INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Objektorientierte Analyse und Implementierung von Managementinformation und -funktionalität ausgewählter systemnaher Basisdienste

Erwin Hainzinger

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Markus Gutschmidt
Abgabedatum:	15. Mai. 1995

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 12. Mai 1995

(Erwin Hainzinger)

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Systemmanagement

Früher stellten wenige Großrechner, die über Terminals zugänglich waren, Rechenleistung bereit. Ständig wachsende Anforderungen an das Rechensystem erforderten auch eine ständige Erweiterung des Systems. Durch die gestiegene Rechenleistung von Workstations, deren gesunkenen Anschaffungskosten und die bessere Erweiterbarkeit von Workstationverbunden wurde die Anschaffung von Workstations gefördert. Damit war es auch möglich, dem Anwender direkt an seinem Arbeitsplatz ein Rechensystem bereitzustellen. Außerdem erhöhte sich die Ausfallsicherheit des Gesamtsystems.

Ermöglicht wurde diese Entwicklung durch die Einführung von Netzen mit hohen Datenübertragsraten und standardisierten Schnittstellen. Aufgrund des rasch wachsenden Netzes und des Heterogenität erhöht sich aber auch der Administrationsaufwand. Deshalb wurde das Netzmanagement entwickelt, dessen Ziel ein effektiver und effizienter Ablauf der Kommunikation über das Rechnernetz ist. Dazu gehört die zentrale Überwachung und Verwaltung aller Hard- und Softwarekomponenten, die verantwortlich für den Kommunikationsablauf sind, was das Rechensystem selbst und verteilte Systeme ausschließt.

Aber die Vernetzung brachte noch einen weiteren Vorteil. Auf den isolierten Rechnern mußten alle Anwendungen und Daten vorhanden sein. Jetzt konnten verteilte Anwendungen eingesetzt werden, die unteranderem für einen gemeinsamen Datenbestand sorgten und die gemeinsame Nutzung teuerer Ressourcen ermöglichten.

Wegen der Vielzahl von Rechnern ist es nicht mehr vertretbar und durchführbar, jeden Rechner einzeln zu verwalten. Es muß ein Mechanismus entwickelt werden, der es gestattet mehrere Rechner gleichzeitig und zentral zu verwalten, pflegen, überwachen und weiterzuentwickeln. Will man jedoch ganze Cluster von Systemen von einer zentralen Stelle aus administrieren, so werden dazu Konzepte und Mechanismen benötigt, wie sie beim Netzmanagement schon längere Zeit bekannt und erprobt sind ([Hege 93]).Für diese Bereiche ist das Systemmanagement zuständig. Letztlich soll es dem Systemadministrator bei der Planung und dem reibungslosen Betrieb von Endsystemen und verteilten Systemen unterstützen sowie auf auftretende Probleme aufmerksam machen.

Bei genauerer Betrachtung des Systemmanagements stellt man fest, daß die Aufgaben gar nicht so verschieden von denen des Netzmanagements sind und deshalb viele Konzepte daraus übernommen werden können. Die Grenze zwischen Netz- und Systemmanagement ist darüberhinaus bei einem verteilten System fließend.

1.2 Motivation und Überblick

Ein Rechnersystem bietet eine Vielzahl von Basisdiensten an, wie den Nachrichtendienst (Mail), Druckdienst (LPD), Dateidienst (Fileserver) oder Namensdienst (DNS). Allen Diensten ist gemeinsam, daß an sie Aufträge geschickt werden, die sie bearbeiten und beantworten sollen. Werden an sie mehrere Aufträge gleichzeitig geschickt, können sie entweder den ersten annehmen und alle folgenden abweisen oder sie legen die folgenden Aufträge in einem Puffer ab. Aus diesem Puffer können sie die Aufträge dann der Reihe nach bearbeiten, wobei sich bei manchen Diensten diese Reihenfolge beeinflußen läßt.

Für den Systemverwalter ist es nun interessant zu erfahren, ob der Dienst aktiv ist und wenn ja, wie hoch die Belastung, Fehlerhäufigkeit und Bearbeitungsgeschwindigkeit des Servers ist. Anhand dieser Daten kann der Administrator seine zukünfige Planung ausrichten, entscheiden, ob die Konfiguration geändert werden muß und bei Fehlern eingreifen.

Dabei ist es sinnvoll nicht für jeden Basisdienst ein eigenes Management zu entwicklen. Vielmehr sollte ein dienstunabhäniges Modell gefunden werden, auf das die Basisdienste abgebildet werden können und durch das die gleiche Managementinformation und -funktionalität für alle Basisdienste bereitsteht. Somit kann der Systemverwalter alle Basisdienste auf die gleiche Art administieren und muß sich nicht für jeden Basisdienst spezielles Wissen aneignen.

In dieser Diplomarbeit wird nun untersucht, inwieweit das in Kapitel 2 vorgestellte Objektmodell in der Lage ist, die Basisdienste abzubilden und mit Managementinformation und -funktionalität auszustatten.

Dafür wurden zwei Basisdienste ausgewählt, der Druck- und der Nachrichtendienst, welche in Kapitel 3 ausführlich untersucht werden. Begonnen wird dabei mit der Entwicklung der funktionalen Modelle, um anschließend die spezifischen Objektmodelle aufstellen zu können.

Um einen Basisdienst zu managen, muß als erstes entschieden werden, welche Managementinformation und -funktionalität nötig ist. Dies wird in Kapitel 4 untersucht und den einzelnen Komponenten des Objektmodells zugewiesen. Wie diese Daten von Basisdiensten angeboten werden, wird dann im folgenden noch für die beiden ausgewählten Basisdienste erläutert.

Schließlich wird in Kapitel 5 eine Abbildungsvorschrift entwickelt, um das Objektmodell in das Internet Informationsmodell abbilden zu können. Hierbei wird auf eine allgemeine Abbildungsvorschrift wert gelegt, um alle entwickelten Objektmodelle nach dem gleichen Prinzip abbilden zu können, egal welche Komponenten das Objektmodell im einzelnen enthält. Am Ende des Kapitels wird gezeigt, wie die ausgewählten Basisdienste abgebildet werden und deren Struktur im Internet Informationsmodell aussieht.

Nachdem bekannt ist, welche Variablen das Informationsmodell enthalten soll und wie sie in der MIB anzuordnen sind, kann ein Subagent implementiert werden. Allerdings wird der Subagent nur noch für den Druckdienst entwickelt und in Kapitel 6 beschrieben. Um die nötigen Informationen vom Druckdienst zu erhalten, mußten auch dessen Programme (lpr, lpd) verändert werden.

Kapitel 7 schließt mit einer Zusammenfassung der wichtigsten Erkenntnisse aus dieser Arbeit und gibt noch einen kurzen Ausblick auf Managementbereiche, die in dieser Arbeit nicht untersucht wurden.

Damit der Managementapplikation bekannt ist, welche Variablen sich im Subagenten für den Druckdienst befinden, wird ihr eine SNMP MIB Beschreibung bereitgestellt. Diese Beschreibung befindet sich auch in Anhang A wieder.

Kapitel 2

Ein Objektmodell zur Analyse des Managements systemnaher Dienste

Für die unterschiedlichen Basisdienste, die durch verschiedene Server realisiert werden, ist es nötig einen einheitlichen Managementrahmen zu finden. Dazu wird in diesem Kapitel ein generisches Objektmodell für Server vorgestellt, worauf sich alle Basisdienste abbilden lassen sollten. Später werden auch noch Managementanforderungen aufgestellt, die von den spezialisierten Objektmodellen — das Objektmodell für einen speziellen Basisdienst — verlangt werden.

Das Objektmodell wird in [Guts 95] entwickelt und ausführlich erläutert. Dieses Kapitel faßt die für diese Arbeit wichtigen Elemente daraus zusammen.

2.1 Das Objektmodell

Um einen einheitlichen Managementrahmen für Basisdienste zu haben, wurde das Objektmodell entwickelt. Es muß fexibel genug sein, die verschiedenen Server abbilden zu können, darf aber nicht zu kompliziert und umfangreich sein, damit sich die spezialisierten Objektmodelle nicht grundsätzlich voneinander unterscheiden. Um zu zeigen, wie realitätsnah die Abbildung geschieht, wird sie im folgenden Kapitel für zwei ausgewählte Basisdienste durchgeführt.

Bei der Entwicklung des Objektmodells wird eine objektorientierte Analyse angewendet. Sie bietet den Vorteil einer ganzheitlichen Sicht auf die Struktur, die Informationen (Daten) und der Funktionalität eines Problemraumes. Eine ausführliche Beschreibung der zentralen Konzepte der Objektorientierung findet man in Abschnitt 5.1.1, hier werden sie nur kurz genannt:

- Klassenkonzept
  Durch einen Objekttyp lassen sich gleichartige Objekte zusammenfassen und gemeinsam definieren.
- Kapselung
  Die Attribute eines Objekts können nur durch die Operationen eines Objekts verändert werden.
- Vererbung
  Objekte einer Klasse können durch Vererbung auf die Attribute und Methoden einer bereit vorher definierten Klasse zurückgreifen, ohne diese neu zu definieren. So lassen sich grundlegende Informationen in einer Oberklasse bereitstellen, die für konkrete Objekte durch Vererbung erweitert werden können.

Bei der Vorstellung des Objektmodells wird auch der Begriff Klasse-&-Objekt eingeführt. Um diesen erklären zu können, werden zuerst die Begriffe Klasse und Objekt vorgestellt.

Klasse: Die Klasse ist eine abstrakte Beschreibung der Daten und des Verhaltens der Objekte.

Objekt: Die Abstraktion von etwas im Problemraum. Die Kapselung der Attribute und den Diensten darauf.

Klasse-&-Objekte: Dieser Ausdruck bezeichnet eine Klasse mit allen Objekten, die von dieser Klasse stammen. Ein Beispiel: Bei der Abbildung der Tastatur eines Computers gibt es auch eine Klasse „Taste“. Sie definiert die Attribute und Methoden einer Taste. Für jede real vorhandene Taste, wird ein Objekt der Klasse „Taste“ erzeugt, die die Attribute einer speziellen Taste der Tastatur beinhaltet. Alle diese Objekte und die Klasse selbst sind gemeint, wenn von Klasse-&-Objekte „Taste“ gesprochen wird.

Um das Objektmodell für einen konkreten Server aufzustellen, müssen als erstes die funktionalen Komponenten des Servers gefunden werden. Diese Komponenten werden im Objektmodell zu Klassen-&-Objekte überführt, anschließend die Struktur (Beziehungen zwischen den Objekten) festgelegt und zuletzt die Attribute und Services (Methoden) der Objekte ermittelt.

Bei der Untersuchung realer Server lassen sich im Grunde drei Arten funktionaler Komponenten finden. Es gibt die Warteschlange, die Aufträge sammelt und solange hält bis

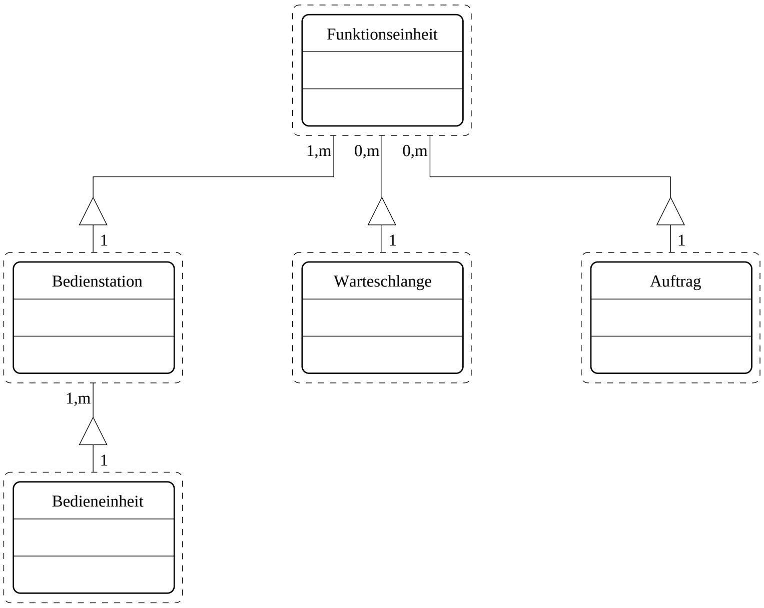

Abbildung 2.1: Das Grundgerüst des Objektmodells

benötigte Ressourcen frei und für die Aufnahme eines neuen Auftrags bereit sind. Weiter gibt es Bedienstationen, welche für die Bearbeitung des Auftrags zuständig sind. Die vollständige Bearbeitung des Auftrags kann dabei in mehrere Schritte zerlegt und auf mehrere Bedienstationen verteilt sein. Die eigentliche Bearbeitung der Aufträge übernehmen die Bedieneinheiten, die in den Bedienstationen enthalten sind. Zuguterletzt gibt es den Auftrag selbst, er enthält die Daten die bearbeitet werden müssen und die Informationen zur korrekten Steuerung der Bearbeitung.

Im Objektmodell wird den bisher aufgezählten Komponenten noch die Komponente „Funktionseinheit" übergeordnet, sie bildet den Server selbst ab, d.h. alle anderen Komponenten des Objektmodells sind in der Funktionseinheit enthalten.

In dem bisher entwickelten Objektmodell fehlen noch die Verbindungen zwischen den Komponenten, die erst später eingeführt werden. Die Abbildung 2.1 zeigt ein erstes Klassenmodell, in der die Klassen durch Kästchen dargestellt sind. Jedes Kästchen ist in drei Bereiche unterteilt, im oberen befindet sich der Klassenname, im mittleren können die Attribute der Klasse angegeben werden und im unteren die Methoden. Die gestrichelten Kästchen um die Klassen symbolisieren die Objekte, die von der Klasse abgeleitet wurden. Somit ist ein inneres Kästchen mit einem äußeren gestrichelten Kästchen eine Klasse-&-Objekt des Objektmodells.

Da die Funktionseinheit alle Komponenten des Objektmodells enthält, wird sie auch als oberstes Element im Objektmodell eingezeichnet. Die Enthaltensein–Relationen (Containment) sind in der Abbildung durch die Verbindungen mit dem Dreieck in der Verbindungslinie gekennzeichnet. Die Zahlenangabe bei der Verbindungslinie am Objekt gibt an, wie häufig dieses Objekt im übergeordneten Objekt enthalten sein kann bzw. wie häufig das zugehörige Objekt in diesem enthalten sein kann. Welche und wieviele in einem Server enthalten sind, muß für jeden Server einzeln entschieden werden. Allerdings hat jeder Server wenigstens eine Bedienstation, daneben können noch Warteschlangen und Aufträge in der Funktionseinheit enthalten sein. Jedoch dürfen diese Klassen-&-Objekte in keiner weiteren Funktionseinheit mehr beinhaltet sein.

Betrachtet man nun einen konkreten Server genauer, stellt man fest, daß es von den Komponenten des Objektmodells mehrere Typen geben kann. So können z.B. verschiedene Auftragsformate vom Server angenommen und verarbeitet werden. Dies läßt sich im Objektmodell durch die Spezialisierung (Vererbung) der entsprechenden Klassen wiedergeben. In Abbildung 2.2 ist diese Typisierung durch die Linien mit den Halbkreisen dargestellt. Existieren typisierte Klassen, gibt es von der Oberklasse keine Instanz mehr, sie werden zu abstrakten Klassen. Abstrakte Klassen sind in der Abbildung 2.2 durch das Fehlen des gestrichelten Kästchens um die Klasse zu erkennen.

Zusätzlich sind in Abbildung 2.2 die Beziehungen zwischen den Komponenten im Objektmodell zu sehen. Diese Beziehungen werden als gerichtete Verbindungen zwischen den Objektinstanzen ausgedrückt. Sie stellen mögliche Wege des Auftrags durch die Funktionseinheit dar. Welche Verbindungen zwischen den Objekten dabei denkbar sind, wird im folgenden aufgezählt:

Funktionseinheit: Es muß gekennzeichnet werden, von welchen Komponenten des Objektmodells die Aufträge angenommen werden. Durch Verbindungen von der Funktionseinheit zu den Instanzen, die Aufträge annehmen, wird dies dargestellt. Mindestens eine Bedienstation oder Warteschlange muß die Aufträge annehmen, es können aber auch mehrere sein. Zusammengefaßt gibt es von der Funktionseinheit Verbindungen zu

- keiner bis mehreren Warteschlangen
- keiner bis mehreren Bedienstationen

Bedienstation: Eine Bedienstation kann einen Teil einer Auftragsbearbeitung übernehmen und den Auftrag anschließend einem anderen Objekt des Objektmodells weiterreichen, sie kann aber den Auftrag auch abschließen. Schließt sie den Auftrag ab, wird dies durch eine Verbindung zur Funktionseinheit kenntlich gemacht. Zusammenfassend heißt das, einer Bedienstation folgt immer eine weitere Komponente, wobei wenigstens eine Bedienstation für den Auftragsabschluß zuständig ist. Möglich sind Verbindungen von der Bedienstation zu

- keiner oder einer Funktionseinheit (Auftragsabschluß)
- keiner oder mehreren Warteschlangen
- keiner oder mehreren Bedienstationen

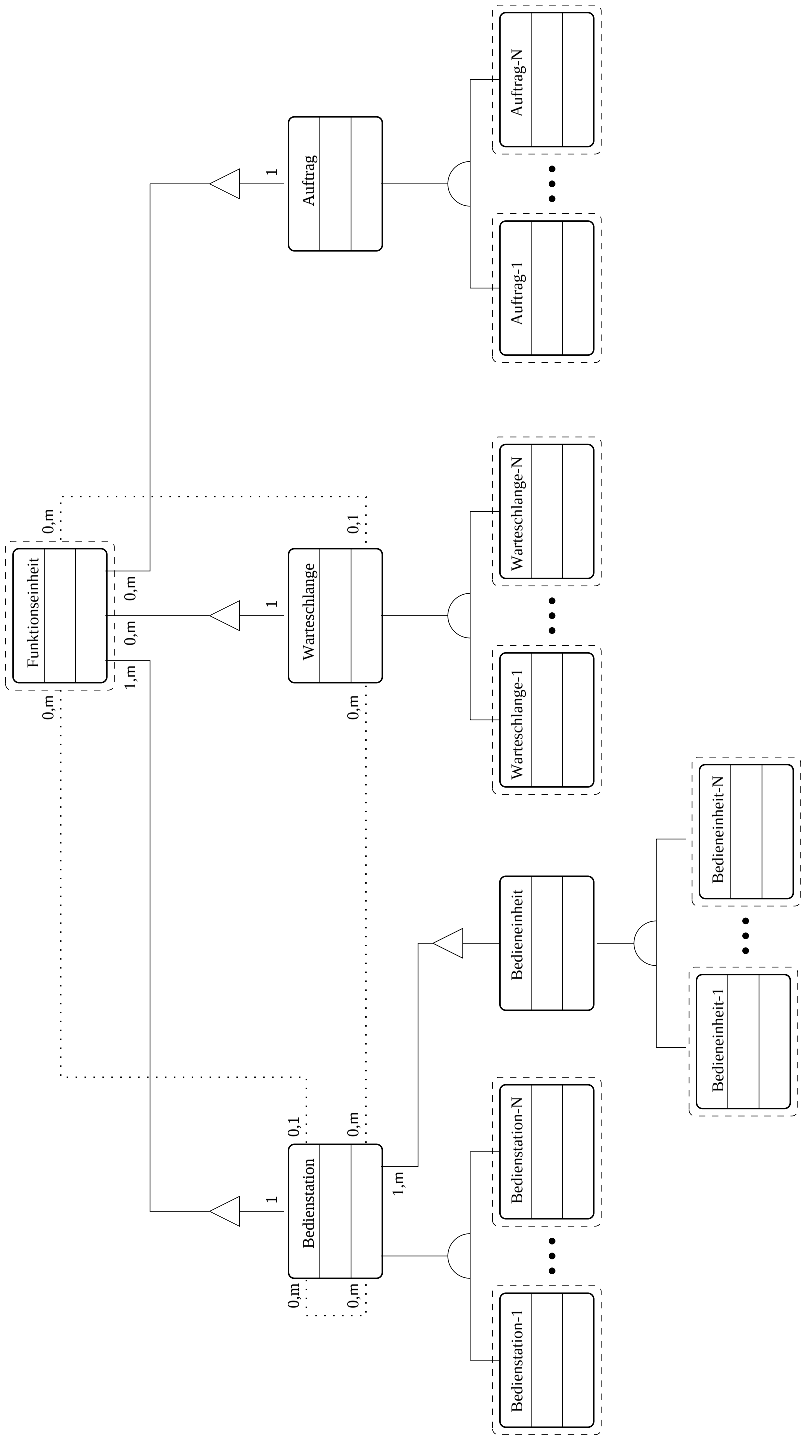

Abbildung 2.2: Das vollständige Objektmodell mit Typisierung und Verbindungen

Warteschlange: Aufträge werden in einer Warteschlange gesammelt, sofern sie nicht sofort von der folgenden Bedienstation bearbeitet werden können. Einer Warteschlange muß also mindestens eine Bedienstation folgen. Von der Warteschlange gibt es deshalb eine Verbindung zu

- einer oder mehreren Bedienstationen

Kapitel 3

Die Objektmodelle für zwei ausgewählte Basisdienste

Das im vorhergehenden Kapitel eingeführte Objektmodell soll in diesem Kapitel auf seine Anwendbarkeit hin untersucht werden. Da aber jeder Server anders aufgebaut ist kann nicht die Allgemeingültigkeit und Anwendbarkeit des Objektmodells bewiesen werden. Ist es aber möglich die Objektmodelle zweier ausgewählter Basisdienste aufzustellen, wofür der Druck- und der Nachrichtendienst ausgewählt wurden, dürfte es bei fast allen anderen Basisdiensten auch gelingen. Um die Objektmodelle zu erhalten, müssen als erstes die funktionalen Modellen aufgestellt und die Informationen und Funktionalität ermittelt werden. In die funktionalen Modelle fließt auch die Prozeßstruktur noch stark mit ein, wohingegen sie bei den Objektmodellen in den Hintergrund tritt.

3.1 Managementinformation und -funktionalität des Drucksystems

Das Drucksystem PLP (Public Line Printer Spooler) von Prof. Patrick Powell vom Electrical and Computer Engineering Dept., San Diego State University ist eine Erweiterung des Standarddrucksystems LPD (Line Printer Spooler) des Berkley–Unix. Es wurde aber größtenteils eigenständig entwickelt, wobei auf die funktionale Ähnlichkeit zu LPD wert gelegt wurde. Eine ausführliche Beschreibung des PLP Drucksystems mit den Erweiterungen, Bedienungs- und Installationsanleitungen findet man in [PLP 94]. Da PLP keine Public Domain Software ist, sind auch die Copyright Bestimmungen zu beachten.

Um Aufträge zu erstellen und diesen entsprechende Optionen mitzugeben, dient das Programm 'lpr'. Es nimmt den Auftrag an, initiiert geforderte Bearbeitungsschritte und reiht den Auftrag in die Warteschlange ein. Sobald das geschehen ist, meldet es dem Druckdämon 'lpd', daß sich ein neuer Auftrag in der Warteschlange befindet. Für die Reihenfolge der

Aufträge in der Warteschlange ist das Warteschlangensteuerungsprogramm (Queuehandler) zuständig. Die weitere Bearbeitung des Auftrags übernimmt nun der Druckdämon. Er startet für jeden Drucker, für den Aufträge vorliegen einen Druckserver (Printserver), der sich die Aufträge aus der Warteschlange holt, sie den Angaben entsprechend bearbeitet und letztendlich dem Drucker zum Ausdrucken zuführt.

3.1.1 Untersuchung des Drucksystems

Verfolgt man einen Druckauftrag durch das Drucksystem bis zum Drucker, erkennt man, daß mehrere Stufen der Bearbeitung durchlaufen werden. Nachdem der Auftrag erstellt wurde, wird geprüft, ob die Auftragsbeschreibung und die geforderten Bearbeitungsschritte zulässig sind. Anschließend können bei Bedarf Präfilter durchlaufen werden, bevor der Auftrag in die Warteschlange eingereiht wird. Die Aufgaben der Filter lauten Konvertierung des Auftragsformats, Abrechnungsinformation (Accounting) sammeln, Datenüberprüfung und Formatierung der Druckdaten. Als Präfilter werden dabei jene Filter bezeichnet, die ein Auftrag durchläuft, bevor er die Warteschlange erreicht, wohingegen die durchlaufenen Filter, nachdem der Auftrag aus der Warteschlange geholt wurde, als Postfilter bezeichnet werden. Wird der Auftrag also von der Warteschlange weitergegeben, werden Postfilter durchlaufen bis der Auftrag endlich den Drucker erreicht. Ein funktionales Modell für das Drucksystem wird in Abbildung 3.1 gezeigt. Im folgenden wird dann die Managementinformation und -funktionalität des Drucksystems den einzelnen Komponenten des Modells zugeteilt.

Im Modell ist zu erkennen, daß sich im Drucksystem mehrere Warteschlangen mit den vor- und nachgestellten Bedienstationen befinden können. Anhand des im Auftrag angegebenen Druckernamens entscheidet der Server, in welche Warteschlange der Auftrag eingereiht werden soll, womit auch festgelegt ist, welche Bedienstationen für die Bearbeitung des Auftrags zuständig sind. Da der Aufbau der Warteschlangen mit den Bedienstationen für jeden Zweig des Modells gleich ist, wird im folgenden nur auf einen Zweig des Drucksystems eingegangen.

3.1.1.1 Auftrag

Wünscht ein Benutzer einen Ausdruck seiner Daten, muß er einen Auftrag an das Drucksystem schicken. Neben den benötigten Angaben können noch viele weitere Angaben zur Bearbeitung des Auftrags mitgegeben werden. Diese Angaben sind optional und werden seltener benötigt.

Benötigte Angaben

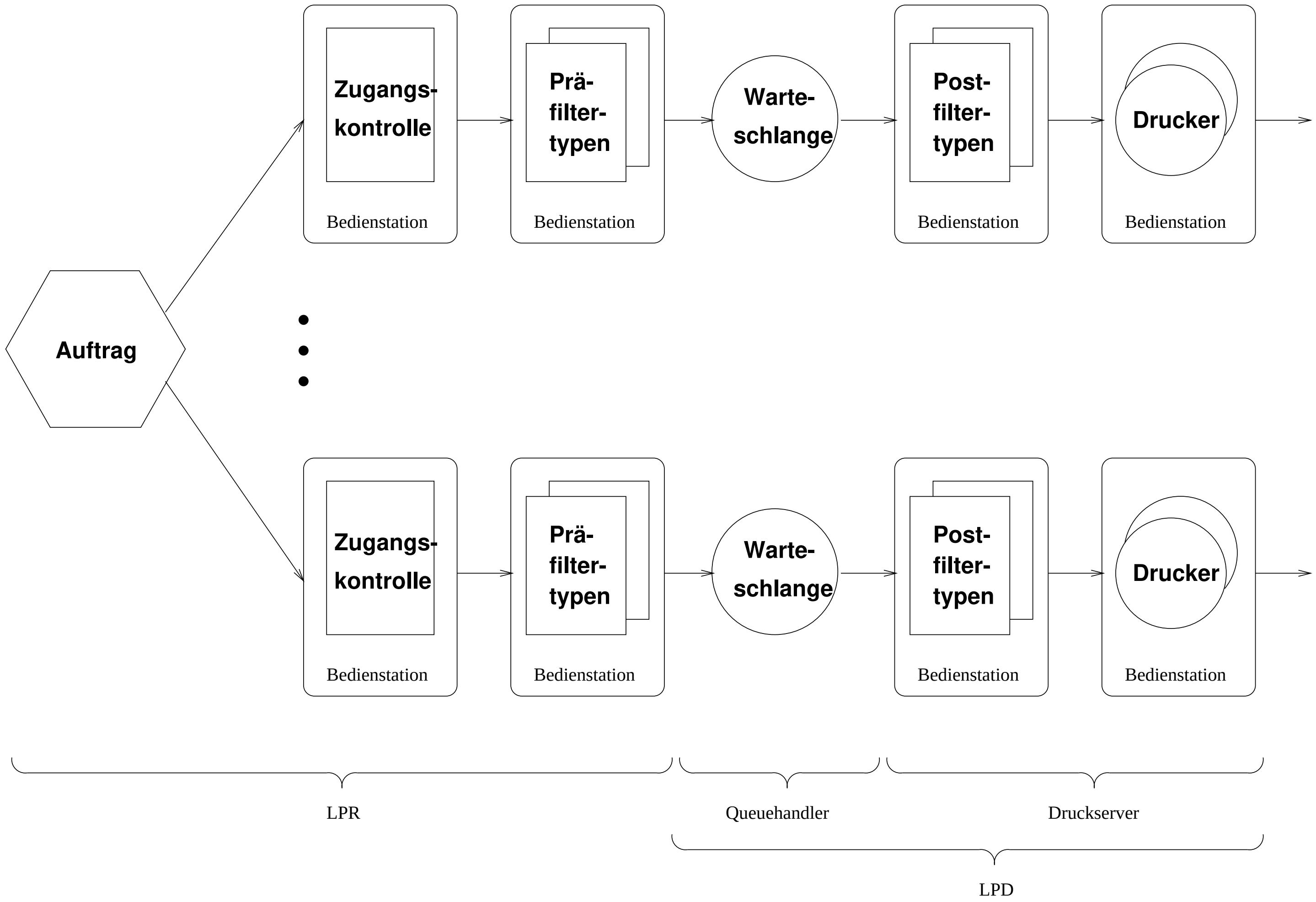

Abbildung 3.1: Ein funktionales Modell des Drucksystems

- Daten
  Als Daten, die gedruckt werden sollen, werden gewöhnlich eine oder mehrere Dateien angegeben, in denen die Daten enthalten sind. Allerdings ist es auch möglich die Daten dem Auftrag direkt zuzuführen (Pipe).

- Druckername
  Da in einem Drucksystem mehrere Drucker vorhanden sein können, muß angegeben werden auf welchem Drucker der Auftrag gedruckt werden soll. Gibt der Auftraggeber keinen Druckernamen an, wird ein fest eingestellter Drucker gewählt. Da jeder Drucker einer Warteschlange zugeordnet ist, entscheidet der Druckername in welcher Warteschlange der Auftrag abgelegt wird. Tatsächlich ist der Warteschlangenname identisch mit dem Druckernamen und sind einer Warteschange mehrere Drucker zugeordnet, können diese nicht mehr einzeln angesprochen werden. Es kommen dann alle Aufträge für die angeschlossenen Drucker in die eine Warteschlange. Jedoch wird hier nicht mehr unterschieden, für welchen Drucker der Auftrag bestimmt war, er wird auf einem beliebigen der Warteschlange zugeordneten Drucker gedruckt werden.

- Priorität
  Jeder Auftrag muß eine Priorität haben, wenn er in die Warteschlange kommt. Es gibt Prioritäten von A - Z, wobei A die höchste Priorität ist. Fehlt die Prioritätsangabe bei der Auftragserstellung, erhält der Auftrag die Priorität Z. Der Systemverwalter hat noch die Möglichkeit die maximale Priorität, die ein Auftraggeber fordern kann, zu begrenzen.

- Datenformat
  Die zu druckenden Daten müssen nicht notwendigerweise im ASCII– oder Postscript–Format vorliegen. Stammen die Daten z.B. von TeX, troff oder plot, muß dies angegeben werden. Mit Hilfe dieser Angabe kann das Drucksystem entscheiden, welche Filter zur Konvertierung es aufrufen muß.

  Das PLP Drucksystem bietet noch ein spezielles Feature, welches andere Drucksysteme nicht haben. Wird angegeben, daß die Dateien Daten in binärer Form enthalten, kann das Drucksystem zum Dateitransfer mißbraucht werden. Denn diese Dateien können nicht gedruckt werden, was ja auch gar nicht beabsichtigt ist. Um die Datei auf einen anderen Rechner zu transferieren wird ein aktiver PLP Druckdämon auf dem entfernten Rechner vorausgesetzt, der die übermittelten Dateien annehmen und abspeichern kann. Diese Form der Datenübertragung macht allerdings nur Sinn, wenn keine anderen Übertragungsprogramme, wie z.B. ftp, zur Verfügung stehen.

Vom Server ermittelte Daten

- Datengröße
  Dieses Attribute gibt die Anzahl der Bytes an, die gedruckt werden müssen.

- Auftraggeber
  Die Benutzerkennung der Person, die diesen Auftrag erzeugt hat. Diese Angabe ist interessant für die Zugangsberechtigung, zur Kostenabrechung und der Kennzeichnung des Ausdrucks. Soll auch der richtige Name des Auftraggebers auf dem Deckblatt erscheinen, wird dieser aus der */etc/passwd* Datei ermittelt.

- Rechnername
  Der Name des Rechners, von dem aus der Auftrag abgeschickt wurde.

Dem Auftrag zugewiesene Attribute
Diese Informationen werden dem Auftrag erst beim Einreihen in die Warteschlange zugeordnet.

- Auftragsnummer
  Wenn der Auftrag in die Warteschlange eingereiht wird, erhält er eine Auftragsnummer. Diese Nummer liegt im Bereich von 1 – 999. Anhand dieser kann der Auftrag im Drucksystem gezielt angesprochen werden.

- Zeit und Datum
  Es ist auch wichtig, die Uhrzeit und das Datum zu wissen, wann genau der Auftrag in die Warteschlange eingereiht wurde.

Optionen für die Auftragsbearbeitung

- Datei löschen
  Nachdem der Auftrag, die Dateien zu drucken, ausgeführt wurde, sollen neben den Dateien in der Warteschlange auch die Orginaldateien beim Auftraggeber gelöscht werden. Aus Sicherheitsgründen ist diese Option nur privilegierten Benutzern gestattet.

- Auftragsendemitteilung
  An den Auftraggeber wird eine Nachricht (Mail) geschickt, sobald der Druckauftrag durchgeführt wurde. Der Auftraggeber kann jedoch auch einen anderen Empfänger der Nachricht angeben.

Optionen für die Filter

- Aufbereitungsart
  Wünscht der Auftraggeber, daß sein Auftrag bereits einen Filter durchläuft bevor der Auftrag in die Warteschlange gelangt, muß er dies ausdrücklich angeben. Allerdings kann nur das Filterprogamm 'pr(1)' aufgerufen werden. Zu beachten ist hierbei

der Unterschied zu den Filterprogrammen des Drucksystems, auf dessen Aufruf der Auftraggeber keinen Einfluß hat. Allerdings können den Filterprogrammen einige Optionen mitgegeben werden.

  - Deckblatt oder Kopfzeile
    Üblicherweise wird bei jedem Ausdruck eines Auftrags ein Deckblatt oder wenigstens eine Kopfzeile erzeugt. Das Aussehen des Deckblattes bzw. der Kopfzeile kann vom Auftraggeber beeinflußt werden, er kann den Ausdruck dieser Angaben sogar unterbinden.
  - Seitenbreite
    Auch die Anzahl der Zeichen pro Zeile, mit der dieser Auftrag formatiert werden soll, kann angegeben werden. Diese Angabe wird von ‘pr(1)’ übernommen. Das Programm ‘pr(1)’ kann allerdings nur Daten im ASCII Format bearbeiten.
  - Titel
    Der Ausdruck kann mit dem angegeben Titel versehen werden. Dieser Titel wird von ‘pr(1)’ übernommen.
  - Auftragsname
    Dem Auftrag kann ein Name zugewiesen werden, dieser Name erscheint auf dem Deckblatt des Ausdruckes.
  - Auftraggeber
    Der vollständige Name des Benutzer, der diesen Auftrag erteilt hat, soll auf dem Ausdruck erscheinen. Dieser Name wir aus der */etc/passwd* Datei ermittelt.

- andere Benutzerkennung
  Privilegierten Benutzern, die möglicherweise mehrere Kennungen in der Domäne besitzen, ist es gestattet für die Kostenabrechung des Auftrags eine andere Benutzerkennung anzugeben.

- weitere Optionen
  Hierdurch kann man dem Eingangsfilter des Druckers weitere Optionen mitgeben. Dies wird hauptsächlich benötigt beim Übertragen auf nicht-Unix Drucker und beim Dateitransfer auf nicht-Unix Rechner.

Optionen für den Drucker
Diese Optionen für den Drucker sind nur gestattet, wenn der Drucker diese Auftragsberarbeitung auch unterstützt. Es gibt allerdings nur wenige Drucker, die dies tun.

- Kopieanzahl
  Möchte der Auftraggeber mehrere Ausdrucke seiner Daten, kann er in der Auftragsbeschreibung angeben, wieviele Kopien er wünscht.

- Einrücktiefe
  Soll der Text nicht direkt am linken Rand gedruckt werden, kann man angeben, um wieviele Zeichen der Drucker den Text einrücken soll.

3.1.1.2 Zugangskontrolle

Bevor nun der erstellte Auftrag in die Warteschlange eingereiht wird, muß kontrolliert werden, ob überhaupt das Recht dafür besteht. Nicht jeder Benutzer darf Aufträge in jede Warteschlange einfügen, außerdem kann sein Druckkontingent bereits erschöpft sein. Aber auch die Auftragsoptionen müssen überprüft werden.

- Druckkontingent
  Für jede Warteschlange kann angegeben werden, wieviele Seiten der Benutzer auf den zugehörigen Druckern drucken darf. Dabei wird die bereits gedruckte Seitenanzahl und die maximale Seitenanzahl gespeichert.

- Zugriffsrechte
  Für jede Warteschlange im Drucksystem müssen Zugriffsrechte vergeben werden. Die Festlegungen werden für alle verfügbaren Warteschlangen in einer speziellen Datei, der *printer_perms* Datei, gespeichert. Desweiteren lassen sich noch zusätzliche Zugriffsrechte bei jeder einzelnen Warteschlange angeben, was aber nicht üblich ist.

  – Rechner
    Die Rechner, von denen aus Aufträge in die Warteschlange geschrieben werden dürfen, lassen sich einschränken. Dabei ist es natürlich möglich, mit Hilfe von Jokerzeichen, mehrere Rechner zusammenzufassen. So lassen sich z.B. alle Rechner einer Domäne unter Angabe des Domänennamens zusammenfassen.

  – Auftraggeber
    Jedem Auftraggeber, der seinen Auftrag in die Warteschlange schreiben will, muß das Recht der Auftragserteilung eingeräumt sein. Dabei kann jeder Auftraggeber einzeln angegeben werden, es lassen sich aber auch Gruppen von Auftraggebern spezifizieren.

- erlaubte Optionen
  Manche Optionen, die dem Auftrag mitgegeben werden können, sind den privilegierten Benutzern vorbehalten. An dieser Stelle muß untersucht werden, ob sich Optionen dieser Art in der Auftragsbeschreibung befinden, und wenn ja, ob der Auftraggeber ein privilegierter Benutzer ist.

- Auftragsformat
  Nicht alle Dateiformate können von den der Warteschlange zugeordneten Druckern verarbeitet werden. Die Aufträge, die in die Warteschlange eingereiht werden, müssen

deshalb ein bestimmtes Format haben. Welche Formate erlaubt sind, ist dem Drucksystem bekanntzugeben. Einige Formate setzen voraus, daß die angegebenen Dateien druckbar sind (z.B. nur ASCII–Zeichen in der Datei), was ebenfalls überprüft werden muß.

- maximale Auftragsgröße
  Hiermit wird angegeben, wie groß der Auftrag maximal sein darf, damit er noch bearbeitet werden kann. Als Auftragsgröße wird dabei die Datengröße mal der Kopienanzahl gerechnet.

- maximale Anzahl der Kopien
  Die Anzahl der maximal erlaubten Kopien eines Ausdrucks kann begrenzt sein, deshalb muß an dieser Stelle überprüft werden, wieviele Kopien der Auftrag wünscht.

3.1.1.3 Präfilter

Wenn ein Auftrag in die Warteschlange geschrieben wird, hat er ein bestimmtes Auftragsformat. Anhand dieses Formats wird entschieden, welche Filter aufgerufen werden und welches Format der Auftrag anschließend hat.

Den Aufruf der Filterprogramme mit den nötigen Parametern übernimmt das Programm 'lpr'. Weiterhin kann man noch angeben, welche Datenformate auf nichtdruckbare Zeichen überprüft werden sollen, bevor sie in die Warteschlange eingereiht werden.

