .
diskXfer ::= integer <s>  { [m : i] diskXfer [s] } .
diskReads ::= integer <s>  { [m : i] diskReads [s] } .
diskBusy ::= integer <s>  { [m : i] diskBusy [s] } .
diskLabel ::= string <s>  { [m : i] diskLabel [s] } .
diskState ::=  (
1{ [m : i] diskState ['enabled'] }  |
2{ [m : i] diskState ['disabled'] }  ) .
diskCapacity ::= integer <s>  { [m : i] diskCapacity [s] } .
diskRemoveable ::=  (
1{ [m : i] diskRemoveable ['removeable'] }  |
2{ [m : i] diskRemoveable ['not-removeable'] }  ) .
diskMedia ::=  (
1{ [m : i] diskMedia ['unknown'] }  |
2{ [m : i] diskMedia ['hardDisk'] }  |
3{ [m : i] diskMedia ['opticalDisks'] }  |
4{ [m : i] diskMedia ['floppyDisk'] }  ) .
diskAccess ::=  (
1{ [m : i] diskAccess ['read-write'] }  |
2{ [m : i] diskAccess ['read-only'] }  ) .



ACTION setSysContact [String newSysContact]
SNMP   ::= '.1.3.6.1.3.100.2.1.2.0 ' newSysContact
TARGET [UnixMachineWithSysMib m].
POST ::= [m] sysContact [newSysContact].

ACTION setSysLocation [String newSysLocation]
SNMP   ::= '.1.3.6.1.3.100.2.1.3.0 ' newSysLocation
TARGET [UnixMachineWithSysMib m].
POST ::= [m] sysLocation [newSysLocation].

ACTION setSysDate [String newSysDate]
SNMP   ::= '.1.3.6.1.3.100.2.1.7.0 ' newSysDate
TARGET [UnixMachineWithSysMib m].
POST ::= [m] sysDate [newSysDate].
```

C Literaturverzeichnis

Ein Teil der hier aufgeführten Texte sind von Hewlett Packard als *Company Confidential* ausgewiesen. Gleichwohl standen sie dem Autor im Verlauf dieser Arbeit zur Verfügung und sind wohl auch anderen Studenten zugänglich.

Literatur

[CDFS90] Case, Davin, Fedor und Schoffstall: *RFC 1157: Simple Network Management Protocol (SNMP).* Technischer Bericht IAB, 1990.

[Dra] Drakos, Nikos: *The LaTeX2html Translator.*

[Dzi95] Dzik, Michael: *Entwurf und Implementierung von Managementfunktionen für HP-Dolphin zum Management von Windows-PC-Netzen.* Fortgeschrittenen-Praktikum, Universität München, voraussichtlich Dezember 1995.

[Fed95] Feder, Adam B.: *A Dolphin Model Development Environment.* Diplomarbeit, Massachusetts Institute of Technology, 1995.

[GN95] Gutschmidt, M. und B. Neumair: *Integration von Netz- und Systemmanagement: Ziele und erste Erfahrungen.* Technischer Bericht Ludwig-Maximilians-Universität München, 1995.

[HH95] Haubelt, Nedo und Rainer Hauck: *Implementierung einer MIB für Systemmanagementaufgaben.* Diplomarbeit, Technische Universität München, 1995.

[Kri94] Krieger, Uwe: *Konzeption einer Managementinformationsbasis für das Management von Unix-Endsystemen.* Diplomarbeit, Technische Universität München, 1994.

[Pel] Pell, Adrian: *Dolphin - A Model Language Specification.*

[Riv] Riviere, Anne: *SNMP MIB Reader.*

[Ros94] Rose, M. T.: *The Simple Book.* Prentice-Hall, 1994.

[Sta93] Stallings, W.: *SNMP, SNMPv2 and CMIP: The Practical Guide to Network Management Standards.* Addison-Wesley, 1993.