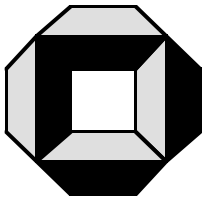

Universität Karlsruhe
Fakultät für Informatik
76128 Karlsruhe

Netzwerk-Management und Hochgeschwindigkeits-Kommunikation

Teil XII

Seminar SS 1995

Herausgeber:
Stefan Dresler
Günter Schäfer
Hajo Wiltfang

Universität Karlsruhe
Institut für Telematik

Interner Bericht 35/95

Kurzfassung

Der vorliegende interne Bericht enthält die Beiträge zum Seminar "Netzwerkmanagement und Hochgeschwindigkeitskommunikation", das im Sommersemester 1995 zum zwölften Mal abgehalten wurde. Entsprechend dem Titel ist der Band in zwei Teile gegliedert.

In dem Teil "Netzwerkmanagement" werden in den ersten beiden Kapiteln Fragen der MIB-Implementierung behandelt. Die beiden folgenden Beiträge führen in Problemstellungen der Verwaltung verteilter Anwendungen ein und stellen jeweils Forschungsergebnisse aus aktuellen Veröffentlichungen vor. Das letzte Kapitel dieses Teiles führt in das Konzept des "Network Management by Delegation" ein.

Der zweite Teil des Seminars befaßt sich mit aktuellen Fragestellungen zum Bereich „Hochgeschwindigkeitskommunikation", insbesondere zu ATM und FDDI. Die ersten beiden Beiträge widmen sich dem Einsatz von Vorwärtsfehlerkorrekturverfahren und der Diskussion zweier Signalisierungsprotokolle in ATM-Netzen. Es folgen eine Vorstellung von Mechanismen zur Bereitstellung eines verbindungslosen Dienstes sowie von Verfahren zur kreditbasierten Flußkontrolle, beide ebenfalls im Bereich ATM-Netze. Die darauffolgenden Beiträge diskutieren Mechanismen zur effizienteren Implementierung von Protokollfunktionen sowie Ansatzpunkte für ein mögliches Hardware/Software Codesign. Das Seminar schließt mit Abhandlungen über Verfahren zur Regelung der Nutzungskontrolle von Diensten in ATM-Netzen und einer Beschreibung des GIGASwitch/FDDI-Systems.

Abstract

This Technical Report includes student papers produced within small lessons called seminar of "Network Management and High Speed Communications". According to the title, the report is divided into two parts.

The first part contains two chapters dealing with questions of implementing management information bases. The next two chapters introduce problems and solutions to the field of application management. In the final contribution to the first part the new paradigm of network management by delegation is explained.

The second part is devoted to questions of high speed communication. Its first two sections discuss the usage of forward error correction methods and signalling in ATM networks. Subsequently, methods for the provision of a connectionless service and credit-based flow control are presented. They are followed by contributions on integrated layer processing and the possibility of hardware/software codesign. The seminar ends with aspects of traffic shaping in ATM networks and a presentation of the GIGASwitch/FDDI system.

Vorwort

Das Seminar "Netzwerk-Management und Hochgeschwindigkeits-Kommunikation" erfreute sich in den letzten Jahren immer größerer Beliebtheit. Gerade heutzutage sind Stichworte wie "Datenautobahn", "Breitband-ISDN" oder "Multi-Media-Kommunikation" in aller Munde. Daher sind die Forschungsgebiete in diesen Bereichen auch von allgemeinem Interesse, so daß sie eine Vielzahl an innovativen Arbeiten aufweisen können, deren Behandlung in anderen Lehrveranstaltungen so detailliert nicht möglich ist.

Jetzt liegt auch der zwölfte Seminarband als interner Bericht vor. Durch das engagierte Mitarbeiten der beteiligten Studenten kann so zumindest ein Ausschnitt aus dem komplexen und umfassenden Themengebiet übersichtlich präsentiert werden. Für den Fleiß und das Engagement der Seminaristen sei daher an dieser Stelle herzlich gedankt.

Die ausgesprochen gute Resonanz bei den Studenten hat uns veranlaßt, auch im Wintersemester 1995/1996 ein derartiges Seminar – natürlich mit geändertem Inhalt – durchzuführen, so daß bald ein weiterer interner Bericht mit neuen Forschungsergebnissen aus aktuellen Tagungsbeiträgen erscheinen wird. Doch vorerst sollen im vorliegenden Band die folgenden Themengebiete vorgestellt werden:

Die Ressourcen eines Netzwerkes werden in einem konzeptionellen Datenspeicher, der sogenannten Management Information Base, fuer das Netzwerkmanagement repräsentiert. Für die Definition von Managementinformation stehen die "Guidelines for the Definition of Managed Objects" von der ISO zur Verfügung. Der Beitrag *Implementierung der Management Information Base: Umsetzung von GDMO-Beschreibungen* führt in grundlegende Problemstellungen der MIB-Implementierung ein und stellt einen Algorithmus für die automatische Erzeugung von C++ Code aus GDMO-Definitionen vor.

Bei der MIB-Implementierung auf der Grundlage von GDMO-Beschreibungen erweist es sich als nachteilig, daß diese keine implementierungsspezifischen Details in GDMO spezifiziert werden können. Der Beitrag *Implementierung der Management Information Base: Erweiterung der GDMO-Notation um implementierungstechnische Hinweise* untersucht drei Ansätze, welche dieses Manko zu beseitigen versuchen.

Zusätzlich zum reinen Netzwerkmanagement hat sich in den vergangenen Jahren ein Bedarf an technischer Unterstützung bei der Verwaltung verteilter Anwendungen entwickelt. In dem Kapitel *Management Verteilter Anwendungen: Der Prisma Ansatz* wird die Notwendigkeit einer solchen Unterstützung motiviert und ein aktueller Ansatz auf diesem Gebiet vorgestellt.

Auch der Beitrag *Management Verteilter Anwendugen: Vergleich zweier Ansätze* behandelt Fragen des Anwendungsmanagements und vergleicht hierzu zwei weitere Ansätze miteinander.

An bestehenden Netzwerkmanagement-Architekturen wird oft kritisiert, daß sie die Aufgaben einseitig zu Lasten der reinen Managementanwendungen verteilen und die Agenten, welche die Managementanwendungen unterstützen sollen, nur für primitive Abfragen und Aufträge nutzen. In dem Kapitel *Network Management by Delegation* wird ein Ansatz beschrieben, welcher diesen Nachteilen durch das Paradigma der Delegation begegnet.

Für die Erbringung eines Dienstes zur Übertragung von Audio- oder Video-Daten kann oft nicht auf Übertragungswiederholung (ARQ) zurückgegriffen werden. Um dennoch akzeptable Fehlerraten zu erhalten, bietet sich in vielen Fällen die Verwendung von fehlerkorrigierenden Verfahren auf der Basis hinzugefügter Redundanz an. Der Abschnitt *Vorwärtsfehlerkorrektur in ATM-Netzen* ist der Beschreibung geeigneter Verfahren dazu gewidmet.

Zum Aufbau von Verbindungen in ATM-Netzen gibt es verschiedene Vorschläge für geeignete Signalisierungsprotokolle. In *Signalisierung in ATM-Netzen* werden die zwei Protokolle DSS2 und CMAP beschrieben und gegenübergestellt.

ATM unterscheidet sich von klassischen LAN-Technologien wie Ethernet und Token-Ring durch die Notwendigkeit des Aufbaus von Verbindungen vor dem Datenaustausch. In vielen Fällen wird jedoch ein verbindungsloser Dienst gewünscht. Hier helfen die in *Bereitstellung eines verbindungslosen Dienstes in ATM-Netzen* vorgestellten Mechanismen.

Zur Vermeidung von Pufferüberläufen in Empfängern greift man zur sogenannten Flußkontrolle. Sie steuert die Datenmenge, die zwischen den Endsystemen ausgetauscht wird. Eine Beschreibung des Verfahrens findet sich in *Kreditbasierte Flußkontrolle in ATM-Netzen.*

Der Beitrag *Integrated Layer Processing* stellt ein Konzept vor, bei dem versucht wird, die Protokollfunktionen innerhalb eines Protokollturmes effektiver zu implementieren. Die Funktionen der einzelnen Protokollschichten werden dabei nicht getrennt, sondern zusammen in einer integrierenden Schleife ausgeführt. So lassen sich viele der aufwendigen Lade- und Speicheroperationen einsparen.

Auch der Beitrag *Ansatzpunkte für Hardware/Software Codesign in Kommunikationssystemen* befaßt sich mit Möglichkeiten zur Steigerung der Verarbeitungsleistung in Kommunikationssystemen. Der Einsatz von Hardware verspricht im allgemeinen eine deutliche Beschleunigung bei der Verarbeitung. Allerdings sollte die Hardware geeignet in die sie umgebende Software eingebettet sein. Beim Codesign wird versucht, dies durch den parallelen Entwurf von Hard- und Software sicherzustellen.

Im Beitrag *Traffic Shaping in ATM-Netzen* wird die Nutzungskontrolle am Netzzugang von ATM-Netzen untersucht. Dabei werden neben den bekannten Verfahren zur Nutzungskontrolle vor allem Verfahren zur Regelung des Datenstromes auf Seiten des Senders vorgestellt. Das Traffic Shaping sorgt in dem Fall für die Anpassung des Anwendungsdatenstromes an die mit dem Netz ausgehandelte Verbindung.

Das GIGAswitch/FDDI-System wird im letzten Beitrag als eine hochleistungsfähige Netzwerkkomponente vorgestellt. Dieser FDDI-Switch stellt eine Brücke mit Routing-Funktionalität dar. Sein Aufbau, seine Funktionsweise und wesentliche Merkmale seiner Implementierung werden erläutert.

Karlsruhe, im Juli 1995

Stefan Dresler, Günter Schäfer und Hajo Wiltfang

Inhaltsverzeichnis

I Netzwerkmanagement

II Hochgeschwindigkeitskommunikation

III Anhang

Teil I

Netzwerkmanagement

Implementierung der Management Information Base: Umsetzung von GDMO-Beschreibungen

Joachim Elstner

Kurzfassung

Im OSI Management Modell werden die relevanten Informationen in einem konzeptuellen Datenspeicher, der *Management Information Base (MIB)*, aufbewahrt. Die Spezifikation der Datentypen erfolgt hierbei aufgrund der *Guidelines for the Definition of Managed Objects (GDMO)*. Die ISO-Norm befaßt sich zwar mit der abstrakten Beschreibung einer MIB durch GDMO, aber nicht mit deren Implementierung.