3.1.1.4 Warteschlange

In der Warteschlange werden die Aufträge gesammelt und sortiert, damit sie an die folgende Bedienstation weitergegeben werden können, sobald die Bedienstation für einen neuen Auftrag bereit ist.

Eigenschaften der Warteschlange

- Warteschlangenname
  Die Warteschlange kann häufig unter mehreren Namen angesprochen werden. Als Name der Warteschlange wird jedoch der erste in der *printcap* Datei angegebene Name verwendet. Ist der Warteschlange nur ein Drucker zugeordnet, lautet auch der Name des Druckers so, gibt es mehrere Drucker, haben diese bei der Administration eigene Namen.

  Der Grund, warum man für ein Warteschlange mehrere Namen vergeben kann, liegt darin, daß man sowohl lange aussagekräftige als auch kurze Namen vergeben sollte.

- Zustand der Warteschlange

  - Warteschlangeneingang
    Der Eingang der Warteschlange läßt sich sperren, damit sich keine neuen Aufträge mehr in die Warteschlange einreihen lassen. Auf bereits in der Warteschlange befindliche Auftäge hat das keinen Einfluß.
  - Warteschlangenausgang
    Die Abarbeitung der Aufträge aus der Warteschlange kann unterbunden werden. Desweiteren kann auch der Prozeß, welcher für die Bearbeitung der Aufträge zuständig ist, abgebrochen oder neu gestartet werden. Ein Neustart ist z.B. bei Problemen bei der Bearbeitung eines Auftrags nötig.

Aufträge in der Warteschlange

- Kontrolldatei
  In dieser Datei befinden sich die nötigen Informationen zum Drucken des Auftrags. Neben den Angaben zum Auftrag, die bei der Auftragserstellung mit angegeben wurden, sind darin die Namen der Datendateien vermerkt.
- Datendateien
  In diesen Dateien befinden sich die zu druckenden Daten. Für jede zu druckende Datei wird eine eigene Datendatei angelegt. Allerdings kann anstelle der Datendatei auch ein Verweis (Link) auf die Orginaldatei existieren.
- Warteschlangenverzeichnis
  Darin werden warteschlangenspezifische Dateien und die Auftragsdateien abgelegt. Zu jedem Eintrag in der *printcap* Datei, welcher eine Warteschlange definiert, muß ein Warteschlangenverzeichnis existieren.
- Löschen von Aufträgen
  Jeder Benutzer, der das Recht hat Aufträge in diese Warteschlange zu schicken, kann seine Aufträge auch wieder aus der Warteschlange entfernen. Privilegierte Benutzer können sogar beliebige Aufträge aus der Warteschlange entfernen. Dabei haben sie verschiedene Möglichkeiten zur Auswahl der zu löschenden Aufträge. Sie können alle Aufträge der Warteschlange, alle Aufträge eines Benutzers oder nur ein bestimmten Auftrag, anhand der Jobnummer, löschen. Sogar bereits in Bearbeitung befindliche Aufträge können abgebrochen und der Rest des Auftrags aus der Warteschlange entfernt werden.

Bedienstrategie

- Auftragsreihenfolge
  Ankommende Aufträge werden anhand ihrer Priorität in der Warteschlange eingereiht. Bei Aufträgen mit gleicher Priorität entscheidet die Zeit der Ankunft der Auf-

träge in der Warteschlange. Allerdings können privilegierte Benutzer die Reihenfolge der Aufträge in der Warteschlange verändern.

Die Auswahl des nächsten zu druckenden Auftrags übernimmt das Warteschlangensteuerungsprogamm (Queuehandler). Dieses ist bereits fest im Druckdämon kodiert, kann aber durch die Angabe eines neuen Warteschlangensteuerungsprogramms außer Kraft gesetzt werden. Allerdings ist es nur dem Systemadministrator gestattet die Bedienstrategie zu verändern.

- maximale Wiederholungen
  Scheitert die Bearbeitung eines Auftrags beim Drucker, wird der Auftrag in die Warteschlange zurückgestellt und es wird später erneut versucht ihn zu bearbeiten. Jedoch sollte ein Neustart nicht beliebig oft wiederholt werden. Gelingt die vollständige Bearbeitung des Auftrags auch nach mehreren Versuchen nicht, wird er gelöscht. Die Anzahl der Versuche läßt sich festlegen.

3.1.1.5 Postfilter

Es gibt verschiedene Filtertypen, einige zur Konvertierung des Datenformats, andere zur Formatierung des fertigen Ausdruckes und manche zur Ermittlung der Abrechnungsinformationen (Accounting). Ein Filterprogamm kann natürlich mehrere Aufgaben gleichzeitig übernehmen, doch ist dies nur von der Implementierung abhängig. Die Konvertierung der Datenformate ist bei Drucksystemen nötig, die mehrere Datenformate annehmen können, deren zugeordnete Drucker jedoch nur ein bestimmtes Format akzeptieren.

- Druckereingang
  Nur wenn der Druckdämon einen Druckserver für einen Drucker starten kann, erhält dieser Aufträge. Ob ein Druckserver gestartet werden kann, hängt ab, ob der Drucker Aufträge annehmen will oder nicht, was sich administrieren läßt. Auf diese Weise kann man einen Drucker von der Warteschlange nehmen bzw. ihn wieder aktivieren. Allerdings gibt es diese Möglichkeit nur, wenn mehrere Drucker einer Warteschlange zugeordnet sind.

- Abrechnungsinformationen
  Da jeder Druckauftrag Kosten verursacht, diese Kosten von den Verursachern aber wieder verlangt werden müssen, ist es nötig die Anzahl der gedruckten Seiten je Benutzer zu erhalten. Bevor nun der Auftrag an den Drucker gesendet wird, kann ermittelt werden, wieviele Seiten der Ausdruck benötigt. Diese Information wird dann in einer Datei gesichert.

- Form des Ausdrucks
  Gewöhnlich wird für jeden Auftrag ein Deckblatt gedruckt. Welche Informationen

darauf erscheinen ist vorkonfiguriert oder wird in der Auftragsbeschreibung angegeben. Ein Filterprogramm übernimmt die Aufgabe, diese Informationen in eine geeignete Form zu bringen und dem Druckauftrag mitzugeben.

Neben dem Deckblatt gibt es noch Kopfzeilen, Anzahl der Zeichen pro Zeile, Seitenlänge, Zeilennummerierung, usw. die bei dem Auftrag berücksichtigt werden können.

- Datenformat für den Drucker
  Jeder Auftrag, der in die Warteschlange kommt, hat ein bestimmtes Format (ASCII, TeX, troff, ...), woraus sich auch die weiteren Bearbeitungsschritte ableiten lassen. Die Auftragsformate in einer Warteschlange lassen sich zwar einschränken, sind aber in der Regel immer noch mehr, als der Drucker verarbeiten kann. Deshalb müssen die Aufträge, bevor sie zum Drucker kommen, noch konvertiert werden. Welche Filter dafür zuständig sind entscheidet das Auftragsformat. Desweiteren gibt es vielleicht noch geräteabhängige Anforderungen an den Auftrag, die jetzt berücksichtigt werden müssen.

3.1.1.6 Bedienstation, -einheit „Drucker“

Für jeden der Warteschlange zugeordneten Drucker wird ein Druckserver gestartet. Dieser ist für die Bearbeitung der Aufträge aus der Warteschlange zuständig. Dazu holt er mit Hilfe des Warteschlangensteuerungsprogramms den nächsten Auftrag aus der Warteschlange, startet die entsprechenden Filter und leitet die Daten zum Drucker oder in eine andere Warteschlange.

Den meisten Warteschlangen ist mindestens eine Bedieneinheit zugeordnet, die die Aufträge aus der Warteschlange abarbeiten kann. Im Drucksystem entspricht eine Bedieneinheit einem Drucker, Plotter oder ähnlichem, es kann aber auch ein Übertragungsprogamm sein, daß den Auftrag in eine andere Warteschlange einreiht. Diese Warteschlange muß auch nicht notwendigerweise im gleichen Drucksytem, sondern kann genauso in einem anderen Drucksytem liegen, zu dem eine Verbindung aufgebaut werden kann.

Dazu muß jedoch gesagt werden, ob das Ziel nun eine Warteschlange oder ein Drucker ist, beide Bearbeitungsverfahren sind sich in ihren Aufgaben und ihrem Verhalten sehr ähnlich. Beide bauen eine Verbindung zum Ziel auf und schicken die Daten über diese Verbindung. Nur die Bedienstation „Warteschlange“ schickt zusätzlich noch die Auftragsbeschreibung mit, wohingegen die Bedienstation „Drucker“ gerätespezifische Kommandos mitschickt.

Es ist aber auch möglich, mehrere Bedieneinheiten für die Aufträge einer Warteschlange bereitzustellen, allerdings sollten diese Bedieneinheiten ähnliche Eigenschaften aufweisen, z.B. alle Drucker, die Daten im Postscriptformat verarbeiten können. Diese Bedieneinheiten lassen sich nun zu einer Bedienstation zusammenfassen und gemeinsam administrieren. Im Drucksystem wird den einzelnen Druckern aber ein so hohe Bedeutung beigemessen, daß sie auch noch einzeln administriert werden können, falls mehrere Drucker einer Warteschlange zugeordnet sind.

- Art der Verbindung
  Die ersten beiden angegebenen Arten beziehen sich eine andere Warteschlange als Auftragsziel, die restlichen auf einen Drucker.
  - andere Warteschlange
    Soll der Auftrag an eine andere Warteschlange weitergereicht werden, wird nur der Name der anderen Warteschlange und der Rechnername angegeben. Bei einer Warteschlange im gleichen Drucksystem reicht der Warteschlangenname. An die angegebene Warteschlange werden dann alle Aufträge geschickt.
  - NFS
    Ein gemeinsames Warteschlangenverzeichnis, das für alle Drucksysteme in der Domäne zugänglich ist, erspart das Kopieren der Daten von einer Warteschlange zur anderen. Der Dämon, welcher für die Warteschlange zuständig ist, erhält eine Nachricht, sobald ein neuer Auftrag in die Warteschlange geschrieben wurde.
  - Ausgabeschnittstelle
    Die Schnittstelle an der die Bedieneinheit „Drucker“ angeschlossen ist und an die die Daten des Auftrags geschickt werden. Bei seriellen Verbindungen können noch die Eigenschaften der Verbindung festgelegt werden, wie Baudrate, Parity, etc.
  - Annex Terminal Server
    Die Druckaufträge können auch an einen Annex Terminal Server gesendet werden, was durch Öffnen eines Sockets zum Server und Senden eines einfachen Kommandos, welches die Ausgabeschnittstelle (Outputline) des Servers festlegt, geschieht.
  - TCP/IP Socket
    Zu Geräten, die nur am Netzwerk auf Verbindungen mithören und alle empfangenen Zeichen auf dem Drucker ausgeben.
  - mehrere Drucker an einer Warteschlange
    Eine Warteschlange kann auch mehrere Drucker bedienen, diese Drucker müssen angegeben und noch durch je einen Eintrag in der *printcap* Datei genauer spezifiziert werden.
- Status des Druckservers
  In einer Datei speichert das Drucksystem den Status des Druckservers. Er ist für die Bearbeitung des Auftrags von der Warteschlange bis zum Drucker zuständig. Wenn man sich über den Inhalt der Warteschlange informiert, wird auch diese Information angezeigt.

- Bedieneinheitname
  Sind einer Warteschlange mehrere Drucker zugeordnet, erhält jeder Drucker einen eigenen Namen, sonst übernimmt die Bedieneinheit den Namen der Warteschlange.

3.1.1.7 Konfiguration

Bisher haben wir uns immer auf eine Warteschlange mit den vor- und nachgestellten Komponenten beschränkt. Es muß aber auch angegeben werden, wieviele und welche Warteschlangen sich im Drucksystem befinden und wie die Drucker auf die Warteschlangen verteilt sind. Diese Angaben befinden sich in der *printcap* Datei des Drucksystems, dort gibt es für jede Warteschlange einen Eintrag, der die zugehörigen Bedienstationen mit den Bedieneinheiten spezifiziert. Die mögliche Konfiguration der einzelnen Komponenten ist bereits bei der Beschreibung der Komponenten angegeben.

Außerdem gibt es noch eine Ereignisaufzeichungsdatei (Log–File), in die das Drucksystem Meldungen während der Bearbeitung von Aufträgen schreibt. Diese ist besonders hilfreich bei auftretenden Fehlern im Drucksystem, denn damit kann der Ablauf der Bearbeitung mit gewissen Einschränkungen nachvollzogen werden.

3.1.2 Das druckerspezifische Objektmodell

Nachdem die Bearbeitungsabläufe im Drucksystem bekannt sind und ein funktionales Modell existiert, deren Komponentenbezeichnungen bereits an das Objektmodell angelehnt sind, wird für den Druckdienst nun das Objektmodell entwickelt.

3.1.2.1 Komponenten

Wenn man sich das funktionale Modell betrachtet, erkennt man vier verschiedene Bedienstationtypen, dies sind die Zugangskontrolle, Drucker, Prä- und Postfilter. Weiter gibt es einen Warteschlangentyp und einen Auftragstyp. Jeder Zweig des funktionalen Modells läßt sich aus diesen Typen bilden, die Anzahl der Zweige ist frei.

Dem Auftrag können viele verschiedene Optionen mitgegeben werden, doch existiert eine Grundmenge, die alle Aufträge haben müssen und auf die wir uns beschränken. Die weiteren Optionen dienen nur der genaueren Auftragsbeschreibung bzw. den Bearbeitungswünschen. Für jede Bedienstation ist ein eigener Bedieneinheitentyp nötig, wobei noch zu sagen ist, daß bei der Bedienstation „Drucker“ die Aufgaben der Bedieneinheiten gleich sind, nur die Art der Aufgabenerfüllung sich etwas unterscheidet. Ihre Aufgabe ist das Übertragen der Druckdaten über eine Verbindung, wobei noch Verbindungsparameter und Übertragungsprotokolle berücksichtigt werden müssen. Dabei wird aber nicht gesagt, welche Art von Verbindung vorhanden ist, es kann z.B. eine serielle Verbindung zum Drucker oder eine TCP–Verbindung zu einer anderen Warteschlange sein.

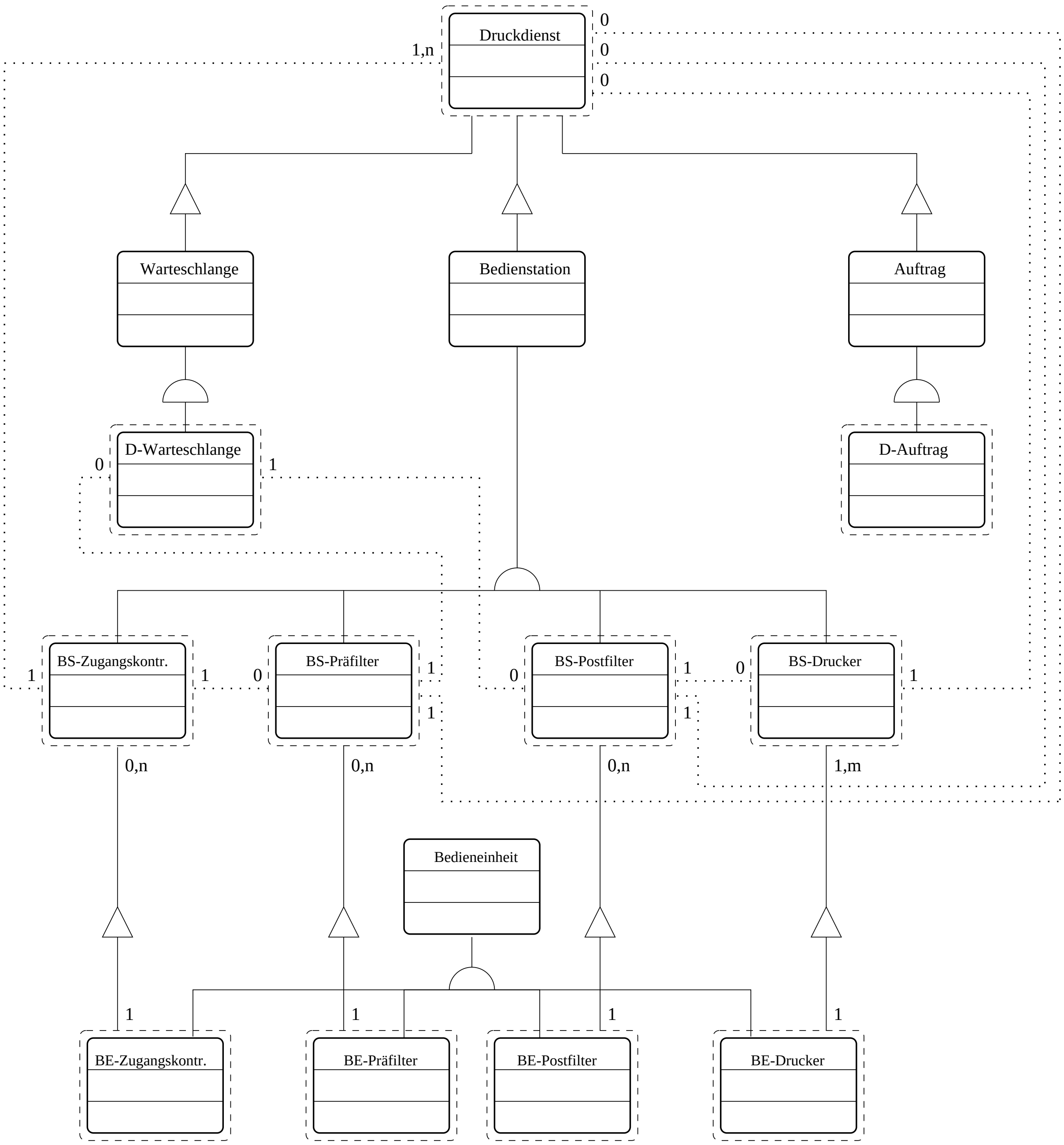

Abbildung 3.2: Das Objektmodell des Drucksystems

Das Objektmodell besagt, daß jede Bedienstation wenigstens eine Bedieneinheit enthalten muß, die den Bearbeitungsschritt ausführt, für welchen die Bedienstation verantwortlich ist. Bei den Bedienstationen Zugangskontrolle, Prä- und Postfilter gibt es jeweils einen Bedieneinheitstyp, der dynamisch erzeugt werden kann und nach Erledigung seiner Aufgabe wieder aufgelöst wird. So kann für jeden ankommenden Auftrag eine Bedieneinheit erzeugt werden, womit sich die Anzahl der gerade vorhandenen Bedieneinheiten nach der Anzahl der gerade in der Bedienstation befindlichen Aufträge richtet. Anders bei der Bedieneinheit „Drucker". Die Anzahl dieses Bedieneinheittyps ist festgelegt durch die Anzahl der zur Verfügung stehenden Drucker bzw. es gibt genau eine Bedieneinheit, die Aufträge an eine andere Warteschlange weiterleitet.

Zusammenfassend kann man sagen, in der Funktionseinheit „Druckdienst" sind die Klassen-&-Objekte Zugangskontrolle, Präfilter, Postfilter und Drucker sowie die Warteschlange und der Auftrag enthalten. Die Bedienstationen enthalten jeweils eine Klasse-&-Objekte „Bedieneinheit".

3.1.2.2 Beziehungen

Dabei erben alle Bedienstationen von der abstrakten Klasse „Bedienstation" die generischen Attribute und Methoden bereitstellt, die im folgenden Kapitel definiert werden. Entsprechend bei den Bedieneinheiten, der D–Warteschlange und dem D–Auftrag. Diese Verfeinerung ist nötig, um dienstspezifische Unterschiede bei den Objekten berücksichtigen zu können. Auch die Enthaltensein–Beziehung zwischen den abstrakten Klassen „Bedienstation" und „Bedieneinheit" ist davon betroffen. Das Einzeichnen dieser Beziehung wird an dieser Stelle unterlassen und erfolgt dafür zwischen den abgeleiteten Komponenten.

Im Objektmodell sind auch bereits die Beziehungen — die gepunkteten Linien — der Komponenten zueinander eingezeichnet, welche den Weg des Auftrags durch die Funktionseinheit darstellen. Daraus ist ersichtlich, daß alle Aufträge für das Drucksystem bei der BS–Zugangskontrolle ankommen. Anschließend folgen die BS–Präfilter, D–Warteschlange, BS–Postfilter und am Schluß die BS–Drucker. Da der Auftrag in jeder Bedienstation beendet werden kann, bestehen auch Beziehungen von allen Bedienstationen zur Funktionseinheit.

Was im Objektmodell nicht ohneweiters erkennbar ist, ist die gleiche Anzahl von Objekten von den Bedienstationklassen und der Warteschlangenklasse. Auch die Zusammengehörigkeit von jeweils einer Bedienstation von den Bedienstationtypen und einer Warteschlange, die einen Zweig im funktionalen Modell bilden, ist nicht leicht ersichtlich. Allerdings ist diese Information in den Beziehungen zwischen den Komponenten des Objektmodells enthalten. Denn die Beziehungen sind zwischen den erzeugten Objekten definiert und nicht zwischen den Klassen.

3.1.2.3 Managementinformationszuordung

Die Managementinformation und -funktionalität der Komponenten im Objektmodell unterscheidet sich etwas von den Angaben bei den Komponenten des funktionalen Modells. So werden die beim Auftrag angegeben Daten und Optionen noch um die Möglichkeit einen Auftrag wieder zu löschen erweitert. Außerdem sind die Kontroll- und Datendatei des Auftrags bekannt.

Bei der BS–Zugangskontrolle und bei der BS–Präfilter kann die Information vom funktionalen Modell direkt übernommen werden. Dies sind bei der BS–Zugangskontrolle die erlaubten Optionen bzw. Grenzwert und die Benutzer und Rechner, die Aufträge in die Warteschlange einreihen dürfen. Bei der BS–Präfilter sind es die vorhandenen Filterprogramme und zu überprüfenden Datenformate.

Die Warteschlange im Objektmodell umfaßt nicht alle Informationen und Funktionen der Warteschlange im funktionalen Modell. Es verbleiben der Warteschlangenname, -verzeichnis, -eingang, enthaltene Aufträge und die Bedienstrategie.

Dagegen können die Informationen für die BS–Postfilter wieder übernommen werden, was der Postfiltereingang, die Filterprogramme und das Datenformat, in welches der Auftrag konvertiert wird, sind.

Bei der Bedienstation „Drucker“ ist angegeben, welche Bedieneinheiten enthalten und von welchem Typ sie sind. Für die Bedieneinheiten der Bedienstation gibt es noch den Namen, Status und Bedieneinheiteingang.

3.1.2.4 Anmerkung

Die Abbildung des Drucksystems auf das Objektmodell läßt sich relativ leicht vollziehen. Das Objektmodell beim Druckdienst ist auch in der Lage alle wesentlichen Informationen aufzunehmen, wobei häufig sogar die Informationen der funktionalen Komponenten auf die Objekte des Objektmodell direkt übertragen werden. Nur wenige Informationen müssen einem anderen Objekt zugeteilt werden. Das Objektmodell ist also gut geeignet den Druckdienst darzustellen.

3.2 Managementinformation und -funktionalität des Nachrichtensystems

Sendmail, Version 8.6.9 von Eric P. Allman von der University of California, Berkeley, ist ein Programm, daß im Internet die Zustellung von Nachrichten übernehmen kann. Es wird gewöhnlich nicht direkt aufgerufen, sondern von einem Programm, wie '/bin/mail' oder

'/bin/elm', daß den Benutzer beim Lesen, der Beantwortung oder Erstellung einer Nachricht unterstützt. Zur Zustellung der Nachricht an den Empfänger wird Sendmail bemüht. Die Aufgaben von Sendmail bei der elektronischen Nachrichtenzustellung sind mit denen der Post bei der Briefzustellung vergleichbar. Sendmail übernimmt also nur die Weiterleitung der Nachricht vom sendenden Rechner zum Zielrechner. Dort wird die Nachricht von einem Sendmail–Dämon empfangen und durch den Aufruf eines geeigneten Programms, wie z.B. '/bin/mail' oder '/bin/deliver', dem Empfänger zugestellt. Ist der sendende Rechner mit dem empfangenden Rechner nicht über das Internet verbunden, sondern über UUCP, kann Sendmail die Übermittlung der Nachricht nicht selbst durchführen, sondern muß das Programm 'uux' zur Übermittlung der Nachricht aufrufen. Entsprechend verfährt Sendmail bei anderen Netzwerkprotokollen oder lehnt die Zustellung der Nachricht ab. Manchmal ist es allerdings nicht möglich oder nötig die Nachricht sofort weiterzuleiten, in diesen Fällen wird die Nachricht in eine Warteschlange geschrieben. Die Bearbeitung der Warteschlange übernimmt ein Sendmail–Dämon, der in zeitlich festlegbaren Abständen versucht die in der Warteschlange befindlichen Nachrichten zuzustellen. Gewöhnlich ist dieser Sendmail–Dämon auch für den Empfang von Nachrichten von anderen Rechnern zuständig, dies kann allerdings auch von einem zweiten Sendmail–Dämon übernommen werden.

3.2.1 Untersuchung des Nachrichtensystems

Anhand des Ablaufs der Bearbeitung eines Nachrichtenauftrags läßt sich ein funktionales Modell aufstellen. Dieses, in Abbildung 3.3 gezeigte Modell, wird im folgenden genauer beschrieben und es werden ihm die Managementinformation und -funktionalität des Nachrichtensystems zugeteilt. Zur Findung dieser Informationen dienten vor allem [Cos 93] und die Dokumentation (Manualpages) der Programme von Sendmail.

3.2.1.1 Auftrag

Sobald eine Nachricht mit Hilfe eines geeigneten Programms erstellt wurde, wird sie an Sendmail als Auftrag übergeben. Dabei müssen einige Informationen an Sendmail mit übergeben werden, andere werden von Sendmail an diesen Auftrag gebunden. Der Auftrag besteht schließlich aus der Empfängerliste, dem Kopf (Header; den Informationen) und der Nachricht (Body) selbst.

Benötigte Angaben

- Absenderadresse
  Bei Übergabe des Auftrags an Sendmail erhält es die Benutzerkennung des Auftraggebers. Zusammen mit dem Domänennamen des Rechners ergibt dies die Absenderadresse. Diese wird zweimal im Kopf des Auftrags vermerkt, in der From–Zeile

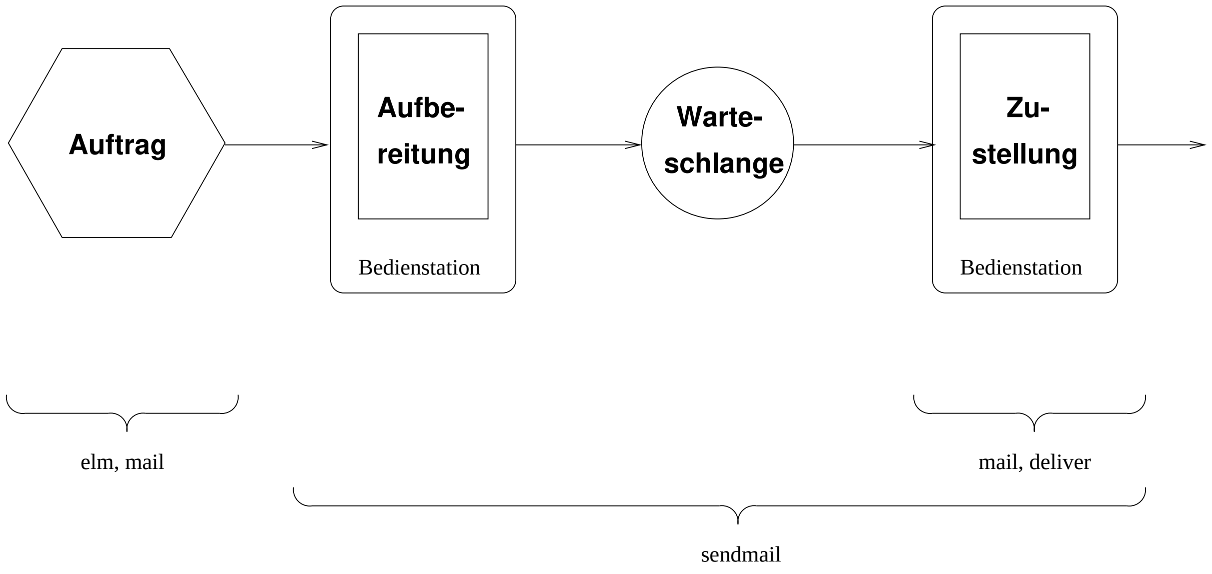

Abbildung 3.3: Das funktionale Modell des Nachrichtendienstes

und der Return–Path–Zeile. Sollte der Empfänger der Nachricht ein lokaler Benutzer sein, kann auch auf die Angabe der Domäne verzichtet werden. Mit Hilfe dieser Angabe kann der Empfänger der Nachricht oder Sendmail selbst eine Nachricht an den Absender zurückschicken. Durch eine entsprechende Konfiguration von Sendmail kann auch noch der richtige Name des Absenders, so wie er in der */etc/passwd* Datei eingetragen ist, dem Auftrag mitgegeben werden. Die Absenderadresse besteht also aus

- der Benutzerkennung,
- der Domäne des sendenden Rechners und
- dem richtigen Namen des Absenders.

• Empfängerliste
Es können beliebig viele Empfänger der Nachricht angegeben werden, wodurch jeder angegebene Empfänger eine Kopie der Nachricht erhält. Hierzu muß für jeden Empfänger die Benutzerkennung und, falls er kein lokaler Benutzer ist, die Domäne angegeben werden. Dabei kann man auch Aliases verwenden, was die Empfängerangabe vereinfacht und verkürzt, diese müssen von Sendmail aber expandiert werden. Somit besteht die Empfängerliste für jeden Empfänger aus

- der Benutzerkennung des Empfängers und
- der Domäne des empfangenden Rechners
- oder aus einem Alias.

Dem Auftrag zugewiesene Attribute

- Nachrichtenlänge
  Die Länge der Nachricht (Body) in Bytes kann vom Auftraggeber im Kopf des Auftrags (Content–Length–Zeile) angegeben werden. Meist fehlt diese Angabe jedoch, weshalb Sendmail die Größe der Nachricht ermittelt und im Kopf des Auftrags einträgt.

- Zeit und Datum
  Im Auftrag wird auch vermerkt, wann der Auftrag zur Zustellung an Sendmail übergeben wurde. Es besteht zwar die Möglichkeit, daß der Absender die Zeit und das Datum selbst angibt, allerdings sollte das nicht die Regel sein.

- Auftragskennung
  Bei Übergabe des Auftrags an Sendmail wird dem Auftrag eine Auftragskennung zugeteilt, welche weltweit eindeutig sein soll. Um dies zu erreichen, setzt sich die Kennung gewöhnlich aus der Uhrzeit in Sekunden, der eindeutigen Warteschlangenkennung in der Domäne und dem Namen der Domäne des Rechners zusammen.

Optionen

- zusätzliche Kopfzeilen
  Für den Absender der Nachricht besteht die Möglichkeit dem Auftrag weitere Informationen mitzugeben. Diese sind nicht nötig, können aber ganz sinnvoll sein. So wird sehr häufig der Betreff (Subject) der Nachricht dem Auftrag mitgegeben. Weitere Beispiele sind die Schlüsselwörter (Keywords) einer Nachricht oder die Kodierungsart (Encrypted), mit welcher die Nachricht verschlüsselt wurde.

- Priorität
  Jeder Auftrag, der an Sendmail übergeben wird, erhält auch eine Priorität. Durch die Priorität wird die Reihenfolge der Bearbeitung der Aufträge in der Warteschlange bestimmt. Es gibt fünf mögliche Prioritäten (special-delivery, first-class, list, bulk, junk), die im Kopf des Auftrags (Precedence) angegeben werden kann. Diesen Prioritäten wird in der Konfigurationsdatei von Sendmail eine Zahl zugewiesen. Fehlt die Prioritätsangabe im Auftrag erhält er standardmäßig die Priorität „first-class“. Die Priorität ist auch entscheidend bei der Fehlerbehandlung. So werden Aufträg mit kleiner Priorität nach einem erfolglosen Zustellungsversuch verworfen, wohingegen Aufträge mit größerer Priorität wieder in die Warteschlange eingebracht werden.

Desweiteren existiert die Möglichkeit, zu verfolgen welche Aufträge bearbeitet wurden sowie einige Informationen von den Nachrichten zu protokollieren. Der Umfang der Ereignisaufzeichnung (Logging) läßt sich bei Sendmail beeinflußen. Dazu bietet Sendmail Ereignisaufzeichnungsstufen an, je niedriger die Stufe, desto weniger Informationen werden protokolliert. Ist die Stufe z.B. auf 9 (LOG_INFO) gesetzt, werden in der Regel für jede verschickte Nachricht drei Zeilen an die Aufzeichnungsdatei (Logfile) angehängt. Jede Zeile beginnt mit dem Datum, der Uhrzeit, dem Rechnernamen und der PID vom Sendmailprogramm, darauf folgt die Kennung der Nachricht, die gerade bearbeitet wird. In der ersten Zeile folgt nun die Message-ID, in der zweiten der Absender, die Nachrichtengröße und die Klasse (Priorität). In der letzten Zeile ist der Empfänger, die Zeitspanne zur Nachrichtenzustellung und der Status des Auftrags vermerkt.

3.2.1.2 Aufbereitung

Sobald Sendmail den Auftrag erhält, eine Nachricht zuzustellen, beginnt es mit der Zerlegung und Untersuchung der Absenderadresse und Empfängerliste. Anschließend wird der Kopf des Auftrags überarbeitet, um zum Schluß den fertig bearbeiteten Auftrag in die Warteschlange schreiben zu können.

- Makrotypen
  Durch einfache Makros kann einem Symbol in der Konfigurationsdatei von Sendmail ein einfacher Text zugewiesen werden. Dadurch ist es möglich den Text einmal zu definieren und ihn an mehreren Stellen in der Konfigurationsdatei, durch Angabe des Symbols, zu verwenden. Einige Makros sind von Sendmail bereits vorbelegt oder müssen vom Systemverwalter definiert werden, andere stehen dem Systemverwalter zur freien Verfügung.

  Um einem Symbol mehrere Strings gleichzeitig zuweisen zu können, gibt es noch Klassenmakros. Ferner existieren Datenbankmakros (Angabe einer Datenbankdatei; hash, dbm, nis), womit Informationen aus einer weiteren Datei geholt werden können.

  Alle diese Makrotypen dienen zur Erleichterung der Konfiguration von Sendmail. Bei der Anwendung der Makros gibt es jedoch Einschränkungen, Klassenmakros können nicht auf der rechten Seite und Datenbankmakros nicht auf der linken Seite von Adreßregeln angewendet werden.

- Aliases
  Die in der Empfängerliste des Auftrags angegebenen Aliases müssen von Sendmail durch die richtigen Adressen ersetzt werden, dabei können sich hinter einem Alias auch mehrere Empfänger verbergen. Die Informationen zur Auflösung dieser Aliases können in einer eigenen Datei oder in der *~/.forward* Datei des Auftraggebers stehen.

- Adressen und Regeln
  Regeln werden in Sendmail benützt, um Empfängeradressen zu verändern, Fehler in der Adressierung zu erkennen und um das richtige Nachrichtenzustellprogramm auszuwählen.

Da die Adressen auf viele verschiedene Arten angegeben werden können, vom Zustellprogramm aber in einer bestimmten Form benötigt werden, müssen sie dementsprechend abgeändert werden. So lautet z.B. die Adresse im Internet–Format ‘friend-@uuhost’ und im UUCP–Format ‘uuhost!friend’.

Eine weitere Aufgabe der Regeln von Sendmail ist das lokale Erkennen von Fehlern in der Adresse. Sollte die Benutzerkennung in der Adresse des Empfängers fehlen, ist es besser die Nachricht gleich zu verwerfen, anstatt vom empfangenden Rechner abweisen zu lassen.

Und schließlich wird anhand der Empfängeradresse das richtige Zustellprogramm ausgewählt. Ist der Empfänger ein lokaler Benutzer, wird ein Programm zur lokalen Zustellung herangezogen, soll die Nachricht an einen entfernten Benutzer gehen, wird das dafür geeignete Programm aufgerufen.

Die einzelnen Regeln stehen aber nicht für sich alleine, sondern werden zu Regelsätzen zusammengefaßt. Dabei stellt jeder Satz eine Art Unterroutine dar. Sendmail definiert die Sätze 0 bis 6 für ganz bestimmte Zwecke. So ermittelt z.B. Regelsatz 0 aus einer Empfängeradresse das geeignete Zustellprogramm, den Benutzernamen des Empfängers und die Domäne des empfangenden Rechners.

- Kopf des Auftrags
  Einige Informationen müssen in jedem Auftrag erscheinen, andere sind optional. Sofern die benötigten Informationen nicht bereits in der Nachricht angegeben sind, übernimmt Sendmail das Erstellen dieser Informationen und schreibt sie in den Kopf des Auftrags. Im Auftrag muß auf jeden Fall vermerkt sein,

  - die Uhrzeit und das Datum der Auftragsübergabe an Sendmail,
  - die Absenderadresse und
  - die Auftragskennung.

  Außerdem wird bei der Zustellung der Nachricht jeder Rechner vermerkt, der in den Zustellvorgang eingebunden war.

- Privilegierte Benutzer
  Manchen Absendern ist es erlaubt, die Absenderadresse selbst explizit anzugeben. Dies ist nötig um bestimmt Arten der Nachrichtenzustellung zu ermöglichen, wie z.B. der Nachrichtenversand über UUCP. Bei UUCP übernimmt ein Pseudo–Benutzer ‘uucp’ die Übergabe der Nachricht, welcher dann fälschlicherweise als Absender eingetragen wird. Durch die explizite Angabe der Absenderadresse wird der Pseudo-Benutzer überschrieben.

3.2.1.3 Warteschlange

Nachdem der Auftrag in der Warteschlange angekommen ist, wird er entweder sofort zugestellt oder verbleibt für eine gewisse Zeit in der Warteschlange. Ein Sendmail–Dämon übernimmt die Zustellung der in der Warteschlange liegenden Nachrichten in regelmäßigen Abständen. Diese Warteschlange wird durch ein Verzeichnis, gewöhnlich *mqueue* bezeichnet, realisiert.

Aufträge in der Warteschlange
Um einen Auftrag in die Warteschlange einzureihen, wird der Auftrag in die Nachricht (Datendatei) und den Kopf (Auftragskontrolldatei) zerlegt. Diese beiden Teile werden dann jeweils in einer eigenen Datei im Verzeichnis der Warteschlange abgelegt. Nachdem die Dateien freigegeben worden sind, ist der Auftrag vollständig in der Warteschlange und kann dort bearbeitet werden. Aus Sicherheitsgründen sollte das Verzeichnis der Warteschlange als Besitzer „root“ und nur die notwendigsten Zugriffsrechte vergeben haben, von CERT wird das Zugriffsrecht 0700 empfohlen.

Neben der bereits erwähnten Datendatei, die die Nachricht enthält, und der Warteschlangenkontolldatei, mit dem Kopf des Auftrags, werden noch weitere Informationen zur Abwicklung der Zustellung einer Nachricht benötigt. Einige dieser Informationen werden zusätzlich in die Auftragskontrolldatei eingetragen, andere in eigenen Dateien. Diese Dateien nennen sich temporäre Auftragskontrolldatei und Fehlerdatei.

- Datendatei
  Darin befindet sich nur die Nachricht des Auftrags. Sollte das Verzeichnis der Warteschlange für jeden Benutzer lesbar sein, sollte auf alle Fälle die Datendatei vor unberechtigten Zugriffen geschützt sein, denn diese Datei beinhaltet die Nachricht, die mitunter sensible oder persönliche Daten enthält. Daraus folgt auch, daß der Inhalt dieser Datei für das Management des Nachrichtensystems nicht relevant ist.

- Auftragskontrolldatei
  Neben dem Kopf des Auftrags werden in dieser Datei noch weitere Informationen gespeichert. Manche dieser Informationen sind bereits im Kopf des Auftrags enthalten, werden aber nochmals explizit angegeben.

  - die Adresse des Absenders
  - die Adreßliste der Empfänger
  - die Priorität des Auftrags
  - die Zeit, wann die Nachricht das erste Mal in die Warteschlange kam
  - den Grund, warum die Nachricht in die Warteschlange kam.
  - den Namen der Datendatei
  - das Format der Nachricht (ein Zeichen kann sowohl mit 8 Bit als auch mit 7 Bit kodiert werden)
  - den Besitzer der ~/*.forward* Datei, wenn die Nachricht weitergeleitet wurde

- die Adresse, an die die Fehlermeldungen von Sendmail geschickt werden sollen
- wenn die Nachricht nicht sofort zugestellt werden konnte, ob bereits eine Meldung dieses Problems an den Absender geschickt wurde
- das Protokoll, der ersten von Sendmail empfangenen Nachricht, z.B. SMTP

Jede dieser Informationen steht in einer eigenen Zeile, eingeleitet von einem Großbuchstaben, der die Art der Information festlegt. Dabei muß nicht jede der oben aufgeführten Informationen in der Auftragskontrolldatei zu finden sein, andere können auch mehrmals erscheinen. Zu erwähnen bleibt noch, daß sich das Aussehen der Auftragskontrolldatei von Version zu Version von Sendmail ändern kann. Da es sich bei der Auftragskontrolldatei um eine interne Schnittstelle von Sendmail handelt, wird die Änderung auch nicht ausdrücklich bekanntgegeben.

- temporäre Auftragskontrolldatei
  Sobald versucht wird eine Nachricht aus der Warteschlange zu verschicken, ist es häufig nötig den Inhalt der Auftragskontrolldatei zu verändern. Dies ist der Fall, wenn die Zustellung scheiterte oder nicht für alle Empfänger gelang. In beiden Fällen muß mindestens die Priorität des Auftrags erhöht werden. Damit die Orginal–Auftragskontrolldatei nicht beschädigt wird oder bei einem Systemabsturz verloren geht, erzeugt Sendmail eine temporäre Kopie dieser Datei. Nach erfolgreicher Änderung in der temporären Datei wird diese in die Auftragskontrolldatei umbenannt.

- Fehlerdatei
  Treten bei der Zustellung der Nachricht Fehler auf, werden diese in der Fehlerdatei gesammelt und nach Beendigung des Zustellversuches an den Absender der Nachricht geschickt.

Bedienstrategie
Während der Zeit sammeln sich Aufträge in der Warteschlange an und bleiben dort solange, bis Sendmail die Warteschlangenbearbeitung startet. Dies kann entweder ein Sendmail–Dämon übernehmen, der in periodischen Zeitabständen die Warteschlange abarbeitet, oder ein Aufruf von Sendmail von der Kommandozeile aus. Jedesmal wenn Sendmail die Warteschlange bearbeitet, muß als erstes die Reihenfolge der Aufträge in der Warteschlange festgelegt werden, nach der die Aufträge bearbeitet werden. Dazu liest Sendmail von jedem Auftrag die Auftragskontrolldatei, wobei Aufträge, bei denen die Auftragskontrolldatei nicht gelesen werden kann, ignoriert werden. Allerdings wird nicht jeder Auftrag gleich behandelt, es gibt drei verschiedene Möglichkeiten, wann ein Auftrag an der Reihe ist.

- sofortige Zustellung
  Gewöhnlich veranlaßt das Sendmailprogramm, welches den Auftrag in die Warteschlange einreihte, die sofortige Zustellung der Nachricht. Dabei werden die in der Warteschlange befindlichen Aufträge nicht beachtet. Jedoch wird nicht immer diese Zustellart ausgeführt, es gibt einige Gründe, die die sofortige Zustellung verhindern.

  - Das momentane Scheitern der Zustellung der Nachricht. Ist die Nachricht für mehrere Empfänger bestimmt, wird nur für die nicht erreichten Empfänger die Nachricht wieder in die Warteschlange eingefügt.
  - Die Option '-odq' in der Kommandozeile beim Start von Sendmail. Dadurch wird die sofortige Zustellung der Nachricht nicht erwünscht.
  - Ein höherer Belastungswert (load–Wert) des Systems als der in der Konfiguration von Sendmail angegebene Wert.

- periodische Zustellung
  Dies entspricht der periodischen Abarbeitung der Warteschlange. Nachdem die Reihenfolge der Aufträge festgelegt ist, beginnt die Bearbeitung der einzelnen Aufträge. Begonnen wird mit dem Sperren der Auftragskontrolldatei, damit der Auftrag nicht von zwei Sendmailprozessen gleichzeitig bearbeitet werden kann. Darauf wird die Verweilzeit des Auftrags in der Warteschlange mit der maximalen Verweilzeit verglichen, wurde sie überschritten, wird der Auftrag storniert. Jetzt kann die Nachricht an jeden Empfänger geschickt werden. Werden nicht alle Empfänger erreicht, müssen die nicht erreichten Empfänger wieder in der Auftragskontrolldatei eingetragen werden, alle anderen werden gelöscht. Dann kann der Auftrag wieder in die Warteschlange zurückgestellt werden und bei der nächsten Warteschlangenbearbeitung erneut aufgegriffen werden. Bei einem erfolglosen Zustellungsversuch wird auch noch die Priorität des Auftrags erhöht und es kann eine entsprechende Fehlermeldung an den Absender der Nachricht geschickt werden. Erreichte die Nachricht alle Empfänger, werden die Dateien für diesen Auftrag aus dem Verzeichnis der Warteschlange entfernt, was dem Entfernen des Auftrags aus dem Server gleichkommt. Sobald der Zustellversuch für einen Auftrag abgeschlossen ist, wird mit dem nächsten Auftrag begonnen.

- individuelle Zustellung
  Durch einen geeigneten Aufruf von Sendmail lassen sich auch bestimmte Aufträge aus der Warteschlange herausgreifen und die Zustellung für diese durchführen. So läßt sich ein bestimmter Auftrag angeben, es kann versucht werden alle Aufträge eines Absenders oder auch alle Aufträge für einen bestimmten Empfänger zu verschicken. Natürlich lassen sich diese Optionen auch miteinander verknüpfen, infragekommmende Aufträge müssen alle Optionen erfüllen. Eine weitere Möglichkeit ist die Angabe eines anderen Verzeichnisses für eine Warteschlange, womit Aufträge zurückgestellt und später verschickt werden können.

Mit entsprechenden Optionen beim Aufruf von Sendmail erhält man eine Übersicht über alle in der Warteschlange befindlichen Aufträge. Es wird die Anzahl und die Reihenfolge der Aufträge dargeboten. Zusätzlich werden für jeden Auftrag

- die Kennung,

- ob der Auftrag gesperrt ist,

- ob die Systemlast zu hoch ist,
- die Priorität,
- die Größe der Datendatei,
- ob bereits eine Meldung an den Absender gesendet wurde,
- das Datum und die Uhrzeit, wann der Auftrag in die Warteschlange kam,
- der Absender der Nachricht und
- die Empfängerliste der Nachricht

angezeigt.

3.2.1.4 Zustellung

Sendmail übernimmt normalerweise die Zustellung der Nachrichten an den Empfänger nicht selbst. Zur Zustellung wird ein geeignetes Programm aufgerufen, daß die Nachricht in den Briefkasten (Mailbox) des Empfängers schreibt. Es besteht zwar die Möglichkeit Sendmail dazu zu veranlassen, die Nachricht an eine Datei anzuhängen, es sollte jedoch nicht die Regel sein. Außerdem erhält der Empfänger der Nachricht noch den Kopf des Auftrags zugestellt.

- lokale Zustellung
  Befindet sich der Briefkasten des Empfängers auf dem gleichen Rechner, auf dem die Nachricht abgeschickt wurde, wird die Nachricht lokal ablegt. Gewöhnlich übernimmt die Zustellung der Nachricht, in den Briefkasten des Empfängers, das Programm '/bin/mail'. Es ist allerdings auch möglich die Nachricht an ein Programm zu schicken, das auf diesem Rechner aktiv ist. Genauso verfährt der Sendmail–Dämon, wenn er eine Nachricht von einem anderen Rechner für einen lokalen Benutzer erhält.
- Zustellung über das Netzwerk
  Ist der Empfänger der Nachricht nicht lokal auf diesem Rechner zu finden, muß die Adresse und der Weg zum Zielrechner (Rechner, auf dem der Empfänger seinen Briefkasten hat) ermittelt werden. Der Auftrag kann an ein zentrales Nachrichtenzustellsystem (mail delivery hub) gesendet werden oder direkt zum Zielrechner. In einem TCP/IP Netzwerk kann Sendmail diese Aufgabe übernehmen, muß sich dafür aber die IP–Adresse des empfangenden Rechners besorgen, dabei kann Sendmail auch auf DNS (Domain Name System) zurückgreifen. Auf dem empfangenden Rechner ist ein Sendmail–Dämon für die Annahme des Auftrags verantwortlich. Bei UUCP Netzwerken übergibt Sendmail diese Aufgabe einem geeigneten Programm, ebenso bei anderen Netzwerkprotokollen.

Sendmail kann auf Wunsch das Zustellen der Nachrichten, in die vorher erzeugte Datei *sendmail.st*, mitprotokollieren. In der Datei findet man dann den Zeitraum der Überwachung, als auch für jede Zustellart (Mailer, Bedieneinheit) die Anzahl und Gesamtgröße der empfangenen sowie abgesandten Nachrichten.

3.2.2 Das nachrichtenspezifische Objektmodell

Die gleiche Vorgehensweise, die bereits bei der Erstellung des druckerspezifischen Objektmodells durchgeführt wurde, wird auch bei der Erstellung des nachrichtenspezifischen Objektmodells angewandt.

3.2.2.1 Komponenten

Bei der Überführung des funktionalen Modells des Nachrichtendienstes in ein Objektmodell des Nachrichtendienstes muß als erstes ermittelt werden, welche Komponenten benötigt werden.

Da es im Nachrichtensystem genau eine Warteschlange und einen Auftragstyp gibt, stehen diese beiden Komponenten bereits fest. Bevor der Auftrag die Warteschlange erreicht, muß er bereits einen Bearbeitungsschritt durchlaufen. Dafür wird eine Bedienstation „Aufbereitung“ benötigt, die verantwortlich für die Bearbeitung ist. Zur Durchführung der Bearbeitung ist eine Bedieneinheit vonnöten. Die Anzahl der Bedieneinheiten richtet sich nach der Auftragsanzahl in der Bedienstation.

Die Weiterverarbeitung der Aufträge nach der N–Warteschlange geschieht wieder in einer Bedienstation, die die Zustellung der Nachricht übernimmt. Dabei kann der Empfänger der Nachricht sein Postfach lokal auf dem gleichen Rechner haben oder auf einem entfernten Rechner. Hier muß man unterscheiden zwischen einem Programmaufruf zur Zustellung der lokalen Nachrichten und der Übermittlung der Nachricht an einen anderes Nachrichtensystem. Um dies im Objektmodell zu berücksichtigen, werden zwei Bedienstationstypen „Lokal“ und „Entfernt“ dafür bereitgestellt. Nur so kann man unterscheiden, wieviele Nachrichten lokal zugestellt und wieviele an ein anderes Nachrichtensystem weitergeleitet werden konnten. Der Bedieneinheitstyp ist für beide Bedienstationen wieder gleich. Er übernimmt die Zustellung der Nachricht, wobei er dafür ein Programm aufrufen kann oder die Zustellung selbst ausführt, beide Möglichkeiten sind bei beiden Bedienstationen möglich, es ist nur von der Konfiguration und dem Übertragungsweg abhängig. Durch die Auftragsanzahl wird auch wieder die Bedieneinheitenanzahl bestimmt.

3.2.2.2 Beziehungen

Im Nachrichtensystem gibt es von den Komponenten des Objektmodells nur je ein Objekt, ausgenommen beim N–Auftrag und den Bedieneinheiten. D.h. alle Aufträge kommen bei

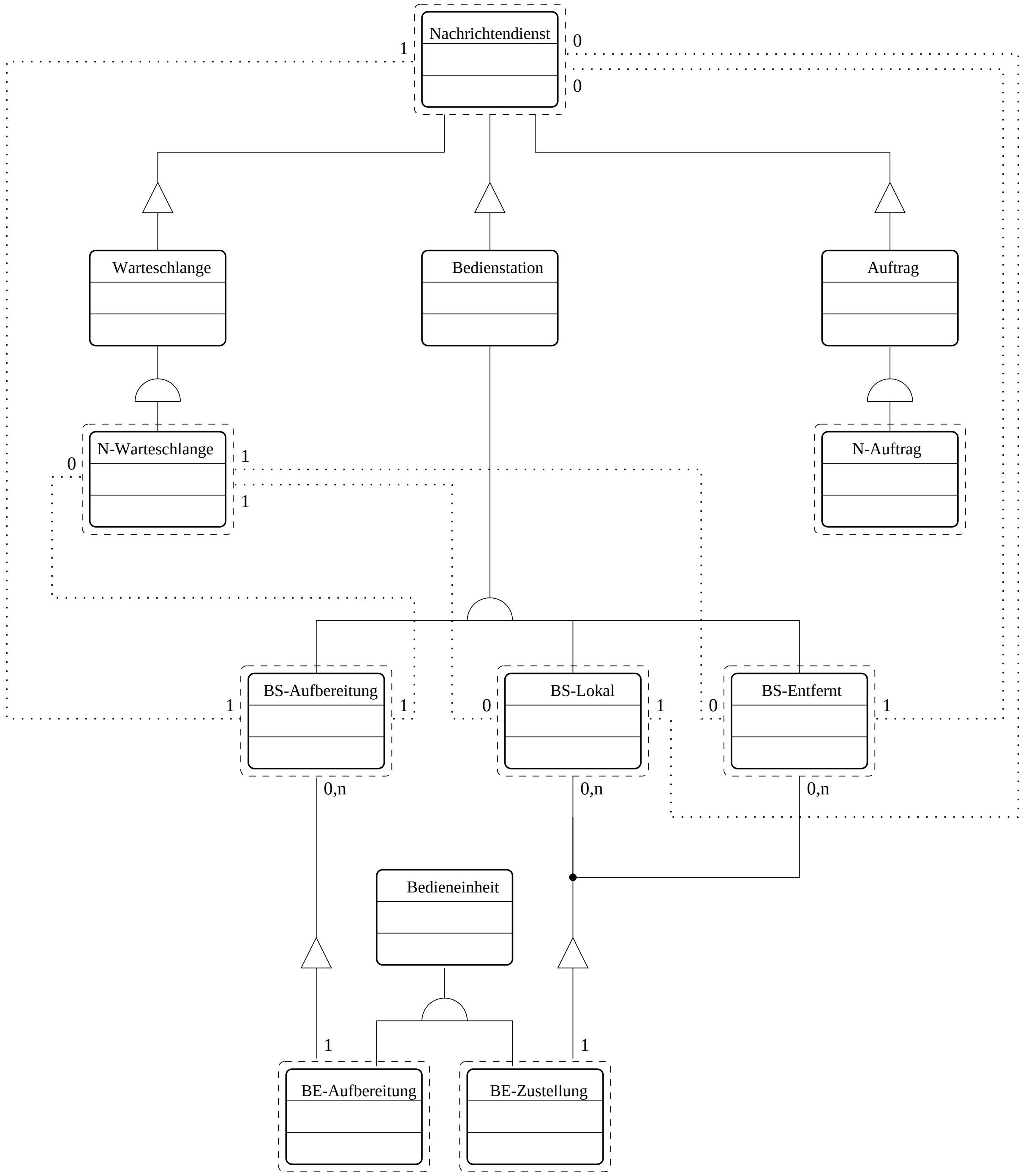

Abbildung 3.4: Das Objektmodell des Nachrichtendienstes

der einzigen Bedienstation „Aufbereitung“ an, werden darauf in die Warteschlange eingereiht, von wo sie weiter in eine der Bedienstationen „Lokal“ oder „Entfernt“ gereicht werden. Ist die Zustellung erfolgreich, wird der Auftrag beendet, sonst wird er wieder in die Warteschlange zurückgestellt. Auch wenn die Bearbeitung des Auftrags in einer Bedienstation aufgrund eines Problems scheitert oder abgewiesen wird, ist der Auftrag beendet. Anhand dieser Auftragsbearbeitung ergeben sich die Beziehungen zwischen den Komponenten des Objektmodells. Wenn der Auftrag an die Bedienstation „Lokal“ oder „Entfernt“ weitergereicht wird, verbleibt er allerdings noch in der Warteschlange und verläßt diese erst, nachdem die erfolgreiche Zustellung abgeschlossen ist.

Die generischen Attribute und Methoden erben die Bedienstationen, die Bedieneinheiten, die N-Warteschlange und der N-Auftrag wieder von ihren entsprechenden abstrakten Klassen. Somit erhält man das in Abbildung 3.4 gezeigte Objektmodell für den Nachrichtendienst.

3.2.2.3 Managementinformationszuordnung

Schließlich können den Komponenten des nachrichtenspezifischen Objektmodells die Managementinformation und -funktionalität der Komponenten des funktionalen Modells zugewiesen werden.

Dabei können die Attribute des Auftrags im funktionalen Modell direkt für den Auftrag im Objektmodell übernommen werden. Für die Bedienstation „Aufbereitung“ existieren Informationen über definierte Makros, Aliases, Regeln und nötige Eintragungen im Kopf des Auftrags, was hauptsächlich Konfigurationsinformationen sind. Der Warteschlange kann nichts zugeteilt werden, die Informationen im funktionalen Modell für die Warteschlange müssen teilweise dem Auftrag zugeschrieben werden oder sie sind abhängig vom Systemzustand und den Aufrufparametern von Sendmail. Die Bedienstationen „Lokal“ und „Entfernt“ bieten die Anzahl und Gesamtgröße der zugestellten Nachrichten an.

3.2.2.4 Anmerkung

Ähnliches, was beim druckerspezifischen Objektmodell gesagt wurde, gilt auch hier. Die Abbildung vom funktionalen Modell ins Objektmodell ist einfach zu bewerkstelligen. Es ist auch wieder realitätsnah und kann den Nachrichtendienst gut abbilden. Bei der Zuordnung der Managementdaten müssen zwar ein paar Informationen anderen Komponenten zugewiesen werden, doch im allgemeinen stimmt es beim funktionalen Modell und dem Objektmodell überein.

Leider sind die meisten Daten des Nachrichtensystems für das Konfigurationsmanagement bestimmt. Andere Managementbereiche werden stark vernachlässigt oder völlig übergangen. Vom Druckdienst wurden mehr Managementdaten von verschiedenen Managementbereichen bereitgestellt.

Allerdings hat dies kein Auswirkung auf das Objektmodell. Für beide Basisdienste konnte ein spezielles Objektmodell entwickelt werden, die die untersuchten Server sehr gut darstellen.

Kapitel 4

Managementinformation und -funktionalität

Nachdem gezeigt wurde, wie die Abbildung von Basisdiensten auf das Objektmodell aussieht, ist es nötig Attribute anzugeben, mithilfe deren das Systemmanagement realisiert werden kann. Dazu werden aus den allgemeinen Managementanforderungen an Basisdienste generische Managementinformation und -funktionalität für das Objektmodell abgeleitet. Diese wird dann in den abstrakten Klassen des Objektmodells allen Basisdiensten bereitgestellt.

Der nächste Schritt wäre die Untersuchung dienstspezifischer Managementanforderungen, was in dieser Diplomarbeit aber übersprungen wird. Jeder Dienst hat gewisse Eigenheiten, auf die das Management speziell eingehen muß. Diese Eigenheiten werden bei der Verfeinerung der abstrakten Klassen berücksichtigt, wo neue Attribute und Methoden aufgenommen bzw. überdeckt werden können.

Um die Untersuchung dienstspezifischer Managementanforderungen auslassen zu können, wurde bei der Auswahl der Programme für die beiden ausgewählten Basisdienste darauf geachtet, daß sie keine Sonderformen ihres Dienstes darstellen. Nun „sendmail“ ist das Standardprogramm für den Nachrichtendienst im Internet schlechthin und die Funktionalität des PLP–Drucksystems wurde gegenüber dem Berkley–Drucksystem nur etwas erweitert.

Anhand der ausgewählten Programme ist es möglich implementierungsspezifische Managementanforderungen herauszufinden und die implementierungsspezifische Managementinformation und -funktionalität zu bestimmen.

Nachdem im folgenden Abschnitt die generische Managementinformation und -funktionalität im Objektmodell erläutert wird, werden in den beiden folgenden Abschnitten die Managementanforderungen an das Druck- und Nachrichtensystem aufgestellt. Dazu wird versucht die generischen Managementanforderungen durch die implementierungsspezifischen

Managementanforderungen der jeweiligen Dienste zu erfüllen. Darüberhinaus entdeckte Managementanforderungen müssen in eigenen Attributen berücksichtigt und den Komponenten des Objektmodell zugeordnet werden.

Wird die generische Information bzw. Funktionalität weder angeboten noch kann sie in vertretbarem Rahmen implementiert werden, muß man leider auf sie verzichten.

Für jede Komponente des Objektmodells und für das Auftragsmanagement muß untersucht werden, welche Attribute sie bereitstellen sollen, welche Werte diese Attribute annehmen können und wie diese Werte zu interpretieren sind.

Attribute, die die Anzahl eines bestimmten Ereignisses angeben, beziehen sich dabei immer auf ein Zeitintervall. Dieses Zeitintervall beginnt mit dem letzten Zurücksetzen bzw. Neustart des Agenten, der die Attribute bereitstellt und endet mit dem Augenblick der Anfrage an den Agenten.

4.1 Objektorientierte Analyse ausgewählter Managementanforderungen

Die Managementanforderungen an einen Server wurden in [Guts 95] untersucht und ausführlich dargestellt, die für diese Diplomarbeit nötigen Punkte werden hier kurz erläutert. Bei der Suche nach den Managementanforderungen wurden grundlegende Modellierungsarbeiten sowie etablierte Managementmodelle für einzelne Anforderungen berücksichtigt. Zur Zuweisung der benötigten Managementanforderungen und -funktionalität an Komponenten von Basisdiensten wurde das bereits vorgestellte Objektmodell herangezogen.

Die Managementanforderung und –funktionalität läßt sich in vier Bereiche einteilen:

- Leistungsmanagement
- Statusmanagement
- Auftragsmanagement
- Konfigurationsmanagement

4.1.1 Leistungsmanagement

Zur Überwachung des Leistungsverhaltens eines Servers und zum Sammeln statistischer Daten wird das Leistungsmanagement herangezogen. Das Ziel des Leistungsmanagements ist ein „gut“ laufendes Gesamtsystem, in diesem Fall der Server. Anhand der erhaltenen Daten ist ein Eingreifen zur Verbesserung des Leistungsverhaltens möglich. Die gesammelten Daten werden aber auch bei der weiteren Planung des Systems herangezogen.

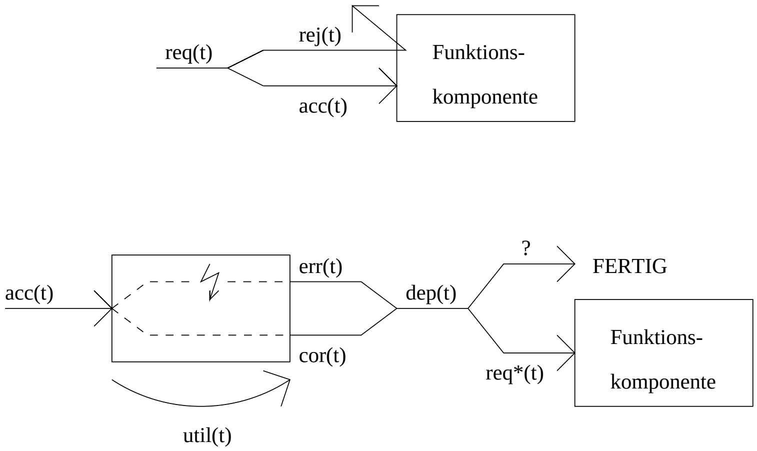

Abbildung 4.1: Die Attribute für das Leistungsmanagement

Um das Leistungsverhalten beurteilen zu können, sind die Größen, wie z.B. der Durchsatz, die Auslastung, die Antwortzeit und die Dienstgüte (QoS) interessant. Diese Größen sind Managementinformationen, die von der Ressource in geeigneter Form bereitgestellt werden müssen.

Dabei stützt man sich auf die QoS–Größen Last, Durchsatz, Füllung, Dauer und Genauigkeit. Es wird vorausgesetzt, daß $t > t_0$ ist, wobei t_0 den letzten Rücksetzzeitpunkt der Zähler darstellt. In Abbildung 4.1 sind die Attribute grafisch dargestellt und werden im folgenden genauer erläutert.

- Lastüberwachung

  req(t): Ist die Anzahl der im Intervall $(t_0, t($ bei der Funktionskomponente eingetroffenen Aufträge.

  acc(t): Ist die Anzahl der im Intervall $(t_0, t($ in der Funktionskomponente angenommenen Aufträge.

  rej(t): Ist die Anzahl der im Intervall $(t_0, t($ in der Funktionskomponente abgewiesenen Aufträge. Dazu zählen auch die Aufträge, die nach unmittelbarer Auftragsannahme wieder abgelehnt werden, ohne daß eine relevante Ressourcenbindung stattgefunden hat. Diese Information berechnet sich aus der Differenz von *req(t)* und *acc(t)*. Ein Grund für das Ablehnen von Aufträgen kann z.B. ein Mangel an Ressourcen sein oder die zeitweilig Sperrung des Servers.

- Durchsatzüberwachung und Füllung

  dep(t): Ist die Anzahl der im Intervall $(t_0, t($ in der Funktionskomponente beobachtbaren Meldungen über die Beendigung von Aufträgen. Auf welche Art die

Aufträge beendet werden spielt dabei keine Rolle. Sie können an die nächste Komponente im Objektmodell weitergeleitet werden, die Funktionseinheit ordnungsgemäß verlassen, wenn die vollständig Bearbeitung des Auftrags erledigt ist oder sie werden in der Komponente wegen eines Fehlers verworfen. Dabei muß auch nicht notwendigerweise eine Meldung an den Auftraggeber erfolgen.

util(t): Diese Information berechnet sich aus $acc(t) - dep(t)$. Warum diese Größe als eigenständige Basisgröße angeboten werden soll, liegt in der Zeitdifferenz beim aufeinanderfolgenden Zugriff auf acc(t) und dep(t) und der möglicherweise daraus resultierenden Ergebnisverfälschung. Der Wert dieses Attributs gibt die Anzahl der in der Komponente enthaltenen Aufträge wieder.

- Dauer

del(t): Ist die Bearbeitungszeit des letzten im Intervall $(t_0, t($ von der Funktionskomponente beendeten Auftrags. Einige Server stellen diese Information bereits zur Verfügung, bei anderen kann sich leicht ermittelt werden.

- Genauigkeit

cor(t): Ist die Anzahl der im Intervall $(t_0, t($ von der Funktionskomponente korrekt bearbeiteten Aufträge. Korrekt ist ein Auftrag in einer Komponente bearbeitet, wenn er im Server beendet oder weitergeleitet wurde.

err(t): Das Ergebnis aus *dep(t) - cor(t)* gibt die Anzahl der fehlerhaft bearbeiteten Aufträge. Damit sind die Aufträge gemeint, die von der Funktionskomponente angenommen wurden, durch ein Problem in der Funktionskomponente jedoch nicht korrekt bearbeitet werden konnten.

In [Guts 95] wird weiter dargelegt, welche Komponenten des Objektmodells, welche Informationen anbieten sollen. Kurz gesagt, die Funktionseinheit, die Warteschlange und die Bedienstation sollten alle Attribute anbieten. Bei den Bedieneinheiten ist es abhängig von der Art der Bedieneinheit und für Aufträge sind die Attribute ungeeignet. Das gleiche gilt für die Attribute des Statusmanagement, die im folgenden Abschnitt aufgeführt werden.

Sollte eine Funktionseinheit mehrere Auftragsklassen annehmen können und eine Betrachtung für einzelne Auftragsklassen gewünscht werden, so muß für jede Klasse diese Information bereitgestellt werden. Ausgenommen davon ist die Funktionseinheit, sie gewährleistet eine Überwachung über alle Auftragsklassen.

4.1.2 Statusmanagement

Ein wichtiger Aspekt für Systemverwalter ist die Überwachung, ob ein Server noch läuft oder nicht mehr. Diese Aufgabe fällt in den Bereich des Statusmanagements, daß die

Möglichkeit bereitstellt, den Status der zu managenden Ressource zu ermitteln und diesen bei Bedarf auch zu ändern.

Die Suche nach den Attributen für das Statusmanagement des Objektmodells stützt sich stark auf einen Standard für das Netzmanagement, der *State Management Function* [ISO 10164-2], aber auch auf das DSIS Modell [DSIS 94].

Dabei zeigt sich, daß die *State Management Function* bereits Attribute und Meldungen definiert, die für das Statusmanagement des Objektmodells geeignet sind. Im Großen und Ganzen kann der Standard für das Netzmanagement auch auf das Objektmodell angewendet werden. Durch das DSIS Modell kommen lediglich noch zwei weitere Werte für die Attribute hinzu.

Das Statusmodell in [ISO 10164-2] zerlegt den Status einer Ressource in drei Faktoren. Die dafür möglichen Attribute wurden noch um einige erweitert.

Betriebsbereitschaft: Sie zeigt an, ob eine Ressource vorhanden ist bzw. in Betrieb ist oder nicht.

Mögliche Werte für das Attribut der Betriebsbereitschaft sind „enabled", „disabled" oder „unkown". Ist eine Ressource teilweise oder ganz betriebsbereit wird es durch den Wert „enabled" signalisiert, „disabled" bedeutet, daß die Ressource nicht betriebsbereit ist. Den Wert „unkown" nimmt es nur an, wenn sich der Zustand der Ressource nicht ermitteln läßt. Das Management kann dieses Attribut nur lesen, aber nicht verändern.

Nutzung: Dadurch wird angezeigt, ob die Ressource frei, beschäftigt oder voll ausgelastet ist.

Das Attribut der Nutzung kann vier Werte annehmen, „idle", „active" „busy" und „unkown". Daß die Ressource frei ist, wird durch den Wert „idle" angezeigt, „active" deutet die Belegung der Ressource an, wobei jedoch noch Kapazitäten frei sind, und „busy" zeigt die völlige Auslastung der Ressource an. Wieder wird durch den Wert „unkown" gezeigt, den Zustand nicht ermitteln zu können.

Administration: Sie zeigt die Erlaubnis der Nutzung einer Ressource an. Durch das Verändern des Wertes vom Management kann der Zustand der Ressource beeinflußt werden.

Für das Attribut des Administrationszustandes sind mehrere Werte möglich. Diese Werte sind „not applicable", „unlocked", „shutting down" und „locked". „Not applicable" zeigt an, daß dieser Wert für den realen Server nicht verändert werden kann. Ist der Ressource nicht erlaubt ihren Dienst zu erbringen, erscheint der Wert „locked", falls es ihr erlaubt ist „unlocked". Soll die Ressource die bereits angenommenen Aufträge noch bearbeiten, aber keine neuen Aufträge mehr annehmen, wird dies durch den Wert „shutting down" erreicht.

Der Administrationszustand ist unabhängig von der Betriebsfähigkeit und der Nutzung, er dient dem Management zur Steuerung der Ressource.

Natürlich müssen die Attribute in konkreten Servern nicht alle möglichen Attribute annehmen können. Sie können je nach Bedarf eingeschränkt oder erweitert werden. So können noch Statusattribute für *alarm status, procedural status, availability status, control status, standby status und unkown status* hinzukommen, welche die obigen Attribute noch detailierter darstellen können.

4.1.3 Auftragsmanagement

Das Auftragsmanagement dient zur Beobachtung und Beeinflussung einzelner Aufträge im System. Dazu zählt auch die Aufzeichnung der Aktivitäten des Servers bei der Bearbeitung eines Auftrags. Desweiteren wird beim Fehler- und Leistungsmanagement, wenn es sich nur über ein kurzes Zeitintervall erstreckt, auf das Auftragsmanagement zurückgegriffen.

Weitere Gründe für das Auftragsmanagement sind zum einen die Aufzeichnung der Auftragsbearbeitung zu Dokumentationszwecken oder für eine spätere Auswertung. Sowie die Überwachung von Aufträgen, die längere Zeit (mindestens ein paar Minuten) in der Funktionseinheit sein können und eine relevante Ressourcenbindung beanspruchen.

Dieser längere Zeitraum ist nötig, damit die Managementapplikation eingreifen kann. Dazu gehört, den Systemstand zu ermitteln, das Problem zu orten und zu reagieren.

Für einen Auftrag sind dabei folgende Informationen interessant:

- Eine eindeutige Identifizierung des Auftrags
- Die Eintreffzeit des Auftrags in der Funktionseinheit
- Der Name des Absenders und des absendenden Rechners
- Der aktuelle Bearbeitungszustand eines Auftrags, der durch seine „Position“ innerhalb der Funktionseinheit bestimmt wird, d.h. in welchen Komponenten des Objektmodells er sich gerade befindet. Der Auftrag muß stets mindestens in einer Bedienstation oder Warteschlange enthalten sein.
- Die Angabe der Größe läßt sich nicht allgemeingültig realisieren. Sie schwankt zwischen der Größe
  - der Daten des Auftrags,
  - der Auftragsbeschreibung und
  - des belegten virtuellen Speichers von ablaufenden Prozessen.

Deshalb wird die Beschreibung der Größe der späteren Verfeinerung der Objekte für einen konkreten Server überlassen.

Bei jeder Erzeugung oder Entfernung eines Auftragsobjekts muß der lokale Managementagent informiert werden. Dabei soll der Objektidentifikator, der Grund der Meldung, der Auftraggeber und der auftraggebende Rechner mitgeteilt werden.

Außerdem muß der Auftrag eine Methode bereitstellen, mit der er gelöscht werden kann. Das Anhalten eines Auftrags geschieht bei einer Bedienstation.

Wenn wir uns wieder dem Objektmodell zuwenden, läßt sich dabei feststellen, daß in der Funktionseinheit sowieso alle Aufträge enthalten sind. Bei den anderen Komponenten wird die Position des Auftrags durch eine Enthaltensein–Beziehung zwischen dem Auftrag und der Komponente festgelegt. Die Anzahl der in einer Komponente enthaltenen Aufträge kann dabei begrenzt sein.

In Abbildung 4.2 ist nochmals das Objektmodell zu sehen, darin ist vermerkt welche Attribute die einzelnen Komponenten des Objektmodells anbieten sollten. Dabei steht 'S' für Statusattribute, 'L' für Leistungsattribute und 'A' für Auftragsattribute. Desweiteren werden abgehende Meldungen von Komponenten des Objektmodells an den lokalen Managementagenten durch gepunktete Pfeile ausgedrückt.

4.1.4 Konfigurationsmanagement

Ein weiterer großer Punkt ist das Konfigurationsmanagement. Im Rahmen dieser Diplomarbeit wird darauf aber nicht weiter eingegangen. Die Realisierung des Konfigurationsmanagements befaßt sich hauptsächlich mit dem Auslesen und Darstellen der Konfigurationsdaten aus den Konfigurationsdateien. Die Änderung der Konfigurationsdaten von der Managementstation aus bewirkt häufig eine Änderung in der Konfigurationsdatei. Daraufhin muß in der Regel der Server neu gestartet werden, damit die Änderungen in der Konfigurationsdatei vom Server gelesen und wirksam werden. Die Konfiguration eines Servers, ohne den Umweg über die Konfigurationsdatei, wird von den Servern im allgemeinen nicht angeboten, ausgenommen von ein paar Parametern beim Starten des Servers.

Ein allgemeingültiges Konfigurationsmanagement für verschiedene Basisdienste zu finden bereitet außerdem große Probleme. Sobald man Konfigurationsparameter der verschiedenen Basisdienste betrachtet, stellt man fest, daß jeder Basisdienst spezielle für seine Zwecke nützliche Optionen besitzt, die bei anderen Basisdiensten sinnlos sind. Letztendlich muß man sich auf das Lesen der Konfigurationsdatei beschränken und kann erst bei der Verfeinerung des Objektmodells eines realen Servers das Konfigurationsmanagement konkretisieren. Was bedeutet, jeder Konfigurationsparameter wird als eigene Variable dargestellt.

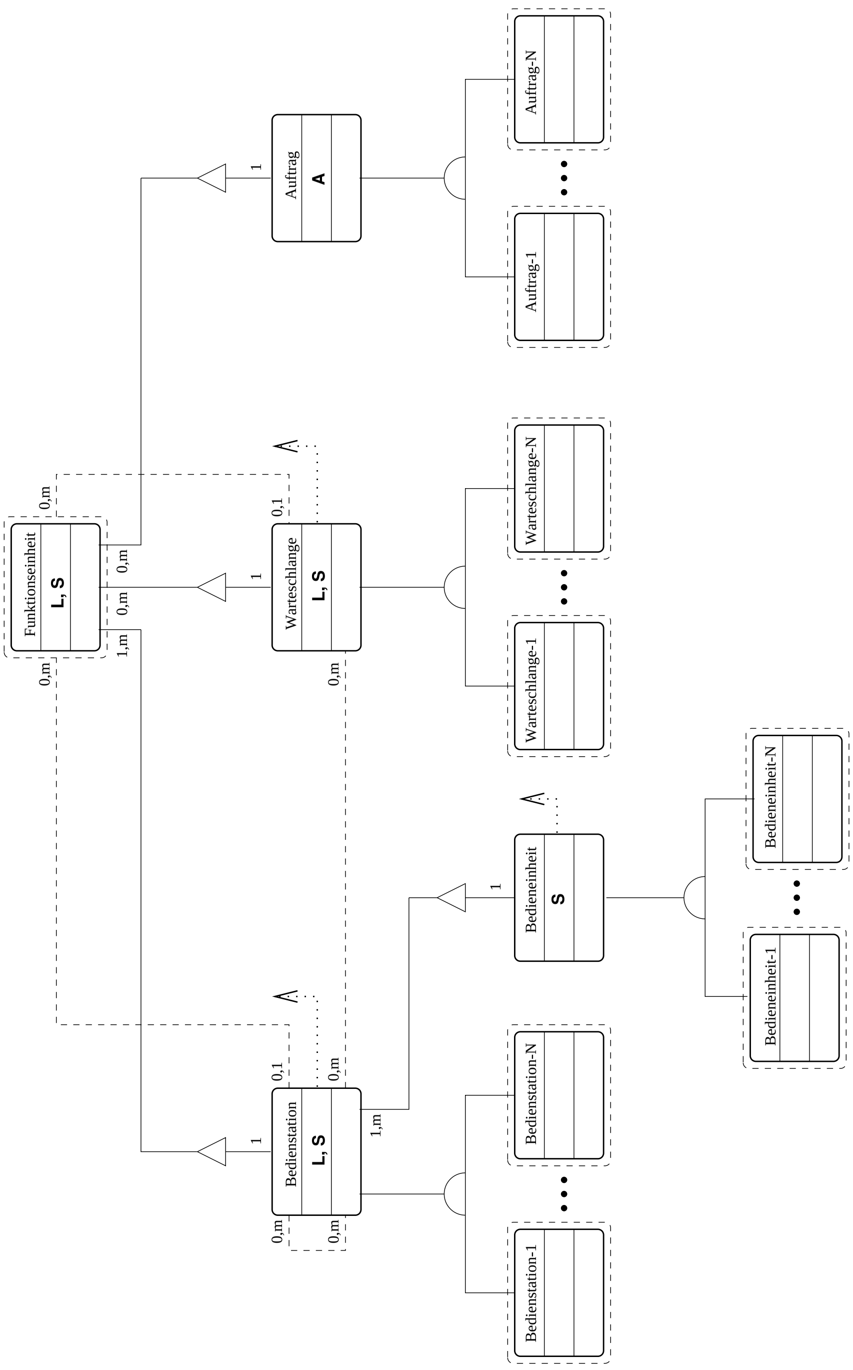

Abbildung 4.2: Das Objektmodell mit Meldungen an den Managementagenten

4.2 Managementanforderungen an das Drucksystem

Eine wichtige Aufgabe des Managements ist die Verwaltung des Drucksystems. Es muß kontrolliert werden, ob das Drucksystem noch ordungsgemäß seine Arbeit verrichtet, in welchem Zustand es sich befindet, wie ausgelastet es ist und natürlich muß das Drucksystem vom Management aus gesteuert werden können.