Aus diesem Grund beschäftigt sich der Beitrag mit der Darstellung und Speicherung der Informationen, sowie mit der Partitionierung einer MIB und den damit verbundenen Problemen der Datenintegrität, Wartung und Datensicherheit. Zur Entlastung des Programmierers bei einer Implementierung wird im Anschluß ein Algorithmus zur Umsetzung von GDMO-Beschreibungen in C++-Klassen vorgestellt. Dabei zeigt es sich, daß erstens nur Prototypen für die Klassen generiert werden können und zweitens aufgrund gewisser Eigenschaften der Beschreibungssprache keine vollständige Transformation möglich ist.

1 Einleitung

Relativ früh wurde die Notwendigkeit eines Netzwerkmanagements erkannt. Deshalb entwickelte die ISO im Rahmen von OSI *ein Managementmodell*, das einer Managementanwendung eine spezifizierte Schnittstelle zum Zugriff auf die entsprechenden Informationen zur Verfügung stellt [Sei94b]. Dabei muß beachtet werden, daß die managementrelevanten Daten über die ganzen Netzwerkressourcen verteilt sein können.

Das Modell gliedert sich in einen *Manager* und einen oder mehrere *Agenten* (Abb. 1). Auf jeder Netzwerkkomponente, die Managementinformationen enthält, wird ein Agent instanziiert. Dieser enthält eine Datenbank, in der die Informationen der Netzwerkkomponente verwaltet werden. Die Informationen bestehen aus Attributen, Operationen und Meldungen zur Benachrichtigung der Anwendung bei bestimmten Ereignissen. Dieser Aufbau legt die Realisierung als Objekte nahe. Eine abstrakte Beschreibung der Eigenschaften eines Datenobjektes wird *Managed Object (MO)* genannt. Ein MO enthält dabei keine Angaben über die Implementierung und ist eine Instanz einer *Managed Object Class (MOC)*, die von einer anderen MOC abgeleitet sein kann.

Der Manager hat die Aufgabe, die einzelnen Datenbanken zu einer logischen Datenbank zusammenzufassen und deren Objekte einer Managementanwendung zugänglich zu machen. Dazu wird von der neuen Datenbank mittels der MOs eine abstrakte Beschreibung, die *Management Information Base (MIB)*, erstellt. Die MIB dient als Schnittstellendefinition zwischen dem Manager und der Managementanwendung. Damit der Manager auf

die einzelnen Datenobjekte zugreifen kann, muß er mit den Agenten über das Netzwerk kommunizieren.

Die Implementierung einer MIB besteht nun aus dem Entwurf von entsprechenden Modulen für die Agenten und den Manager. Dazu wird mit GDMO zuerst die MIB spezifiziert und anschließend aufgrund der Spezifikation in einer gängigen Programmiersprache realisiert. Man wünscht sich eine möglichst weitestgehend automatische Umsetzung der GDMO-Beschreibung, um den Aufwand zu reduzieren und Fehler zu vermeiden. Im Abschnitt 3.2 wird dazu ein Algorithmus vorgestellt, nachdem zuvor einige grundlegende Aspekte der Implementierung besprochen worden sind. Die Grundlage des Beitrags bilden die beiden Arbeiten [Bap91] und [Sou94].

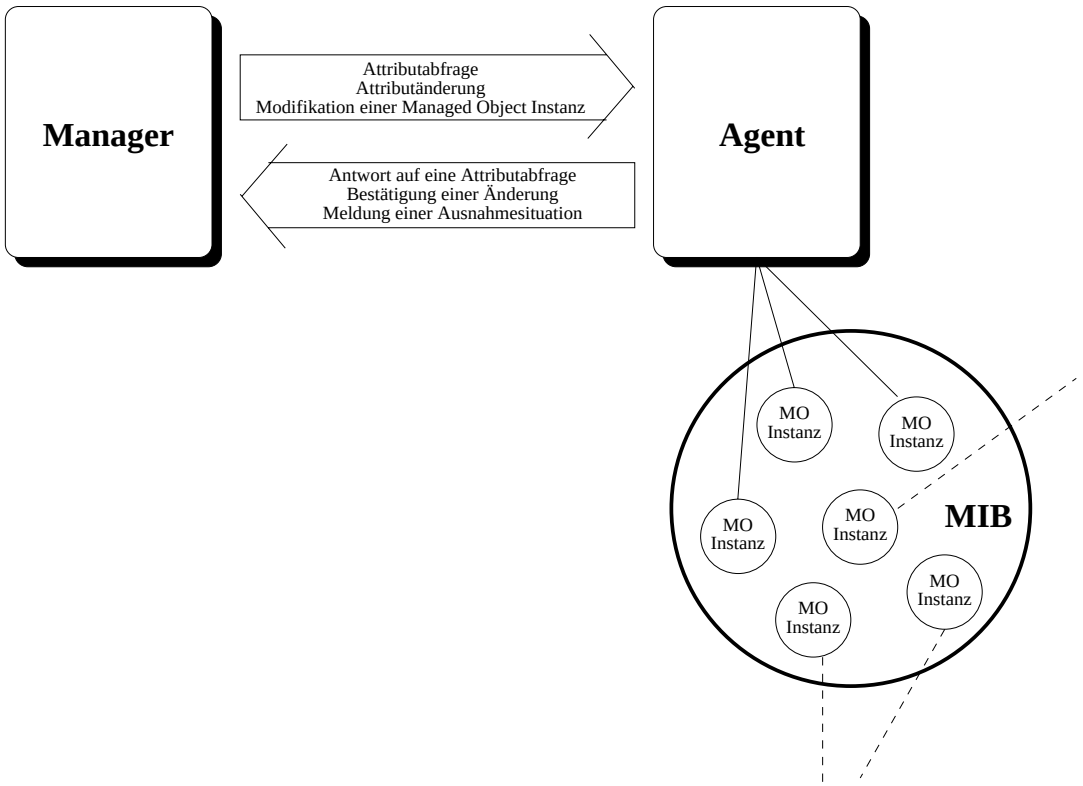

Abbildung 1. OSI Management Modell

2 Implementierung einer MIB

2.1 Beschreibung von MOs

Der Aufbau von MO's legt die Beschreibung mit Hilfe einer *objekt-orientierten Sprache* nahe. Dazu müssen die Objekte mit ihren Attributen und ihrem Verhalten identifiziert werden. Dies ist zwar in den meisten Fällen offensichtlich, aber es gibt auch Situationen, die einige Probleme bereiten. So läßt sich z.B. ein Zähler zur Bestimmung der Anzahl der übertragenden Datenpakete leicht durch ein Objekt mit einem entsprechend typisierten Attribut und Operationen zum Setzen und Lesen des Zählerstandes modellieren.

Im Gegensatz dazu sind Beziehungen zwischen MOs abgesehen von der Vererbung nur mit Tricks oder Erweiterungen darstellbar. Beispiele für sinnvolle und wünschenswerte Beziehungen sind:

1. Enthaltenseinrelation: Manche Objekte müssen physikalisch oder logisch in einer anderen Objektinstanz vorhanden sein.
2. Verknüpfungen: Solche Beziehungen werden zum Beispiel zur Modellierung von Netzwerktopologien benötigt.
3. Berechtigungen: Diese Beziehungen dienen zur Realisierung von Zugriffsrechten.

2.2 Speicherung der Datenobjekte auf einer Netzwerkkomponente

Je nach Aufbau einer Netzwerkkomponente gibt es unterschiedliche Möglichkeiten die Datenobjekte zu speichern. Man kann die Daten in einem ROM, einem RAM oder einem Massenspeicher (z.B. Festplatte oder Diskette) ablegen. Alle genannten Arten unterscheiden sich bzgl. der Schnelligkeit des Zugriffs, der Veränderbarkeit der Daten und eines möglichen Datenverlustes bei Stromausfall (siehe Abb. 2).

Speicherart	Zugriff	Veränderbarkeit	möglicher Datenverlust
ROM	mittel	nein	nein
RAM	schnell	ja	ja
Disk	langsam	ja	nein

Abbildung 2. Vergleich der Speichermedien

Die *Datenbank* in einem Agenten kann objekt-orientiert, relational oder in speziell angepaßten Formaten aufgebaut sein.

Das *objekt-orientierte Format* bietet sich an, weil einmal die Daten selber objekt-orientiert aufgebaut sind und zweitens neben den Attributen auch die Operationen mitverwaltet werden können. Auf der anderen Seite hat eine MIB durchschnittlich nur 3 Vererbungsstufen und nützt deshalb die Fähigkeiten einer OO-Datenbank nur schlecht aus. Außerdem gibt es bis heute keinen standardisierten Zugriff auf eine solche Datenbank.

Dieser wird aber von vielen stabilen und schnellen Implementierungen für *relationale Datenbanken* angeboten (z.B. SQL). Nachteilig wirkt sich aber aus, daß die oben beschriebenen Beziehungen zuerst umgewandelt werden müssen, damit sie im relationalen Modell darstellbar sind. Die Umwandlung kann zu einem beachtlichen Effizienzverlust führen.

Mit *speziell angepaßten Formaten* kann eine effiziente Implementierung erreicht werden. Der Entwicklungsaufwand und besonders die Wartungskosten (die Datenbank muß bei Änderungen immer wieder angepaßt werden) sind aber hoch.

In der Praxis verwendet man nicht nur eine, sondern Kombinationen der aufgeführten Realisierungsformen. Dadurch können die Vorteile der Varianten individuell für die Ma-

nagementinformationen ausgenutzt werden. Eine typische Konfiguration besteht zum Beispiel aus:

1. ROMs zur Speicherung von statischen Informationen und Operationen,
2. RAMs zur Verwaltung von dynamisch veränderbaren Informationen und
3. einer auf Massenspeichern basierten relationalen Datenbank.

2.3 Patitionierung der MIB

Die Informationen der Datenobjekte können auf den Agenten und/oder in einer *zentralen Datenbank* auf einer Managementstation verwaltet werden.

Im letzteren Fall kommen die Netzwerkkomponenten ohne Speicher und mit weniger Logik aus. Es entsteht aber ein erhöhter Kommunikationsaufwand, da Änderungen sofort an die zentrale Station weitergeleitet werden müssen. Dies wirkt sich negativ auf die Leistung des Netzes aus und beeinträchtigt die Reaktionszeit der MOs erheblich. Wenn das Netzwerk aber hierarchisch aufgebaut ist, kann die Belastung gemindert werden, indem Komponenten auf mittleren Hierarchiestufen die Verwaltung der Daten für die darunterliegenden Ressourcen übernehmen. Die einzelnen Managementinformationen werden nun nicht mehr über das gesamte Netz, sondern nur noch innerhalb eines gewissen Segmentes übertragen. Bei Ausfall der Managementstation fällt das ganze Management aus.