4.2.1 Funktionseinheit

Die Funktionseinheit gewährt einen Überblick über den Druckdienst, denn in der Funktionseinheit sind alle Komponenten des Objektmodells enthalten. Die Werte sind teilweise mit den Werten der einzelnen Komponenten des Objektmodell identisch oder addieren sich aus den Werten mehrerer Komponenten.

Leistungsmanagement

req(t): Neue Aufträge werden dem Drucksystem entweder mit dem Programm 'lpr' zugestellt oder kommen von der Warteschlange eines entfernten Rechners. Beide Fälle müssen berücksichtigt werden und die Summe beider ergibt die Anzahl der eingetroffenen Aufträge. Diese Werte werden vom Drucksystem bisher jedoch nicht angeboten und müssen deshalb noch implementiert werden.

acc(t): Nicht immer ist es dem Drucksystem möglich neue Aufträge anzunehmen oder ist es bereit dazu. Allerdings sollte angegeben werden, wieviele Aufträge die Funktionseinheit zur Bearbeitung angenommen hat. Dieser Wert wird von der Funktionseinheit allerdings nicht bereitgestellt.

rej(t): Auch Aufträge die abgewiesen wurden, sollen gezählt werden. Im Drucksystem werden gewöhnlich keine Aufträge abgelehnt, außer der Administrator sperrt den Zugang neuer Aufträge, was mit dem Sperren der Warteschlange identisch ist.

dep(t): Gibt die Anzahl aller in der Funktionseinheit beendeten Aufträge an. Wird in einer Komponente des Objektmodells ein Auftrag beendet, d.h. aus dem Server entfernt und nicht an eine andere Komponente des Objektmodells weitergeleitet, erhöht sich dieser Zähler um eins. Dabei wird nicht unterschieden, ob der Auftrag korrekt gedruckt oder ob er in einer Bedieneinheit abgewiesen wurde, aufgrund von nichterlaubten Anforderungen, fehlerhaften Aufträgen oder nicht möglichen Operationen in der Bedieneinheit.

util(t): Um zu erfahren, wieviele Aufträge sich momentan in der Funktionseinheit befinden, wird die Differenz aus den angenommenen Aufträgen und den abgeschlossenen Aufträgen gebildet.

del(t): Jedesmal, wenn ein Auftrag in der Funktionseinheit beendet wird, muß die Zeitspanne von der Auftragsankunft bis zu seiner Beendigung ermittelt und gespeichert werden. Auf welche Art und aus welchem Grund der Auftrag beendet wurde spielt keine Rolle. Sobald ein Auftrag beendet wird, wird dessen Bearbeitungsdauer hier vermerkt. Inwieweit diese Information jedoch Sinn macht, ist abhänig vom Drucksystem. Da an dieser Stelle nicht angegeben wird, welcher Auftrag beendet wurde, existieren auch keine Daten über den Auftrag. Es ist nicht bekannt, ob der Auftrag bereits in der Zugangskontrolle abgewiesen wurde und er deshalb ein kurze Bearbeitungsdauer hatte oder ob es ein großer Druckauftrag bei einem langsamen Drucker war und deshalb sehr viel Zeit benötigte.

cor(t): Zu den korrekt bearbeiteten Aufträgen zählen alle Aufträge die vollständig gedruckt oder in einer Komponente des Objektmodell vorzeitig beendet wurden.

err(t): Dieses Attribut gibt die Anzahl der abgebrochen Auftragsbearbeitungen wieder. Der Grund hierfür liegt an Problemen der Funktionseinheit, z.B. nicht genügend Speicherplatz auf der Festplatte. Berechnet wird der Wert aus der Differenz von den angenommenen Aufträgen und den korrekt beendeten Aufträgen.

Statusmanagement

Betriebsbereitschaft: Drei Werte sind nötig um die Betriebsbereitschaft der Funktionseinheit darzustellen. Sie kann unbekannt sein (unkown), der Server kann für Aufträge bereit sein (enabled) oder der Server ist nicht aktiv (disabled). Die Betriebsbereitschaft des Drucksystems wird hier vom Druckdämon abhängig gemacht, denn ohne einen aktiven Druckdämon können keine Aufträge gedruckt werden, obwohl das Einreihen neuer Aufträge in die Warteschlange möglich ist. Dies liegt aber an der Realisierung des Drucksystems durch mehrere Prozesse.

Allerdings sind die Werte „unkown" und „disabled" nur darstellbar, wenn der Server und der Agent als getrennte Prozesse existieren. Sind sie durch einen Prozeß realisiert, bedeutet der Ausfall des Servers auch den Ausfall des Managementagenten und das Attribut „Betriebsbereitschaft" kann den Wert „unkown" oder „disabled" nicht mehr annehmen.

Nutzung: Wie stark das Drucksystem ausgelastet ist bzw. ob noch Ressourcen für weitere Aufträge frei sind, wird durch das Attribut „Nutzung" angezeigt. Befindet sich zur Zeit kein Auftrag im Drucksystem, nimmt dieses Attribut den Wert „idle" an, sind Aufträge im System aber noch unbenutzte Drucker vorhanden, den Wert „active", sind alle Drucker belegt, den Wert „busy". Kann der Agent den Zustand des Systems nicht exakt feststellen, erhält das Attribut den Wert „unkown".

Administration: Zur Steuerung des Druckdämons dient dieses Attribut. Kann der Agent nicht feststellen, in welchem Zustand sich der Druckdämon befindet, nimmt das Attribut den Wert „unkown" an. „Unlocked" startet den Druckdämon, wenn er nicht bereits aktive ist, d.h. Aufträge können gedruckt werden. Mit „locked“ kann der Druckdämon wieder beendet werden.

Zusätzliche Attribute

Name der Funktionseinheit: Eine Zeichenkette, welche den Namen und die Versionsnummer des verwendeten Drucksystems angibt, erleichtert die Identifizierung des Systems.

Ort der Funktionseinheit: Es besteht auch die Möglichkeit, daß der Agent nicht auf dem gleichen Recher wie das Drucksystem läuft, deshalb kann hier der entsprechende Ort angegeben werden. Laufen beide Prozesse auf dem gleichen Rechner, ist diese Information bereits an anderer Stelle in der MIB enthalten.

Startzeitpunkt des Agenten: Alle Attribute des Leistungsmanagements sind Zähler bzw. Pegel zwischen zwei Attributen. Um die Werte dieser Attribute richtig beurteilen zu können, ist es nötig zu wissen über welchen Zeitraum diese Zähler aktiv waren. Beim Starten des Agenten werden alle Attribute des Leistungsmanagements auf den Wert „0“ gesetzt, was somit der Beginn des Zeitintervalls ist. Der Startzeitpunkt des Agenten gibt an, wann der Agent gestartet wurde.

4.2.2 Bedienstationen

Im Drucksystem gibt es vier verschiedene Bedienstationtypen. Die Bedienstationen „Präfilter“ und „Postfilter“ sind für das Leistungsmanagement zwar von geringem Interesse, sollten der Vollständigkeit aber nicht übergangen werden. Der Ressourcenbedarf dieser Bedienstationen beschränkt sich auf eine niedrige Prozessor- und Speicherbelastung. Desweiteren ist es unwahrscheinlich, daß eine dieser Bedienstationen Probleme bei der Bearbeitung eines Auftrags verursacht. Interessanter sind die Bedienstationen „Zugangskontrolle“ und „Drucker“.

Im Drucksystem kann es von jedem Bedienstationtyp mehrere Objekte geben. Die Anzahl richtet sich nach der Anzahl der Warteschlangen, den ihr wird von jedem Bedienstationtyp genau eine Bedienstation zugeordnet, d.h. die Anzahl der Bedienstationen pro Typ ist gleich der Anzahl der Warteschlangen. Die eigentliche Bearbeitung des Auftrags geschieht nicht in den Bedienstationen, sondern in den enthaltenen Bedieneinheiten. Die Anzahl der Bedieneinheiten pro Bedienstation ist nicht begrenzt und richtet sich nach der Anzahl der Aufträge in der Bedienstation. Eine Ausnahme bildet die Bedienstation „Drucker“, deren Anzahl an Bedieneinheiten durch die Anzahl der verfügbaren Drucker vorgegeben ist. Von den Bedieneinheiten „Drucker“ sollten die Informationen auch nochmals für jeden Drucker einzeln angeboten werden, denn sie stellen eine große Ressourcenbindung dar.

4.2.2.1 Zugangskontrolle

Leistungsmanagement

req(t): Wenn ein Auftrag dem Drucksystem zugeführt wird, trifft er als erstes bei der Zugangskontrolle ein. Dabei gibt es mehrere Weg für den Auftrag in das Drucksystem zu gelangen. Ein Anwender, der etwas drucken möchte, bedient sich des Programms 'lpr', um einen Auftrag dem lokale Drucksystem zuzuführen. Wird der Auftrag von einem entfernten Rechner übertragen, ist der Druckdämon für die Weiterleitung des Auftrags an die Zugangskontrolle zuständig. Außerdem kann eine Warteschlange an die Zugangskontrolle einer anderen Warteschlange im gleichen Drucksystem Aufträge schicken, dabei findet aber nur noch eine eingeschränkte Zugangskontrolle statt, denn die Druckberechtigung wird nicht mehr überprüft.

Jedoch kommt jeder Auftrag als erstes bei einer Zugangskontrolle an, d.h. nur bei den Zugangskontrollen können Aufträge vom Server angenommen werden.

acc(t): Die Zugangskontrolle nimmt fast immer Aufträge an, sogar wenn der Druckdämon nicht aktiv ist, kein Druckserver gestartet werden kann oder der angeschlossene Drucker abgeschaltet ist. Ein Auftrag gilt noch nicht als angenommen, wenn die Bedienstation den Auftrag nur entgegengenommen hat. Erst mit dem Beginn der Bearbeitung des Auftrags, bei der auch eine relevante Ressourcenbindung eintritt, ist der Auftrag angenommen.

rej(t): Nur wenn die zugehörige Warteschlange der Zugangskontrolle für neue Aufträge gesperrt oder das Verzeichnis der Warteschlange nicht zugänglich ist, lehnt die Zugangskontrolle die Annahme des Auftrags ab.

dep(t): Nachdem ein Auftrag angenommen wurde, wird die Zugangsberechtigung, das Druckkontingent, die Auftragsgröße, Kopienanzahl, Auftragsformat und die angegebenen Optionen auf Erlaubnis kontrolliert. Scheitert der Auftrag an einer dieser Bearbeitungschritte wird er beendet, sonst wird er an die folgende Bedienstation „Präfilter" weitergeleitet. Trat bei der Bearbeitung ein Fehler im Programmablauf auf, wird der Auftrag auch beendet. Auf jeden Fall wird dieser Zähler um 1 erhöht.

util(t): Wieviele Auftäge sich in der Bedienstation befinden bzw. wieviele Bedieneinheiten gerade existieren, wird durch dieses Attribut wiedergegeben.

del(t): Der Ressourcenbedarf der Bedienstation „Zugangskontrolle" ist gering und die Bearbeitungszeit eines Auftrags minimal. Die Bearbeitungsdauer eines Auftrags ist auch nur unwesentlich von der Auftragsgröße abhängig, deshalb müßte sie für jeden Auftrag inetwa gleich sein.

cor(t) und err(t): Da die Bearbeitungsschritte in der Zugangskontrolle nicht sonderlich aufwendig und komplex sind und solange keine größeren Probleme im Rechensystem auftreten, wird es wohl sehr selten vorkommen, daß sich dieser Pegel erhöht. Bemerkt man jedoch eine rasches Anwachsen des Pegels, sollte die Ursache ermittelt werden. Es könnte sein, daß auf die *printcap* Datei, *printer_perm* Datei oder das Verzeichnis der Warteschlange nicht zugegriffen werden kann.

Statusmanagement

Betriebsbereitschaft: Grundsätzlich ist die Zugangskontrolle immer betriebsbereit, weshalb der Wert immer „enabled“ lautet. Solange keine Aufträge zur Bearbeitung anstehen, existiert auch keine Bedieneinheit in der Zugangskontrolle. Wird jedoch ein Auftrag erzeugt und mit Hilfe des Programms ‘lpr’ versucht diesen in die Warteschlange einzureihen, wird auch eine Bedieneinheit in der Zugangskontrolle erzeugt, welche die Bearbeitung des Auftrags übernimmt.

Nutzung: Bei der Nutzung der Zugangskontrolle können nur die Zustände „active“ und „idle“ angegeben werden. Befindet sich momentan kein Auftrag in Bearbeitung, d.h. es existiert auch keine Bedieneinheit, wird der Wert „idle“ zurückgegeben. Sobald ein Auftrag bearbeitet wird, ist der Wert des Attributs „active“. Ist die Bearbeitung des Auftrags beendet, beendet sich auch die Bedienstation wieder. Die Anzahl der parallel laufenden Bedieneinheiten ist nicht begrenzt, deshalb kann das Attribut niemals den Wert „busy“ annehmen.

Administration: Die Administration der Zugangskontrolle ist nicht möglich und auch nicht nötig. Eine Bedieneinheit der Zugangskontrolle bindet wenig Ressourcen, kann beliebig oft erzeugt werden und ist nicht lange aktiv. Sollte die Warteschlange gespeert sein, nimmt die Zugangskontrolle keinen Auftrag mehr an, weshalb alle Aufträge abgelehnt werden müssen. Deshalb erhält dieses Attribut den Wert „not applicable“.

Zusätzliche Attribute

Name der Bedienstation: Im Namen der Bedienstation wird angegeben, welcher Typ von Bedienstation vorhanden ist und welcher Warteschlange diese Bedienstation zugeordnet ist. Besonders die Angabe der zugeordneten Warteschlange ist wichtig, denn die Zugangsberechtigungen sind auch abhänig von der betroffenen Warteschlange.

4.2.2.2 Prä- und Postfilter

Diese beiden Bedienstationtypen sind sich in ihrem Verhalten sehr ähnlich, auch bieten sie die gleichen Informationsarten an. Bei der Implementierung des Managementagenten werden sie natürlich getrennt und völlig unabhängig voneinander realisiert, nur zur Beschreibung der Informationen und Funktionalität werden sie hier zusammengefaßt.

Leistungsmanagement

req(t), acc(t) und rej(t): Beide Bedienstationtypen sind nicht in der Lage Aufträge abzulehnen. Jeden Auftrag, der ihnen zugestellt wird, müssen sie annehmen und bearbeiten. Deshalb ist die Anzahl der ankommenden Aufträge (req(t)) gleich den angenommenen Aufträge (acc(t)) und es wird nie ein Auftrag abgelehnt.

dep(t): Aufträge sind in den Bedienstationen beendet, wenn sie an die nächste Komponente des Objektmodells weitergeleitet wurden, wenn ein Bearbeitungsfehler auftrat oder der Auftrag geforderte Optionen nicht erfüllte.

util(t): Die Anzahl der Aufträge in der Bedienstation ist mit der Anzahl der gerade existierenden Bedieneinheiten in der Bedienstation identisch.

del(t): Die Bearbeitungsdauer des letzten bearbeiteten Auftrags hängt vom Auftragsformat und der Auftragsgröße ab. Welche Filter aufgerufen werden müssen, wird anhand des Auftragsformats und der Konfiguration der Bedienstation bestimmt. Beim Präfilter besteht zusätzlich die Möglichkeit, den Filter ‘pr’ zu starten, wenn es in der Auftragsbeschreibung gewünscht wird. Dies ist der einzige Filter, den der Auftraggeber explizit angeben kann.

cor(t): Sobald ein Auftrag in der Funktionseinheit beendet wurde und dies nicht durch einen Programmabsturz geschah, wird dieser Zähler um eins erhöht.

err(t): Ein Auftrag, der das Progamm in einen unvorhergesehenen Zustand versetzt, wie Absturz oder Endlosschleife, kann nicht korrekt bearbeitet werden. Ähnlich wie bei der Zugangskontrolle, sollte sich dieser Pegel nur selten ändern. Mögliche Gründe für das Anwachsen des Pegels sind defekte oder fehlende Filterprogramme, aber auch Druckdaten, die vom Filterprogramm nicht bearbeitet werden können.

Statusmanagement

Betriebsbereitschaft: Die Betriebsbereitschaft des Präfilters ist immer gewährleistet, wohingegen beim Postfilter ein aktiver Druckserver, der vom Druckdämon gestartet wird, vorausgesetzt ist. Denn der Druckserver versorgt den Postfilter mit einem neuen Auftrag aus der Warteschlange. Wohingegen beim Präfilter das Programm ‘lpr’ zuständig ist, das ja vom Auftraggeber jederzeit gestartet werden kann.

Nutzung: Befinden sich in der Bedienstation Aufträge, wird der Wert des Attributes auf „active“ gesetzt, den Wert „idle“ nimmt es an, wenn keine Aufträge vorhanden sind. Da für die Anzahl der Aufträge keine Obergrenze existiert, kann es niemals den Wert „busy“ annehmen.

Administration: Nur der Bedienstationtyp „Postfilter“ kann administriert werden. Beim Präfilter ist der Wert des Attributs deshalb „not applicable“. Da es auch keinen Sinn macht die Bedienstation zu sperren, wird die Möglichkeit der Administration dieser Bedienstation auch nicht nachträglich implementiert.

Beim Präfilter gibt es die Möglichkeit Aufträge aus der Warteschlange anzunehmen, was durch den Wert „unlocked“ signalisiert wird. Dagegen zeigt der Wert „shutting down“ an, daß zur Zeit keine Aufträge angenommen, aber gerade in Bearbeitung befindliche Aufträge noch beendet werden.

Zusätzliche Attribute

Name der Bedienstation: Der Typ der Bedienstation sowie der Name der zugehörigen Warteschlange, erleichtern die Identifizierung dieser Bedienstation.

4.2.2.3 Drucker

Die Werte für den Bedienstationtyp „Drucker“ setzen sich aus den Werten der einzelnen Bedieneinheiten der Bedienstation zusammen. Einen Überblick über alle an einer Warteschlange verfügbaren Drucker zu haben ist sicher von Interesse, aber auch die einzelnen Drucker sollten die Informationen bereitstellen.

Die Bedieneinheiten des Bedienstationtyps „Drucker“ müssen aber nicht unbedingt Drucker sein. Die Aufgabe der Bedienstation kann auch lauten, die Aufträge in eine andere Warteschlange weiterzuleiten. Diese Warteschlange kann im gleichen Drucksystem enthalten sein, womit mehrere Warteschlangen einen Drucker bedienen können. Die Warteschlange kann aber auch im Drucksystem eines anderen Rechners sein, weshalb mit dem Druckdämon des entfernten Drucksystems Kontakt aufgenommen werden muß, um ihm den Auftrag schicken zu können.

Leistungsmanagement

req(t): Alle Aufträge, die bei dieser Bedienstation gedruckt oder einer anderen Warteschlange zugestellt werden, sollen gezählt werden. Der Auftraggeber hat durch seine Angabe in der Auftragsbeschreibung sich für diese Bedienstation entschieden. Bei der Anzahl der bearbeiteten Aufträge spielt auch die Anzahl und Verarbeitungsgeschwindigkeit der enthaltenen Bedieneinheiten eine Rolle.

acc(t): Dieser Wert ist immer gleich dem Wert von req(t), denn wenn ein Drucker keine Aufträge annehmen kann oder will, werden ihm auch keine Aufträge zugestellt.

rej(t): Da keine Aufträge abgewiesen werden, ist dieser Wert immer null. Um ein homogenes Objektmodell zu erhalten, wird dieser Wert auch angeboten.

dep(t): Die Bedienstation „Drucker“ ist die letzte Komponente im Objektmodell, deshalb müssen alle Aufträge spätestens hier beendet werden und können nicht an eine weitere Komponente des Objektmodells geschickt werden. Die Bedienstation ist auch für das Entfernen des Auftrags aus der Funktionseinheit (Löschen der Dateien im Warteschlangenverzeichnis) zuständig. Weiter muß sie die Zustellung einer Mail an den Auftraggeber initieren, falls dies gewünscht wurde.

util(t): In der Bedienstation können maximal soviele Aufträge enthalten sein, wie Bedieneinheiten vorhanden sind, was wiederum von der Anzahl verfügbarer Drucker abhängt.

del(t): Damit erhält man die Bearbeitungszeit des letzten in der Bedienstation beendeten Auftrags. Allerdings muß dieser Wert immer in Relation mit der Auftragsgröße gesehen werden.

cor(t): Aufträge, die ausgedruckt oder wegen einer fehlerhaften Auftragsbeschreibung beendet wurden, werden hier gezählt. Dabei gibt es viele Fehlerquellen die berücksichtigt werden müssen und ein korrektes Bearbeiten des Auftrags verhindern.

err(t): Die Bedieneinheiten der Bedienstation sind sehr anfällig für Fehler. Es kann keine Verbindung zum entfernten Rechner oder zum Drucker hergestellt werden. Der Drucker wird ausgeschaltet, hat einen Papierstau oder kein Papier mehr und die Bedieneinheit kann dieses Problem nicht abfangen.

Zu beachten sind hierbei Aufträge, die wieder in die Warteschlange zurückgestellt wurden, weil z.B. keine Verbindung zu einem entfernten Drucksystem hergestellt werden konnte. Ist die Häufigkeit des Zurückstellens für einen Auftrag begrenzt, wird er erst nach Erreichen dieser Häufigkeit entfernt und darf beim Warteschlangenausgang gezählt werden, in dieser Bedienstation wird er jedesmal gezählt.

Statusmanagement

Betriebsbereitschaft: Solange wenigstens eine Bedieneinheit „Drucker“ der Bedienstation „Drucker“ aktiv (on-line) ist, ist die Bedienstation betriebsbereit, was durch den Attributwert „enabled“ signalisiert wird. Ist keine Bedieneinheit Drucker mehr verfügbar, nimmt das Attribut den Wert „disabled“ an. Bei einigen Druckern ist es allerdings nicht möglich den Status des Druckers abzufragen, erst beim Scheitern der Zustellung der Druckdaten an den Drucker erkennt man das Problem. Kann der Zustand des Druckers nicht abgefragt werden, aber der Verbindungsaufbau ist möglich, erhält das Attribut bereits den Wert „enabled“.

Nutzung: Bearbeitet zur Zeit kein Drucker einen Auftrag, wird das Attribut auf „idle“ gesetzt, sollt bereits mindestens ein Auftrag bearbeitet werden, aber noch freie Drucker vorhanden sein, auf „active“. Sind alle Drucker belegt wird dem Attribut der Wert „busy“ zugewiesen.

Administration: Kann die Bedienstation keine neuen Aufträge mehr annehmen (shutting down) wird dies durch das Sperren des Postfilters erreicht, womit auch die Bedienstation „Drucker“ keine Aufträge mehr erhält. Den Attributwert „locked“ könnte man als das sofortige Beenden der Auftragsbearbeitung bei allen Bedieneinheiten „Drucker“ sehen, besser ist jedoch eine explizite Beendigung der Bedieneinheit selbst. Auch um wenigstens eine Bedieneinheit „Drucker“ aktiv zu haben, wendet man sich lieber an eine Bedieneinheit direkt, anstatt in der Bedienstation den Wert „unlocked“ zu setzen. Infolgedessen ist dieses Attribut auf „not applicable“ gesetzt.

Zusätzliche Attribute

Name der Bedienstation: Der Typ der Bedienstation und der Name der Warteschlange, aus welcher die Aufträge entnommen werden.

4.2.3 Bedieneinheit „Drucker“

Sind einer Warteschlange mehrere Drucker zugeordnet, ist es interessant Informationen für jeden einzelnen Drucker zu haben. Dabei sollten die gleichen Informationen wie bei der Bedienstation „Drucker“ bereitgestellt werden. Ist nur ein Drucker vorhanden, ist die Information der meisten Attribute mit der der Bedienstation „Drucker“ identisch, die Attribute der Administration unterscheiden sich aber. Die Informationen der einzelnen Drucker sind wichtig, denn der Drucker ist die Ressource, die für den Engpaß im Drucksystem verantwortlich ist. Der Ausfall oder die Beeinträchtigung eines Druckers stellt gewöhnlich eine erhebliche Verschlechterung des Drucksystems dar.

Leistungsmanagement
Die Attribute für das Leistungsmanagement entsprechen den Attributen des Leistungsmanagements bei der Bedienstation „Drucker“ und werden deshalb hier nicht nochmals erläutert.

Statusmanagement

Betriebsbereitschaft: Ist der Drucker abgeschaltet, im Zustand „off–line“, die Verbindung zu ihm unterbrochen, etc., nimmt das Attribut den Wert „disabled“ an. Können Aufträge an den Drucker geschickt werden, den Wert „enabled“. Dabei wird vorausgesetzt, daß der Zustand des Druckers abgefragt werden kann, sonst kann die Betriebsbereitschaft nicht ermittelt werden.

Nutzung: Nur zwei Attributwerte sind bei der Nutzung möglich, denn die Bedieneinheit kann immer nur einen Auftrag bearbeiten. Ist die Bedieneinheit nicht beschäftigt, wird der Wert „idle“ geliefert, sobald ein Auftrag bearbeitet wird der Wert „busy“.

Administration: Ist der Druckserver eines Druckers aktiv, so ist der Wert des Attributes „unlocked“. Soll jedoch nach Beendigung des gerade aktiven Auftrags der Druckserver beendet werden, muß dieses Attribut auf „shutting down“ gesetzt werden. Wird das Attribut auf „locked“ gesetzt, muß der Druckserver sofort beendet werden, d.h. auch die gerade aktive Bearbeitung eines Auftrags wird beendet.

Zusätzliche Attribute

Name der Bedienstation: Ist einer Warteschlange nur ein Drucker zugeordnet, ist der Name des Druckers mit dem Namen der Warteschlange identisch. Gibt es allerdings mehrere Drucker, erhält jeder Drucker einen eigenen Namen. Neben dem Namen des Druckers wird deshalb auch noch der Name der zugehörigen Warteschlange mitangegeben.

Status der Bedienstation: Als Status der Bedienstation wird der Status des Druckservers angesehen. Der Status ist eine Beschreibung, welche in einer Datei in der zugehörigen Warteschlange abgelegt ist.

Aufstellungsort: Nur bei Druckern kann angegeben werden, wo sie aufgestellt sind. Sie müssen nicht notwendigerweise im gleichen Raum wie der Rechner aufgestellt sein. Ist die Aufgabe der Bedieneinheit die Übertragung des Auftrags an eine andere Warteschlange, enthält dieser Eintrag den Namen der anderen Warteschlange.

Verbindungsart: Der Drucker kann an das Drucksystem über verschiedene Verbindungsarten angeschlossen sein. Es können eine serielle Verbindung, eine TCP Verbindung, usw. sein, aber es kann auch eine andere Warteschlange sein, deren Name und der Name des Rechners dann bereits im Attribut „Aufstellungsort“ vermerkt ist.

4.2.4 Warteschlange

Die Warteschlange gleicht das Zeitproblem von besetzten Ressourcen und neu ankommenden Aufträgen aus. Sie sammelt alle neu ankommenden Aufträge, sortiert sie unter Berücksichtigung ihre Bedienstrategie, um die Aufträge an die besetzten Ressourcen weitergeben zu können, sobald diese frei und für einen neuen Auftrag bereit sind.

Leistungsmanagement

req(t): Wurde in das Verzeichnis, welches die Warteschlange realisiert, ein Auftrag eingebracht, erhält der Druckdämon eine Meldung, daß ein neuer Auftrag zur Bearbeitung ansteht. Den für diese Warteschlange zuständigen Druckservern wird die Meldung weitergereicht und gleichzeitig wird der Wert dieser Variablen um eins erhöht.

acc(t): Möchte man das Einreihen von neuen Aufträgen in die Warteschlange verhindern, geschieht dies durch das Sperren der Warteschlange. Da bereits bei der Zugangskontrolle überprüft wird, ob die Warteschlange gesperrt ist, werden ihr auch keine Aufträge mehr zugestellt, wenn sie gespeert ist, weshalb der Wert dieses Attributes immer gleich dem von req(t) ist.

rej(t): Der Wert dieses Attributs ist immer null.

dep(t): Aufträge werden erst aus der Warteschlange entfernt, wenn sie entweder korrekt gedruckt oder aufgrund einer Bearbeitungs- oder Auftragsoption aus dem Drucksystem entfernt wurden.

util(t): Die Anzahl der in der Warteschlange befindlichen Aufträge wird durch dieses Attribut angegeben.

del(t): Hierin ist die Bearbeitungsdauer des letzten bearbeiteten Auftrags abgelegt. Diese Zeitspanne reicht vom Einreihen des Auftrags in die Warteschlange bis zum Entfernen des Auftrags aus der Warteschlange, was erst geschieht, wenn der Auftrag fertig bearbeitet wurde und die Funktionseinheit verläßt.

cor(t): Korrekt wurde ein Auftrag bearbeitet, wenn er gedruckt wurde und seine Daten aus dem Warteschlangenverzeichnis gelöscht sind oder er vom Drucksystem abgewiesen wurde, aufgrund unzulässiger Optionen oder sonstigem.

err(t): Konnte das Drucksystem dagegen die Bearbeitung wegen Software- oder Hardwarefehler nicht vollständig durchführen, erhöht sich der Wert dieser Variablen um eins.

Statusmanagement

Betriebsbereitschaft: Die Betriebsbereitschaft der Warteschlange ist abhängig vom Verzeichnis, das diese Warteschlange realisiert. Solange dieses Verzeichnis zugänglich ist, um neue Aufträge in die Warteschlange zu schreiben, hat das Attribut den Wert „enabled“. Sollte das Verzeichnis nicht zugänglich sein, den Wert „disabled“.

Nutzung: Solange kein Auftrag in der Warteschlange ist, erhält das Attribut den Wert „idle“, kommen jedoch Aufträge in die Warteschlange, den Wert „active“. Den Wert „busy“ kann es nie annehmen, denn die Anzahl der Aufträge in der Warteschlange ist nicht begrenzt. Natürlich kann es passieren, daß die Warteschlange „voll“ ist (nicht mehr genug Plattenplatz), aber das wird nicht berücksichtigt.

Administration: Die Administration der Warteschlange ist ein wichtiger Punkt beim Management des Drucksystems . Damit kann die Annahme und Weiterbearbeitung der Aufträge für einzelne Warteschlangen gesteuert werden. Nimmt die Warteschlange Aufträge an, erhält das Attribut den Wert „unlocked“. Sollen keine neuen Aufträge mehr angenommen werden, kann dies durch den Attributwert „locked“ erreicht werden.

Zusätzliche Attribute

Name der Bedienstation: Für eine Warteschlange können in der *printcap* Datei zwar mehrere Namen angegeben werden, in dieser Variablen ist allerdings nur der erste in der *printcap* Datei angegebene Name abgelegt. Deshalb sollte der erste angegebene Name aus einer aussagekräftigen Zeichenkombination bestehen, welcher auf den zugehörigen Druckertyp und Aufstellungsort deutet, wie z.B. psLRZ345, was auf einen Postscript–Drucker im Leibniz–Rechnenzentrum im Raum 345 deutet.

4.2.5 Auftragsmanagement

Für den Systemadministrator ist es wichtig genauere Angaben über die in der Funktionseinheit enthaltenen Aufträge zu haben. Weiter sollte jeder Auftrag in der Funktionseinheit lokalisierbar sein, d.h. bekannt sein in welchen Komponeten des Objektmodells er sich gerade befindet. Um diese Ansprüche zu erfüllen, sind folgende Informationen von nöten:

Identifizierung: Jeder Auftrag muß in der Funktionseinheit eindeutig identifizierbar sein, dazu dient im Drucksystem die Auftragsnummer, die jeder Auftrag beim Eingang ins System erhält.

Auftragseingangszeit: Diese Zeit bietet das Drucksystem nicht an und muß deshalb noch ermittelt werden.

Auftraggeber: Jeder Auftrag hat einen Auftraggeber, der einen Ausdruck seiner Daten wünscht. Dem Drucksystem muß natürlich bekannt sein, wer der Auftraggeber ist. Neben der Benutzerkennung des Auftraggebers sollte auch noch der Rechnername, von dem der Auftrag abgeschickt wurde, angegeben werden.

Auftragsposition in der Funktionseinheit: Die Position des Auftrags in der Funktionseinheit wird durch die Angabe der Komponenten des Objektmodells, in denen der Auftrag sich gerade befindet, mitgeteilt. Beim Drucksystem kann er in bis zu drei Komponenten gleichzeitig enthalten sein. Die drei gleichzeitig möglichen Komponenten sind die Warteschlange und die Bedienstationen „Postfilter“ und „Drucker“.

Auftragsgröße: Beim Drucksystem wird als Auftragsgröße die zu druckende Datenmenge gesehen. Die Auftragsbeschreibung kann aufgrund ihrer geringen Größe vernachlässigt werden.

Löschen eines Auftrags: Das Drucksystem stellt mehrere Methoden bei der Auswahl von Aufträgen, die gelöscht werden sollen, zur Verfügung. Allerdings reicht es einen bestimmten Auftrag löschen zu können. Möchte man mehrere Aufträge löschen ist es erforderlich jeden einzeln zu löschen. Wird ein Auftrag gelöscht, müssen sowohl die Auftragsdaten und die Auftragsbeschreibung gelöscht werden, er muß aber auch aus allen Komponenten des Objektmodells entfernt werden, in denen er sich zur Zeit befindet. Wurde bereits damit begonnen, die Druckdaten an einen Drucker zu übermitteln, muß auch der Drucker zurückgesetzt werden.

Zusätzliche Attribute

Warteschlangeneingangszeit: Im Drucksystem selbst ist die Zeit, wann der Auftrag in die Warteschlange eingereiht wird, ausschlaggebend. Anhand dieser Zeit wird die Reihenfolge der Aufträge in der Warteschlange festgelegt und auch bei den Managementwerkzeugen des Drucksystems wird auf diese Zeit zurückgegriffen. Um mit diesen Werkzeugen und der Reihenfolgefestlegung von Aufträgen konform zu sein, wird deshalb auch der Eintreffzeitpunkt in der Warteschlange angegeben.

Priorität: Zur Festlegung der Reihenfolge der Aufträge in der Warteschlange dient die Priorität. Durch das Ändern der Priorität eines Auftrags kann der Systemverwalter den Auftrag vorziehen oder zurückstellen.

4.2.6 Zusammenfassung

Die vom Objektmodell geforderte Managementinformation und -funktionalität kann vom Drucksystem häufig nicht erbracht werden. Besonders die Attribute für das Leistungsmanagement wurden völlig übergangen, allerdings können sie nachträglich ohne größeren Aufwand implementiert werden. Dazu müssen aber geeignete Meldungen vom Drucksystem erzeugt und dem Subagenten geschickt werden. Wie diese Meldungen aussehen und von wo sie im Drucksystem abgeschickt werden müssen, wird in Abschnitt 6.5 beschrieben.

Bei den Statusattributen und dem Auftragsmanagement sieht es besser aus. Viele dieser Informationen und Funktionalität wird von den Managementtools des Drucksystems angeboten und kann in gleicher Weise auch für den Subagenten verwendet werden. Bei allen Bedienstationen die Möglichkeiten der Statusattribute voll auszuschöpfen, ist dabei aber weder nötig noch realisierbar, was auch vom Objektmodell nicht verlangt wird.

Es gibt aber auch Managementanforderungen an den Druckdienst, die vom Objektmodell nicht abgedeckt sind und deshalb noch aufgenommen werden müssen. Da es von einigen Objekttypen mehrere Objekte gibt, wird ein eindeutiger Name für jedes Objekt benötigt, um dem Anwender die Unterscheidung zwischen den Objekten zu erleichtern. Besonders die Zusammengehörigkeit von Objekten in einem Zweig des funktionalen Modells sollte erkennbar sein. Natürlich ist die Zusammengehörigkeit in den Beziehungen der Objekte zueinander enthalten, existiert jedoch keine geeignete Darstellungsform dieser Beziehungen, ist sie für den Anwender schwer durchschaubar. Geeignete Namen der Objekte sind eine erste Hilfestellung.

Sehr wichtig ist vor allem, den genauen Startzeitpunkt der Überwachung zu kennen. Viele der Attribute sind Zähler, die das Eintreten eines Ereignisses in einem Zeitintervall zählen. Ist die Länge des Zeitintervalls unbekannt, können die Werte schwer interpretiert werden.

Für die Bedieneinheit „Drucker" wurden außerdem noch alle Attribute des Leistungsmanagements aufgenommen. Aufgrund des Ressourcenengpasses, den der Drucker darstellt, ist es wichtig den Durchsatz und die Fehlerhäufigkeit dieser Komponente zu kennen. Dazu kommen noch Attribute für den Status und den Verbindungstyp des Druckes.

Auch der Auftrag benötigt zwei zusätzliche Attribute. Um die Reihenfolge der Aufträge in der Warteschlange festzulegen, sind die Priorität und Warteschlangeneingangszeit eines Auftrags ausschlaggebend. Über diese Attribute kann der Systemverwalter auch Einfluß auf die Auftragsreihenfolge nehmen.

Die vom Druckdienst angebotene Managementinformation und -funktionalität dient hauptsächlich zu seiner Steuerung und Konfiguration sowie den darin enthaltenen Aufträgen. Leistungs- und Statusattribute werden weniger bereitgestellt, da sie für die Funktionsweise des Druckdienstes von untergeordnetem Interesse sind.

4.3 Managementanforderungen an das Nachrichtensystem

Nachdem im vorangegangen Abschnitt die Managementanforderungen an den Druckdienst erläutert wurden, werden sie in diesem Kapitel für den Nachrichtendienst aufgezählt. Beim Druckdienst geschah dies sehr ausführlich, weshalb der Sinn der Attribute jetzt sicher geläufig ist und sie hier nur noch kurz beschrieben werden müssen.

4.3.1 Funktionseinheit

Die Zusammenfassung der Attributwerte für den Nachrichtendienst ist einfach zu bewerkstelligen. Zum einen gibt es nur drei Bedienstationtypen mit genau je einem Objekt und zum anderen kann das Nachrichtensystem nur gering beeinflußt werden.

Leistungsmanagement

req(t): Neue Aufträge werden dem Nachrichtensystem entweder mit einem Programm, wie 'elm' oder 'mail' zugestellt, sie können aber auch von einem Sendmaildämon eines entfernten Rechners kommen. Dies sind die einzigen Möglichkeiten einen Auftrag dem Nachrichtendienst zuzustellen.

acc(t) und rej(t): Das Nachrichtensystem ist nicht in der Lage Aufträge abzulehnen, weshalb acc(t) gleich dem Wert von rej(t) ist. Daraus folgt auch, daß rej(t) immer den Wert „0“ besitzt.

dep(t), util(t), del(t), cor(t) und err(t): Diese Attribute entsprechen der Definition.

Statusmanagement

Betriebsbereitschaft: Ist der Sendmaildämon aktiv, ist die Funktionseinheit betriebsbereit. Dieser Sendmaildämon muß die Annahme von Nachrichten von entfernten Sendmailsystemen übernehmen, aber auch regelmäßig die Zustellung der in der Warteschlange befindlichen Aufträge veranlassen. Ist kein Sendmaildämon aktiv, können trotzdem Aufträge bearbeitet und Nachrichten verschickt werden. Denn mit jedem Auftrag wird ein Sendmailprozeß gestartet, der versucht die Nachricht zuzustellen. Nur wenn dies mißlingt oder nicht gewünscht wurde, bleibt der Auftrag in der Warteschlange liegen und muß durch den Sendmaildämon zugestellt werden.

Nutzung: Wird zur Zeit kein Auftrag bearbeitet, kann das Attribut den Wert „idle“ annehmen. Sonst kann es nur noch den Wert „active“ erhalten, denn in der Funktionseinheit existiert keine Obergrenze für die Auftragsanzahl.

Administration: Nur der Druckdämon kann administriert werden. Durch Setzen des Attributs auf „locked“ wird der Nachrichtendämon beendet und Aufträge verbleiben in der Warteschlange, wenn sie nicht sofort bei der Auftragsübergabe zugestellt wurden. „Unlocked“ startet den Dämon wieder.