Sollen die *Informationen auf den Agenten* gespeichert werden, so müssen diese über veränderbare Speicher verfügen und werden dadurch komplexer. Sie bieten aber dann eine kurze Reaktionszeit. Handelt es sich um einen flüchtigen Speicher wie z.B. RAM, ist eine Sicherung der Daten gegen Verlust notwendig. Eine Möglichkeit wäre, daß sie in der Managementstation ein zweites Mal gespeichert werden. Man handelt sich aber dann Konsistenzprobleme ein (siehe Abschnitt 2.4).

Systemweite Informationen werden meistens in einer zentralen Datenbank gesichert. Tritt eine häufigere Benutzung von bestimmten Daten auf, so kann es sinnvoll sein, sie zusätzlich in Agenten abzulegen. Es müssen dann wie oben Mechanismen zur Wahrung der Konsistenz eingeführt werden.

Die Bestimmung der besten Aufteilung erfolgt in Abhängigkeit von der Antwortzeit, der Bandbreite, der Auslastung des Netzes, sowie Analysen der Abfragenstrukturen der MO's.

2.4 Integrität der Daten

Um Fehler zu vermeiden, ist die *Konsistenz der Daten* zu gewährleisten. Wie oben gesehen, gibt es Situationen in denen Daten aus Effizienz- oder Ausfallgründen mehrfach gespeichert werden. Die Einführung von Kontrollmechanismen ist in diesem Fall unausweichlich. Eine Möglichkeit besteht darin die Infomationen eines Objektes zu sperren und dann ein simultanes Updating durchzuführen. Da dazu einige Kommunikation über das Netzwerk erfolgt und das MO während dieser Zeit gesperrt ist, liegt die Reaktionszeit des MO's relativ hoch. Kann jedoch eine gewisse kurzzeitige Inkonsistenz in Kauf

genommen werden, dann bietet es sich an einen periodischen Polling-Mechanismus zu verwenden. Dabei werden in bestimmten Zeitabständen die Objekte aktualisiert. Dieses Verfahren beeinflußt die Reaktionszeit nicht.

Neben der Datenkonsistenz muß auch die *Integrität der Existenzabhängigkeiten* gegeben sein. Unter einer Existenzabhängigkeit versteht man, wenn ein MO zur Erfüllung seiner Aufgabe die Existenz einer Instanz eines anderen MO's voraussetzt. Beim Löschen von Instanzen könnte die Integrität verletzt werden. Dieses Problem wird mit Hilfe eines Referenzobjekts gelöst. Es hat die Aufgabe die Referenzen auf ein MO zu zählen und in Abhängigkeit des Zählerstandes zu entscheiden, ob das MO gelöscht werden soll oder nicht.

Die *Konsistenz der Attribute* ist ein weiterer wichtiger Punkt. Die Attribute haben teilweise einen eingeschränkten Wertebereich, der nicht überschritten werden sollte. Der Agent muß ungültige Werte abweisen und entsprechende Fehlermeldungen weitergeben. Ein Beispiel dafür ist ein Netzwerkadapter, der nur Übertragungsraten von 4, 10 und 100 MBit/s unterstützt, dem aber ein Wert von 155 MBit/s oder 20 MBit/s zugewiesen wird.

2.5 Wartung der MIB

Änderungen an der Partitionierung der MIB oder an den MOs selber sind im Laufe der Zeit nicht ausschließbar. Ein Grund wäre eine neue Implementierung eines MO's, wodurch die Daten nun an einem anderen Ort gespeichert werden. Aber auch Veränderungen in der Abfragestruktur der MOs können eine Wartung notwendig machen. In einem solchen Fall muß man die Verteilung der Daten dahingehend überdenken, ob es nicht sinnvoll geworden ist, bestimmte MOs aus der zentralen Datenbank auf eine Netzwerkkomponente zu verlagern und dafür andere Daten des Agenten jetzt zentral zu verwalten.

Das Löschen und Hinzufügen von MOs und Änderungen der internen Repräsentation sollte durch die Implementierung unterstützt werden. Dadurch braucht das Management nicht jedesmal angehalten und neu initialisiert werden.

Das Ziel besteht darin, die Einwirkungen auf eine Anwendung durch eine Wartung so gering wie möglich zu halten. Dies fällt bei der Verteilung der Informationen relativ leicht. Anpassungen der Anwendung sind aber meistens unausweichlich, wenn bestehende Eigenschaften der MO's geändert oder Instanzen gelöscht werden

2.6 Backup und Recovery

Man möchte bei einer Störung des Systems den Datenverlust möglichst gering halten. Um dies zu gewährleisten, müssen von dem Managementsystem Backups erstellt werden. Man unterteilt die Daten einmal in die MIB Beschreibung und zweitens in die aktuellen Werte der Attribute.

Die MIB ändert sich nicht ständig, sondern nur bei einer Wartung. Da dies in der Regel nicht so häufig geschieht, benötigt man keine periodisch erstellten Backups, sondern kann sie explizit erzeugen.

Im Gegensatz dazu sind die Attribute der Datenobjekte einer laufenden Änderung unterworfen. Es bietet sich hier ein periodisches Backup an, wobei die Zeitabstände ein Kompromiß zwischen der Netzwerkbelastung durch das Backup und der Aktualität der gesicherten Daten darstellen.

2.7 Weiterführende Anforderungen an eine MIB Implementierung

Neben grundsätzlichen Aspekten bei der Implementierung einer MIB, wird auch die Unterstützung von weiterführenden Fähigkeiten gefordert. Diese werden im Rahmen der Ausarbeitung nur erwähnt. In einer MIB sollte ein *Sicherheitsmanagement* zur Regelung von Zugriffsrechten auf die MOs modelliert werden können. Von Interesse ist es auch Benutzer zu *Gruppen* zusammenzufassen. Dadurch sind zum Beispiel Projektgruppen verwaltbar und es kann einer Gruppe mit hohem Kommunikationsaufwand eine bestimmte Übertragungskapazität zur Verfügung gestellt werden. Weiterhin ist eine *History-Funktion* für manche MOs für den Systemoperator notwendig. Damit sind Änderungen an der Konfiguration nachvollziehbar. Dies spielt eine große Rolle, wenn ein Managementprogramm in Abhängigkeit des Netzwerkzustandes automatisch die Konfiguration ändert.

3 Umsetzung von GDMO-Beschreibungen

3.1 Einführung in GDMO

Zur Beschreibung von Managed Objects wurden von der ISO die *Guidelines for the Definition of Managed Objects (GDMO)* eingeführt. GDMO ist eine Makroerweiterung von *ASN.1* und besteht aus 9 Syntax-Schemata, sogenannten *Templates* [Ehr92, Sei94b]. Jedes einzelne Template kann mit dem Konstrukt REGISTER AS... in der MIB mit einer weltweit eindeutigen Kennung registriert werden.

Managed Object Class Template: Dieses Template führt ein neues Objekt ein und legt durch Angabe der Oberklasse(n) die Position im Ableitungsbaum fest. Es werden zusätzlich noch Verweise auf sogenannte Packages aufgelistet. Die Packages beinhalten die Eigenschaften und das Verhalten des Objekts. Die Einbindung eines Packages muß nicht immer zwangsläufig stattfinden, sondern kann auch von einer Bedingung abhängig gemacht werden.

```
classname MANAGED OBJECT CLASS
        DERIVED FROM class(es);
        CHARACTERIZED BY package(s);
        CONDITIONAL PACKAGES
           packages(s) PRESENT IF condition(s);
REGISTERED AS object-identifier;
```

Package Template: Das Package Template erlaubt es eine Menge von Attributen, Attributgruppen, Operationen und Meldungen durch Aufzählung der Bezeichner zu

einem Modul zusammenzufassen. Ein solches Modul ist in anderen MOs wiederverwendbar. Die Attribute können dabei mit einer Eigenschaftsliste versehen werden, in der Grundoperationen (GET, REPLACE, ADD oder REMOVE), Default- und Initialwerte (DEFAULT VALUE und INITIAL VALUE) festgelegt sind.

```
packagename PACKAGE
        ATTRIBUTES attribute(s);
        ATTRIBUTE GROUPS groupattribut(s);
        ACTIONS action(s);
        NOTIFICATIONS notification(s);
        BEHAVIOUR behaviour-template(s);
REGISTERED AS object-identifier;
```

Parameter Template: Mit diesem Template spezifiziert man Parameter, die im Zusammenhang mit Attributen, Aktionen und Meldungen in einem Fehlerfall ausgetauscht werden. Ihre Gültigkeit kann auf einen der Kontexte ACTION-INFO, ACTION-REPLY, EVENT-INFO, EVENT-REPLY und SPECIFIC-ERROR eingeschränkt sein. Der Eintrag type-reference ist ein Name eines ASN.1-Typs, der den Typ des Parameters festlegt.

```
parametername PARAMETER
        WITH SYNTAX type-reference;
        CONTEXT context-type;
        BEHAVIOUR behaviour-template(s);
REGISTERED AS object-identifier;
```

Attribute Template: Das Template definiert ein Attribut, dessen Typ von einem anderen Attribut abgeleitet sein kann oder mit ASN.1 spezifiziert wird. Vergleichsoperatoren sind hinter „MATCHES FOR“ zu definieren. Zur Auswahl stehen EQUALITY, ORDERING, SUBSTRINGS, SET-COMPARISON und SET-INTERSECTION. Das Feld PARAMETERS definiert Argumente, die einer Fehlermeldung mitgegeben werden.

```
attributename ATTRIBUTE
         DERIVED FROM attribute;
        or
         WITH SYNTAX type-reference;
        MATCHES FOR qualifier(s);
        PARAMETERS parameter-template(s);
        BEHAVIOUR behaviour-template(s);
REGISTERED AS object-identifier;
```

Attribute Group Template: Attribute können zu Gruppen zusammengefaßt werden. Solche Gruppen sind als unveränderlich kennzeichenbar, wodurch eine spätere Erweiterung unmöglich wird. Das Template nimmt auch einen Kommentar auf, in dem meistens der Grund für die Gruppierung angegeben ist.

```
groupname ATTRIBUTE GROUP
        GROUP ELEMENTS attribute(s);
        FIXED;
        DESCRIPTION text;
REGISTERED AS object-identifier;
```