Zusätzliche Attribute
Die Attribute **Name** und **Ort** der Funktionseinheit sowie die **Laufzeit** des Agenten sind auch hier nötig. Die Attribute sind bereits beim Drucksystem ausführlich beschrieben und werden hier nicht nochmals erklärt.

4.3.2 Bedienstationen

Die Bedienstationen „Lokal“ und „Entfernt“ im Drucksystem sind sich sehr ähnlich und werden deshalb auch gemeinsam behandelt. Daneben gibt es noch die Bedienstation „Aufbereitung“. Allerdings bieten alle Bedienstationen wenig Administrationsmöglichkeiten.

4.3.2.1 Aufbereitung

Leistungsmanagement

req(t): Alle Aufträge, ob sie nun von einem anderen Nachrichtensystem kommen oder von einem lokalen Anwender stammen, müssen als erstes in dieser Bedienstation bearbeitet werden. Deshalb werden bei dieser Bedienstation alle Aufträge gezählt, die dem Nachrichtensystem übergeben werden.

acc(t) und rej(t): Aufträge werden immer angenommen, weshalb der Wert von acc(t) mit dem Wert von req(t) identisch ist. Daraus folgt auch der Wert „0“ für rej(t).

dep(t): Nachdem der Kopf des Auftrags geprüft und vervollständigt, Aliases aufgelöst und die Regeln angewandt wurden, kann der Auftrag an die Warteschlange weitergegeben werden. Auch Aufträge, die hier von einem anderen Nachrichtensystem empfangen wurden, müssen überprüft werden, ob die Daten vollständig sind und an welche Empfänger die Nachricht geschickt werden soll. Scheitert die Bearbeitung des Auftrags kann auch eine Meldung an den Absender erfolgen.

util(t): Wieviele Auftäge sich in der Bedienstation befinden, bzw. wieviele Bedieneinheiten gerade existieren wird durch dieses Attribut wiedergegeben.

del(t), cor(t) und err(t): Die Bearbeitungsdauer eines Auftrags ist nur kurz und auch von der Auftragsgröße nur unwesentlich abhängig. Fehler in der Bedienstation werden auch nur sehr selten auftreten, denn der Ressourcenbedarf ist nicht sehr hoch und die Bearbeitung von niedriger Komplexität.

Statusmanagement

Betriebsbereitschaft: Die teilweise Betriebsbereitschaft der Bedienstation ist immer gewährleistet, solange es möglich ist das Programm ‘sendmail’ zu starten und das Verzeichnis, welches die Warteschlange realisiert, existiert und zugänglich ist. Ist allerdings der Sendmaildämon nicht aktiv, können keine Aufträge von anderen Nachrichtensystemen angenommen werden, nur lokale Benutzer können Nachrichten verschicken. Deshalb bezieht sich dieses Attribut auf den Sendmaildämon, welcher Aufträge von anderen Nachrichtensystemen annehmen kann.

Nutzung: Bei der Nutzung der Aufbereitung können nur die Zustände „active“ und „idle“ angegeben werden. Ist die Bearbeitung des Auftrags beendet, beendet sich auch die Bedieneinheit wieder. Die Anzahl der gleichzeitig existierenden Bedieneinheiten ist nicht begrenzt, deshalb kann das Attribut niemals den Wert „busy“ annehmen.

Administration: Die Administration der Aufbereitung ist zuständig für die Empfangsbereitschaft des Nachrichtensystems von Aufträgen von anderen Nachrichtensystemen. Durch Setzen des Wertes auf „unlocked“ wird der Dämon gestartet, falls er nicht bereits aktiv ist. Umgekehrt wird der Dämon beendet, wenn der Wert auf „locked“ gesetzt wird.

Zusätzliche Attribute

Name der Bedienstation: Da im Drucksystem nur eine Bedienstation „Aufbereitung“ existiert, reicht bereits die Angabe des Bedienstationtyps.

4.3.2.2 Lokal und Entfernt

Die beiden Bedienstationen unterscheiden sich durch die Zustellart für Nachrichten, sie kann lokal auf diesem Rechner erfolgen oder an ein anderes Rechnersystem übermittelt werden. Damit festgehalten werden kann, wieviele Nachrichten an ein anderes Nachrichtensystem weitergeleitet und wieviele direkt zustellen wurden, sind zwei getrennte Bedienstationen vonnöten.

Leistungsmanagement

req(t): Die Aufträge aus der Warteschlange verteilen sich auf die beiden Bedienstationen „Lokal“ und „Entfernt“, wieviele Aufträge jede einzelne erhielt, ist in diesen Zählern vermerkt.

acc(t) und rej(t): Auch diese Bedienstationen können keine Aufträge ablehnen, weshalb acc(t) = req(t) und rej(t) = 0.

dep(t): Die Bedienstationen sind verantwortlich für den Abschluß des Auftrags, was heißt Aufträge, die hier beendet werden, verlassen auch die Funktionseinheit. Es gibt jedoch Aufträge die aufgrund von vorübergehenden Problemen nicht abgeschlossen werden konnten, die Bedienstation aber verließen, nicht jedoch die Warteschlange. Später wird dann abermals versucht diese Auftäge korrekt zu beenden.

util(t): Die Anzahl der Bedieneinheiten richtet sich wieder nach der Anzahl der Aufträge in der Bedienstation.

del(t): Damit erhält man die Bearbeitungszeit des letzten in der Bedienstation beendeten Auftrags.

cor(t): Aufträge, die ausgedruckt oder wegen einer fehlerhaften Auftragsbeschreibung beendet wurden, werden hier gezählt.

err(t): Kann eine Nachricht nach mehreren Versuchen immer noch nicht zugestellt werden, wird sie letztendlich verworfen. Wann das geschieht und wie, ist abhänig von der Konfiguration und der Priorität der Nachricht. Gründe für erfolglose Zustellversuche können sein, Probleme mit dem Netzwerk, kein aktiver Sendmaildämon auf dem entfernten Rechner oder ein nicht vorhandenes Postfach.

Statusmanagement

Betriebsbereitschaft: Für die sofortige Bearbeitung von Aufträgen lokaler Benutzer sind die Bedienstationen immer betriebsbereit. Bei der Bearbeitung von Aufträgen aus der Warteschlange, die dort schon einige Zeit liegen, und Aufträgen von entfernten Nachrichtensystemen ist es abhängig vom Sendmaildämon. Der Sendmaildämon wird aber bereits bei der Bedienstation „Aufbereitung“ und der Warteschlange behandelt, deshalb erhält das Attribut hier den Wert „unkown“.

Nutzung und Administration: Die Nutzung kann die Werte „idle“ und „active“ annehmen, die Administration nur den Wert „not applicable“.

Zusätzliche Attribute

Name der Bedienstation: Da es im Drucksystem nur je ein Objekt von diesen beiden Bedienstationtypen gibt, reicht bereits die Angabe des Bedienstationtyps.

4.3.3 Warteschlange

Aufträge in der Warteschlange verlassen diese in der selben Reihenfolge wieder, wie sie eingetroffen sind. Ausgenommen davon sind Aufträge, die sofort zugestellt werden sollen und Aufträge, die von anderen Nachrichtensystemen kommen. Sie werden in der Warteschlange nur kurz zwischengespeichert um bei einem Systemabsturz nicht verloren zu gehen. Man kann sich überlegen, diese Aufträge bei den Attributen der Warteschlange nicht mitzuzählen, da sie sich nicht nach der Bedienstrategie der Warteschlange richten. Andererseits möchte man wissen wieviele Aufträge die Warteschlange passiert haben. Deshalb wird die Bedienstrategie der Warteschlange erweitert. Es gibt Aufträge, die sofort weitergeleitet und solche, die hinter den bereits wartenden Aufträgen eingereiht werden.

Leistungsmanagement

req(t), acc(t) und rej(t): Die Dateien eines neuen Auftrags werden nur in das Verzeichnis, welches die Warteschlange realisiert, kopiert. Damit ist der Auftrag von der Warteschlange akzepiert und bereit für die Weiterleitung, deshalb kann er auch nicht abgelehnt werden.

dep(t): Aufträge werden erst aus der Warteschlange entfernt, wenn sie entweder korrekt zugestellt oder aufgrund einer Bearbeitungs- oder Auftragsoption aus dem Nachrichtensystem entfernt wurden.

util(t): Die Anzahl der in der Warteschlange befindlichen Aufträge wird durch dieses Attribut angegeben.

del(t): Hierin ist die Bearbeitungsdauer des letzten bearbeiteten Auftrags abgelegt. Diese Bearbeitungsdauer umfaßt die Aufenthaltszeit in der Warteschlange und fast die volle Bearbeitungszeit in einer der Bedienstationen „Lokal“ oder „Entfernt“. Wird mehrmals versucht die Nachricht zuzustellen, summieren sich die Zeiten.

cor(t) und err(t): Nachdem der Auftrag die Warteschlange verlassen hat, da er korrekt zugestellt oder verworfen wurde, kann der Wert von cor(t) inkrementiert werden, sonst err(t).

Statusmanagement

Betriebsbereitschaft: Die Betriebsbereitschaft der Warteschlange bezieht sich hier nur auf Aufträge, die sich bereits einige Zeit in der Warteschlange befinden. Aufträge, die sofort weitergeleitet werden, sind nicht auf einen aktiven Sendmaildämon angewiesen, der regelmäßig versucht die Aufträge weiterzugeben. Damit ist auch schon gesagt worauf es ankommt, auf einen aktiven Sendmaildämon, der für die Warteschlange zuständig ist.

Nutzung: Nur die Werte „idle“ und „active“ sind möglich, denn die Anzahl der Aufträge in der Warteschlange ist nicht begrenzt.

Administration: Wird der Sendmaildämon beendet, was durch Setzen des Attributs auf „locked“ geschieht, verbleiben die Aufträge in der Warteschlange, wenn sie nicht sofort zugestellt werden können. Zum Starten des Dämons wird der Wert auf „unlocked“ gesetzt. Es ist jedoch nicht möglich zu verhindern, daß neue Aufträge in die Warteschlange eingereiht werden.

Zusätzliche Attribute

Name der Bedienstation: Da es im Drucksystem genau eine Warteschlange gibt, reicht bereits die Angabe „Nachrichtensystemwarteschlange“.

Zeitintervall: Hiermit kann man dem Sendmaildämon mitteilen, in welchen Zeitintervallen er versuchen soll, die Aufträge aus der Warteschlange weiterzuleiten.

Warteschlangenverzeichnis: Die Aufträge in der Warteschlange sind festgelegt durch die Auftragsdateien im Warteschlangenverzeichnis. Möchte man die Aufträge einer Warteschlange für einige Zeit zurückstellen, da die Aufträge z.B. an ein Nachrichtensystem geschickt werden sollen, welches aufgrund von Netzwerkproblemen für einige Tage nicht erreichbar ist, wird einfach ein neues Warteschlangenverzeichnis angegeben, in welches forthin alle neuen Aufträge kommen und auch weitergeleitet werden. Sollen die Aufträge aus der alten Warteschlange zugestellt werden, wird nur wieder das Warteschlangenverzeichnis auf den alten Namen zurückgestellt.

4.3.4 Auftragsmanagement

Die Aufträge im Nachrichtensystem müssen durch die Attribute des Auftragsmanagements noch genauer beschrieben werden. Dafür existieren folgende Attribute:

Identifizierung: Zur Identifizierung des Auftrags erhält jeder Auftrag eine weltweit eindeutige Bezeichnung, die sich aus dem Datum, der Uhrzeit, der Nachrichtensystemkennung und dem Domänennamen ergibt.

Auftragseingangszeit: In jedem Auftrag ist auch abgelegt, wann der Auftrag dem Nachrichtensystem übergeben wurde. Allerdings kann der Auftraggeber diesen Eintrag bei der Auftragserstellung auch selbst angeben, was zu Ungenauigkeiten führt. Deshalb ist es besser sich diese Zeit selbst zu besorgen.

Auftraggeber: Jeder Auftrag hat einen Auftraggeber, der diese Nachricht verschicken möchte. Der Name des Auftraggebers ist im Kopf des Auftrags angegeben und kann von dort ermittelt werden. Allerdings ist es nicht schwierig einen falschen Namen anzugeben.

Auftragsposition in der Funktionseinheit: Die Position des Auftrags in der Funktionseinheit wird durch Angabe der Komponenten des Objektmodells, in denen er sich gerade befindet, festgelegt. Beim Nachrichtensystem kann er in einer oder zwei Komponenten gleichzeitig enthalten sein. Die möglichen zwei Komponenten sind die Warteschlange und eine der Bedienstationen „Lokal“ oder „Entfernt“.

Auftragsgröße: Beim Nachrichtensystem wird als Auftragsgröße die Nachrichtengröße gesehen. Die Empfängerliste und der Kopf des Auftrags werden aufgrund ihrer meist einigermaßen konstanten Größe vernachlässigt.

Löschen eines Auftrags: Es ist nicht möglich einen Auftrag zu löschen, der einmal im Nachrichtensystem angelangt ist. Nur durch Löschen der Auftragsdateien im Warteschlangenverzeichnis verschwindet der Auftrag im Nachrichtensystem. Befindet er sich aber gerade auch in einer Bedienstation, führt dies zu Fehlern.

4.3.5 Zusammenfassung

Da der Agent für das Nachrichtensystem nicht implementiert wurde, können keine sicheren Aussagen darüber gemacht werden. Bei der Bereitstellung der generischen Managementinformation und -funktionalität vom Nachrichtensystem scheint es sich aber ähnlich zu verhalten wie beim Drucksystem.

Genauso bei den implementierungsspezifischen Managementanforderungen des Drucksystems. Auch hier kommt man nicht ohne zusätzliche Attribute aus, um nicht sinnvolle Managementinformation zu verlieren. Einige Attribute entsprechen dabei denen des Drucksystems, andere sind speziell für den Nachrichtendienst vonnöten, wie z.B. das Zeitintervall für die Nachrichtenzustellung.

Kapitel 5

Abbildung des Objektmodells in SNMP

Damit die Managementanwendung auf die Attribute des Objektmodells zugreifen kann, müssen diese in einer MIB zusammengefaßt und angeboten werden. Heute gibt es zwei Informationsmodelle, in denen MIBs beschrieben werden können, die Internet MIB und die ISO/CCITT GDMO MIB. Während die GDMO MIB bereits objektorientierte Konzepte enthält und dadurch die Abbildung des Objektmodells auf die GDMO MIB vereinfacht, gibt es bei der Abbildung zur Internet MIB mehrere Hürden zu überwinden. Deshalb und weil im LAN–Bereich die meisten Managementanwendungen (HP OpenView, Spectrum) SNMP basiert sind, wird in diesem Kapitel eine Abbildungsvorschrift für das Objektmodell in eine Internet MIB vorgestellt.

Wie diese Abbildungsvorschrift, angewandt auf unsere beiden Basisdienste, ein Objektmodell in eine Internet MIB verwandelt, wird am Ende diese Kapitels gezeigt.

5.1 Die Abbildungsvorschrift

Wie bereits in der Einleitung dieses Kapitels angedeutet wurde, ist das Objektmodell von einer GDMO MIB gar nicht so verschieden. Beide enthalten objektorientierte Konzepte, für die es gilt sie in einer Internet MIB darzustellen. Die Abbildung einer GDMO MIB in eine Internet MIB wurde bereits in [New 94] herausgearbeitet und umgekehrt die Abbildung einer Internet MIB in eine GDMO MIB in [Lab 94]. Bei der Abbildung des Objektmodells in eine Internet MIB wurde nun versucht sich so weit wie möglich an die Richtlinien diese IIMC Dokumente zu halten.

5.1.1 Konzepte und Relationen im Objektmodell

In Kapitel 2 wurden die zentralen Konzepte der Objektorientierung bereits kurz angesprochen. Um die Konzepte im Objektmodell nun in die SNMP Welt abbilden zu können ist es wichtig sich im klaren über die Begriffe zu sein. Nur so kann entschieden werden, welche Konzept abgebildet werden sollen und können. Zur Erklärung der Begriffe wurde auf die Quellen [Cox 87], [Clau 93], [Booch 91], [Kemp 90] und [New 94] zurückgegriffen.

5.1.1.1 Objekt

Bei näherer Betrachtung der Objekte im Objektmodell stellt man fest, daß so ein Objekt folgende Eigenschaften besitzt:

- seine Attribute
- seine Methoden
- Meldungen vom Objekt, auch spontan

Die Attribute beschreiben den Zustand und das Wissen eines konkreten Objekts. Typischerweise sind die Attribute einfache Variablen oder Tabellen. Jedes Attribut hat einen Typ, einen Wertebereich und möglicherweise eine Einschränkung der Zugriffsrechte.

Anweisungen (Methoden) an das Objekt dienen zum Lesen oder Verändern von Attributen. Mit dem Ändern von Attributen können Aktionen des Objekts angestoßen werden. So kann man durch Setzen des Attributs „Administration“ des Objekts „Drucker“ auf „down“ das Abschalten des Druckers bewirken. Das Objekt kann aber auch Meldungen an andere Objekte senden oder auf bestimmte Ereignisse warten. In unserem Objektmodell wird der Auftrag, genauer eine Meldung, an das folgende Objekt geschickt.

Treten unvorhergesehene Änderungen bei einem Attribut auf oder wird ein Schwellwert überschritten, kann das Objekt auch von sich aus Meldungen erzeugen und abschicken. Z.B. soll das Objekt „Drucker“ eine Meldung machen, wenn der repräsentierte Drucker eine Fehlermeldung kundtut. Meldungen von einem Objekt oder für ein Objekt sollten immer den Namen des sendenden und des empfangenden Objekts enthalten sowie die notwendigen Parameter zur Angabe des Grundes.

Durch das Verbot der direkten Veränderung von Attributen von „außen“ kann das Objekt seine Attribute schützen und gleichzeitig die innere Struktur verbergen. Das Objekt muß eine klar definierte Schnittstelle anbieten, die einfach und standardisiert erscheinen sollte. Damit ist es möglich die innere Struktur zu ändern, ohne andere Programmodule zu beeinflußen. Dieses Konzept wird als Kapselung bezeichnet.

Zusammenfassen kann man sagen, daß ein Objekt eine Datensammlung ist, welche einen Gegenstand repräsentiert. Dieser Gegenstand kann ein greifbarer Körper, ein Softwaremodul oder ein abstraktes Ding sein. Mit den Attributen und Methoden läßt sich dieses Objekt näher beschreiben und charakterisieren.

5.1.1.2 Klassenkonzept

Meist wird es eine Vielzahl von Objekten geben, die vom Typ her gleich sind, aber verschiedene Gegenstände ausdrücken. So wird es im Drucksystem viele Objekte „Warteschlange“ geben, die alle ein Warteschlangenverzeichnis, Aufträge in der Warteschlange und einen Namen haben. Diese Informationen müssen für jedes Objekt in ihren Attributen abgelegt werden. Um nicht jedes Objekt neu zu definieren, wird ein vollständiges Konstruktionsschema für Objekte gleichen Types entworfen. Man nennt dieses Konstruktionsschema Klasse, in der die Attribute und Methoden definiert werden.

Eine Klasse „Drucker“ etwa definiert alle Attribute, Meldungen und Aktionen eines Druckers. Um nun einen realen Drucker zu verwalten, wird ein Objekt der Klasse erzeugt. Jedes Objekt einer Klasse unterscheidet sich von den anderen durch seinen Namen und möglicherweise in den Werten der Attribute. Beachten sollte man hierbei die Beziehung zwischen dem Objekt und dem realen Gegenstand. Es ist sinnlos ein Objekt zu erzeugen, für das kein zugehöriger realer Gegenstand existiert. Andererseits ist ein realer Gegenstand ohne zugehörigem erzeugten Objekt für die Software nicht sichtbar. In einem Objekt einer Klasse sind die Werte der in der Klasse definierten Attribute für einen konkreten Gegenstand gespeichert, deshalb muß jedem Objekt Speicherplatz für seine Attribute bereitgestellt werden. Die Methoden einer Klasse sind für alle Objekte gleich und müssen nicht für jedes Objekt erzeugt werden, sie sind in der Klassendefinition festgelegt und stehen jedem Objekt zur Verfügung.

Vererbung
Der Mechanismus der Vererbung erweitert das Klassenkonzept um die Möglichkeit aus allgemeinen (abstrakten) oder bereits existierenden Klassen spezielle Klassen zu bilden. Die allgemeine Klasse wird als Oberklasse (Superclass) bezeichnet und die davon abgeleitete spezielle Klasse als Unterklasse (Subclass). Das Prinzip der Vererbung ist nur eine Vereinfachung bei der Definition von Klassen, es reduziert den Programmieraufwand. Denn in einer Unterklasse werden nur die Unterschiede zur Oberklasse, von der sie ja alle Attribute und Methoden erbt, angegeben. Möglich ist dabei

- neue Attribute und Methoden zu definieren,
- die Attribute und Methoden der Oberklasse zu überschreiben oder
- den Zugriff auf Attribute und Methoden der Oberklasse einzuschränken.

Der Mechanismus der Vererbung ist rekursiv, d.h. eine Oberklasse kann selbst wieder Unterklasse einer anderen Oberklasse sein. Somit entsteht eine Vererbungshierarchie, bei der eine Unterklasse von allen ihren Oberklassen erbt. Gewöhnlich kann oder sollte eine Klasse aber nur von einer Klasse direkt erben.

Enthaltensein-Relation (Containment)
Komplexere Objekte können auch aus mehreren Objekten zusammengesetzt sein und werden dann zusammengesetzte Objekte genannt. Ein Objekt „Auto“ beispielsweise setzt sich aus den Objekten Reifen, Motor, Karosserie, Sitz, Steuerrad, Getriebe, usw. zusammen. Die Enthaltensein-Relation wird dabei durch einen Zeiger auf ein Objekt dargestellt. Der Zeiger ist ein Attribut im zusammengesetzten Objekt, der auf das enthaltene Objekt zeigt. Natürlich kann ein zusammengesetztes Objekt in einem weiteren Objekt enthalten sein, wodurch sich komplexe Strukturen abbilden lassen. Nicht erlaubt sind natürlich Verbindungen, die einen Ring bilden, außerdem darf jedes Objekt nur in einem anderen Objekt enthalten sein. Somit entsteht bei zusammengesetzten Objekten eine baumartige Struktur.

Der Unterschied zwischen Vererbung und der Enthaltensein-Relation sollte jetzt geklärt sein. Die Vererbung wird auf Klassen angewandt und dient zur Minimierung des Programmieraufwandes bei der Definition vieler verschiedener Objekte, wobei letztendlich eine Vererbungshierarchie zwischen Klassen entsteht. Zwischen den Objekten existiert die Enthaltensein–Relation, damit lassen sich komplexe Datenstrukturen konstruieren.

5.1.2 Strukturierungsmöglichkeiten in SNMP

Die Informationen im Internetmanagement werden in einer MIB strukturiert. Dazu wird jedem Objekt ein Name, eine Syntax und eine Kodierung zugewiesen. Der Name bezeichnet die Stelle in der MIB, wo der Wert der Variablen zu finden ist. Die Syntax des Objektidentifiers definiert die abstrakte Datenstruktur des Objekts und die Kodierung legt die Wertdarstellung der Information zur Übertragung über das Netz fest.

5.1.2.1 Namen

Ein SNMP–Objekt kann nur aus einem Attribut bestehen. Dieses wird mit Hilfe des Names in der MIB angeordnet. Da die MIB eine Baumstruktur besitzt, kann man auch Gruppen definieren. Dabei umfaßt eine Gruppe alle direkt unter einem Ast definierten Variablen.

5.1.2.2 Syntax

Um die Attribute des Objektmodells im Internet darzustellen, stehen uns Grundtypen, konstruierende Typen und definierte Typen zur Verfügung.

Es gibt die Grundtypen NULL, OCTET STRING, OBJECT IDENTIFIER und INTEGER. Auf diesen Typen bauen alle anderen Typen auf. Der Typ INTEGER kann noch zum aufzählenden INTEGER erweitert werden.

Die konstruierenden Typen dienen zur Erstellung von Listen und Tabellen, zur Verfügung stehen dabei SEQUENCE und SEQUENCE OF. Es ist zur Zeit die einzige Form der Datenstrukturierung: eine einfache zweidimensionale Tabelle mit einfachen Werteinträgen. Dazu existiert auch noch der INDEXPART, mit Hilfe dessen eine oder mehrere Spalten der Tabelle als eindeutige Zeileneintragbezeichner ausgewiesen sind. Ferner kann für jeden Eintrag in der Tabelle ein Defaultwert angegeben werden, eingeleitet durch das Schlüsselwort DEFVAL.

Zusätzlich können aus den Grundtypen neue Typen definiert werden, um den Werteintrag genauer zu spezifizieren. So existieren bereits die definierten Typen IPADDRESS, NETWORKADDRESS, COUNTER, GAUGE, TIMETICKS und OPAQUE.

5.1.2.3 Resümee

Zusammenfassend muß man allerdings erkennen, daß die Strukturierungsmöglichkeiten in SNMP sehr beschränkt sind. Komplexere Attribute darzustellen bereit bereits erhebliche Schwierigkeiten und ist auf direktem Weg nicht möglich. Relationen zwischen den Attributen oder Objekten abzubilden ist in SNMP nicht vorgesehen.

5.1.3 Konventionen zur Abbildung des Objektmodells in SNMP

Nachdem nun die Konzepte und Relationen des Objektmodells und die Strukturierungsmöglichkeiten in SNMP bekannt sind, kann damit begonnen werden eine Abbildungsvorschrift aufzustellen.

5.1.3.1 Klassenkonzept

In [RFC 1212] wird empfohlen für jedes mehrfach auftretende Objekt eine Tabelle zu definieren. Da Klassen gewöhnlich mehrere Objekte haben, werden sie auf Tabellen abgebildet. Die Tabellenabbildung bietet noch weitere Vorteile:

- jedem Wert eines neuen Objekts kann ein Defaultwert zugewiesen werden
- es ist nicht nötig Spezialfälle, wie Objekte die nur einmal erscheinen, zu berücksichtigen

Allerdings müssen komplexere Attribute noch genauer betrachtet werden, denn diese lassen sich nicht durch einen einfachen Typ in der Tabelle darstellen.

Deshalb wird jede Klasse des Objektmodells auf eine Objektgruppe in der Internet MIB abgebildet und im folgenden Klassengruppe genannt. In dieser Klassengruppe gibt es mindestens eine Internet MIB Tabelle, sind mehrere Tabellen in der Klassengruppe stehen sie miteinander in Beziehung.

- Die Klasse wird als Ganzes in die Klassentabelle abgebildet. Die Spalten der Klassentabelle enthalten die Attribute der Objektklasse. Daneben gibt es noch Spalten, in denen ein Verweis auf komplexere Attribute (Nebentabelle) abgelegt wird und welche für die Wiedergabe der Beziehungen zwischen den Objekten.
- Ferner keine oder mehrere Nebentabellen, die die komplexen Attributwerte enthalten oder Beziehungen beschreiben.

In der Klassentabelle wird für jedes Objekt einer Klasse eine Zeile angelegt, so ein einzelner Eintrag nennt sich Klasseneintrag. Zum Klasseneintrag gehören auch Einträg in den Nebentabellen, um die komplexen Attribute des Objekts abzubilden. Somit ist ein Objekt einer Klasse in der Internet MIB ein Eintrag in die Klassentabelle und den dazugehörigen Nebentabellen. Um einen einzelnen Klasseneintrag in der SNMP Objektgruppe auszuwählen, dient ein Klasseninstanzzeiger, der durch eine OID angegeben wird.

Auf diese Weise werden alle Klassen des Objektmodells in Klassengruppen abgebildet. Wenn die Klassentabelle erstellt wird, wird sie an den Ast 1 unterhalb der Klassengruppe angehängt. An den folgenden Ästen, beginnend bei 2 unterhalb der Klassengruppe, werden alle Nebentabellen angeordnet, so wie es in Abbildung 5.1 gezeigt ist. Die Variable x steht dabei für den OID, unter welchem die Klassengruppe in der Internet MIB registriert ist.

Das Konzept der Nebentabelle läßt sich rekursiv fortsetzen, d.h. eine Nebentabelle kann weiter Nebentabellen haben usf. Diese Nebentabellen von Nebentabellen werden aber in der Internet MIB nicht tiefer verschachtelt, auch sie hängen direkt an einem Ast unterhalb der Klassengruppe.

Beachte: In der Klassengruppe sind keine einfachen Variablen enthalten, denn alle Attribute einer Klasse sind in der Klassentabelle oder in einer Nebentabelle eingetragen.

Struktur der Klassentabelle
In der Klassentabelle sind folgende Spalten und zugehörige Äste unterhalb der Klassentabelle zu reservieren.

Ast 0 darf nicht verwendet werden nach Definition der Structure of Management Information.

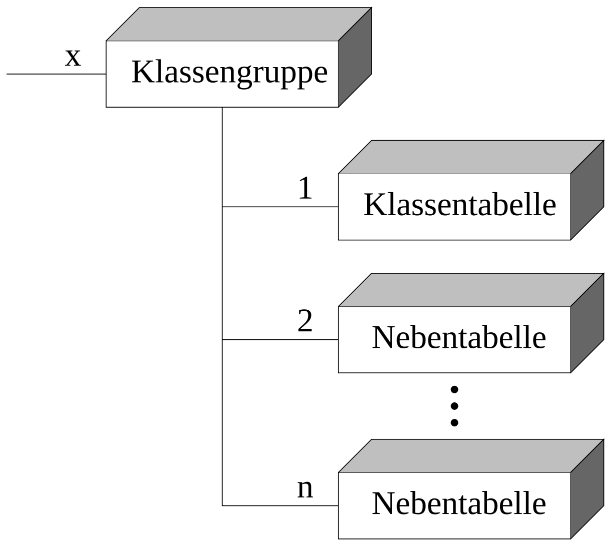

Abbildung 5.1: Klasse in der Internet MIB

Ast 1 beinhaltet den Klasseneintragindex (Tabellenindex). Er ist ein eindeutiger Index für einen Eintrag in der Tabelle, gewöhnlich ein Integerwert. Natürlich könnte man auch eine bereits vorhandene Spalte der Tabelle als Indexelement auszeichnen, allerdings muß das Element eindeutig und sollte nicht zu lang sein. Besonders die Anforderung der Eindeutigkeit erfordert eine gründliche Überprüfung der Spaltenauswahl, was durch eine extra erzeugte Indexspalte umgangen wird.

Der bereits oben erwähnte Klasseninstanzzeiger zum Verweis auf eine Instanz einer Klasse setzt sich somit aus der OID des Klasseneintrags, dem Wert „1" (Tabellenindexspalte) und dem Wert des Tabellenindexes für die betroffene Instanz zusammen.

Ast 2 bis n+m ist von den n-1 von einer Oberklasse geerbten Attributen der Klasse zu besetzen, gefolgt von den m Attributen der Klasse selbst. D.h. alle Attribute der umzuwandelnden Klasse werden zu Spalten in der Klassentabelle.

Ast n+m+1 kann für einen Zeiger auf eine Nebentabelle verwendet werden, in der Verweise auf alle zur Zeit in diesem Objekt befindlichen Aufträge enthalten sind (Auftrag–Enthaltensein–Relation).

Ast n+m+2 wird für einen Zeiger auf eine Nebentabelle verwendet, in der alle möglichen Objekte aufgeführt sind, an die der Auftrag weitergeleitet werden kann (Ziele–Relation).

Ast n+m+3 ist wieder ein Zeiger auf eine Nebentabelle, in der ein Verweis auf alle Objekt existiert, die in diesem Objekt enthalten sind (Bedieneinheiten–Enthaltensein–Relation).

Ast n+m+4 ist nochmals ein Zeiger auf eine Nebentabelle, in der ein Verweis auf ein Objekt existiert, die dieses Objekt enthält. Man benötigt diese Nebentabelle um die Eltern–Relation von Objekten auszudrücken. Der Zeiger in der Nebentabelle zeigt auf jenes Objekt, welches dieses Objekt beinhaltet und darf deshalb für die Managementstation nur lesbar sein (Eltern–Relation).

Ast n+m+5 ist nötig, sollte es der Managementstation gestattet sein, Objekte zu erzeugen oder zu löschen. Dazu bedient sie sich der Spalte „Zeilenstatus", welche deshalb les- und schreibbar sein muß.

Selbstverständlich müssen nicht alle Äste in einer realen Klassentabelle vorhanden sein. Nur Äste die wirklich benötigt werden, müssen auch vorhanden sein. Die wirklich vorhandenen Äste werden dann mit 1 beginnend fortlaufend durchnummeriert, ohne eine Nummer zu überspringen.

Struktur der Nebentabelle für komplexe Attribute
Komplexere Attribute der Klasse müssen aus mehreren einfachen Variablen zusammengesetzt werden, welche dann in einer eigenen Tabelle, der Nebentabelle, zu speichern sind. Dazu kommen noch einige Spalten zur Verwaltung.

Ast 0 darf nicht verwendet werden nach Definition der Structure of Management Information.

Ast 1 ist nötig, wenn mehrere Einträge pro Eintrag in der zugehörigen Elterntabelle erlaubt sind. Diese Elterntabelle ist entweder die Klassentabelle oder eine andere Nebentabelle. In diesem Ast sollte dann ein Tabellenindex für diese Nebentabelle stehen, der zusammen mit dem Index der Elterntabelle einen Eintrag eindeutig auswählt.

Ast 2 bis k dient für die einfachen Variablen, die zusammen das komplexe Attribut der Klasse darstellen.

Ast k+1 und k+2 sind nötig bei einer rekursiven Struktur des umzuwandelnden Attributes. Wie es der Fall ist, wenn sich das Attribut direkt oder indirekt selbst enthält. Im Ast k+1 ist dann ein Rekursionszeiger abgelegt, der auf das rekursiv verschachtelte Attribut zeigt und im Ast k+2 ein Flag, das anzeigt, ob auf diesen Eintrag ein Rekursionszeiger zeigt. Wenn der Managementstation erlaubt ist die Struktur zu verändern, ist der Wert des Rekursionszeigers les- und schreibbar, sonst sind beide nur lesbar.

Ast k+3 ist nötig, sollte es der Managementstation gestattet sein, Objekte zu erzeugen oder zu löschen. Dazu bedient sie sich der Spalte „Zeilenstatus", welche deshalb les- und schreibbar sein muß.

Index der Tabelle
Wichtig ist dabei, daß jede Klassen- oder Nebentabelle mindestens einen Tabellenindex hat, der meist ein nichtzugreifbarer Wert vom Typ Integer ist. Seine einzige Aufgabe ist die eindeutige Identifizierung eines Tabelleneintrags. Während die Klassentabelle genau einen Tabellenindex benötigt, wird bei Nebentabellen der Tabellenindex der Klassentabelle verwendet. Sind für einen Eintrag in der Klassentabelle allerdings mehrere Einträge in der Nebentabelle möglich, so muß auch in der Nebentabelle ein eigener Tabellenindex angelegt werden. In Kombination mit dem Tabellenindex der Klassentabelle entsteht damit ein eindeutiger Verweis auf einen Eintrag der Nebentabelle. Das Prinzip der Nebentabelle kann sich rekursiv fortsetzen, womit sich der Schlüssel für einen Eintrag um weitere Tabellenindexe verlängern kann.

5.1.3.2 Enthaltensein–Relation und Eltern–Relation

Mit Hilfe des Klasseninstanzzeigers können alle Relationen innerhalb des Objektmodells in der Internet MIB dargestellt werden, dies beinhaltet auch die Enthaltensein–Relation. Um dieses Problem zu lösen, gibt es mehrere Ansätze:

1. Ein Zeiger in der Klassentabelle verweist auf eine Nebentabelle. Durch den Index der Klassentabelle ist sichergestellt, daß Einträge in der Klassentabelle ihre zugehörigen Einträge in der Nebentabelle finden. Somit können die Einträge in der Nebentabelle als in Einträgen der Klassentabelle enthalten sein angesehen werden.

2. Denkbar wäre auch eine neue Nebentabelle, in der alle Kindobjekte eines Objektes durch einen Klasseninstanzzeiger angegeben werden.

3. Auch eine neue Objektgruppe, in der die Enthaltensein–Relationen aller Objekte eingetragen werden, ist möglich.

4. Ein weiterer Ansatz ist ein Zeiger auf eine andere Objektgruppe, in der die enthaltenen Objekte abgelegt sind. Damit kann auf beliebige Objekte in der MIB verwiesen werden, auch auf Objekte die außerhalb des Objektmodells liegen.

Da in unserem Objektmodell nur zwei wirkliche Enthaltensein–Relationen vorliegt, nämlich zwischen der Bedienstation und ihren Bedieneinheiten bzw. Aufträgen, würde sich bereits der erste Ansatz eignen. Leider wird dadurch die Klasse „Bedieneinheit“ nicht in einer eigenen Klassengruppe dargestellt, sondern als Nebentabelle in der Klassengruppe des Elternobjekts.

Für die Relation zwischen der Bedienstation und ihren Bedieneinheiten eine eigene Objektgruppe anzulegen, in der diese Relationen eingetragen sind, ist zu aufwendig. Wären im Objektmodell mehrere Enthaltensein–Relationen enthalten, die möglicherweise mehrstufig sind und auch mehrere verschiedene Objekte betreffen, wäre die Objektgruppe sicher eine elegante Lösung.

Nur einen Zeiger auf eine Objektgruppe anzugeben, in der die enthaltenen Objekte angegeben sind, ist zwar sehr schön, es müssen aber unter Umständen sehr viele Objektgruppen angelegt werden. Jedes Objekt das Kindobjekte haben kann, braucht eine eigene Objektgruppe. Diese Objektgruppen sind in ihrer Struktur jedoch gleich und sollten besser zu einer Objektgruppe vereint werden. Außerdem können Objektgruppen nicht dynamisch erzeugt werden, was heißt die Anzahl der Elternobjekte ist fest.
Vereint man alle Kindobjektgruppen gleicher Struktur zu einer Objektgruppe, existieren in der Objektgruppe für ein Elternobjekt neben seinen Kindobjekten noch Kindobjekte anderer Elternobjekt. Das erfordert die Untersuchung jedes Eintrags in der Objektgruppe, ob dieses auch das zugehörige Kindobjekt ist.
Sind nur wenige Objekte, deren Anzahl feststeht und die Objekte enthalten können, vorhanden oder gibt es nur wenige Objekte, die in anderen Objekten enthalten sein können, sollte man sich diesen Lösungsansatz anschauen.

Übrig bleibt noch die Möglichkeit in einer Klassengruppe, deren Einträge andere Objekte enthalten können, eine weitere Nebentabelle einzuführen, in der die Klasseninstanzzeiger auf die Kindobjekte abgelegt sind. Um in dieser Nebentabelle die zugehörigen Einträge eines Klassentabelleneintrags zu finden, bedienen wir uns wieder dem Index der Klassentabelle. Somit ist ein zügiger Zugriff auf das enthaltene Objekt gewährleistet, alle Klassen des Objektmodells können nach der Abbildungsvorschrift für Klassen konvertiert werden, die Anzahl ist nicht fix und die Information über die Kindobjekte ist direkt in der Klassengruppe des Elternobjekts enthalten.

Das Gegenstück der Enthaltensein–Relation ist die Eltern–Relation. Von jedem Objekt möchte man natürlich auch wissen, in welchem Objekt es enthalten ist. Das Objekt kann nur in einem anderen Objekt enthalten sein, weshalb ein Zeiger auf das andere Objekt in der Klassentabelle reichen würde. Es spricht aber nichts dagegen, die gleiche Methode wie bei der Enthaltensein–Relation zu verwenden. Dadurch wird die Abbildungsvorschrift homogener.

5.1.3.3 Angabe des Auftragsdurchlaufs

Auch die Reihenfolge der Komponenten des Objektmodells, die ein Auftrag durchläuft, wird durch eine Relation angegeben. Sie wird im folgenden Ziele–Relation genannt. Für jede Komponente im Objektmodell muß angegeben werden, wohin es den Auftrag weiterleiten soll. Gewöhnlich gibt es nur ein Ziel, weshalb eine Spalte in der Klassentabelle für einen Klasseninstanzzeiger ausreichen würde. Aber manche Komponenten haben mehrere Ziele zur Auswahl, abhängig vom Auftrag, dem Betriebszustand der Funktionseinheit oder sonst was. In diesem Fall müssen mehrere Klasseninstanzzeiger angegeben werden, was sich am Besten mit der gleichen Methode wie bei der Enthaltensein–Relation realisieren läßt.

5.1.3.4 Aufträge in den Komponenten der Funktionseinheit

Um Aufträge in der Funktionseinheit zu lokalisieren, wird angegeben in welchen Komponenten sie sich befinden. Allerdings wird auch bei den Komponenten angegeben, welche Aufträge sie gerade bearbeiten. Dabei muß man unterscheiden, ob die Komponente nur einen Auftrag gleichzeitig bearbeiten kann, wie die Bedieneinheit „Drucker“, oder ob auch mehrere Aufträge bearbeitet werden können, so wie bei der Warteschlange. Kann nur ein Auftrag angenommen werden, würde bereits eine Spalte in der Klassentabelle reichen, in die ein Klasseninstanzzeiger auf den Auftrag abgelegt wird. Bei mehreren Aufträgen in einer Komponente muß für jeden Auftrag ein Klasseninstanzzeiger gespeichert werden, womit wir wieder bei der Methode für die Enthaltensein–Relation angelangt sind. Damit nun nicht für jede Komponente entschieden werden muß, welche Methode zu verwenden ist und um ein homogens Modell zu erhalten wird die gleiche Methode wie bei der Enthaltensein–Relation verwendet. Kann nur ein Auftrag angenommen werden, ist diese Methode zwar etwas aufwendiger, es gehen aber keine Informationen verloren und der Zugriff auf diese Informationen ist bei allen Komponenten identisch.

5.1.3.5 Struktur der Nebentabelle für eine Relation

Zur Darstellung aller Relationen wird das gleiche Prinzip angewandt. In der Klassentabelle existiert ein Zeiger auf eine Nebentabelle, in der die in Beziehung stehenden Objekte durch einen Klasseninstanzzeiger aufgeführt sind.

Ast 0 darf nicht verwendet werden nach Definition der Structure of Management Information.

Ast 1 wird für den Tabellenindex benötigt. Denn für einen Eintrag in der Klassentabelle können in dieser Tabelle mehrere zugehörige Einträge existieren, weshalb der Index der Klassentabelle alleine nicht mehr reicht. Gibt es nur einen Eintrag, so wie bei einer Bedieneinheit, die nur einen Auftrag enthalten kann, wird kein zusätzlicher Index benötigt und kann entfallen.

Ast 2 dient zur Aufnahme des Klasseninstanzzeigers. Dieser Zeiger ist ein Verweis auf das in Relation stehende Objekt.

5.1.3.6 Vererbung

Wie bereits oben geschildert wurde, ist das Prinzip der Vererbung eine Vereinfachung des Programmieraufwandes, denn eine Klasse kann die Attribute und Methoden einer Oberklasse erben. Wenn wir nun eine Klasse in die Internet MIB abbilden, müssen wir auch die geerbten Attribute berücksichtigen. Um keine Attribute zu übersehen und um die Attribute einigermaßen zu struktuieren, werden als erstes die Attribute der Oberklasse eingetragen.

Erbt die Klasse von mehreren Oberklassen wird in der Reihenfolge vorgegangen, wie die Reihenfolge der Oberklassen angegeben ist. Hat eine Oberklasse selbst wieder Oberklassen, werden als erstes deren Attribute berücksichtigt. Als letzte werden die Attribute der Klasse in die Tabelle eingetragen, womit man schließlich die in der Abbildung 5.2 gezeigte Struktur erhält. Bei der Abbildung einer konkreten Klasse müssen nicht alle in der Abbildung 5.2 gezeigten Strukturelemente auftreten.

5.1.3.7 Typkonventionen

Die Typen der einfachen Attribute der Objekte werden in der Internet MIB auf die primitiven Variablentypen abgebildet, wie INTEGER, OBJECT IDENTIFIER, NULL oder OCTET STRING. Daneben gibt es noch definierte Typen, wie COUNTER, GAUGE, IPADDRESS, TIMETICKS, usw. Um jedoch kenntlich zu machen, das der Wert in der Spalte auf eine Nebentabelle oder Klasseninstanz verweist, müssen noch Textkonventionen (textual conventions) vereinbart werden. Damit läßt sich auch angeben, für was die Nebentabelle bzw. der Klasseninstanzzeiger benützt wird, ob zur Darstellung komplexer Attribute, einer bestimmten Relation oder dem Eltenobjekt. Festhalten läßt sich dabei, daß alle Textkonventionen vom Typ OBJECT IDENTIFIER sind. Der Tabelle 5.1 kann man entnehmen welche Konventionen für welchen Zweck definiert wurden.

5.1.4 Anordung der Klassen des Objektmodells in der Internet MIB

Bisher haben wir uns damit beschäfigt die einzelnen Klassen des Objektmodells in die Internet MIB abzubilden. Jedoch wissen wir bis jetzt nicht, wie diese einzelnen Klassen im Registrierungsbaum einzuordnen sind. Noch haben wir uns überlegt, wie die verschiedenen Typen von Bedienstationen, Bedieneinheiten, Warteschlangen und Aufträge sinnvoll gruppiert werden können.

Die Funktionseinheit nimmt eine besondere Stellung im Objektmodell ein. Sie ist keine eigentliche Komponente eines konkreten Servers, vielmehr dient sie zur Beschreibung des Servers als Ganzes. Die Funktionseinheit umfaßt alle Komponenten des Objektmodells, was auch in der MIB ausgedrückt werden soll. Im Objektmodell wird dies durch eine Enthaltenseinrelation gezeigt, da jedoch alle Komponenten enthalten sind, ist es nicht nötig diese einzeln aufzuzählen. Man definiert für den Server ein Gruppe in der alle Komponenten enthalten sind. Um diese Gruppe nicht mit extrem vielen Variablen zu füllen, wird sie weiter untergliedert.

Ast 1 ist eine Gruppe mit den Attributen der Funktionseinheit. Da es von der Klasse „Funktionseinheit“ genau eine Instanz gibt, ist es nicht nötig dafür eine Klassentabelle anzulegen. In der Gruppe der Funktionseinheit gibt es einfache Variablen aber auch Tabellen für komplexere Attribute und Relationen.

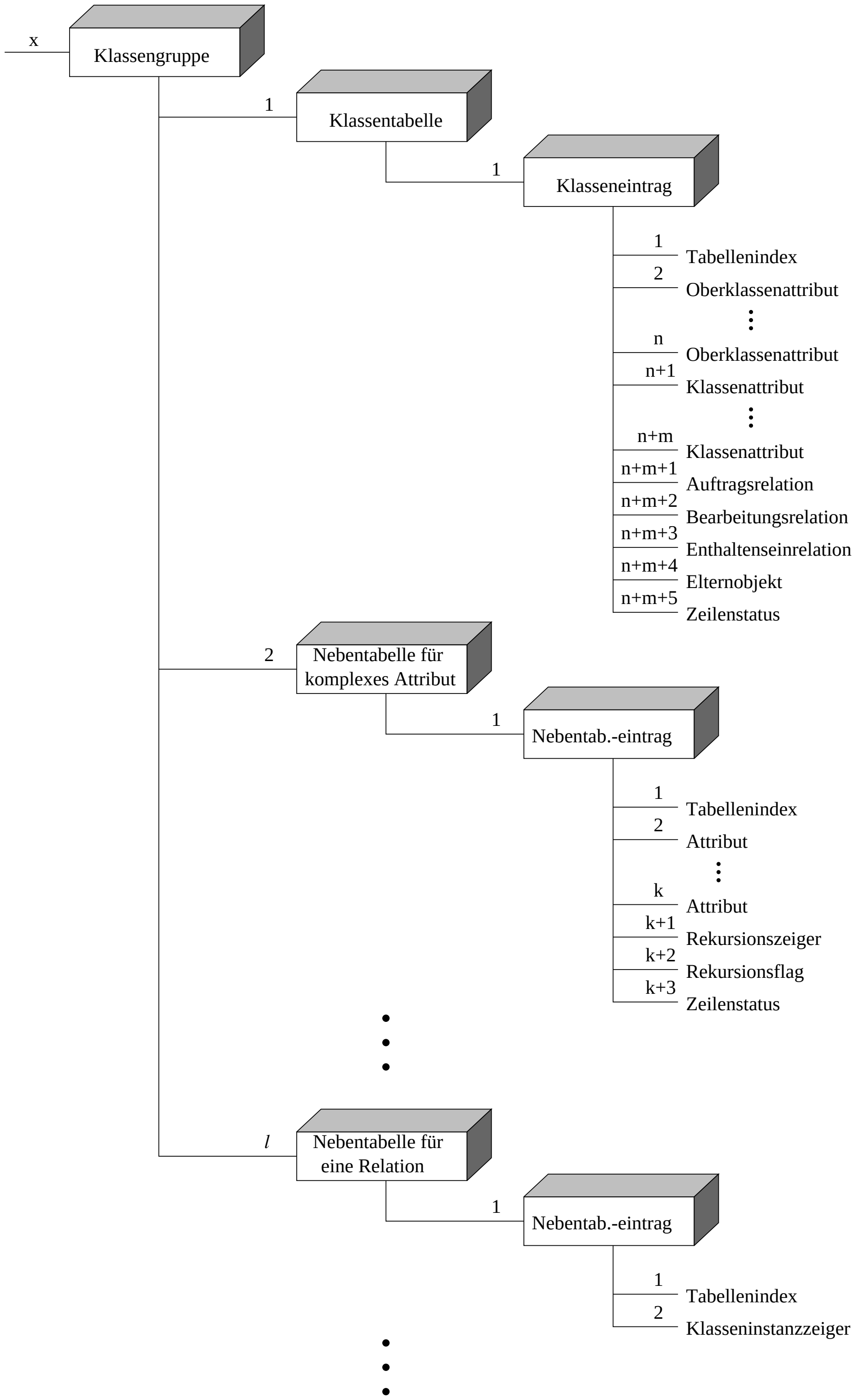

Abbildung 5.2: Struktur der Klasse in der Internet MIB

Textkonvention	Beschreibung
TableIndex	Er stellt einen kompakten und einfachen (Sub)Index für eine Tabelle bereit.
Jobs	Darin ist ein Zeiger auf eine Nebentabelle mit den zur Zeit in Bearbeitung befindlichen Aufträgen.
Destination	Darin ist ein Zeiger auf eine Nebentabelle in der die möglichen nächsten Objekte, an die der Auftrag geschickt werden soll stehen.
Containment	Deutet auf einen Zeiger auf eine Nebentabelle, in der alle enthaltenen Objekte aufgeführt sind.
Parent	Deutet auf einen Zeiger auf eine Nebentabelle, in der das Objekt aufgeführt ist, das dieses enthält.
Complex	Verweist auf eine Nebentabelle, in der mehrere Variablen zur Darstellung eines komplexen Attributes abgelegt sind. In der Nebentabelle gibt es genau einen solchen Eintrag.
MultiComplex	Verweist auf eine Nebentabelle, in der mehrere Variablen zur Darstellung eines komplexen Attributes abgelegt sind. Dabei können in der Nebentabelle beliebig viele zugehörige Einträge stehen.
Parent	Verweist auf ein Objekt, das dieses Objekt enthält.
Recursion	Verweist auf einen Eintrag in der gleichen Tabelle.

Tabelle 5.1: Textkonventionen

Ast 2 ist die Gruppe, in der alle Bedienstationen aufgeführt sind. Für jeden Typ einer Bedienstation wird eine eigene Untergruppe in dieser Gruppe angelegt. Die Untergruppe entspricht dabei der oben beschriebenen Klassengruppe. Gewöhnlich hat ein Server mehrere Bedienstationtypen, weshalb hier auch mehrere Untergruppen anzutreffen sind.

Ast 3 ist die Gruppe der Bedieneinheiten. Darin sind wieder Untergruppen mit den verschiedenen Bedieneinheittypen zu finden. Bei realen Servern wird es selten vorkommen, daß die Bedieneinheiten einzeln administriert werden, weshalb in der Gruppe häufig keine Klassengruppe definiert sein wird.

Ast 4 ist die Gruppe der Warteschlangen. Das unterschiedliche Typen von Warteschlangen in einem Basisdienst realisiert sind, ist wohl sehr selten. Da es jedoch nicht ausgeschlossen werden kann und um alle Komponenten des Objektmodells gleich zu behandeln, wird auch hier die Klassengruppe nicht direkt unter der Servergruppe eingehängt, sondern wird eine Gruppe „Warteschlangen“ angelegt, unter der die Klassengruppe der Warteschlange hängt.

Ast 5 ist die Gruppe der Aufträge. Hier kann es wieder verschiedene Auftragsklassen geben, weshalb die verschiedenen Klassengruppen unterhalb dieser Gruppe eingehängt werden.

Somit erhält man eine Servergruppe mit fünf Untergruppen. Eine Untergruppe enthält die Attribute der Funktionseinheit, weiter gibt es die vier Untergruppen Bedienstationen, Bedieneinheiten, Warteschlangen und Aufträge, die jeweils ihre entsprechenden Klassengruppen enthalten, wie es in Abbildung 5.3 abgebildet ist.

5.2 Die MIB–Struktur für den Druckdienst

Bei der Abbildung des druckspezifischen Objektmodells werden alle Komponenten des Objektmodells zu Klassengruppen. Ausgenommen davon bleiben die Funktionseinheit, die gesondert behandelt wird, und alle Bedieneinheiten, die für das Management nicht relevant sind. Beim Druckdienst ist nur die Bedieneinheit „Drucker“ von Interesse und wird berücksichtigt. Die so erhaltenen Klassengruppen werden dann unter den entsprechenden Hauptgruppen eingehängt, wie es in Abbildung 5.3 vorgegeben ist.

Anschließend muß für alle Klassengruppen entschieden werden, welche Nebentabellen sie neben der Klassentabelle enthalten sollen. In Abbildung 5.4 sind diese bereits eingezeichnet, man sieht die Baumstruktur mit den Haupt- und Klassengruppen, sowie die Tabellen in den Klassengruppen. Auch die Nummerierung der Äste ist bereits angegeben.

Einfache Attribute des Objektmodells, welche durch einfache Variablen dargestellt werden können, dessen Variablentyp bereits in der Internet MIB verfügbar ist, können in die

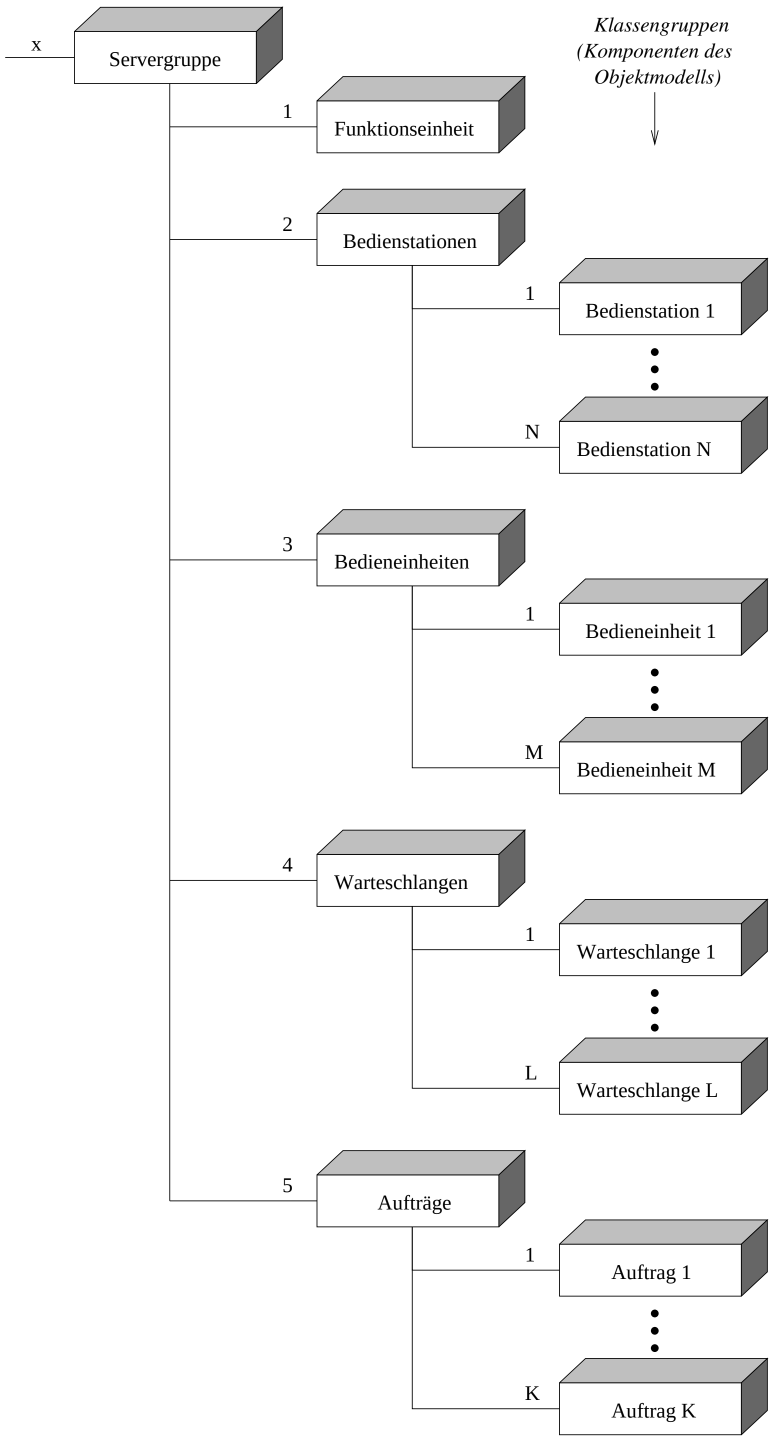

Abbildung 5.3: Die Klassen des Objektmodells im MIB-Baum

Klassentabelle des entsprechenden Objekts aufgenommen werden. Da im Objektmodell glücklicherweise keine komplexen Attribute vorkommen, benötigen wir auch keine Nebentabelle um solche darzustellen. Somit können alle Attribute, die bei den Managementanforderungen an das Drucksystem definiert wurden, in die entsprechenden Klassentabellen übertragen werden.

Zur Darstellung der Beziehungen zwischen den Objekten sind Nebentabellen vonnöten. Jede Nebentabelle, die in einer Klassengruppe eingefügt wird, benötigt auch ein zugehöriges Attribut in der Klassentabelle, das auf die Nebentabelle verweist.

Als nächstes müssen wir uns überlegen, welche Komponenten des Objektmodells Aufträge enthalten können. Dies sind natürlich die Bedienstationen, die Bedieneinheit „Drucker“ und die Warteschlange. Jede dieser Klassengruppen erhält deshalb eine Nebentabelle „Enthältaufträgetabelle“.

Gleiches gilt für die Angabe des Auftragsdurchlaufs. Jede Komponente muß angeben, wohin sie die Aufträge weiterleitet. Davon sind wieder die Bedienstationen und die Warteschlange betroffen. Auch hierfür erhält jede Klassengruppe eine Nebentabelle „Zieletabelle“, in der diese Information abgelegt wird.

Die Bedieneinheit „Drucker“ und die Auftäge sind immer in einer anderen Komponente des Objektmodells enthalten. Welche dies sind, muß wieder in je einer Nebentabelle „Enthaltenseintabelle“ in ihrer Klassengruppe vermerkt werden.

Bei den Komponenten ist bereits angegeben, welche Aufträge sie enthalten. Welche Bedieneinheiten die Bedienstation „Drucker“ enthält, fehlt allerdings noch. Dies wird in der Nebentabelle „Enthältbedieneinheitentabelle“ nachgeholt.

Schließlich kommen wir noch zur Funktionseinheit, die ja eine Sonderstellung einnimmt. Sie ist die einzige Komponente des Objektmodells, die keine Klassentabelle benötigt. Ihre Attribute werden einfach unter der Hauptgruppe „Funktionseinheit“ eingehängt. Um allerdings anzugeben, bei welchen Komponenten des Objektmodells Aufträge angenommen werden, wird eine Tabelle benötigt, die der Zieletabelle entspricht.

5.3 Die MIB–Struktur für den Nachrichtendienst

Aus den gleichen Überlegungen, wie sie bereits beim Druckdienst angestellt wurden, ergibt sich die MIB–Struktur für den Nachrichtendienst, wie sie in Abbildung 5.5 zu sehen ist.

Dabei fällt allerdings auf, daß keine Hauptgruppe „Bedieneinheiten“ existiert, was auch nicht nötig ist, weil keine Informationen einer Bedieneinheit dargestellt werden müssen.

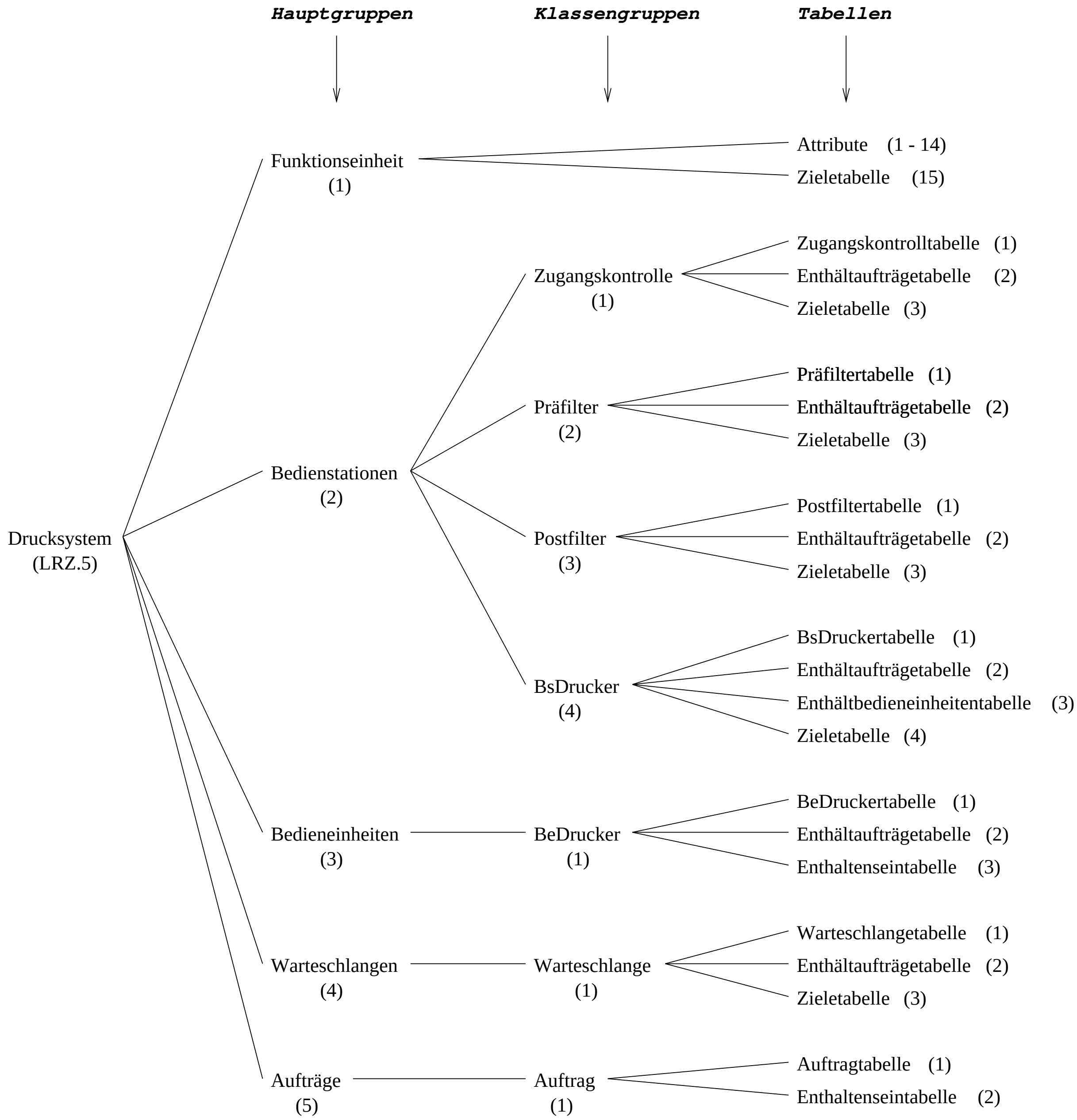

Abbildung 5.4: Die Internet MIB Struktur für den Druckdienst

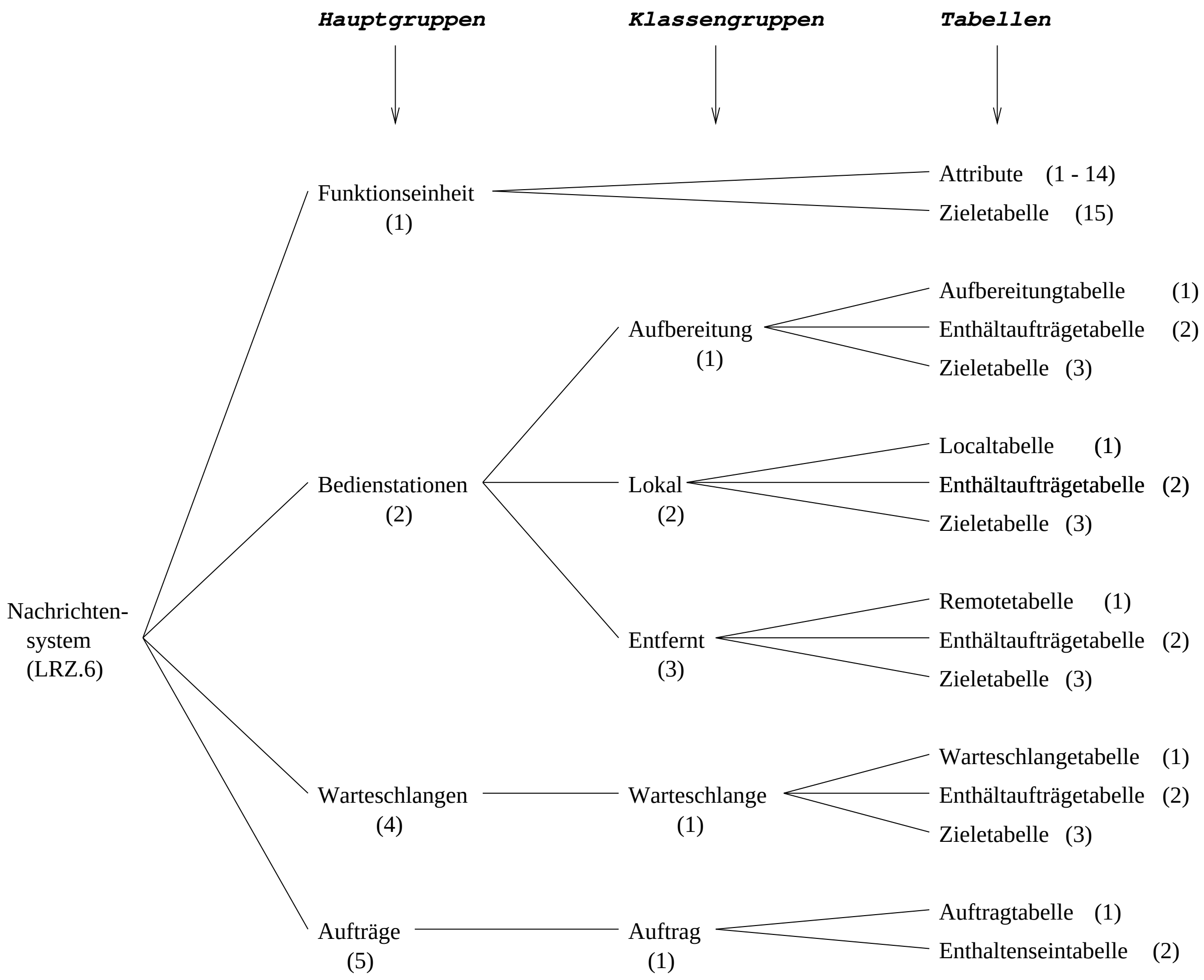

Abbildung 5.5: Die Internet MIB Struktur für den Nachrichtendienst

Kapitel 6

Beschreibung des Subagenten

Die Überwachung von Endsystemen geschieht in TCP/IP–basierten Netzen gewöhnlich durch Agenten, die auf den Endsystemen laufen. Diese Agenten sammeln die Managementinformationen und können Aktionen anstoßen. Die Kommunikation zwischen dem Agenten und der Managementstation läuft dabei über ein SNMP–Protokoll ab.

Hier wurde der CMU–Agent gewählt, der sowohl SNMPv1- als auch SNMPv2–sprachig ist. Außerdem bietet er die Möglichkeit, die MIB auf mehrere Prozesse aufzuteilen, wodurch die MIB des Drucksystems als ein eigener Prozeß realisiert werden kann. Ermöglicht wird dies durch die DPI–Schnittstelle des CMU–Agenten. Über diese Schnittstelle kann der Agent Anfragen nach bestimmten Variablen an seine Subagenten weiterleiten.

Jetzt könnte man natürlich den Druckdämon um eine DPI–Schnittstelle erweitern und auch die MIB im Druckdämon implementieren. Dadurch erspart man sich einen weiteren Dämon und den Informationsaustausch zwischen einem separaten Subagenten und dem Druckdämon. Jedoch ist bei einem Absturz des Druckdämons dann auch die Administrationsmöglichkeit beendet.

Das DPI–Protokoll wird in [RFC 1592] spezifiziert und in [DPI API] werden bereits Funktionen dafür bereitgestellt, die die Schnittstelle realisieren. Die Implementierung und Funktionalität der DPI–Schnittstelle im CMU–Agenten läßt sich in [Hain 95] nachlesen. Ferner gibt es einen Subagenten [HaHa 94] für Systemmanagementaufgaben, der für die Zusammenarbeit mit dem CMU–Agenten erprobt ist.

Viele Funktionen des Subagenten, besonders die zur Implementierung der DPI–Schnittstelle, stammen von [DPI API]. Deshalb werden diese Funktionen nicht nochmals beschrieben, da sie bereits dort genau erläutert sind. Der Aufbau des Subagenten und weitere Funktionen wurden vom Subagenten [HaHa 94] entnommen, weshalb auch diese Informationen besser dort nachzulesen sind.

6.1 Konstanten und Datenstrukturen

In den Dateien *snmp_dpi.h*, *dpi_printer.h* und *variablen.c* sind die wichtigsten Konstanten und Datenstrukturen zu finden. Die Datei *snmp_dpi.h* stammt vom Beispielsubagenten von IBM und enthält alle nötigen Informationen zur Implementierung der DPI–Schnittstelle. Sucht man die Konstanten, welche speziell für diesen Subagenten definiert wurden, und die Strukturen, in denen die Werte der Variablen der MIB gespeichert sind, wird man in der Datei *dpi_printer.c* fündig. Welche Variablen in der MIB enthalten sind, ist in dem Feld „Variablendaten“ in der Datei *variablen.c* angegeben.

Die in der Datei *dpi_printer.c* angegebenen Konstanten definieren den Namen des Subagenten, das Object Identifier Präfix aller enthaltenen Variablen sowie einige Konstanten, die die maximale Länge bzw. Anzahl von OIDs, Indexes, Namen, Warteschlangen, Druckern, usw. angeben. Weiter gibt es Konstanten, die den Wert einiger Attribute der MIB definieren. Für jede Tabelle der MIB wurde außerdem eine Struktur definiert, die einen kompletten Eintrag dieser Tabelle aufnehmen kann. Ein kompletter Eintrag entspricht einer Zeile der Tabelle. Da eine Tabelle natürlich mehrere Einträge haben kann, wird für jede Klassentabelle und Nebentabelle ein Feld bereitgestellt, daß eine fest vorgegeben Anzahl von solchen Einträgen aufnehmen kann. Jede Tabellenzeile benötigt einen Index, mit dem sie eindeutig angesprochen werden kann. Dazu wird die Feldposition des Eintrags im für diese Tabelle bereitgestellten Feld verwendet.

In den Tabellen, besonders den Nebentabellen, werden ständig Einträge gelöscht bzw. neue angelegt, deshalb muß darauf geachtet werden, daß sich die Indexe der vorhandenen Einträge nicht ändern. Weshalb beim Löschen eines Eintrags nicht alle folgenden Einträge nachgezogen werden, sondern auf ihrer Position im Feld verbleiben. Die freigewordene Position kann später von einem neuen Eintrag belegt werden.

Für jede Variable der MIB ist eine Struktur „Variablen_daten“ vorhanden, in der ihr Object Identifier, der Typ der Variablen und Zeiger auf eine GET, GETNEXT und SET Funktion abgelegt sind. Die Strukturen aller Variablen sind nach aufsteigendem OID im Feld „Variablendaten“ in der Datei *variablen.c* sortiert. Der Zeiger auf eine GETNEXT Funktion existiert nur, wenn es sich bei der Variablen um eine Tabellenvariable handelt, sonst steht an dieser Stelle NULL. Variablen, die nur lesbar sind, benötigen keine SET Funktion, deshalb steht auch hier NULL.
Bei Tabellenvariablen liefert die GETNEXT Funktion nicht den gesuchten Wert, sondern nur den Index der gesuchten Variablen in der Tabelle. Mit Hilfe dieses Indexes kann der Wert der Variablen mit der GET Funktion ermittelt werden. Handelt es sich um eine einfache Variable bei der GETNEXT Anfrage wird nur der nächstfolgende Eintrag im Feld „Variablendaten“ genommen, denn die Einträge sind ja nach aufsteigendem OID sortiert. In der Abbildung 6.1 ist ein Beispieleintrag im Feld „Variablendaten“ zu sehen. Dabei wird auch noch genauer auf die Form der OID–Angabe eingegangen. Die Länge des Feldes für den Object Identifier ist fest. Ist der Object Identifier kürzer als das Feld, werden die nichtbenutzten Stellen mit negativen Zahlen aufgefüllt und die letzte Stelle des Feldes gibt

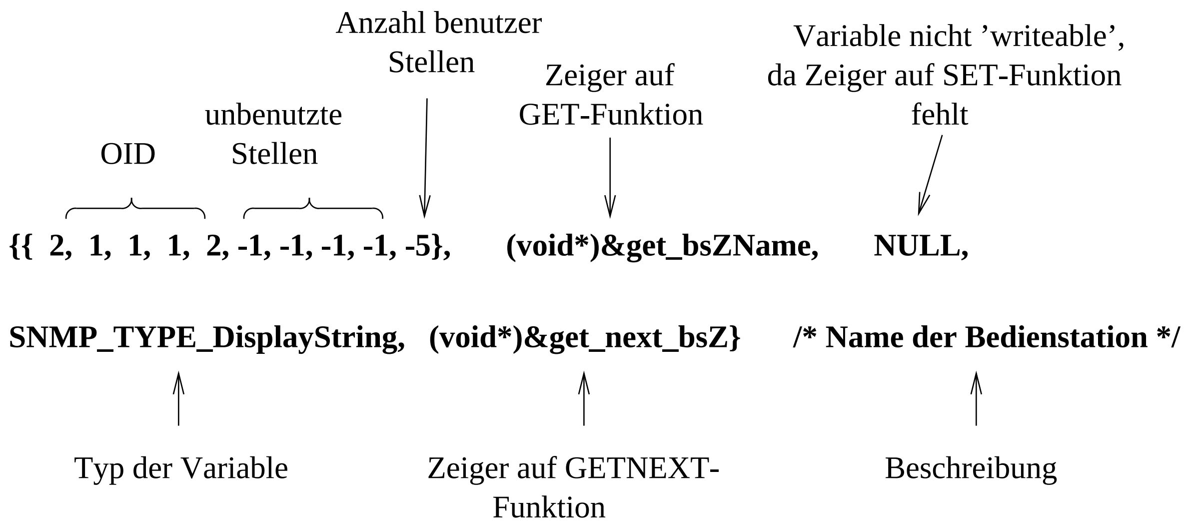

Abbildung 6.1: Der Eintrag für eine Variable in „Variablendaten“

die Anzahl der besetzten Stellen als negierte Zahl an.

6.2 Initialisierung des Subagenten

Bevor der Subagent seine Arbeit korrekt ausführen kann, sind einige Werte zu initialisieren. Als erstes muß ermittelt werden, wo sich die Konfigurationsdateien des Drucksystems befinden, um daraus die real existierende Struktur des Drucksystems zu bestimmen. Die Funktion „Readconf()“ übernimmt das Auslesen der */etc/plp.conf* Datei, in der unteranderem die Namen mit den vollständigen Pfaden der *printcap* und *printer_perms* Datei abgelegt sind.

Nachdem bekannt ist, wo sich die *printcap* Datei befindet, wird die Funktion „printcap_lesen()“ aufgerufen. Sie übernimmt die Abbildung der Struktur des Drucksystems auf die Datenstruktur des Subagenten. Häufig benötigte Daten aus der *printcap* Datei werden übernommen, ferner werden alle Attribute der MIB initialisiert.

Begonnen wird mit dem Initialisieren aller Werte, die unabhängig von der Drucksystemstruktur sind, dazu zählen die Attribute der Funktionseinheit, die Verweise von Klassentabellen auf Nebentabellen und einige Zähler, die beim Abbilden der Struktur inkrementiert werden.

Alsdann wird ermittelt, welche Warteschlangen vorhanden sind. Für jede Warteschlange müssen alle angegebenen Namen gespeichert werden, wobei allerdings nur der erste Name in die MIB eingetragen wird. Desweiteren werden noch die Lock-, Spool- und Statusdatei der Warteschlange festgehalten und alle Attribute der Warteschlange in der MIB initialisiert.

Zu jeder Warteschlange gehören außerdem vier Bedienstationen, dies sind die Zugangskontrolle, der Prä- und Postfilter sowie der Drucker. Auch sie müssen angelegt und mit den Grundwerten ihrer Variablen versorgt werden, dazu gehören auch die Verbindungen zueinander. Weiter muß bei der Bedienstation „Drucker“ noch festgestellt werden, welche Bedieneinheiten vorhanden sind.

Es gibt zwei Typen von Bedieneinheiten, zum einen einen Drucker, zum anderen ein Übertragungsprogramm, das den Auftrag an eine andere Warteschlange weiterleitet. Ist die vorhandene Bedieneinheit kein Drucker, besteht der Name der Bedieneinheit aus dem Zielwarteschlange- und Rechnernamen, bei dem die Warteschlange existiert. Gibt es nur eine Bedieneinheit „Drucker“, ist der Name der Bedieneinheit mit dem Namen der Warteschlange identisch. Sobald mehrere Bedieneinheiten vorhanden sind, hat jede Bedieneinheit ihren eigenen Namen.

Während der Abbildung der Struktur wurden alle Namen der Warteschlangen und der Drucker gespeichert. Außerdem wurde jedem Namen der Index seiner Warteschlange zugeordnet. So kann später, anhand des in einem Auftrag angegebenen Names, die richtige Warteschlange ermittelt werden. Ist die Warteschlange bekannt, kann auch die zuständige Bedienstation herausgefunden werden.

Zum Schluß muß noch ein Port angelegt werden, bei dem Meldungen vom Drucksystem angenommen werden. Dadurch erfährt der Subagent, welche Tätigkeiten im Drucksystem stattfinden.

Der Subagent kennt jetzt den Aufbau des Drucksystems und seine MIB ist initalisiert, nun kann er sich bei einem DPI–fähigen Agenten anmelden, um seinen Aufgaben nachzukommen.

6.3 Bearbeitungsverlauf

Nachdem der Subagent gestartet wurde und seine Werte initialisiert sind, kann er sich bei einem Agenten, der auf dem gleichen Rechner läuft, anmelden. War die Anmeldung erfolgreich, wartet er auf ankommende Pakete vom Agenten und Drucksystem. Erhält er eins, beginnt er sofort mit der Bearbeitung dieses Paketes. Bei Paketen vom Drucksystem muß er die Werte der Variablen der MIB entsprechen ändern. Kommt das Paket vom Agenten muß er nach der Bearbeitung auch noch ein Antwortpaket zusammenstellen und dem Agenten zurücksenden. Möchte der Agent die Zusammenarbeit beendet, teilt er dies dem Subagenten durch ein entsprechendes Paket mit und bricht die Verbindung ab. Der Subagent nimmt die Möglichkeit des Verbindungsabbruchs nicht wahr.