Action Template: Dieses Template definiert eine Operation durch Angabe eines Funktionsnamens und des ASN.1-Typs für den Eingabe- und den Ausgabeparameter. Weiterhin wird festgelegt, ob die Operation bestätigt oder unbestätigt ausgeführt werden soll. Es besteht die Möglichkeit Parameter für eine etwaige Fehlermeldung zu definieren.

```
actionname ACTION
        WITH INFORMATION SYNTAX type-reference;
        WITH REPLY SYNTAX type-reference;
        MODE CONFIRMED;
        PARAMETERS parameter-template(s);
        BEHAVIOUR behaviour-template(s);
REGISTERED AS object-identifier;
```

Notification Template: Mit diesem Template werden Meldungen spezifiziert. Der Aufbau ist ähnlich wie beim Action Template.

```
notificationname NOTIFICATION
        WITH INFORMATION SYNTAX type-reference;
        WITH REPLY SYNTAX type-reference;
        PARAMETERS parameter-template(s);
        BEHAVIOUR behaviour-template(s);
REGISTERED AS object-identifier;
```

Name Binding Template: Das Name Binding Template dient zur Realisierung einer Enthaltenseinrelation. Es gibt an, daß die Klasse superior-class eine Instanz der Klasse subordinate-class enthält und auf diese über das angegebene Attribut zugreifen kann. Die dynamische Erzeugung und das Löschen einer Instanz kann durch Angabe der Modifier WITH-REFERENCE-OBJECT (Angabe eines Referenzobjekts zur Festlegung des Initialwertes), WITH-AUTOMATIC-INSTANCE-NAMING (automatische Erzeugung des Instanzennamens bei der Instanziierung), ONLY-IF-NO-CONTAINED-OBJECTS (nur dann Löschen, wenn die Instanz keine anderen Instanzen enthält) und DELETES-CONTAINED-OBJECTS (enthaltene Instanzen werden mitgelöscht) beeinflußt werden.

```
namebinding-name NAME BINDING
        SUBORDINATE OBJECT CLASS subordinate-class;
        SUPERIOR OBJECT CLASS superior-class;
```

```
        WITH ATTRIBUTE attribute;
        CREATE create-modifier(s);
        DELETE delete-modifier;
        BEHAVIOUR behaviour-template(s);
REGISTERED AS object-identifier;
```

Behaviour Template: Das Template ermöglicht eine umgangssprachliche oder auch formale Beschreibung von Packages, Operationen, Attributen und Name Bindings. Fast allen Templates kann ein Verweis auf ein Behaviour Template mitgegeben werden.

```
behaviour BEHAVIOUR
        DEFINED AS behaviour-description;
```

3.2 Umsetzung von GDMO in C++-Klassen

Man wünscht sich eine möglichst automatische Umsetzung der GDMO-Beschreibungen, um die Implementierung zu vereinfachen. Dabei muß beachtet werden, daß mit GDMO der Aufbau eines Objekts, aber nicht dessen Verhalten spezifizierbar ist. Aus diesem Grund kann ein Algorithmus nur Prototypen für die Klassen generieren.

Beispielhaft soll hier der Algorithmus von Soukouti [Sou94] vorgestellt werden. Der Grundgedanke besteht darin, jede MOC auf eine C++-Klasse abzubilden. Dem Managed Object Class Template wird dazu der Klassenname, die Oberklasse und die Packagebezeichner entnommen und ein entsprechender Klassenrahmen in C++ generiert. Bei der Umsetzung werden grundsätzlich alle Packages eingebunden, egal ob eine Bedingung angegeben wurde oder nicht. Die Abbildungen 3, 4 und 5 enthalten ein Beispiel, indem für die MOC printer eine von device abgeleitete C++-Klasse erzeugt wurde. In den Zeilen 2-6 von Abb. 3 werden dazu zuerst ein paar Datentypen in ASN.1 angegeben. Anschließend folgen Templates für eine MOC, ein Package, Attribute, Aktionen und eine Meldung.

Die Transformation der einzelnen Attribute aus den Packages verläuft in 3 Schritten.

1. Für die Attribute werden Variablen gleichen Namens in die C++-Klasse eingefügt. Den Datentyp ermittelt der Algorithmus aufgrund der Angaben im entsprechenden Attribute Template entweder durch Kopieren oder mit Hilfe eines ASN.1-Compilers.

2. Die Grundoperationen aus der Eigenschaftsliste werden in Methoden, deren Name sich aus dem Attributnamen und dem Funktionsbezeichner zusammensetzen, transformiert.

3. Für die Vergleichsfunktionen führt der Algorithmus boolesche Funktionen ein, denen als Parameter ein Zeiger auf einen Wert, der den gleichen Typ, wie das Attribut hat, übergeben wird. Auch hier sind die Methodennamen nach einem Schema aufgebaut. Sie bestehen aus dem Attributnamen, dem Operatornamen und dem Fragewort „Is“.

Zum Beispiel wird das Attribut font (Abb. 3, Z. 14, 20-23) im Package printerPackage auf eine Variable (Abb. 5, Z. 5) mit den dazugehörigen Funktionen Get_font, Replace_font und Is_font_Equal (Abb. 5, Z. 12-14) abgebildet.

Zur Umsetzung der Aktionen und Meldungen ermittelt der Algorithmus die benötigten Ein- und Ausgabeparameter mit ihren Typen aus dem Action Template, überprüft den Kontext und erzeugt entsprechende Funktionsprototypen. Es wird dabei nicht beachtet, ob die Methode bestätigt oder unbestätigt ausgeführt werden soll, da dies im Sourcecode der einzelnen Methoden integriert werden muß, und nicht im Prototypen realisierbar ist. Die Aktion Configure (Abb. 3, Z. 35-38) zum Beispiel hat einen Eingabeparameter vom Typ ConfigureInfo und keinen Ausgabeparameter. Dies spiegelt sich im Prototypen der entsprechenden Methode wieder (Abb. 5, Z. 19).

Die Informationen aus Name Binding Templates werden derzeitig nicht verwendet (siehe Abschnitt 4).

4 Diskussion

In Abschnitt 2 sind einige wichtige Aspekte der Implementierung (Beschreibung und Speicherung der Datenobjekte, Partionierung und Wartung der MIB, Integrität und Backup der Daten) besprochen worden. Sie bilden nur einen kleinen Ausschnitt und man wird beim Programmieren noch auf viele Probleme stoßen.

Die ISO hat sich darauf festgelegt die Datenobjekte objekt-orientiert zu beschreiben. Dieser Ansatz wird den Bedürfnissen einer Implementierung nicht ganz gerecht, da Beziehungen, die mannigfaltig auftreten können, nicht darstellbar sind. Eine Erweiterung wurde bereits mit GDMO eingeführt. Das Name Binding Template ermöglicht zwar Enthaltenseinrelationen anzugeben, aber ein allgemeineres Konzept, mit dem auch andere Beziehungen modellierbar sind, wäre sinnvoll.

Eine Definition von MOs sollte möglichst in einer formalen Spezifikationssprache erfolgen, damit Mißverständnisse vermieden werden. GDMO weist diesbezüglich einige Lücken auf. Die Bedingungen für das Einbinden von Packages und die Beschreibung von Operationen mittels Behaviour-Template können zum Beispiel in Umgangssprache verfaßt sein. Dies macht eine Interpretation durch ein Programm und damit eine vollständig automatische Umsetzung von GDMO-Beschreibungen unmöglich. Der in Abschnitt 3.2 angegebene Algorithmus behilft sich dadurch, daß grundsätzlich alle Packages eingebunden und für die Aktionen keine entsprechenden Methoden generiert werden. Eine manuelle Überarbeitung des Sourcecodes ist unabdingbar. Das Programm unterstützt den Benutzer dabei, indem es Kommentare einfügt. Man könnte aber auch den Algorithmus dahingehend erweitern, daß Präprozessoranweisungen in den Sourcecode integriert werden. Die Einbindung der Packages ließe sich dann über Compilerdirektiven steuern.

Ein großes Problem bei der Implementierung ist die Partionierung der MIB. Sie ist sehr stark von der Hardware abhängig und kann deshalb nicht durch Tools unterstützt werden. Maßnahmen zur Einhaltung der Datenkonsistenz werden maßgebend von ihr beeinflußt. Um eine hohe Effizienz zu erreichen ist für jedes einzelne Objekt zu entscheiden, ob, und wenn ja, welcher Kontrollmechanismus benötigt wird.

```
1 Example DEFINITIONS ::= BEGIN

2  SpeedType  ::= INTEGER
3  FontType   ::= ENUMERATED{times(0), system(1), roman(2), helvetica(3)}
4  OperationalStatusTyp  ::= BOOLEAN
5  ConfigureInfo  ::= SEQUENCE{speed SpeedType, font FontType}
6  PrintinError  ::= ENUMERATED{jam(0), NoMorePaper(1), noMoreToner(2)}
7  ...
8  printer MANAGED OBJECT CLASS
9          DERIVED FROM device;
10         CHARACTERIZED BY printerPackage;
11 REGISTERED AS {2 1 1};
12
13 printerPackage PACKAGE
14         ATTRIBUTES speed GET-REPLACE, font GET-REPLACE,
15              operationalStatus GET;
16         ACTIONS Configure, activate, deactivate;
17         NOTIFICATIONS printingError;
18 REGISTERED AS {2 2 1};
19
20 font ATTRIBUTE
21         WITH ATTRIBUTE SYNTAX FontType;
22         MATCHES FOR EQUALITY;
23 REGISTERED AS {2 3 1};
24
25 speed ATTRIBUTE
26         WITH ATTRIBUTE SYNTAX SpeedType;
27         MATCHES FOR EQUALITY, ORDERING;
28 REGISTERED AS {2 3 2};
29
30 operationalStatus ATTRIBUTE
31         WITH ATTRIBUTE SYNTAX OperationalStatusType;
32         MATCHES FOR EQUALITY;
33 REGISTERED AS {2 3 3};
34
35 Configure ACTION
36         MODE CONFIRMED;
37         WITH INFORMATION SYNTAX ConfigureInfo;
38 REGISTERED AS {2 4 1};
39
40 activate ACTION
41         MODE CONFIRMED;
42 REGISTERED AS {2 4 2}
43
```

Abbildung 3. Beispiel einer MOC in GDMO

```
44  deactivate ACTION
45        MODE CONFIRMED;
46  REGISTERED AS {2 4 3};
47
48  printingError NOTIFICATION
49          WITH INFORMATION SYNTAX PrintingError;
50  REGISTERED AS {2 5 1};
51  END
```

Abbildung 4. Fortsetzung der MOC in GDMO

```
 1  class printer: public device {
 2   protected:
 3    /* attribute definition */
 4    SpeedType             *speed;
 5    FontType              *font;
 6    OperationalStatusType *operationalStatus;
 7   public:
 8    /* operation on attributes */
 9    SpeedType             *GET_speed();
10    void                  *REPLACE_speed(SpeedType *Value);
11    boolean               Is_speed_Equal(SpeedType *Value);
12    FontType              *GET_font();
13    void                  *REPLACE_font(FontType *Value);
14    boolean               Is_font_Equal(FontType *Value);
15    OperationalStatusType *GET_operationalStatus();
16    boolean               Is_operationalStatus_Equal
17                                (OperationalStatusType *Value);
18    /* actions of this class */
19    void                  Configure(ConfigureInfo *Information);
20    void                  activate();
21    void                  deactivate();
22    /* notifications of this class */
23    void                  printingError(PrintingError *Information);
24  };
```

Abbildung 5. Aus der MOC erzeugte C++-Klasse.