6.3.1 Anmelden beim DPI–Agenten

Damit die MIB des Subagenten vom Agenten berücksichtigt wird, muß sie als erstes dem Agenten bekannt gegeben werden. Dazu schickt der Subagent ein DPI OPEN Paket an den Agenten, um eine TCP–Verbindung mit ihm aufzubauen. Vorausgesetzt wird allerdings, daß beide Prozesse auf dem gleichen Rechner laufen. An welchen Port dieses Paket zu schicken ist, hat der Subagent zuvor durch eine SNMP GET Anfrage beim Agenten ermittelt.

Sobald die Verbindung steht, kann der Subagent seine Teilbäume beim Agenten registrieren. Dazu schickt er ein DPI REGISTER Paket für jeden vorhandenen Teilbaum an den Agenten. In diesem Subagenten existiert nur ein Teilbaum, der Teilbaum für den Druckdienst, der dem Agenten bekannt sein muß, um zu wissen, welche Variablen sich in dieser Teil–MIB befinden. Als Teilbaum wird dabei die Struktur von Abbildung 5.4 bezeichnet und der Name ist der Pfad von der MIB–Wurzel bis zum Beginn dieser Struktur.

Welche Werte der Agent in den verschiedenen DPI Paketen erwartet, ist in [RFC 1592] und [Hain 95] nachzulesen. Möchte man wissen, wie der Subagent die Pakete erzeugt und welche Werte er dafür benötigt, wende man sich bitte an [HaHa 94].

6.3.2 Verarbeitung von DPI–Paketen

Grundsätzlich können vom Agenten DPI GET, GETNEXT und SET Anfragen kommen, DPI GETBULK Pakete wurden bei der Registrierung dieses Teilbaums ausgeschlossen. Dazu kommen noch DPI Pakete, welche nicht direkt die MIB betreffen, aber für die Kommunikation zwischen Agent und Subagent verantwortlich sind. Im folgenden wird nur auf die DPI GET, GETNEXT und SET Anfragen eingegangen, dabei wird auch nur das Wichtigste erwähnt und die Änderungen gegenüber dem Subagenten [HaHa 94] für Systemmanagementaufgaben angesprochen.

6.3.2.1 DPI GET Anfrage

Um den Wert einer Variablen zu ermitteln, muß als erstes der als String vorliegende OID in ein Feld mit Zahlen zerlegt werden. Ein Beispiel: “1.3.6.1.3.100.5” wird zu 1, 3, 6, 1, 3, 100, 5 konvertiert. Mit Hilfe des OID wird im Feld „Variablendaten“ der entsprechende Eintrag gesucht. In diesem Eintrag findet man nun einen Zeiger auf eine Funktion zum Lesen des Wertes der Variablen. Ist die Variable allerdings eine Tabellenvariable wird noch ihr Index in der Tabelle benötigt. Dieser Index muß beim Aufruf der Funktion mitangegeben werden. Damit jeder Funktionsaufruf einheitlich gestaltet werden kann, wird dieser Index einfach in eine Zahl konvertiert und an den OID angehängt. Die betroffene Funktion kennt die Länge ihres OIDs und übernimmt alle folgenden Zahlen als Funktionsparameter.

Das Ergebnis der Anfrage wird dem Agenten durch ein DPI RESPONSE Paket mitgeteilt.

6.3.2.2 DPI GETNEXT Anfrage

Bei einer GETNEXT Anfrage gestaltet sich die Suche nach der lexiographisch folgenden Variablen mitunter sehr schwierig. Bei einfachen Variablen kann einfach der im Feld „Variablendaten“ folgende Eintrag als gesuchte Variable genommen werden, da dieses Feld lexiographisch sortiert ist.

Ist die angegebende oder gesuchte Variable allerdings eine Tabellenvariable, muß in der Tabelle die Variable mit nächstgrößerem OID gefunden werden. Dazu gibt es für jede Tabelle eine Funktion, die den Index des gesuchten Tabelleneintrags liefert.

Ist die richtige Variable bzw. der Index einer Tabellenvariable gefunden, kann ihr Wert mit der Lesefunktion ermittelt werden. Liefert die Lesefunktion keinen Wert, wird von neuem versucht einen Nachfolger zu finden.

6.3.2.3 DPI SET Anfrage

Das Gegenstück zur DPI GET Anfrage ist die DPI SET Anfrage, damit lassen sich die Werte von Variablen setzen. Der Ablauf der Bearbeitung dieser Anfrage entspricht deshalb der von DPI GET Anfragen, nur daß der Wert geschrieben und nicht gelesen wird. Der zu setzende Wert, die Länge des Wertes und die auszuführende SET Operation werden der Schreibfunktion in Parametern mitübergeben. Der Index bei Tabellenvariablen wird wieder in eine Zahl konvertiert und dem OID angehängt. Bei SET Anfragen wird zwischen drei SET Operationen unterschieden. Dies können sein eine SET, COMMIT oder UNDO Operation.

- SET Paket
  Bei einem SET Paket wird der Wert der Variablen noch nicht verändert. Es wird nur geprüft ob es möglich wäre, der Typ der Variablen richtig ist und der Wert im Wertebereich liegt.

- COMMIT Paket
  Jetzt wird er Wert der Variablen tatsächlich gesetzt. Zuvor wird jedoch der alte Wert gerettet um ihn evtl. wieder herstellen zu können. Dieser Wert wird aber nur bis zum nächsten Eintreffen eines SET Paketes gespeichert.

- UNDO Paket
  Muß der Wert einer Variablen wieder auf den alten Wert zurückgesetzt werden, kann dies durch ein UNDO Paket erreicht werden. Der alte Wert wurde ja beim COMMIT Paket gerettet, scheitert das Zurücksetzen trotzdem, muß eine Fehlermeldung erfolgen.

6.3.3 Verarbeitung von Paketen vom Drucksystem

Kommt ein Paket vom Drucksystem an, wird die Funktion „paket_auswerten()“ aufgerufen. Sie befindet sich in der Datei *paketauswertung.c*. In diesem Paket ist der Name einer Warteschlange angegeben, der Objekttyp, die Auftragsnummer, der Typ und die Art der Meldung sowie der Zeitpunkt der Meldung.

Anhand des Warteschlangenamens wird der Zweigindex ermittelt, was die Funktion „get_-index()“ übernimmt. Darauf muß entschieden werden, für welches Objekt im Zweig die Variablen aktualisiert werden müssen. Der Grund der Meldung und die genaue Spezifikation des Grundes bestimmen die betroffenen Variablen. Mehr dazu wird in Abschnitt 6.5 gesagt.

Neben den Variablen müssen aber auch die Nebentabellen und die Beziehungen des Auftrags zu den Objekten aktualisiert werden. Dies kann zum Löschen oder Anlegen von Einträgen in der Auftragstabelle und den Nebentabellen führen. Dabei ist zu beachten, daß sich die Indexe der bereits vorhandenen Einträge nicht ändern dürfen.

Erreicht den Subagenten ein Paket, daß er keinem Objekt zuordnen kann, wird eine Fehlermeldung ausgegeben.

6.3.4 Abmelden beim DPI–Agenten

Beide Seiten, der Agent und der Subagent, können die Verbindung beenden. Meist wird allerdings der Agent die Verbindung abbrechen wollen, aufgrund der Registrierung anderer Teilbäume oder dem Beenden des Agenten.

Der Subagent selbst wird sich gewöhnlich nicht abmelden, außer er stürzt ab, wobei er sich dabei nicht mehr korrekt abmelden kann und das Beenden des Subagenten vom Agenten selbst bemerkt werden muß.

6.4 Beschreibung der Funktionen

In diesem Abschnitt werden nur die Funktionen beschrieben, die unmittelbar mit den Variablen der MIB zu tun haben, d.h. auf die im Feld „Variablendaten“ angegebenen Funktionen. Alle anderen wichtigen Funktionen können aus den weiter oben zitierten Quellen entnommen werden.

Viele dieser Funktionen sind auch einfach aufgebaut und in ähnlicher Form mehrfach vorhanden, weshalb sie nur kurz angesprochen werden. Nur umfangreiche und komplexe Funktionen für die Variablen eines Objekts werden bei den entsprechenden Abschnitten erklärt.

6.4.1 Allgemeines

Hier wird beschrieben, welche Eigenheiten die GET, GETNEXT und SET Anfragen haben. Außerdem werden bereits einige Funktionen erklärt, die in leicht abgeänderter Form in fast allen Objekten des Objektmodells enthalten sind. Dazu gehören vor allem die Funktionen zum Lesen der Attribute des Leistungsmanagements.

Alle Funktionen wurden auf die Dateien *funktionseinheit.c*, *zugangskontrolle.c praefilter.c*, w*arteschlange.c*, *postfilter.c*, *BSdrucker.c* und *BEdrucker.c* verteilt, je nachdem für welches Objekt sie zuständig sind.

6.4.1.1 GET Funktionen

Das Lesen von Werten aus der MIB wurde auf zwei Arten von Funktionen reduziert:

- char* get_VARIABLENNAME(char* Liste_der_Parameter)
- int* get_VARIABLENNAME(char* Liste_der_Parameter)

Welche davon angewandt wird, richtet sich nach den Typ der gesuchten Variablen. Wie die 'Liste_der_Parameter' aufgebaut ist, wurde bereits beschrieben, der OID gefolgt von weiteren Parametern, abgelegt im Integerformat. Die Funktion legt das Ergebnis in einem global bereitgestellten Puffer ab und liefert nur noch einen Zeiger darauf zurück.

Alle Funktionen zum Lesen der Attribute 'Name der Komponenten', req, acc, rej, dep, util, del, cor und err , beschränken sich auf das Lesen dieser Werte aus der entsprechenden Struktur im Subagenten oder bilden die Differenz zweier Variablen. Aufgrund dessen werden sie nicht näher beschrieben.

Bei der Vergabe von Funktionsnamen wurden ein paar Konventionen eingehalten. Alle Funktionsnamen werden durch das Schlüsselwort „get_“ eingeleitet, gefolgt von einem oder zwei Kleinbuchstaben zur Angabe der Hauptgruppe, daran keiner bis zwei Buchstaben für die Klassengruppe und schließlich der Variablenname.

Hauptgruppenkürzel		Klassengruppenkürzel	
fe	Funktionseinheit	Z	Zugangskontrolle
bs	Bedienstationen	Pr	Präfilter
be	Bedieneinheiten	Po	Postfilter
ws	Warteschlangen	D	Drucker
a	Auftrag		

So lautet z.B. die Funktion zum Lesen des Namens der Bedienstation Zugangskontrolle „get_bsZName()“.

6.4.1.2 GETNEXT Funktionen

Die GETNEXT Funktionen dienen zur Findung des nächsten Index in einer Tabelle. Der Funktionsname wird aus „get_next_“, Hauptgruppenkürzel, falls nötig Klassengruppenkürzel und bei Nebentabellen Nebentabellenkürzel gebildet. Die Nebentabelle für Aufträge in der Klassengruppe Zugangskontrolle lautet deshalb „get_next_bsZAuft()“.

Nebentabellenkürzel	
Auft	Nebentabelle für Aufträge
Ziel	Nebentabelle für gerichtete Verbindungen im Objektmodell
Con	Nebentabelle für Objekte, in denen dieses enthalten ist
Be	Nebentabelle für enthaltene Bedieneinheiten

Einen Rückgabewert liefert die Funktion nicht, stattdessen werden die mitübergebenen Parameter verändert. Diese Parameter liegen wieder in einer ‘Liste_der_Parameter’ vor, wie bereits bei GET Funktionen.

In diesem Subagenten gibt es zwei Arten von Tabellen. Tabellen mit nur einem Index, wie die Klassentabellen und Nebentabellen die genau einen zugehörigen Eintrag zu einem Klassentabelleneintrag haben und Nebentabellen mit zwei Indexe, die beliebig viele zugehörige Einträge zu einem Klassentabelleneintrag haben können.

Bei Tabellen mit nur einem Index gibt es drei Möglichkeiten:

- Es wird der kleinste Index gesucht,
- der folgende Index des vorgegeben wird verlangt oder
- es gibt keinen größeren Index in der Tabelle als der vorgegeben und deshalb wird der Wert „-1“ zurückgegeben.

Der Index eines Eintrags errechnet sich aus der Position des Eintrags im bereitgestellten Feld um eins inkrementiert.

Nur ein paar Nebentabellen haben zwei Indexe, wovon der erste der Index der Klassentabelle ist und auch gesondert gespeichert werden muß, der zweite errechnet sich wieder aus der Position des Eintrags. Im Prinzip gibt es wieder die drei Möglichkeiten wie bei Tabellen mit einem Index, nur daß die Suche nach den nächsten Indexpaar sich aufwendiger gestaltet. Der erste Index darf erst erhöht werden, wenn kein größerer Wert mehr für den zweiten Index existiert.

6.4.1.3 SET Funktionen

Die Namensfindung der SET Funktionen geschieht auf die gleiche Weise, wie bereits bei den GET Funktionen, nur daß sie mit dem Schlüsselwort „set_“ beginnen. Als Rückgabewert erhält man einen Integerwert, der aussagt, wie das Setzen der Variablen verlief. Beim Aufruf der Funktion werden zusätzliche Parameter benötigt. Der OID und vorhandene Indexe sind wieder in der ‘Liste_der_Parameter’ enthalten, dazu gibt es Parameter für den zu setzenden Wert, seinen Speicherbedarf in Byte und welche Form von SET ausgeführt werden soll. Damit hat die Funktionsdeklaration die Form:

int set_VARIABLENNAME(int *Liste_der_Parameter, char *Wert,
unsigned short Wertlänge, int set_or_commit)

Zu Beginn aller SET Funktionen wird überprüft, ob der zu setzende Wert im definierten Wertebereich liegt. Anschließend muß bei Tabellenvariablen noch der angegebene Index überprüft werden.

6.4.2 Funktionseinheit

get_feBbs() Die Funktionseinheit ist betriebsbereit, wenn der Druckdämon aktiv ist. Feststellen läßt sich dies anhand der Lock–Datei, die der Dämon beim Start sperrt, um den Start anderer Druckdämons zu verhindern.

get_feN() Befinden sich in der Klassentabelle der Aufträge noch Einträge, ist das Drucksystem beschäftigt, sonst nicht. Befinden sich Aufträge im Drucksystem, muß auch überprüft werden, ob alle Bedieneinheiten besetzt sind oder wenigstens noch eine einen Auftrag annehmen kann.

get_feA() Zur Ermittlung des Wertes des Administrationsattributs wird in gleicher Weise wie bei der Betriebsbereitschaft vorgegangen.

set_fePlace() Beim Start des Subagenten ist dieses Attribut mit einem Standardwert besetzt, es kann aber vom Administrator mit der Raumbezeichnung, in dem der Rechner aufgestellt ist, besetzt werden. Der Wert wird dabei nur in einer vorgegebenen Variablen gespeichert.

set_feA() Zum Starten des Druckdämons muß eine Shell geöffnet und das Kommando zum Starten des Dämons abgesetzt werden. Soll er beendet werden, bedient man sich des Shellscripts ‘kall’, womit sich der angegebene Prozeß beendet läßt.

6.4.3 Zugangskontrolle und Präfilter

get_bsZN() Befinden sich in der zugehörigen Nebentabelle „Aufträge“ Einträge, ist die Bedienstation beschäftigt, sonst nicht.

6.4.4 Warteschlange

get_wsBbs() In jedem Warteschlangenverzeichnis befindet sich eine Lock–Datei. Der Name und Pfad dieser Datei wurde bei der Initalisierung des Subagenten gespeichert. Solange diese Datei zugänglich ist, ist auch die Warteschlange betriebsbereit. Um dies festzustellen wird die Datei kurz zum Schreiben geöffnet und sofort wieder geschlossen.

get_wsA() Je nachdem, wie die Zugriffsrechte bei der Lock–Datei der Warteschlange gesetzt sind, entscheidet sich, ob Aufträge in die Warteschlange eingereiht werden dürfen oder nicht. Mit Hilfe der Funktion „Checklockfile()“ werden diese Rechte gelesen und durch die Auswertung dieser Rechte erhält man das gesuchte Ergebnis.

set_wsA() Während bei der Funktion 'get_wsA()' die Zugriffsrechte nur gelesen und ausgewertet wurden, müssen sie hier anschließend noch entsprechend gesetzt werden.

6.4.5 Postfilter

get_bsPoBbs() Solange ein Druckserver aktiv ist, kann der Postfilter seine Arbeit erledigen. Stehen neue Aufträge an, übernimmt der Druckdämon das Starten eines Druckservers. An dieser Stelle wird aber überprüft, ob ein aktiver Druckserver existiert oder nicht.

get_bsPoA() und set_bsPoA() Diese beiden Funktionen arbeiten genauso wie die entsprechenden bei der Warteschlange, deshalb erübrigt sich hier eine Beschreibung.

6.4.6 Bedienstation „Drucker“

get_bsDBbs() Die Betriebsbereitschaft der Bedienstation „Drucker“ ist abhängig von den enthaltenen Bedieneinheiten „Drucker“. Deshalb müssen alle enthaltenen Bedieneinheiten überprüft werden, ob sie betriebsbereit sind. Daraus ergibt sich dann der Wert für die Bedienstation „Drucker“.

6.4.7 Bedieneinheit „Drucker“

get_beDBbs() Wenn es möglich ist ein Verbindung zum Drucker aufzubauen, ist auch die Bedieneinheit betriebsbereit. Dabei muß jedoch zwischen einem realen Drucker und einer entfernten Warteschlange unterschieden werden.

get_beDA() Besitzt eine Bedienstation mehrere Bedieneinheiten, können diese nochmals getrennt administriert werden. Dazu werden wieder die Zugriffsrechte speziell angelegter Dateien verwendet. Durch das Lesen und Auswerten der Zugriffsrechte dieser Dateien kann der Wert dieses Attributs ermittelt werden.

set_beDO() Der Drucker muß nicht notwendigerweise im gleichen Raum wie der Rechner stehen. Deshalb kann der Administrator hier die Raumbezeichnung eintragen.

set_beDA() Wie bereits bei der Funktion „get_beDA()“ gesagt, existiert diese Möglichkeit nur für einige Drucker. Diese können gesperrt oder freigegeben werden, was mit Hilfe der Zugriffsrechte spezieller Dateien realisiert wird.

6.4.8 Auftrag

get_awsTime(), get_aSize(), get_aPrio() und get_aUser() Den gesuchten Wert besorgen sich alle Funktionen aus der Auftragskontrolldatei, die im Warteschlangenverzeichnis abgelegt ist. Dazu werden alle Kontrolldateien einer Warteschlange gelesen, bis die mit der richtigen Auftragsnummer gefunden ist.

get_aA() Es besteht die Möglichkeit Aufträge in der Warteschlange festzuhalten, d.h. sie werden nicht weitergeleitet. Ob dies der Fall ist wird wiederum über die Zugriffsrechte der Auftragskontrolldatei ermittelt.

set_aP() Für den Auftrag kann eine beliebige Prioität gesetzt werden. Sie muß nur in der Kontrolldatei des Auftrags eingetragen werden. Soll der Auftrag allen anderen vorgezogen werden, wird auch seine Warteschlangeneingangszeit manipuliert.

set_aA() Neben dem Festhalten oder Freigeben von Aufträgen, was durch Setzen der Zugriffsrechte geschieht, kann ein Auftrag auch gelöscht werden. Dazu werden nur seine Auftragskontrolldatei und die Datendateien gelöscht.

6.5 Meldungen vom Drucksystem

Damit der Subagent über die Vorgänge im Drucksystem unterrichtet ist, erhält der Subagent ständig Meldungen. Jedes Objekt des Objektmodell erstellt eine Meldung, wenn es einen Auftrag erhält und wenn der Auftrag das Objekt wieder verläßt. In der Meldung ist auch angegeben, wie die Bearbeitung verlief und der genaue Zeitpunkt der Meldungserstellung. Somit weiß der Subagent, wo sich die Auftäge gerade befinden und wie die Bearbeitung verlief. Das Drucksystem greift zur Erstellung der Meldungen auf die Funktion „send_agent_packet()“ zurück. Diese erzeugt ein Paket mit den erforderlichen Informationen und schickt es an den Subagenten.

Die Funktion ist in der Datei *agent_meldungen.c* implementiert und ihre Funktionsdeklaration lautet:

void send_agent_packet(char *drucker, int objekt, int job, int typ, int art)

Die Informationen werden nur durch Leerzeichen getrennt aneinandergereiht und dann über eine UDP–Verbindung an den Subagenten geschickt. Im einzelnen sind folgende Daten nötig:

- Der Name der Warteschlange, für die dieser Auftrag bestimmt ist. Dadurch wird zwischen den Objekten eines Objekttyps unterschieden.
- Der Objekttyp, von dem diese Meldung ausgeht.
- Die Auftragsnummer, des Auftrags, der gerade die Komponente erreicht bzw. verlassen hat.
- Der Grund der Meldung unterscheidet zwischen Auftragsannahme und -abgabe.
- Eine genauere Spezifizierung des Grundes. Wurde der Auftrag abgewiesen, beendet aufgrund nichterlaubter Optionen oder wurde er korrekt weitergeleitet.
- Der Zeitpunkt der Meldungserstellung, damit auch die Verweilzeit eines Auftrags in einer Komponente ermittelt werden kann.

Welche Werte für den Objekttyp, den Grund und der genaueren Spezifikation möglich sind, wird in den anschließenden Tabellen aufgeführt. Bei der Auftragsannahme gibt es keine genauere Spezifikation des Grundes, nur beim Auftragsabschluß.

Objekttyp (objekt)	
1	Zugangskontrolle
2	Präfilter
3	Warteschlange
4	Postfilter
5	BS–Drucker
6	BE–Drucker

Meldungsgrund (typ)	
1	Auftragsannahme
2	Auftragsabschluß

Meldunsspezifikation (art)		
1	NORMAL	Auftrag wurde an nächste Objekt weitergeleitet
2	BEENDET	Auftrag wurde in Funktionseinheit beendet
3	ABGELEHNT	Auftrag wurde nicht angenommen
4	ERROR	Fehler in der Auftragsbeschreibung

Hier anzugeben, wo überall die Funktion „send_agent_packet()“ im Drucksystem aufgerufen wird, wäre sehr umfangreich und wahrscheinlich schwer verständlich, da der Kontext nicht bekannt ist. Besser man betrachtet sich die Quelldateien und sucht nach den Funktionsaufrufen.

6.6 Installation und Aufrufsyntax

Im Hauptverzeichnis des Subagenten befindet sich das Makefile. Die Quelldateien des Subagenten sind auf drei Unterverzeichnisse verteilt. Im Unterverzeichnis *dpi* befinden sich die Grundfunktionen des Subagenten, wozu auch die Funktionen der DPI–Schnittstelle gehören. Funktionen zum Lesen und Schreiben der MIB–Variablen befinden sich in den Dateien im Unterverzeichnis *funktionen*. Diese Funktionen wurden auch weiter oben erklärt. Zuletzt gibt es noch ein Unterverzeichnis *library*, worin die Basisfunktionen des Drucksystems enthalten sind. Im Hauptverzeichnis existieren auch noch die Dateien *config.h* und *lp.h*, diese werden von den Libraryfunktionen des Drucksystems benötigt.

Beim Starten des Subagenten muß darauf geachtet werden, daß auf dem gleichen Rechner bereits ein DPI–fähiger Agent läuft, bei dem er sich anmelden kann. Befindet sich der Agent auf einem anderen Rechner, muß dies im Quellcode des Subagenten berücksichtigt werden.

Ein einfacher Aufruf von 'make' erzeugt den ausführbaren Subagenten. Anschließend können durch den Aufruf von 'make clear' alle überflüssigen Dateien wieder entfernt werden.

Zum Starten des Subagenten sollte man Root–Rechte besitzen, sonst funktioniert der Subagent nicht richtig. Dabei können auch einige Optionen mitangegeben werden.

dpi_printer [**-d**[**1**|**2**]] [**-write**] [**-cache**]

-d Schaltet den Debug–Modus ein, wodurch sich verfolgen läßt, welche DPI Pakete ankommen und abgeschickt werden. Wie umfangreich die Darstellung des Inhalts der Pakete erfolgen soll, kann durch die Nummer bestimmt werden.

-write Nur wenn diese Option gesetzt ist können die SET Funktionen aufgerufen werden. Fehlt sie, sind nur Leseoperationen erlaubt.

-cache Damit der Cache verwendet wird.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

In dieser Arbeit ist gezeigt worden, daß das vorgestellte Objektmodell in der Lage ist einen Basisdienst abbilden zu können. Von zwei ausgewählten Basisdiensten, dem Druck- und Nachrichtendienst wurde das spezifische Objektmodell aufgestellt und die Managementinformation und -funktionalität mit Werten belegt. Weiter wurde gezeigt, wie das Objektmodell in eine Internet MIB abgebildet werden kann.

Nachdem die ausgewählten Basisdienste untersucht und ein funktionales Modell aufgestellt wurde, bereitete es keine großen Schwierigkeiten mehr, dieses in ein Objektmodell zu transferieren. Daraus läßt sich ersehen, daß das Objektmodell gute geeignet für die Darstellung eines Basisdienstes ist.

Mehr Probleme gibt es bei der Erfüllung der Managementanforderungen des Objektmodells an den Basisdienst. So können einige Attribute nicht mit Werten gefüllt werden, andere nur durch größere Eingriffe in den Basisdienst. Obwohl die Basisdienste viele Informationen anbieten, sind diese meist nicht für das Last- und Statusmanagement geeignet, sondern mehr zur lokalen Verwaltung des Dienstes. Die gebotene Funktionalität der Basisdienste sollte jedoch zur Administration der Dienste ausreichen und darüberhinaus vom Objektmodell geforderte Funktionalität kann entfallen. Denn das Objektmodell versucht alle Administrationsmöglichkeiten zu berücksichtigen, die natürlich nicht alle von den Basisdiensten angeboten werden.

Um jedoch eine homogene Struktur zu erreichen, sollen Attribute, deren Wert konstant ist oder nicht ermittelt werden kann, nicht wegfallen. Besser ist es, auch diese Attribute aufzunehmen und mit einem Wert zu belegen, der anzeigt, dieses Attribut nicht unterstützen zu können.

Zum Schluß folgt die Abbildung des Objektmodells in eine Internet MIB. Auch dabei wurde Wert auf eine Abbildungsvorschrift gelegt, die es erlaubt beliebige Objektmodelle abzubilden ohne vorher große Überlegungen anzustellen. Dabei wurden bewußt in kauf genommen, Strukturen zu erhalten, die die MIB vergrößern und für einige Basisdienste unnötig sind. Dazu gehören auch die Nebentabellen, die für einige Klassengruppen überflüssig sind. Allerdings erhält man damit eine Struktur, die für alle Basisdienste identisch ist und es ermöglicht universelle Managementapplikationen zu entwickeln und anzuwenden.

Im Anhang A dieser Arbeit ist die MIB für den Druckdienst aufgestellt und bereits von einem DPI–Subagenten realisiert. Anhand dieses Beispiels dürfte es auch keine größeren Probleme mehr bei der Erstellung von MIBs für andere Basisdienste geben. Auch für den Nachrichtendienst wurde die Managementanforderung in dieser Arbeit aufbereitet.

In Abbildung 5.4 sind die Haupt- und Untergruppen der MIB für den Druckdienst zu sehen. Die Hauptgruppen sind in jeder MIB, die aus einem Objektmodell hervorgeht zu finden. An diese Hauptgruppen sind dann die vorhandenen Komponenten des abzubildenden Objektmodells anzuhängen.

7.2 Ausblick

Die Bereitstellung eines Objektmodells ist bereits ein großer Schritt in die Richtung des Managements von Basisdiensten. Es stellt eine gemeinsame Grundlage bereit, auf der aufgebaut werden kann. Wichtig ist jetzt, daß sich die Entwickler von Basisdiensten auch mit dem Objektmodell und den Managementanforderungen beschäftigen, um in zukünftigen Entwicklungen diese Anforderungen bereitstellen zu können.

Durch die starke Anlehnung an das Netzmanagement, besonders bei der Auswahl der Attribute, dürfte es auch nicht sehr aufwendig sein, geeignete Managementapplikationen zu entwicklen, die die Informationsdarstellung für den Endanwender übersichtlicher macht.

Mit der Realisierung eines Subagenten für den Druckdienst ist es auch möglich, diese MIB anderen Agenten zugänglich zu machen, sofern sie über eine DPI–Schnittstelle verfügen. Der Druckdienst ist sicher einer der aufwendigsten Basisdienste die es zu administrieren gilt, aber es werden sicher bald andere Basisdienste folgen.

In dem bisherigen Modell wurde das Konfigurationsmanagement und Accountingmanagement nicht berücksichtigt, auch das Fehlermanagement wurde nicht vollständig abgedeckt. Inwieweit diese Bereiche abgedeckt werden sollen und können bedarf noch einer genaueren Untersuchung.

Anhang A

Die MIB für den Druckdienst

Um möglichst aussagekräftige Variablennamen zu erhalten, diese aber nicht zu lang werden sollten, wurden ein paar Konventionen eingehalten.

Der Name einer Klassentabellenvariablen setzt sich aus den Komponenten „pri", Hauptgruppenkürzel, wenn nötig Klassengruppenkürzel und dem Attributnamen zusammen. Bei Nebentabellenvariablen wird vor dem Attributnamen noch „NT" und der Name der Tabelle eingefügt. Die meisten Kürzel wurden schon in Abschnitt 6.4.1 vorgestellt, die restlichen sind leicht erkennbar.

Die exakte Beschreibung der Variablen erfolgte bereits in Kapitel 4, weshalb hier nur die Beschreibung der MIB in ASN.1 Notation zu finden ist.

Im MIB–Modul werden auch einige Variablen vom IIMComibtransConventions–Modul importiert. Eine gekürzte Fassung dieses Modules wurde deshalb noch angehängt.

```
MIBprintserverSNMPv1 DEFINITIONS ::= BEGIN

    -- Objects in this MIB are implemented in the local SNMP agent.

        IMPORTS
    -- MODULE-IDENTITY, OBJECT-TYPE, snmpModules, enterprises
    -- FROM SNMPv2-SMI

            Complex, Pointer, TableIndex, MultiComplex, Parent
                FROM IIMComibtransConventions

            mgmt, NetworkAddress, IpAddress, Counter, Gauge, TimeTicks
                FROM RFC1155-SMI

            InstancePointer, TimeStamp
                FROM SNMPv2-TC

            OBJECT-TYPE
                FROM RFC-1212;

    -- LRZ-Teilbaum wird eingehaengt unter enterprises.100 !!!

        lrz OBJECT IDENTIFIER ::= { enterprises 100 }

    -- Druckdienst-Teilbaum wird eingehaengt unter lrz.5 !!!

    -- printserverMIB MODULE-IDENTITY
    -- LAST-UPDATED "9401210000Z"
    -- ORGANIZATION "TU Muenchen"
    -- CONTACT-INFO " Erwin Hainzinger
    -- Postal:  TU Muenchen
    -- E-mail:  hainzing@informatik.tu-muenchen.de"
    -- DESCRIPTION "MIB module describing PRINTSERVER objects."
    -- ::= { snmpModules x }


    -- textual conventions
        DisplayString ::= OCTET STRING (SIZE (0..255))
        -- This data type is used to model textual information taken
        -- from the NVT ASCII character set. By convention, objects
        -- with this syntax are declared as having.
```

```
    Jobs ::= Pointer
    Destination ::= Pointer
    Containment ::= Pointer
    Recursion ::= Pointer
    -- This data types are used to point to sidetables where
    -- pointers to the included jobs or objects, the next object
    -- for the job can be found.

  printServer OBJECT IDENTIFIER ::= { lrz 5 }

  funktionseinheit OBJECT IDENTIFIER ::= { printServer 1 }
  bedienstationen OBJECT IDENTIFIER ::= { printServer 2 }
  bedieneinheiten OBJECT IDENTIFIER ::= { printServer 3 }
  warteschlangen OBJECT IDENTIFIER ::= { printServer 4 }
  auftraege OBJECT IDENTIFIER ::= { printServer 5 }

  bsZugangskontrolle OBJECT IDENTIFIER ::= { bedienstationen 1 }
  bsPraefilter OBJECT IDENTIFIER ::= { bedienstationen 2 }
  bsPostfilter OBJECT IDENTIFIER ::= { bedienstationen 3 }
  bsDrucker OBJECT IDENTIFIER ::= { bedienstationen 4 }

  beDrucker OBJECT IDENTIFIER ::= { bedieneinheiten 1 }

  warteschlange OBJECT IDENTIFIER ::= { warteschlangen 1 }

  auftrag OBJECT IDENTIFIER ::= { auftraege 1 }



-- ***********************************************************
-- FUNKIONSEINHEIT Printsystem
-- ***********************************************************

  priFEName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A textual description of the entity.
              This value should include the full name
              and version of the Program.  It is mandatory
              that this only contain printable ASCII
              characters."
  ::= { funktionseinheit 1 }
```

```
priFETime OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The time (in hundredths of a second)
                since the subagent was started."
::= { funktionseinheit 2 }

priFEPlace OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The physical location of the computer."
::= { funktionseinheit 3 }

priFEReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { funktionseinheit 4 }

priFEAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { funktionseinheit 5 }

priFERej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { funktionseinheit 6 }

priFEDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { funktionseinheit 7 }
```

```
priFEUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
                included in this object."
::= { funktionseinheit 8 }

priFEDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { funktionseinheit 9 }

priFECors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { funktionseinheit 10 }

priFEErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { funktionseinheit 11 }

priFEBbs OBJECT-TYPE
        SYNTAX INTEGER {
                        enabled(1),
                        disabled(2),
                        unkown(3)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of
                the printdemon (lpd).  Only if the printdemon
                is active, printjobs can be handled."
```

```
    ::= { funktionseinheit 12}

    priFEN OBJECT-TYPE
            SYNTAX INTEGER {
                            idle(1),
                            active(2),
                            busy(4)
                            }
            ACCESS read-only
            STATUS mandatory
            DESCRIPTION "The use of the managed object.  If all
                    devices 'printer' are used, the printsystem
                    ist 'busy', else it's 'idle' or 'active'."
    ::= { funktionseinheit 13}

    priFEA OBJECT-TYPE
            SYNTAX INTEGER {
                            locked(2),
                            unlocked(4)
                            }
            ACCESS read-write
            STATUS mandatory
            DESCRIPTION "The desired state of the printdemon (lpd).
                    Here it is possible to start and kill the
                    printdemon."
    ::= { funktionseinheit 14}


-- Tabelle Auftraege
-- *****************************************************

    priFEfollowRecordTable OBJECT-TYPE
            SYNTAX SEQUENCE OF PriFEfollowRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This table contains links to objects,
                    who can accept new job.  Only the objects
                    'access' can assume new jobs, therefore only
                    links to 'access' objects are allowed."
    ::= { funktionseinheit 15 }

    priFEfollowRecordEntry OBJECT-TYPE
            SYNTAX PriFEfollowRecordEntry
```

```
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priFEfollowRecordTable."
        INDEX { priFEfollowTableIndex }
::= { priFEfollowRecordTable 1 }

PriFEfollowRecordEntry ::= SEQUENCE
    {
        priFEfollowTableIndex TableIndex,
        priFEfollowObject InstancePointer
    }

priFEfollowTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is the index of the
                priFEfollowRecordTable."
::= { priFEfollowRecordEntry 1 }


priFEfollowObject OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to an object
                in the accesstable."
::= { priFEfollowRecordEntry 2 }
```

```
-- ************************************************************
-- BEDIENSTATION Zugangskontrolle
-- ************************************************************

-- Klassentabelle fuer Zugangskontrolle
```

```
priBSaccessRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSaccessRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This managed object is used to represent
                information about the success of printjobs to
                be assumed by the printsystem."
::= { bsZugangskontrolle 1 }

priBSaccessRecordEntry OBJECT-TYPE
        SYNTAX PriBSaccessRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSaccessRecordTable."
        INDEX { priBSaccTableIndex }
::= { priBSaccessRecordTable 1 }

PriBSaccessRecordEntry ::= SEQUENCE
    {
        priBSaccTableIndex TableIndex,
        priBSaccName DisplayString,
        priBSaccReqs Counter,
        priBSaccAccs Counter,
        priBSaccRej Gauge,
        priBSaccDeps Counter,
        priBSaccUtil Gauge,
        priBSaccDel TimeTicks,
        priBSaccCors Counter,
        priBSaccErr Gauge,
        priBSaccBbs INTEGER,
        priBSaccN INTEGER,
        priBSaccA INTERGER,
        priBSaccZiele Destination,
        priBSaccjob Jobs
    }

priBSaccTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priBSaccessRecordTable."
```

```
::= { priBSaccessRecordEntry 1 }


priBSaccName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This is the name of the
                managed object.  It also contains the
                name of the coresponding queue."
::= { priBSaccessRecordEntry 2 }

priBSaccReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { priBSaccessRecordEntry 3 }

priBSaccAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { priBSaccessRecordEntry 4 }

priBSaccRej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { priBSaccessRecordEntry 5 }

priBSaccDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { priBSaccessRecordEntry 6 }

priBSaccUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
```

```
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
                included in this object."
::= { priBSaccessRecordEntry 7 }

priBSaccDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { priBSaccessRecordEntry 8 }

priBSaccCors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { priBSaccessRecordEntry 9 }

priBSaccErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { priBSaccessRecordEntry 10 }

priBSaccBbs OBJECT-TYPE
        SYNTAX INTEGER {
                        enabled(1)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of
                the object 'access'.  This object can always
                assume jobs, therefore the value ist 'enabled'."
        DEFVAL { 1 }
::= { priBSaccessRecordEntry 11}

priBSaccN OBJECT-TYPE
        SYNTAX INTEGER {
                        idle(1),
```

```
                    active(2)
                    }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The use of the managed object.  But this
                object has no limit for the number of jobs."
    ::= { priBSaccessRecordEntry 12}

    priBSaccA OBJECT-TYPE
        SYNTAX INTEGER {
                    not-applicable(1)
                    }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "It's not possible to administrate
                this object."
        DEFVAL { 1 }
    ::= { priBSaccessRecordEntry 13}

    priBSaccZiele OBJECT-TYPE
        SYNTAX Destination
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to objects, who will get the job next,
                specified."
        DEFVAL {priBSaccessNTfollowRecordEntry}
    ::= { priBSaccessRecordEntry 14}

    priBSaccjob OBJECT-TYPE
        SYNTAX Jobs
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to all jobs included in this object."
        DEFVAL {priBSaccessNTjobsRecordEntry}
    ::= { priBSaccessRecordEntry 15}


-- Nebentabelle Auftraege
-- ****************************************************

    priBSaccessNTjobsRecordTable OBJECT-TYPE
```

```
        SYNTAX SEQUENCE OF PriBSaccessNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to all jobs
                currently being proccessed in this object."
::= { bsZugangskontrolle 2 }

priBSaccessNTjobsRecordEntry OBJECT-TYPE
        SYNTAX PriBSaccessNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSaccessNTjobsRecordTable."
        INDEX { priBSaccTableIndex, priBSaccNTjobsTableIndex }
::= { priBSaccessNTjobsRecordTable 1 }

PriBSaccessNTjobsRecordEntry ::= SEQUENCE
   {
        priBSaccNTjobsTableIndex TableIndex,
        priBSaccNTjobsJob InstancePointer
   }

priBSaccNTjobsTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is the second index of the
                priBSaccessRecordTable."
::= { priBSaccessNTjobsRecordEntry 1 }


priBSaccNTjobsJob OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular job
                in the jobtable."
::= { priBSaccessNTjobsRecordEntry 2 }



-- Nebentabelle Ziele
-- ****************************************************
```

```
priBSaccessNTfollowRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSaccessNTfollowRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to objects,
                where the job has to be send, after he is
                finished in this object."
::= { bsZugangskontrolle 3 }

priBSaccessNTfollowRecordEntry OBJECT-TYPE
        SYNTAX PriBSaccessNTfollowRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSaccessNTjobsRecordTable."
        INDEX { priBSaccTableIndex }
::= { priBSaccessNTfollowRecordTable 1 }

PriBSaccessNTfollowRecordEntry ::= SEQUENCE
    {
        priBSaccNTfollowobject InstancePointer
    }

priBSaccNTfollowobject OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular object."
::= { priBSaccessNTfollowRecordEntry 1 }
```

```
-- ********************************************************
-- BEDIENSTATION Praefilter
-- ********************************************************

-- Klassentabelle fuer Praefilter

    priBSprefilterRecordTable OBJECT-TYPE
```

```
        SYNTAX SEQUENCE OF PriBSprefilterRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This managed object is used to represent
                information about the preproccessing of
                printingdata."
::= { bsPraefilter 1 }

priBSprefilterRecordEntry OBJECT-TYPE
        SYNTAX PriBSprefilterRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSprefilterRecordTable."
        INDEX { priBSpreTableIndex }
::= { priBSprefilterRecordTable 1 }

PriBSprefilterRecordEntry ::= SEQUENCE
    {
        priBSpreTableIndex TableIndex,
        priBSpreName DisplayString,
        priBSpreReqs Counter,
        priBSpreAccs Counter,
        priBSpreRej Gauge,
        priBSpreDeps Counter,
        priBSpreUtil Gauge,
        priBSpreDel TimeTicks,
        priBSpreCors Counter,
        priBSpreErr Gauge,
        priBSpreBbs INTEGER,
        priBSpreN INTEGER,
        priBSpreA INTERGER,
        priBSpreZiele Destination,
        priBSprejob Jobs
    }

priBSpreTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priBSprefilterRecordTable."
::= { priBSprefilterRecordEntry 1 }
```

```
priBSpreName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This is the name of the
                managed object."
::= { priBSprefilterRecordEntry 2 }

priBSpreReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { priBSprefilterRecordEntry 3 }

priBSpreAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { priBSprefilterRecordEntry 4 }

priBSpreRej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { priBSprefilterRecordEntry 5 }

priBSpreDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { priBSprefilterRecordEntry 6 }

priBSpreUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
```

```
                included in this object."
::= { priBSprefilterRecordEntry 7 }

priBSpreDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { priBSprefilterRecordEntry 8 }

priBSpreCors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { priBSprefilterRecordEntry 9 }

priBSpreErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { priBSprefilterRecordEntry 10 }

priBSpreBbs OBJECT-TYPE
        SYNTAX INTEGER {
                        enabled(1)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of this
                object.  The usage of this object is always
                guaranteed."
        DEFVAL { 1 }
::= { priBSprefilterRecordEntry 11}

priBSpreN OBJECT-TYPE
        SYNTAX INTEGER {
                        idle(1),
                        active(2)
                        }
```

```
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The use of the managed object.  The number
                of jobs in this object is not limited.  For that
                reason only 'active' and 'idle' are possible."
::= { priBSprefilterRecordEntry 12}

priBSpreA OBJECT-TYPE
        SYNTAX INTEGER {
                        not-applicable(1)
                        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "It's not possible to administrate this
                object."
        DEFVAL { 1 }
::= { priBSprefilterRecordEntry 13}

priBSpreZiele OBJECT-TYPE
        SYNTAX Destination
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to objects, who will get the job next,
                specified."
        DEFVAL {priBSprefilterNTfollowRecordEntry}
::= { priBSprefilterRecordEntry 14}

priBSprejob OBJECT-TYPE
        SYNTAX Jobs
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to all jobs included in this object."
        DEFVAL {priBSprefilterNTjobsRecordEntry}
::= { priBSprefilterRecordEntry 15}


-- Nebentabelle Auftraege
-- *******************************************************

priBSprefilterNTjobsRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSprefilterNTjobsRecordEntry
```

```
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to all jobs
                currently being proccessed in this object."
::= { bsPraefilter 2 }

priBSprefilterNTjobsRecordEntry OBJECT-TYPE
        SYNTAX PriBSprefilterNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSprefilterNTjobsRecordTable."
        INDEX { priBSpreTableIndex, priBSpreNTjobsTableIndex }
::= { priBSprefilterNTjobsRecordTable 1 }

PriBSprefilterNTjobsRecordEntry ::= SEQUENCE
    {
        priBSpreNTjobsTableIndex TableIndex,
        priBSpreNTjobsJob InstancePointer
    }

priBSpreNTjobsTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is the second index of the
                priBSprefilterRecordTable."
::= { priBSprefilterNTjobsRecordEntry 1 }


priBSpreNTjobsJob OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular job
                in the jobtable."
::= { priBSprefilterNTjobsRecordEntry 2 }



-- Nebentabelle Ziele
-- ****************************************************
```

```
priBSprefilterNTfollowRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSprefilterNTfollowRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to objects,
                where the job has to be send, after he is
                finished in this object."
::= { bsPraefilter 3 }

priBSprefilterNTfollowRecordEntry OBJECT-TYPE
        SYNTAX PriBSprefilterNTfollowRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSprefilterNTjobsRecordTable."
        INDEX { priBSpreTableIndex }
::= { priBSprefilterNTfollowRecordTable 1 }

PriBSprefilterNTfollowRecordEntry ::= SEQUENCE
   {
        priBSpreNTfollowobject InstancePointer
   }

priBSpreNTfollowobject OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular object."
::= { priBSprefilterNTfollowRecordEntry 1 }




-- *********************************************************
-- BEDIENSTATION Postfilter
-- *********************************************************

-- Klassentabelle fuer Postfilter

priBSpostfilterRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSpostfilterRecordEntry
```

```
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This managed object is used to represent
                information about the postproccessing of
                printingdata."
::= { bsPostfilter 1 }

priBSpostfilterRecordEntry OBJECT-TYPE
        SYNTAX PriBSpostfilterRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSpostfilterRecordTable."
        INDEX { priBSpostTableIndex }
::= { priBSpostfilterRecordTable 1 }

PriBSpostfilterRecordEntry ::= SEQUENCE
    {
        priBSpostTableIndex TableIndex,
        priBSpostName DisplayString,
        priBSpostReqs Counter,
        priBSpostAccs Counter,
        priBSpostRej Gauge,
        priBSpostDeps Counter,
        priBSpostUtil Gauge,
        priBSpostDel TimeTicks,
        priBSpostCors Counter,
        priBSpostErr Gauge,
        priBSpostBbs INTEGER,
        priBSpostN INTEGER,
        priBSpostA INTERGER,
        priBSpostZiele Destination,
        priBSpostjob Jobs
    }

priBSpostTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priBSpostfilterRecordTable."
::= { priBSpostfilterRecordEntry 1 }
```

```
priBSpostName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This is the name of the
                managed object."
::= { priBSpostfilterRecordEntry 2 }

priBSpostReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { priBSpostfilterRecordEntry 3 }

priBSpostAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { priBSpostfilterRecordEntry 4 }

priBSpostRej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { priBSpostfilterRecordEntry 5 }

priBSpostDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { priBSpostfilterRecordEntry 6 }

priBSpostUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
                included in this object."
```

```
::= { priBSpostfilterRecordEntry 7 }

priBSpostDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { priBSpostfilterRecordEntry 8 }

priBSpostCors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { priBSpostfilterRecordEntry 9 }

priBSpostErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { priBSpostfilterRecordEntry 10 }

priBSpostBbs OBJECT-TYPE
        SYNTAX INTEGER {
                        enabled(1),
                        disabled(2)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of this
                object.  If a printserver is active, the value
                is 'enabled'.  Because this object only get
                jobs, if a printserver exists."
::= { priBSpostfilterRecordEntry 11}

priBSpostN OBJECT-TYPE
        SYNTAX INTEGER {
                        idle(1),
                        active(2)
                        }
```

```
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The use of the managed object.  The number
                of jobs in this object is not limited.  For that
                reason only 'active' and 'idle' are possible."
::= { priBSpostfilterRecordEntry 12}

priBSpostA OBJECT-TYPE
        SYNTAX INTEGER {
                        unlocked(4),
                        shutting-down(5)
                        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "The desired state of this object.  If jobs
                from the queue shut be processed, this value
                must be 'unlocked' else jobs from the queue are
                not accepted."
::= { priBSpostfilterRecordEntry 13}

priBSpostZiele OBJECT-TYPE
        SYNTAX Destination
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to objects, who will get the job next,
                specified."
        DEFVAL {priBSpostfilterNTfollowRecordEntry}
::= { priBSpostfilterRecordEntry 14}

priBSpostjob OBJECT-TYPE
        SYNTAX Jobs
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to all jobs included in this object."
        DEFVAL {priBSpostfilterNTjobsRecordEntry}
::= { priBSpostfilterRecordEntry 15}


-- Nebentabelle Auftraege
-- *******************************************************
```

```
priBSpostfilterNTjobsRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSpostfilterNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to all jobs
                currently being proccessed in this object."
::= { bsPostfilter 2 }

priBSpostfilterNTjobsRecordEntry OBJECT-TYPE
        SYNTAX PriBSpostfilterNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSpostfilterNTjobsRecordTable."
        INDEX { priBSpostTableIndex, priBSpostNTjobsTableIndex }
::= { priBSpostfilterNTjobsRecordTable 1 }

PriBSpostfilterNTjobsRecordEntry ::= SEQUENCE
    {
        priBSpostNTjobsTableIndex TableIndex,
        priBSpostNTjobsJob InstancePointer
    }

priBSpostNTjobsTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is the second index of the
                priBSpostfilterRecordTable."
::= { priBSpostfilterNTjobsRecordEntry 1 }


priBSpostNTjobsJob OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular job
                in the jobtable."
::= { priBSpostfilterNTjobsRecordEntry 2 }



-- Nebentabelle Ziele
```

```
-- *****************************************************

   priBSpostfilterNTfollowRecordTable OBJECT-TYPE
           SYNTAX SEQUENCE OF PriBSpostfilterNTfollowRecordEntry
           ACCESS not-accessible
           STATUS mandatory
           DESCRIPTION "This table contains links to objects,
                   where the job has to be send, after he is
                   finished in this object."
   ::= { bsPostfilter 3 }

   priBSpostfilterNTfollowRecordEntry OBJECT-TYPE
           SYNTAX PriBSpostfilterNTfollowRecordEntry
           ACCESS not-accessible
           STATUS mandatory
           DESCRIPTION "This represents an entry in the
                   priBSpostfilterNTjobsRecordTable."
           INDEX { priBSpostTableIndex }
   ::= { priBSpostfilterNTfollowRecordTable 1 }

   PriBSpostfilterNTfollowRecordEntry ::= SEQUENCE
      {
           priBSpostNTfollowobject InstancePointer
      }

   priBSpostNTfollowobject OBJECT-TYPE
           SYNTAX InstancePointer
           ACCESS read-only
           STATUS mandatory
           DESCRIPTION "This pointer points to a particular object."
   ::= { priBSpostfilterNTfollowRecordEntry 1 }
```

```
-- *********************************************************
-- BEDIENSTATION Drucker
```

```
-- *********************************************************

-- Klassentabelle fuer Drucker

    priBSprinterRecordTable OBJECT-TYPE
            SYNTAX SEQUENCE OF PriBSprinterRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This managed object is used to represent
                    information about the printer station."
    ::= { bsDrucker 1 }

    priBSprinterRecordEntry OBJECT-TYPE
            SYNTAX PriBSprinterRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This represents an entry in the
                    priBSprinterRecordTable."
            INDEX { priBSpriTableIndex }
    ::= { priBSprinterRecordTable 1 }

    PriBSprinterRecordEntry ::= SEQUENCE
        {
            priBSpriTableIndex TableIndex,
            priBSpriName DisplayString,
            priBSpriReqs Counter,
            priBSpriAccs Counter,
            priBSpriRej Gauge,
            priBSpriDeps Counter,
            priBSpriUtil Gauge,
            priBSpriDel TimeTicks,
            priBSpriCors Counter,
            priBSpriErr Gauge,
            priBSpriBbs INTEGER,
            priBSpriN INTEGER,
            priBSpriA INTERGER,
            priBSpriZiele Destination,
            priBSprijob Jobs,
            priBSpricontain Containment
        }

    priBSpriTableIndex OBJECT-TYPE
            SYNTAX TableIndex
```

```
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priBSprinterRecordTable."
::= { priBSprinterRecordEntry 1 }


priBSpriName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This is the name of the
                managed object."
::= { priBSprinterRecordEntry 2 }

priBSpriReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { priBSprinterRecordEntry 3 }

priBSpriAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { priBSprinterRecordEntry 4 }

priBSpriRej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { priBSprinterRecordEntry 5 }

priBSpriDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { priBSprinterRecordEntry 6 }
```

```
priBSpriUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
                included in this object."
::= { priBSprinterRecordEntry 7 }

priBSpriDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { priBSprinterRecordEntry 8 }

priBSpriCors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { priBSprinterRecordEntry 9 }

priBSpriErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { priBSprinterRecordEntry 10 }

priBSpriBbs OBJECT-TYPE
        SYNTAX INTEGER {
                        enabled(1),
                        disabled(2)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of
                the printers.  If a connection at least
                to one printer-device can be made the
                station is 'enabled'."
::= { priBSprinterRecordEntry 11}
```

```
priBSpriN OBJECT-TYPE
        SYNTAX INTEGER {
                        idle(1),
                        active(2),
                        busy(4)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The use of the managed object.  If all
                printer-devices are occupied, no more new jobs
                can be assumed.  But if there are more
                printer-devices available perhaps no all are
                occupied."
::= { priBSprinterRecordEntry 12}

priBSpriA OBJECT-TYPE
        SYNTAX INTEGER {
                        not-applicable(1)
                        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "It's not possible to administrate
                this object."
::= { priBSprinterRecordEntry 13}

priBSpriZiele OBJECT-TYPE
        SYNTAX Destination
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to objects, who will get the job next,
                specified."
        DEFVAL {priBSprinterNTfollowRecordEntry}
::= { priBSprinterRecordEntry 14}

priBSprijob OBJECT-TYPE
        SYNTAX Jobs
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to all jobs included in this object."
        DEFVAL {priBSprinterNTjobsRecordEntry}
```

```
    ::= { priBSprinterRecordEntry 15}

    priBSpricontain OBJECT-TYPE
            SYNTAX Containment
            ACCESS read-only
            STATUS mandatory
            DESCRIPTION "A pointer to a sidetable.  In the sidetable
                    are links to all printers included in this object."
            DEFVAL {priBSprinterNTcontainRecordEntry}
    ::= { priBSprinterRecordEntry 16}


-- Nebentabelle Auftraege
-- ***************************************************

    priBSprinterNTjobsRecordTable OBJECT-TYPE
            SYNTAX SEQUENCE OF PriBSprinterNTjobsRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This table contains links to all jobs
                    currently being proccessed in this object."
    ::= { bsDrucker 2 }

    priBSprinterNTjobsRecordEntry OBJECT-TYPE
            SYNTAX PriBSprinterNTjobsRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This represents an entry in the
                    priBSprinterNTjobsRecordTable."
            INDEX { priBSpriTableIndex, priBSpriNTjobsTableIndex }
    ::= { priBSprinterNTjobsRecordTable 1 }

    PriBSprinterNTjobsRecordEntry ::= SEQUENCE
        {
            priBSpriNTjobsTableIndex TableIndex,
            priBSpriNTjobsJob InstancePointer
        }

    priBSpriNTjobsTableIndex OBJECT-TYPE
            SYNTAX TableIndex
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This is the second index of the
```

```
                priBSprinterRecordTable."
    ::= { priBSprinterNTjobsRecordEntry 1 }


    priBSpriNTjobsJob OBJECT-TYPE
            SYNTAX InstancePointer
            ACCESS read-only
            STATUS mandatory
            DESCRIPTION "This pointer points to a particular job
                in the jobtable."
    ::= { priBSprinterNTjobsRecordEntry 2 }



-- Nebentabelle Ziele
-- ****************************************************

    priBSprinterNTfollowRecordTable OBJECT-TYPE
            SYNTAX SEQUENCE OF PriBSprinterNTfollowRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This table contains links to objects,
                where the job has to be send, after he is
                finished in this object."
    ::= { bsDrucker 3 }

    priBSprinterNTfollowRecordEntry OBJECT-TYPE
            SYNTAX PriBSprinterNTfollowRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This represents an entry in the
                priBSprinterNTjobsRecordTable."
            INDEX { priBSpriTableIndex }
    ::= { priBSprinterNTfollowRecordTable 1 }

    PriBSprinterNTfollowRecordEntry ::= SEQUENCE
        {
            priBSpriNTfollowobject InstancePointer
        }

    priBSpriNTfollowobject OBJECT-TYPE
            SYNTAX InstancePointer
            ACCESS read-only
```

```
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular object."
    ::= { priBSprinterNTfollowRecordEntry 1 }


-- Nebentabelle Bedieneinheiten
-- ****************************************************

    priBSprinterNTunitsRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBSprinterNTunitsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to all units
                contained in this object."
    ::= { bsDrucker 4 }

    priBSprinterNTunitsRecordEntry OBJECT-TYPE
        SYNTAX PriBSprinterNTunitsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBSprinterNTunitsRecordTable."
        INDEX { priBSpriTableIndex, priBSpriNTunitsTableIndex }
    ::= { priBSprinterNTunitsRecordTable 1 }

    PriBSprinterNTunitsRecordEntry ::= SEQUENCE
      {
        priBSpriNTunitsTableIndex TableIndex,
        priBSpriNTunitsUnit InstancePointer
      }

    priBSpriNTunitsTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is the second index of the
                priBSprinterRecordTable."
    ::= { priBSprinterNTunitsRecordEntry 1 }


    priBSpriNTunitsUnit OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
```

```
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular printer
                in the printer-device table."
    ::= { priBSprinterNTunitsRecordEntry 2 }



-- ********************************************************
-- BEDIENEINHEIT Drucker
-- ********************************************************

-- Klassentabelle fuer Drucker

    priBEprinterRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBEprinterRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This managed object is used to represent
                information about the printer station."
    ::= { beDrucker 1 }

    priBEprinterRecordEntry OBJECT-TYPE
        SYNTAX PriBEprinterRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBEprinterRecordTable."
        INDEX { priBEpriTableIndex }
    ::= { priBEprinterRecordTable 1 }

    PriBEprinterRecordEntry ::= SEQUENCE
      {
        priBEpriTableIndex TableIndex,
        priBEpriName DisplayString,
        priBEpriReqs Counter,
        priBEpriAccs Counter,
        priBEpriRej Gauge,
        priBEpriDeps Counter,
        priBEpriUtil Gauge,
        priBEpriDel TimeTicks,
        priBEpriCors Counter,
        priBEpriErr Gauge,
        priBEpriBbs INTEGER,
```

```
        priBEpriN INTEGER,
        priBEpriA INTERGER,
        priBEpriPlace DisplayString,
        priBEpriState DisplayString,
        priBEpriConnect INTEGER,
        priBEprijob Jobs,
        priBEpriParent Parent
    }

priBEpriTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priBEprinterRecordTable."
::= { priBEprinterRecordEntry 1 }


priBEpriName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This is the name of the
                managed object."
::= { priBEprinterRecordEntry 2 }

priBEpriReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { priBEprinterRecordEntry 3 }

priBEpriAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { priBEprinterRecordEntry 4 }

priBEpriRej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
```

```
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { priBEprinterRecordEntry 5 }

priBEpriDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { priBEprinterRecordEntry 6 }

priBEpriUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
                included in this object."
::= { priBEprinterRecordEntry 7 }

priBEpriDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { priBEprinterRecordEntry 8 }

priBEpriCors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { priBEprinterRecordEntry 9 }

priBEpriErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { priBEprinterRecordEntry 10 }

priBEpriBbs OBJECT-TYPE
```

```
        SYNTAX INTEGER {
                        enabled(1),
                        disabled(2),
                        unkown(3)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of
                the printer.  Only if it's possible to ask the
                printer about his state this variable can
                be offerd."
::= { priBEprinterRecordEntry 11}

priBEpriN OBJECT-TYPE
        SYNTAX INTEGER {
                        idle(1),
                        busy(4)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The use of the managed object.  A printer-
                device can only handle one job at the same time."
::= { priBEprinterRecordEntry 12}

priBEpriA OBJECT-TYPE
        SYNTAX INTEGER {
                        locked(2),
                        unlocked(4),
                        shutting-down(5)
                        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "The desired state of the printer-device.  To
                stop the printer immediately, 'locked' is used.  To
                stop the printer after he has finished the actual
                job, 'shutting down'."
::= { priBEprinterRecordEntry 13}

priBEpriPlace OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The physical location of the printer."
```

```
        ::= { priBEprinterRecordEntry 14}

        priBEpriState OBJECT-TYPE
                SYNTAX DisplayString
                ACCESS read-only
                STATUS mandatory
                DESCRIPTION "This value is taken from the
printerstatefile
                        in the queuedirectory."
        ::= { priBEprinterRecordEntry 15}

        priBEpriConnect OBJECT-TYPE
                SYNTAX INTEGER {
                                serial(1),
                                network(2),
                                other-queue(3)
                                }
                ACCESS read-only
                STATUS mandatory
                DESCRIPTION "There are serveral possibilities for
                        printjobs to be proccessed.  The job can be
                        moved into another queue, printed at the
                        printer by this printsystem or by another
                        printsystem."
        ::= { priBEprinterRecordEntry 16}

        priBEprijob OBJECT-TYPE
                SYNTAX Jobs
                ACCESS read-only
                STATUS mandatory
                DESCRIPTION "A pointer to a sidetable.  In the sidetable
                        are links to all jobs included in this object."
                DEFVAL {priBEprinterNTjobsRecordEntry}
        ::= { priBEprinterRecordEntry 17}

        priBEpriParent OBJECT-TYPE
                SYNTAX Parent
                ACCESS read-only
                STATUS mandatory
                DESCRIPTION "A pointer to a sidetable.  In the sidetable
                        are a link to an objects who include this object."
                DEFVAL {priBEprinterNTjobsRecordEntry}
        ::= { priBEprinterRecordEntry 18}
```

```
-- Nebentabelle Jobs
-- ****************************************************

    priBEprinterNTjobsRecordTable OBJECT-TYPE
            SYNTAX SEQUENCE OF PriBEprinterNTjobsRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This table contains links to all jobs
                    currently being proccessed in this object."
    ::= { bsDrucker 2 }

    priBEprinterNTjobsRecordEntry OBJECT-TYPE
            SYNTAX PriBEprinterNTjobsRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This represents an entry in the
                    priBEprinterNTjobsRecordTable."
            INDEX { priBEpriTableIndex }
    ::= { priBEprinterNTjobsRecordTable 1 }

    PriBEprinterNTjobsRecordEntry ::= SEQUENCE
        {
            priBEpriNTjobsobject InstancePointer
        }

    priBEpriNTjobsobject OBJECT-TYPE
            SYNTAX InstancePointer
            ACCESS read-only
            STATUS mandatory
            DESCRIPTION "This pointer points to a particular job
                    in the jobtable."
    ::= { priBEprinterNTjobsRecordEntry 1 }




-- Nebentabelle Parent
-- ****************************************************
```

```
priBEprinterNTparentRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriBEprinterNTparentRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to all objects
                who include this object."
::= { bsDrucker 3 }

priBEprinterNTparentRecordEntry OBJECT-TYPE
        SYNTAX PriBEprinterNTparentRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priBEprinterNTparentRecordTable."
        INDEX { priBEpriTableIndex }
::= { priBEprinterNTparentRecordTable 1 }

PriBEprinterNTparentRecordEntry ::= SEQUENCE
    {
        priBEpriNTparentobject InstancePointer
    }

priBEpriNTparentobject OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular object."
::= { priBEprinterNTparentRecordEntry 1 }


-- *******************************************************
-- WARTESCHLANGE
-- *******************************************************

-- Klassentabelle fuer Warteschlange

priWSRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriWSRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This managed object is used to represent
                information about the printer station."
```

```
::= { warteschlange 1 }

priWSRecordEntry OBJECT-TYPE
        SYNTAX PriWSRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priWSRecordTable."
        INDEX { priWSTableIndex }
::= { priWSRecordTable 1 }

PriWSRecordEntry ::= SEQUENCE
    {
        priWSTableIndex TableIndex,
        priWSName DisplayString,
        priWSReqs Counter,
        priWSAccs Counter,
        priWSRej Gauge,
        priWSDeps Counter,
        priWSUtil Gauge,
        priWSDel TimeTicks,
        priWSCors Counter,
        priWSErr Gauge,
        priWSBbs INTEGER,
        priWSN INTEGER,
        priWSA INTERGER,
        priWSfollow Destination,
        priWSjob Jobs
    }

priWSTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priWSRecordTable."
::= { priWSRecordEntry 1 }


priWSName OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
```

```
        DESCRIPTION "This is the name of the
                managed object."
::= { priWSRecordEntry 2 }

priWSReqs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of arrived jobs."
::= { priWSRecordEntry 3 }

priWSAccs OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of accepted jobs."
::= { priWSRecordEntry 4 }

priWSRej OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of rejected jobs."
::= { priWSRecordEntry 5 }

priWSDeps OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of depended jobs."
::= { priWSRecordEntry 6 }

priWSUtil OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs actually
                included in this object."
::= { priWSRecordEntry 7 }

priWSDel OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
```

```
        STATUS mandatory
        DESCRIPTION "The delay of the latest job
                finished in this object."
::= { priWSRecordEntry 8 }

priWSCors OBJECT-TYPE
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of correct finished jobs."
::= { priWSRecordEntry 9 }

priWSErr OBJECT-TYPE
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The number of jobs finished with
                an error."
::= { priWSRecordEntry 10 }

priWSBbs OBJECT-TYPE
        SYNTAX INTEGER {
                        enabled(1),
                        disabled(2)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The current operational state of
                the queue.  The queue is 'enabled' if the
                directory of the queue is accessible."
::= { priWSRecordEntry 11}

priWSN OBJECT-TYPE
        SYNTAX INTEGER {
                        idle(1),
                        active(2)
                        }
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The use of the managed object.  If the
                queue is empty, the value 'idle' is displayed."
::= { priWSRecordEntry 12}
```

```
priWSA OBJECT-TYPE
        SYNTAX INTEGER {
                        locked(2),
                        unlocked(4)
                        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "The desired state of the queue.  It's
                possible to lock the queue that no more new
                jobs will arrive at the queue."
::= { priWSRecordEntry 13}

priWSfollow OBJECT-TYPE
        SYNTAX Destination
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to objects, who will get the job next,
                specified."
        DEFVAL {priWSNTfollowRecordEntry}
::= { priWSRecordEntry 14}

priWSjob OBJECT-TYPE
        SYNTAX Jobs
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A pointer to a sidetable.  In the sidetable
                are links to all jobs included in this object."
        DEFVAL {priWSNTjobsRecordEntry}
::= { priWSRecordEntry 15}



-- Nebentabelle Auftraege
-- ************************************************************

priWSNTjobsRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriWSNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This table contains links to all jobs
                currently being proccessed in this object."
::= { warteschlange 2 }
```

```
priWSNTjobsRecordEntry OBJECT-TYPE
        SYNTAX PriWSNTjobsRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priWSNTjobsRecordTable."
        INDEX { priWSTableIndex, priWSNTjobsTableIndex }
::= { priWSNTjobsRecordTable 1 }

PriWSNTjobsRecordEntry ::= SEQUENCE
    {
        priWSNTjobsTableIndex TableIndex,
        priWSNTjobsJob InstancePointer
    }

priWSNTjobsTableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is the second index of the
                priWSRecordTable."
::= { priWSNTjobsRecordEntry 1 }


priWSNTjobsJob OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular job
                in the jobtable."
::= { priWSNTjobsRecordEntry 2 }




-- Nebentabelle Ziele
-- ******************************************************

priWSNTfollowRecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriWSNTfollowRecordEntry
        ACCESS not-accessible
        STATUS mandatory
```

```
        DESCRIPTION "This table contains links to objects,
                where the job has to be send, after he is
                finished in this object."
    ::= { warteschlange 3 }

    priWSNTfollowRecordEntry OBJECT-TYPE
        SYNTAX PriWSNTfollowRecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priWSNTjobsRecordTable."
        INDEX { priWSTableIndex }
    ::= { priWSNTfollowRecordTable 1 }

    PriWSNTfollowRecordEntry ::= SEQUENCE
      {
        priWSNTfollowobject InstancePointer
      }

    priWSNTfollowobject OBJECT-TYPE
        SYNTAX InstancePointer
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "This pointer points to a particular object."
    ::= { priWSNTfollowRecordEntry 1 }


-- ********************************************************
-- AUFTRAG
-- ********************************************************

-- Klassentabelle fuer Auftrag

    priARecordTable OBJECT-TYPE
        SYNTAX SEQUENCE OF PriARecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This managed object is used to represent
                information about the printer station."
    ::= { auftrag 1 }

    priARecordEntry OBJECT-TYPE
```

```
        SYNTAX PriARecordEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This represents an entry in the
                priARecordTable."
        INDEX { priATableIndex }
::= { priARecordTable 1 }

PriARecordEntry ::= SEQUENCE
   {
        priATableIndex TableIndex,
        priAName INTEGER,
        priATime TimeTicks,
        priAwsTime TimeTicks,
        priASize INTEGER,
        priAPrio INTEGER,
        priAUser DisplayString,
        priAA INTERGER,
        priAparent Parent
   }

priATableIndex OBJECT-TYPE
        SYNTAX TableIndex
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "This is an index of the
                priARecordTable."
::= { priARecordEntry 1 }


priAName OBJECT-TYPE
        SYNTAX INTEGER
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "To distinguish between the jobs in the
                printsystem a unique jobnummer is necessary."
::= { priARecordEntry 2 }

priATime OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The time, when the job arrived at the
```

```
                printsystem."
::= { priARecordEntry 3 }

priAwsTime OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The time, when the job arrived at the
                queue."
::= { priARecordEntry 4 }

priASize OBJECT-TYPE
        SYNTAX INTEGER
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The size in Bytes of the data, who has to
                be printed."
::= { priARecordEntry 5 }

priAPrio OBJECT-TYPE
        SYNTAX INTEGER {
                        first(64),
                        a(65), b(66), c(67), d(68), e(69),
                        f(70), g(71), h(72), i(73), j(74),
                        k(75), l(76), m(77), n(78), o(79),
                        p(80), q(81), r(82), s(83), t(84),
                        u(85), v(86), w(87), x(88), y(89),
                        z(90),
                        last(91)
                        }
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "The priority of the job.  The priority is
                responsible for the rang of the job in the queue."
::= { priARecordEntry 6 }

priAUser OBJECT-TYPE
        SYNTAX DisplayString
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The name of the user who created this job."
::= { priARecordEntry 7 }
```

```
    priAA OBJECT-TYPE
            SYNTAX INTEGER {
                            waiting(1),
                            active(2),
                            unkown(3),
                            hold(4),
                            delete(5)
                            }
            ACCESS read-write
            STATUS mandatory
            DESCRIPTION "The desired state of the job in the queue."
    ::= { priARecordEntry 8}

    priAparent OBJECT-TYPE
            SYNTAX Parent
            ACCESS read-write
            STATUS mandatory
            DESCRIPTION "A pointer to a sidetable.  In the sidetable
                    are links to all objects, who include this job."
            DEFVAL {priANTcontainRecordEntry}
    ::= { priARecordEntry 9}



-- Nebentabelle Parent
-- *******************************************************

    priANTparentRecordTable OBJECT-TYPE
            SYNTAX SEQUENCE OF PriANTparentRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This table contains links to all objects
                    currently include this job."
    ::= { auftrag 2 }

    priANTparentRecordEntry OBJECT-TYPE
            SYNTAX PriANTparentRecordEntry
            ACCESS not-accessible
            STATUS mandatory
            DESCRIPTION "This represents an entry in the
                    priANTparentRecordTable."
            INDEX { priATableIndex, priANTparentTableIndex }
```

```
    ::= { priANTparentRecordTable 1 }

    PriANTparentRecordEntry ::= SEQUENCE
      {
          priANTparentTableIndex TableIndex,
          priANTparentJob InstancePointer
      }

    priANTparentTableIndex OBJECT-TYPE
          SYNTAX TableIndex
          ACCESS not-accessible
          STATUS mandatory
          DESCRIPTION "This is the second index of the
                priARecordTable."
    ::= { priANTparentRecordEntry 1 }


    priANTparentJob OBJECT-TYPE
          SYNTAX InstancePointer
          ACCESS read-only
          STATUS mandatory
          DESCRIPTION "This pointer points to a particular object."
    ::= { priANTparentRecordEntry 2 }


END
```

```
IIMComibtransConventions DEFINITIONS ::= BEGIN

IMPORTS
        omibtransSpt
                FROM IIMComibtransSptComplianceModule
        InstancePointer, TruthValue
                FROM SNMPv2-TC;

iimcomibtransConventions MODULE-IDENTITY
        LAST-UPDATED "9310150000Z"
        ORGANIZATION "Network Management Forum"
        CONTACT-INFO
        "AIII Subteam
        1201 Mt.  Kemble Avenue
        Morristown, NJ 07960-6628 USA
        Tel:  +1 201-425-1900
        Fax:  +1 201-425-1515
        Email:  iimc@thumper.bellcore.com"
        DESCRIPTION
        "This module contains the textual conventions
         that support IIMCOMIBTRANS:

                * Pointer

                * Complex

                * MultiComplex

                * Parent

                * Recursion

                * TableIndex"

::= { omibtransSpt 0 3 }


Pointer ::= TEXTUAL-CONVENTION
        STATUS current
        DESCRIPTION "An object following this convention, when
        present, has one of two usages:

        1) It follows the ObjectInstance textual convention.
```

```
2) It contains the (entire) name of a scalar MIB object
    or the (entire) name of a conceptual column entry.

A Pointer is always defined as a conceptual column entry.
Absence of this entry for a particular conceptual row
indicates the absence of something to point to, i.e.,
end of a pointer chain."
```
SYNTAX OBJECT IDENTIFIER

Complex ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
```
"A Complex object is an invariant read-only pointer to a
side table containing a single row of objects per entry in
the superior table.  Those objects represent translation
of syntax (i.e., CHOICE, SEQUENCE, SET) that is complex
but not multiply occuring.

An object of this type is not-accessible.  Its constant
pointer value (name of the side table) appears in its
DEFVAL clause."
```
SYNTAX OBJECT IDENTIFIER

MultiComplex ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
```
"A MultiComplex object is an invariant read-only
pointer to a side table that may contain multiple rows of
objects per entry in the superior table.  Those rows represent
translation of syntax (i.e., SEQUENCE OF or SET OF) that is
both complex and multiply occuring.

An object of this type is not-accessible.  Its constant
pointer value (name of the side table) appears in its
DEFVAL clause."
```
SYNTAX OBJECT IDENTIFIER

Parent ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
```
"This convention is used to specify the
superior class entry instance when creating a subordinate
class entry.  It may be queried to determine the superior
```

```
        class instance, if any, for an existing class instance.

        If a Parent pointer is not populated for a particular class
        entry, that object is considered to be subordinate to the
        local root, e.g., the internet system.

        The Parent pointer of a historical entry in a class table
        has the value zero (0)."
SYNTAX Pointer


Recursion ::= TEXTUAL-CONVENTION
        STATUS current
        DESCRIPTION "This convention is a refinement of the Pointer
        convention for pointing to nested recursive syntax.  One
        such column shall be defined per side table if that table
        represents syntax that is directly or indirectly recursive.
        For tables in which it is defined, one corresponding
        RecursivelyNested column shall also be defined.

        The recursion pointer points to the next (lower) level of
        nested syntax (absence of a pointer indicates end of chain).
        It may point to a different entry in the same table
        (direct recursion) or to another table (indirect recursion).
        The innermost level of syntax may be in a non-recursive side
        table, i.e., one that does not have a Recursion pointer.

        A Recursive pointer within an active or notInService row
        shall always point to a valid syntactical option that itself
        is active or notInService.  Attempt to point such a Recursive
        object to invalid syntax results in error-status of badValue
        for SNMPv1 and inconsistentValue for SNMPv2.

        When such an object is successfully changed, any syntax
        previously pointed to (directly or indirectly) is
        automatically deleted by the agent."
SYNTAX Pointer


TableIndex ::= TEXTUAL-CONVENTION
        STATUS current
        DESCRIPTION "This convention is used to identify an
        Internet columnar object that represents the index for
```

```
        a conceptual entry in a table.  It may have no particular
        meaning other than to discriminate between entries of the row.

        An object using this textual convention shall be not-accessible."
```

SYNTAX INTEGER

END

Literaturverzeichnis

[Booch 91] Grady Booch, *Objekt Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., 1991

[Clau 93] Clausen, *Objektorientiertes Programmieren*, Springer Verlag, 1993

[Cos 93] Bryan Costales, *Sendmail*, O'Reilly & Accociates, Inc., Sebastopol, Nov. 1993

[Cox 87] Brad J. Cox, Ph.D. *Objekt-Oriented Programming*, Addision-Wesley Publishing Company, 1987

[DPI API] Bert Wijnen, *DPI Version 2.0 API – Programmers Reference*, IBM International Operations, 1994

[DSIS 94] Distributed Support Information Standards Requirements Specification, Nov. 1994

[Guts 95] Markus Gutschmidt, *Ein Objekt- und Dienstmodell für ein integriertes Management systemnaher Dienste in Client/Server-Umgebung*, LMU München, 1995

[HaHa 94] R. Hauck und N. Haubelt, *Implementierung einer MIB für Systemmanagementaufgaben*, Fortgeschrittenenpraktikum, TU–München, 1994

[Hain 95] Erwin Hainzinger, *Erweiterung eines SNMPv2–Agenten um eine Schnittstelle zur Interprozeßkommunikation*, TU–München, 1995

[Hege 93] Heinz-Gerd Hegering, *Netz- und Systemmanagement*, Addison–Wesley, Bonn, Paris, 1993

[ISO 10164-2] *Information Technology — Open Systems Interconnection — Systems Management — Part 2: State Management Funktion*, ISO 10164-2, ISO/IEC, Juni 1993

[Kemp 90] Kempel, Pfander, *Praxis der objektorientierten Programmierung: Grundlagen, Entwurf und Implementierung in C auf IBM AT*, Carl Hanser Verlag München Wien, 1990

[Lab 94] Lee LaBarre, *ISO/CCITT and Internet Management Coexistence (IIMC): Translation of Internet MIB-II (RFC 1213) to ISO/CCITT GDMO MIB*, The MITRE Corporation, Bedford, Februar 1994

[New 94] Owen Newnan, *ISO/CCITT and Internet Management Coexistence (IIMC): Translation of ISO/CCITT GDMO MIBs to Internet MIBs*, US WEST Advanced Technologies, Colorado, Februar 1994

[PLP 94] Prof. Patrick Powell, *PLP - The Public Line Printer Spooler*, Electrical and Computer Engineering Dept., San Diego State University, Dezember 1994

[RFC 1157] J. Case, M. Fedor, M. Schoffstall, und J. Davin, *A Simple Network Management Protocol (SNMP)*, RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, Mai 1990

[RFC 1212] M. Rose und K. McCloghrie, *Concise MIB Definitions*, RFC 1212, Performance Systems International, Hughes LAN Systems, März 1991

[RFC 1213] M. Rose und K. McCloghrie, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC 1213, Performance Systems International, Hughes LAN Systems, März 1991

[RFC 1227] M. Rose, *SNMP MUX Protocol and MIB*, RFC 1227, Performace Systems International, Inc., Mai 1991

[RFC 1448] J. Case, K. McCloghrie, M. RosE, und S. Waldbusser, *Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1448, SNMP Research, Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993

[RFC 1592] B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, und G. Waters, *Simple Network Management Protocol Distributed Protocol Interface Version 2.0*, RFC 1592, T.J. Watson Research Center, IBM Corp., Bell Northern Research, Ltd., März 1994

[Stall 93] William Stallings, *SNMP, SNMPv2 and CMIP - The Practical Guide to Network-Management Standards*, Addison–Wesley Co., Inc., Reading, MA, 1993

Abbildungsverzeichnis