Man könnte meinen, daß Informationen über Speicherung und Verteilung der Daten, sowie Wartung, Backup und Datenkonsistenz in die GDMO-Beschreibung aufgenommen werden sollten, damit die Umsetzung weiter automatisiert werden kann. Es wäre dann möglich über die Klassenprototypen hinaus die Implementierung zu erleichtern. Man darf aber nicht außer Acht lassen, daß die MIB die Schnittstelle zwischen Managementanwendung und Manager, also den Zugriff auf die Daten, und nicht die Implementierung beschreibt. Aus diesem Grund gehören die erwähnten Aspekte nicht in eine GDMO-

Beschreibung und sollten auch nicht mit GDMO spezifizierbar sein.

Es gibt viele Möglichkeiten die Funktionalität einer MIB, wie die Beispiele aus Abschnitt 2.7 zeigen, zu erweitern. Man sollte aber darauf achten, daß nicht zu viele Funktionen eingeführt werden, um die Implementierung übersichtlich zu halten.

Ein konzeptuell bedingtes Problem ist die statische Vorgehensweise. Alle Informationen müssen beim Compilieren vorliegen und können zur Laufzeit nicht mehr geändert werden. Das Fehlen der Dynamik macht das Management starr und unbeweglich. Kurzfristige Anpassungen an Änderungen im Netzwerk sind unmöglich. Die Bedeutung einer Enthaltenseinrelation relativiert sich dadurch, da man in einem statischen Sytem einen besseren Überblick hat und der Compiler bereits vor der Laufzeit die meisten Fehler findet. Deshalb ist es nicht tragisch, daß das Name Binding Template bei der Umsetzung nicht berücksichtigt wird.

5 Ausblick

In der Diskussion und in Abschnitt 2.7 ist deutlich geworden, daß neben der Grundfunktionalität auch weitergehende Anforderungen an eine Implementierung bestehen. Welche Funktionen aber wirklich in der Praxis gebraucht werden und welche nur in Spezialfällen zur Anwendung kommen ist nicht geklärt.

Desweiteren hat sich gezeigt, daß der objekt-orientierte Ansatz zur Beschreibung der MOs nicht ausreicht. Mit einer anderen Darstellungsform könnte man unter Umständen Abhilfe schaffen.

Die Verwendung der Umgangssprache zur Spezifikation beeinträchtigt den Einsatz von GDMO. Die Normungsgremien ITU und ISO haben diese Schwachstelle erkannt und untersuchen derzeit, ob nicht der Einsatz von formalen Spezifikationssprachen möglich ist (Stand August 94) [FHHS93, FZ93, ISO].

Eine höhere Flexibilität, sowie bessere Bedien- und Wartbarkeit eines Managementsystems könnte erreicht werden, wenn nicht, wie in diesem Beitrag, ein statischer, sondern ein dynamischer Ansatz für die Implementierung der MIB benutzt würde. Die Verwendung eines Interpreters oder eines Laufzeitsystems, das compilierte Module dynamisch linken kann, wären zum Beispiel möglich.

Implementierung der Information Base: Erweiterung der GDMO-Notation um implementierungstechnische Hinweise

Martin Meuer

Kurzfassung

Das OSI-Informationsmodell der ISO hat den Nachteil, daß sich Relationen zwischen Ressourcen und ihren abstrakten Abbildern nur unzulänglich beschreiben lassen. Der vorliegende Aufsatz stellt Ansätze zur Erweiterung der OSI-Objektbeschreibung vor, die die Beschreibung dieser Relationen ermöglichen. Das CMO-Konzept basiert auf der disjunkten Zerlegung von Managementobjekten in Teile, so daß die Modellierung dieser Beziehungen auf Relationen zwischen den Teilen zurückgeführt werden kann. Allerdings gelingt es dabei nicht, alle Relationen abzudecken, die den Implementierer von Managementobjekten interessieren. Der Ansatz von Hegering, Abeck und Boehnke zeigt einen Weg, wie auch der Mapping-Aspekt mit in die Modellierung mitaufgenommen werden kann. Schließlich wird mit dem Ansatz von Hegering, Abeck, Noehnke und Heiler ein Konzept vorgestellt, das die Einführung eines neuen Template-Typs vorsieht. Mit Hilfe dieses Templates lassen sich alle implementierungsrelevanten Relationen zwischen Managementobjekten und den zugehörigen Ressourcen beschreiben. Zudem gewährleistet dieses Konzept, im Gegensatz zu den beiden anderen, die Weiterbenutzung bereits vorhandener Objektkataloge.

1 Einleitung

Die Verwaltung zunehmend komplexer und heterogener Netzwerke ist ohne Rechnerunterstützung kaum mehr zu bewältigen, weshalb die Entwicklung bzw. Weiterentwicklung sogenannter offener Netzwerkmanagement(NM)-Architekturen allgemein auf reges Interesse stößt.

Grundlage einer NM-Architektur ist eine Datenbasis, die alle relevanten Managementinformationen enthält, auf die die Managementanwendungen sowohl lesend als auch schreibend zugreifen können sollen. Beispiele für Managementinformationen sind Informationen über reale Netzkomponenten (z.B. Router, Bridge, Multiplexer), Protokollinstanzen, Attribute von Objekten, etc. Offensichtlich sollte die Speicherung und damit auch der Zugriff auf die Managementinformationen in heterogenen Netzwerken geräteunabhängig sein. Dies wiederum impliziert, daß zu jeder Komponente innerhalb des Netzwerkes ein hinreichend abstraktes Modell zu entwickeln ist, das einerseits der Managementanwendung eine einheitliche Schnittstelle bietet und andererseits nur die Eigenschaften einer realen oder logischen Ressource zur Verfügung stellt, die aus Managementsicht relevant sind. Ein solches, für Managementzwecke geeignetes Abbild von Ressourcen, nennt man *Managementobjekt (Managed Object, MO)*. Den konzeptionellen Behälter für die MOs eines Systems bezeichnet man als *Management-Informations-Datenbasis (Management Information Base, MIB)*. Man unterscheidet folgende drei Aspek-

te bei MIB-bezogenen Problemstellungen: das Informationsmodell, die Objektmodellierung und die MIB-Implementierung.
Das Informationsmodell stellt den strukturellen Rahmen für die Beschreibung von MOs zur Verfügung. Die Objektmodellierung bezeichnet die abstrakte Modellierung der Information, die für die Verwaltung eines spezifischen Gerätetyps (wie z.B. Brücke zwischen Ethernet und Tokenring) benötigt wird. Die MIB-Implementierung beinhaltet die Abbildung der in einer MIB zusammengefaßten MOs auf reale Netzwerkkomponenten.
Abschnitt 2.1 zeigt die Einbettung des Informationsmodells innerhalb des Gesamtkomplexes einer NM-Architektur. Im folgenden Abschnitt wird der mächtige OSI-Ansatz eines Informationsmodells vorgestellt. Anschließend werden in 2.3 die Defizite des OSI-Informationsmodells anhand von Beispielen herausgearbeitet. Abschnitt 3 stellt drei unterschiedliche Ansätze vor, wie man die Mängel des OSI-Informationsmodells durch bestimmte Erweiterungen teilweise (Abschnitt 3.1 und 3.2) oder ganz (Abschnitt 3.3) beheben kann. In der Schlußbetrachtung werden schließlich die Ansätze miteinander verglichen.

2 Das Informationsmodell im OSI-Management

2.1 Einordnung des Informationsmodells in eine NM-Architektur

Laut [HABH94] sollte eine NM-Architektur, die für ein integriertes Netzwerkmanagement in heterogener Umgebung geeignet sein soll, zu folgenden Teilaspekten entsprechende Modelle bereitstellen:

- Beschreibung von Managementobjekten (*Informationsmodell*)
- Behandlung und Unterstützung von Organisationsaspekten (*Organisationsmodell*)
- Beschreibung von Kommunikationsvorgängen zu Managementzwecken (*Kommunikationsmodell*)
- Struktur von Managementaufgaben (*Funktionsmodell*)

Abbildung 6 zeigt die OSI-NM-Architektur mit den einzelnen Teilmodellen nach [HA93]. Das Informationsmodell umfaßt die Konzepte zur Beschreibung der MOs, d.h. es beinhaltet ein Modellierungsparadigma (z.B. objektorientierter Ansatz, Entity-Relationship-Modell) und die Spezifikation einer eindeutigen Syntax für die Beschreibung von Managementinformation. Es nimmt somit eine Sonderstellung innerhalb einer NM-Architektur ein, da es die Strukturierung und Definition des Herzstücks einer jeden NM-Architektur, der Datenbasis für Managementinformationen, beinhaltet.

2.2 Das Konzept der OSI

Der ISO-Beschreibungsrahmen, das OSI-Informationsmodell, wird im ISO-Standard 101-165 (*Structure of Management Information*, SMI) beschrieben. Das OSI-Informationsmodell basiert auf dem objektorientierten Ansatz. Objektorientierung bedeutet hierbei, daß

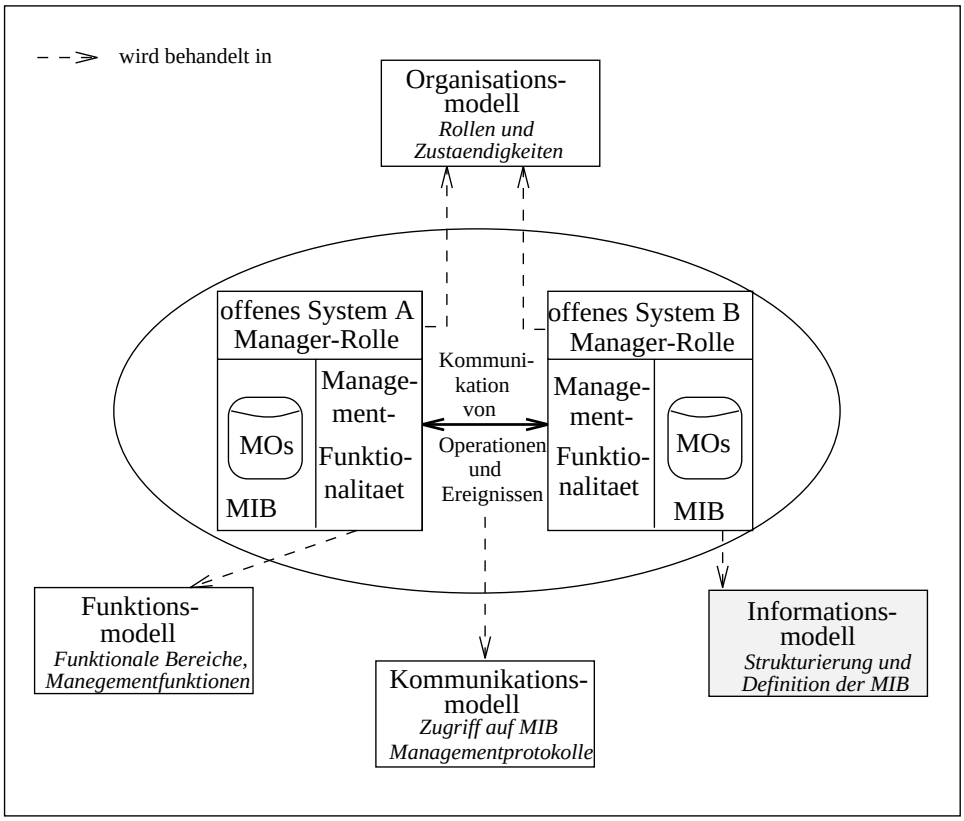

Abbildung 6. Teilmodelle einer NM-Architektur

Datenabstraktion und Vererbungsmechanismen verwendet werden. Als Abstraktionsansatz der MOs wird die Black-Box-Methode (Information Hiding, Encapsulation) benutzt, d.h. das innere Verhalten der zu verwaltenden Ressource wird nur an wohldefinierten Schnittstellen sichtbar, den sogenannten *MO-Boundaries*. In [Sei94c] wird dies wie in Abbildung 7 dargestellt.

Die MOs sind Instanzen von *Management Objekt Klassen (Managed Object Classes,*

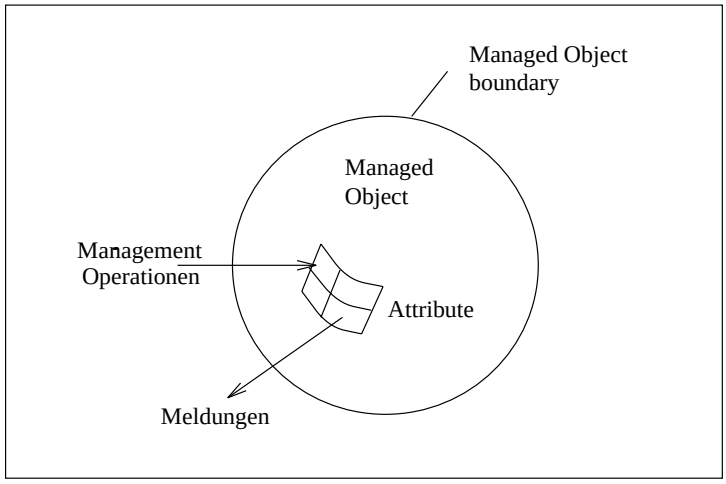

Abbildung 7. Modell eines Managementobjekts

MOC). Die MOCs werden über den Vererbungsmechanismus gebildet, d.h. eine Klasse

erbt automatisch sämtliche Eigenschaften aller Oberklassen innerhalb des Vererbungsbaumes und kann zusätzlich weitere Eigenschaften definieren.
Ein MO ist also charakterisiert durch die an seiner Boundary sichtbaren Attribute und ausführbaren Managementoperationen sowie der Meldungen die das MO aussenden kann. Die Bedeutung der

- Attribute,
- Operationen und Meldungen,
- Reaktion auf Operationen, die auf einem MO ausgeführt werden,
- Umstände, unter denen Meldungen ausgesendet werden,
- Abhängigkeiten zwischen Werten bestimmter Attribute sowie
- Auswirkungen von Beziehungen zu anderen MOs

sind nach [Dit91] durch das Verhalten (*behaviour*) eines MO festgelegt. MOs, die die gleichen Attribute, Managementoperationen, Meldungen und das gleiche Verhalten besitzen, gehören zur selben MOC.
Die Defintion von Eigenschaften einer MOC und die Einordnung in den Vererbungsbaum erfolgt durch die Konkretisierung von Schablonen, sogenannten *Templates*, gemäß den *Guidelines for the Definition of Managed Objects* (GDMO). Diese Templates werden durch eine einfache Template-Metasprache, basierend auf ASN.1-Makromechanismen, beschrieben. Zur Zeit sind folgende 9 generische Templatestrukturen genormt, wobei die einzelnen Templates Verweise auf andere Templates haben können, wodurch ein Wiederbenutzen bereits eingeführter Definitionen ermöglicht wird:

- Managed Object Class Template
- Package Template
- Attribute Template
- Attribute Group Template
- Action Template
- Notification Template
- Parameter Template
- Name Binding Template
- Behaviour Template

Für eine umfassende Beschreibung der einzelnen Templates sei auf [HABH94] oder [Sei94c] verwiesen.
Die Verwaltung der Ressourcen geschieht nun dadurch, daß das eine Ressource modellierende MO mittels der bei dessen Beschreibung definierten Operationen manipuliert

wird. Das betroffene MO stellt also an seiner Dienstschnittstelle (*boundary*) bestimmte Operationen zur Verfügung, die direkt auf das Objekt oder auf seine Attribute einwirken. Operationen, die als Ganzes auf das MO einwirken sind beispielsweise das Erzeugen (*create*) oder das Löschen (*delete*) von MOs. Als attributmanipulierende Operationen seien hier das Lesen (*Get attribute value*), das Schreiben (*Replace attribute value*) und das Initialisieren (*Replace-with-default value*) eines Attributes bzw. seines Wertes angeführt. Die Manipulation eines MO erfolgt über die OSI-Management-Dienstschnittstelle CMIS bzw. über das Managementprotokoll CMIP nach dem *Manager-Agent-Prinzip*.
Es soll nochmals herausgehoben werden, daß das Informationsmodell lediglich einen Beschreibungsrahmen für MOs bereitstellt. Welche Ressourcen letztendlich in eine MIB aufgenommen werden, liegt allein in der Verantwortung des NM-Systembetreibers. Trotzdem sollte deutlich geworden sein, daß sich mit Hilfe des OSI-Beschreibungsrahmens Datenbanken (MIBs) von abstrakten Repräsentationen (MOs) realer oder logischer Ressourcen auf flexible Art erstellen lassen. Daß der OSI-Ansatz jedoch auch Unzulänglichkeiten aufweist, soll im nächsten Abschnitt aufgezeigt werden.

2.3 Defizite des OSI-Konzepts

Das OSI-Informationsmodell sieht die Modellierung von Beziehungen zwischen Ressourcen und ihren abstrakten Abbildern, den MOs, nicht vor. Diese Tatsache begünstigt natürlich die Definition von MOCs, da sich so die MOs auf einem hohen Abstraktionsniveau beschreiben lassen und so unabhängig eine einheitliche logische Sicht auf die Ressourcen möglich ist.
Aus Sicht des MO-Spezifizierers ist dieser Verzicht also sehr bequem, der Implementierer eines MO hingegen bedauert ihn. Ihn interessieren diese Relationen , weil sie möglicherweise Auswirkungen auf seinen Realisierungsentwurf haben. Der Hauptunterschied zwischen der Sicht eines Spezifizierers von MOCs und eines Implementierers eines MO ist dann, daß ersterer seine Sicht auf eine Ressource auf das zugehörige MO beschränkt, während letzterer auch die konkreten Anwendungsprozesse betrachtet und sich überlegen muß, welche Beziehungen zwischen MOs und den zugehörigen Ressourcen bestehen. In [HAB91] werden folgende vier Aspekte angeführt, die sich aus dem Bedürfnis des Implementierers von MOs nach der Möglichkeit des Modellierens von Relationen zwischen MOs ergeben.

1. Verteilungsaspekt (*aspect of distribution*)
2. Aspekt der Lebensdauer (*aspect of lifetime*)
3. Aspekt der Speicherung (*aspect of storage*)
4. Aspekt der Abbildung (*aspect of mapping*)

Die einzelnen Aspekte sollen im folgenden etwas näher beleuchtet werden, um die offensichtlichen Mängel des OSI-Informationsmodells zu verdeutlichen. Dabei wird von einer auf dem Prozeßkonzept basierenden Betriebsystemarchitektur ausgegangen, d.h. die Ressourcen werden betriebsintern durch Prozesse dargestellt. Ein Beispiel aus [Dit91] möge

den Verteilungsaspekt veranschaulichen:
"Es sei angenommen, daß zur Überwachung und Steuerung eines Message Handling Systems gemäß X.400 jeweils eine MOC für die Komponenten Message Transfer Agent (MTA) und User Agent (UA) definiert wird. Bei einer konkreten Implementierung können jedoch MTA und UA durch mehrere Prozesse realisiert sein, so daß auch die Informationen, die in den Attributen eines MO repräsentiert sind, auf verschiedene Prozesse verteilt sein können. Der MO-Implementierer benötigt Hilfsmittel, um zu entscheiden, welche Attribute und Meldungen in einem Prozeß zusammengefaßt sein müssen und wie er bei der Aufsplittung eines MO für die Außenwelt trotzdem die Fiktion eines atomaren MO aufrechterhält."
Der Aspekt der Lebensdauer soll anhand des folgenden Szenarios veranschaulicht werden: Ein Prozeß terminiere wegen eines unbekannten Fehlers. Dann ist es im Rahmen des Fehlermanagements sicherlich notwendig, weiterhin auf in einem MO gespeicherte Informationen zuzugreifen, um die Fehlerursache zu lokalisieren. Es ist also in manchen Fällen unbedingt notwendig, die Lebensdauer eines MO von der Lebensdauer der darunterliegenden Prozesse zu entkoppeln.
Der Aspekt der Speicherung berücksichtigt die unterschiedliche Änderungsdynamik von Attributen. So ist es sinnvoll, dynamische Attribute wie bestimmte Zähler innerhalb der Ressource zu speichern. Statische Informationen werden jedoch besser in einer Managementdatenbasis gehalten.
Der Aspekt der Abbildung beinhaltet die Abbildung der logischen Information eines MO-Teils auf ihren realen Gegenpart.

3 Ansätze zur Erweiterung des OSI-Konzepts

Im vorigen Abschnitt wurde deutlich, daß beim OSI-Konzept eine Unterstützung einer effektiven und effizienten Implementierung der MIB nicht gegeben ist, da dem Modellierer von MOs die Möglichkeit fehlt, sein implementierungsrelevantes Wissen in die MOC-Beschreibung einfließen zu lassen. Die auf den folgenden Seiten vorgestellten Ansätze erweitern das OSI-Konzept der Objektbeschreibungen als Basis für die Objektimplementierung.

3.1 Das Konzept der Composite Managed Objects

Das Konzept der Composite Managed Objects (CMO) berücksichtigt den Verteilungsaspekt und den Aspekt der Lebensdauer, vernachlässigt allerdings den Aspekt der Speicherung und den Aspekt der Abbildung. In [Dit91] wird das CMO-Konzept ausführlich vorgestellt. Dieser Abschnitt soll einen Überblick über diese Arbeit geben.
Das CMO-Konzept geht zunächst davon aus, daß Ressourcen betriebsintern durch Prozesse dargestellt werden. Da die meisten Betriebsystem-Architekturen auf dem Prozeß-Konzept basieren, ist diese Sichtweise durchaus vertretbar. Ziel des CMO-Konzepts ist es also, Relationen zwischen MOs und den assoziierten Prozessen zu modellieren.
Die Grundidee des CMO-Konzepts besteht darin, ein MO derart in disjunkte Teile

(*Parts*) zu zerlegen, daß die Modellierung der Beziehungen zwischen MOs und den zugehörigen Ressourcen auf Relationen zwischen den Parts zurückgeführt werden kann. Solch ein Part ist ebenso wie ein MO durch Attribute, Meldungen und Managementoperationen definiert. Die Ebene der Atomarität, die beim OSI-Konzepts das gesam te MO umfaßt, wird beim CMO-Konzept auf diese definierenden Eigenschaften eines MOs verlagert. Dies wiederum bedeutet, daß jede dieser Eigenschaften genau einem Part zugeordnet wird.
Durch diese Zerlegung eines MOs wird die Modellierung des *Verteilungsaspektes* sehr erleichtert, weil dadurch die Zuordnung von Attributen, Meldungen und Managementoperationen zu einem sogenannten *Resource Object* unterstützt wird. Ein Ressource Objekt ist das Modell eines Prozesses, der eine Ressource realisiert. Dabei wird ein Prozeß durch genau ein Resource Object repräsentiert.
Um auch den *Aspekt der Lebensdauer* zu berücksichtigen, ist beim CMO-Konzept genau ein Part ausgezeichnet, den außerdem jedes CMO beinhalten muß. Dieser *Systems Part* wird getrennt von den Resource Objects in einer speziellen Komponente des Managementsystems gehalten, was ihm eine von den Resource Objects unabhängige Existenz ermöglicht. Der Systems Part beinhaltet diejenigen Teile eines MOs, die auch vor oder nach der Existenz des assozierten Resource Objects verfügbar sein sollen. Diejenigen Bestandteile eines MOs, deren Lebensdauer eng mit der Existenz des zugehörigen Ressource Objects gekoppelt sind, werden in sogenannten *Application-integrated Parts* gehalten. Die Application-integrated Parts enthalten außerdem die Bestandteile eines MOs, die aufgrund ihrer hohen Änderungsdynamik nicht für eine Zuordnung zum Systems Part geeignet sind.

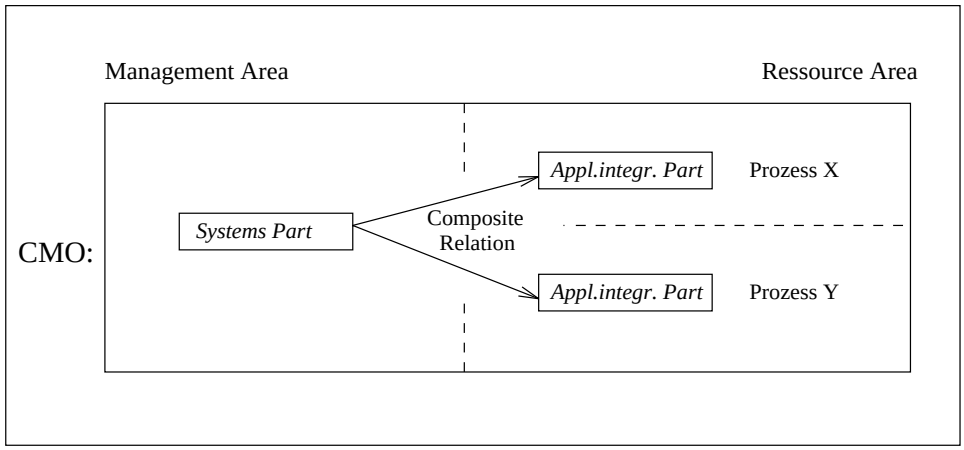

Abbildung 8. Schematischer Aufbau eines CMO

Der Systems Part ist die Schnittstelle eines CMOs zur Managementanwendung, d.h. er stellt alle für das ursprüngliche MO definierte Eigenschaften, wie Attribute, Meldungen und Operationen zur Verfügung. Intern verwaltet der System Part, welche Attribute in welchen Application-integrated Parts liegen, so daß er attributsorientierte Operationen entsprechend weiterreichen kann. Auch für auszuführende Action-Operationen ist dem Systems Part bekannt, in welchem Application-integrated Part die entsprechende Funktion zu finden ist.

Application-integrated Parts haben die Aufgabe, den Zugriff auf die in ihnen vorhandenen Attribute zu ermöglichen und das Eintreten von Ereignissen zu signalisieren.
Es soll hier nochmals herausgestrichen werden, daß durch das Konzept der CMOs nur das Zusammenspiel zwischen den einzelnen Parts eines CMOs unterstützt wird. Die Anbindung an die entsprechenden Resource Objects bleibt weiterhin die Aufgabe des MO-Implementierers.

3.2 Der Ansatz von Hegering, Abeck und Boehnke

Wissend um die Nachteile des CMO-Konzepts, nämlich die Vernachlässigung der Aspekte 3) und 4) präsentierten Hegering, Abeck und Boehnke in [HAB91] eine Methode, die auch den Mapping-Aspekt miteinbezieht. Diese Methode soll im laufenden Abschnitt vorgestellt werden.
Zunächst werden die Begriffe *Fragment* und *Mapping-Types* eingeführt:

1. **Fragment**
   Fragmente sind atomare Teile, die bei der Zerlegung von MOs entstehen und die mit den weiter unten eingeführten Mapping-Types verbunden werden.

2. **Mapping-Types**
   Es gibt mehrere Möglichkeiten, wie Fragmente auf ihre Gegenstücke innerhalb einer realen Ressource abgebildet werden. Durch Kombination der folgenden beiden Kriterien kann man vier unterschiedliche Mapping-Types definieren:

   (a) **Management-initiiert** oder **Ressource-initiiert**
   Management-initiiert bedeutet, daß das Managementsystem auf den aktuellen Inhalt eines Fragments innerhalb einer Ressource lesend oder schreibend zugreifen möchte. Ressource-initiiert bedeutet, daß die Ressource dem Managementsystem ein Ereignis meldet oder einen aktuellen Inhalt eines Fragments zur Verfügung stellt.

   (b) **Direkt oder Indirekt**
   Direkte Abbildung bedeutet, daß der Inhalt eines Fragments ausschließlich innerhalb der Ressource gespeichert wird. Indirekte Abbildung bedeutet demnach, daß das Managementsystem eine Kopie dieses Inhaltes in einer Management-Datenbasis aufbewahrt.

Kombiniert man die beiden Kriterien miteinander, so erhält man vier unterschiedliche Mapping-Types:

- management_initiated_direct
- management_initiated_indirect
- resource_initiated_direct
- resource_initiated_indirect

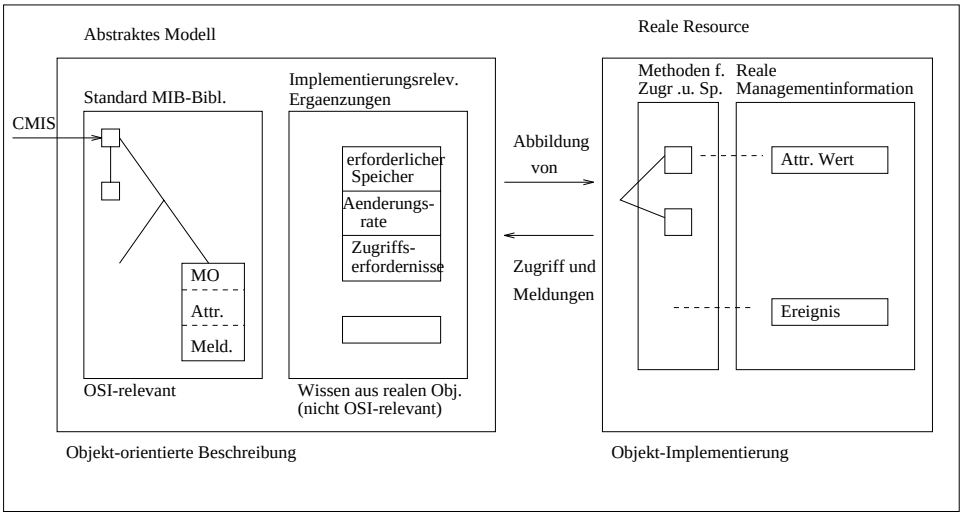

Abbildung 9. Abbildung der objektorientierten Welt in die reale Welt

Die Methode von Hegering, Abeck und Boehnke ist in Abbildung 9 dargestellt und läßt sich mit den eingeführeten Begriffen wie folgt beschreiben:

1. Zerlegung einer MOC in Hinsicht auf die Mapping-Types

2. Systematische Verknüpfung von Mapping-Types mit den Fragmenten gemäß bestimmter Kriterien und Regeln.
   Dieser Punkt hängt stark von den Eigenschaften der einzelnen Fragmente ab. Beispiele für Regeln, wie in [HAB91] aufgeführt, sind:

   – **change rule**: Attribute mit einer hohen dynamischen Änderungsrate benötigen den Mapping-Type management_initiated_direct.

   – **lifetime rule**: Alle Fragmente mit einer von der realen Ressource abweichenden Lebensdauer werden durch management_initiated_indirect oder resource_initiated-_indirect abgebildet.

   Die Autoren von [HAB91] schlagen in diesem Zusammenhang eine Erweiterung der MOC-Bibliotheken um ihre Regeln vor, welche als Gestaltungs-Richtlinien für die Entwickler von Managementanwendungen *und* Modellierer von Ressourcen dienen sollen.

3. Eintragung der Implementierungszusätze in die MOCs mit Hilfe eines erweiterten OSI-Beschreibungsrahmens. Die Erweiterung besteht dabei in der Möglichkeit, innerhalb eines Templates den Attributen einen der vier Mapping-Types zuzuorden.

4. Definition der realen Zugriffsmethode für die ausgewählten Mapping-Types

Die erweiterten Objekt-Klassen-Definitionen aus den Schritten 3) und 4) sollen dem Entwickler von Managementanwendungen als Implementierungsempfehlungen dienen.

3.3 Einführung eines zusätzlichen Template-Typs

Einen von den beiden obigen Konzepten abweichenden Ansatz schlugen Hegering, Abeck, Boehnke und Heiler 1994 in [HABH94] vor. Er sieht die Einführung eines zusätzlichen, zehnten Template-Typs vor, der die implementierungsrelevanten Eigenschaften der Ressourcen beinhaltet. Diese Eigenschaften sollen den Autoren zufolge folgende Implementierungsbereiche abdecken:

- klassenspezifische Verwaltung
- Zusammenspiel des logischen Objekts mit der realen Ressource

Beide Bereiche lassen sich weiter in Teilbereiche in Form von Aspekten aufbrechen, die durch Felder in dem neu eingeführtem Template spezifiziert werden können. Diese Aspekte ermöglichen es, dem Implementierer für die Aufgabe des Object und Ressource Mapping wichtige Eigenschaften vorzugeben.
Die klassenspezifische Verwaltung läßt sich durch folgende Teilbereiche verfeinern:

- Instantiierung (*creation*)
- Löschung (*deletion*)
- Namensgebung (*naming*)
- Anzahl (*count*)
- Abhängigkeiten (*dependencies*)
- Stukturelle Beziehungen (*relations*)

Das Zusammenspiel des logischen Objekts mit der realen Ressource umfaßt nachfolgende Aspekte:

- Lebensdauer (*lifetime*)
- Zugriffsform (*access*)
- Aktualisierungstrategie (*update*)
- Fehler beim Zugriff (*access-error*)

Damit gestaltet sich der *Implementation Aspect*-Templatetype wie in Abbildung 10 dargestellt.

Natürlich sind alle Aspekte innerhalb des Templates optional, da nicht jeder Aspekt für jede Objektklasse relevant ist. Die Einbettung des Implementation Aspect-Templates in den OSI-Beschreibungsrahmen geschieht dadurch, daß in den bestehenden Templates der Objektklassen und Attribute ein entsprechender Verweis auf das neue Template "IMPL-ASPECTS" einzuhängen ist. Diese Erweiterung ist vollständig in den bestehenden OSI-Beschreibungsmechanismus nach GDMO integrierbar. Das besondere an dieser Methode ist ihre *Aufwärtskompabilität* bezüglich des existierenden OSI-Beschreibungsrahmens. Damit können bereits bestehende Objektkataloge problemlos weiterverwendet werden.

```
<impl-label>          IMPL-ASPECTS

    [OBJ-ADMINISTRATION
        [CREATION          <creation-statement>  ;]
        [DELETION          <deletion-statemetn>;]
        [NAMING            <naming-authority>;]
        [COUNT             <number-of-objects>;]
        [DEPENDENCIES      <dependencies-statement> ;]
        [RELATIONS         <relation-statement> ;]
        [RECOVERY          <recovery-statement> ;]
    ]

    [RES-COUPLING
        [LIFETIME[         <lifetime-statement> ;]
        [ACCESS            <access-statement> ;]
        [UPDATE            <update-statement> ;]
        [ACCESS-ERROR      <error-statement>  ;]
    ]

REGISTERED AS   object-identifier ;
```

Abbildung 10. *Implementation Aspect*-Templatetyp

4 Schlußbetrachtung

Der OSI-Beschreibungsrahmen ist trotz seiner Mächtigkeit in seinem jetzigen Zustand für die Implementierung von MIBs nicht ausreichend. Zu schwer wiegt das Fehlen von Beschreibungsmöglichkeiten der implementierungsrelevanten Eigenschaften der MOs. Genau hier setzen die im Abschnitt 3 vorgestellten Arbeiten an. Sie bieten eine Erweiterung des OSI-Beschreibungsrahmens, um die unterschiedlichen Beziehungs-Aspekte zwischen den realen Ressourcen und ihren abstrakten Abbildern, die den Implementierer interessieren, mit in die Modellierung von MOs aufzunehmen.
Das CMO-Konzept scheint noch unausgereift zu sein, berücksichtigt es doch nur den Aspekt der Lebensdauer und den Aspekt der Verteilung. Die in 3.2 vorgestellte Arbeit bezieht außerdem den Aspekt des Mapping mit ein. Beide Ansätze kranken allerdings daran, daß sie bezüglich des bisherigen OSI-Beschreibungsrahmens inkonsistent sind. Das bedeutet, daß es nicht möglich ist, bereits bestehende Objektkataloge mit diesen Ansätzen zu verbinden. Diesbezüglich ist die letzte Methode, die Einführung eines zusätzlichen Template-Typs, die zukunftsträchtigste der drei behandelten Ansätze. Sie berücksichtigt die wesentlichen Implementierungsaspekte der Verteilung, Lebensdauer, Speicherung und Abbildung und löst das Problem der Konsistenz auf elegante, bestechend einfache Art. Bleibt abzuwarten, ob ihr es letztendlich vergönnt sein wird, in die neuen Beschreibungsrichtlinien von Objekten der ISO aufgenommen zu werden. Daß ei-

ne Erweiterung des momentanen Beschreibungsrahmens vonnöten ist und wohl auch in absehbarer Zeit von der ISO initiiert werden wird, scheint jedoch sicher.

Management Verteilter Anwendungen: Der PRISMA-Ansatz

Xiao-Qun Sun

Kurzfassung

In diesem Aufsatz wird ein neues Framework (Teil des PRISMA-Systems) vorgestellt, das die Selektion von ,,managed objects" und deren Integration in verteilten Anwendungen unterstützt. Diese neue Methode unterstützt ein integriertes Framework für Spezifikation und Implementierung der Anwendung, des Managements und kommunikationorientierter Aspekte. Es wird gezeigt, wie man ein ,,managed object" von der Anwendungsspezifikation erhält, und wie eine Ergebnishierarchie durch Analysieren der Anwendungsspezifikation automatisch erzeugt wird.

1 Motivation

Mit der immer stärkeren Verbreitung von Kommunikationsinfrastrukturen, die Rechner aller Größenordnungen und Zweckbestimmungen einschließen, wird das Management von einer fast exponentiell wachsenden Anzahl vernetzter Rechner mit einer wachsenden Heterogenität zu einem Problem von breitem theoretischen und praktischen Interesse.

Es hat sich im Bereich der Entwicklung verteilter Anwendungen herausgestellt, daß auch hier Bedarf für Managementfunktionalität besteht (Anwendungsmanagement).

Wenn mehrere Datensätze, die auf verschiedenen Datenbanken in unterschiedlichen Datenformaten abgelegt sind, von einem Rechner verarbeitet werden sollen, stellt sich folgendes Problem:

Die Daten werden importiert, Datenformate angeglichen, erst dann kann mit der eigentlichen Aufgabe begonnen werden. Oft reicht das Leistungspotential eines Rechners nicht aus und/oder manche Hardwarekomponenten des Rechners werden nicht ausgenutzt.

Das führt dazu, daß Aufgaben nicht gelöst werden können bzw. die Laufzeit mancher Programme unnötig lang ist, da vorhandene Hard- und Software nicht optimal genutzt wird.

Man sollte gegebene Aufgaben so auf verschiedene Rechner verteilen, daß eine möglichst umfassende Gesamtfunktionalität der Systeme entsteht. Durch eine geschickte Verteilung der gegebenen Last erreicht man eine bessere Ausnutzung der vorhandenen Ressourcen und eine Reduzierung der Bearbeitungszeit.

Dies führt uns zum Begriff der verteilten Anwendung. Unter verteilten Anwendungen werden genauer Anwendungsprogramme verstanden, die aus verschiedenen Prozessen bestehen, die ihrerseits auf verschiedenen Rechnern plaziert sind und im Rahmen der Anwendungsbearbeitung miteinander kommunizieren.

Sie kommunizieren zur Laufzeit untereinander und stellen spezielle Anforderung in Bezug auf die Synchronisation der inhärent parallelen Abläufe. Jede Anwendung hat ihre

eigenen Kontrollfunktionen integriert. Die einzelnen Prozesse agieren weitgehend autonom und wickeln ihre Kommunikation nicht etwa über eine zentrale Komponente wie z.B. einen Großrechner ab.

Bei dem Versuch, eine möglichst umfassende Gesamtfunktionalität auf der Basis verschiedener, verteilter Einzelfunktionen zu erbringen, können durch eine komplexe, irreguläre Topologie Komplikationen entstehen.

Also müssen für verteilte Anwendungen Managementmöglichkeiten (Anwendungsmanagement) realisiert werden, um die Anwendung rekonfigurieren oder debuggen zu können. Das Systemmanagement muß um den Kommunikationsaspekt erweitert werden, da ein verteiltes Betriebssystem auf Kommunikationsdiensten aufbaut.

Zu diesem Thema werden im Moment eine Reihe von Forschungsaktivitäten betrieben, welche sich mit unterschiedlichen Teilproblemen wie z.B. Quality of Service, Konfigurationsverwaltung oder Integration von Trading und Managementfunktionalität in verteilten Anwendungen beschäftigen.

2 Die Alternativen

Es gibt nun mehrere Verfahren für die Verwaltung verteilter Anwendungen, obwohl dies verhältnismäßig neu ist.

Ein Ansatz wäre es, die Managementaufgaben in getrennten Managementschnittstellen zu definieren, die Bestandteil der Komponenten sind. ([WMC93])

Ein anderer Ansatz ist ,,Meta", bei welchem man Anwendungen mit *Sensors* und *Actuators* ,,ausrüsten" kann.(siehe Bild 11) *Sensors* erfassen zuerst die benötigen Daten, dann erzeugen *Actuators* die entsprechende Aktionen. ([WMC93])

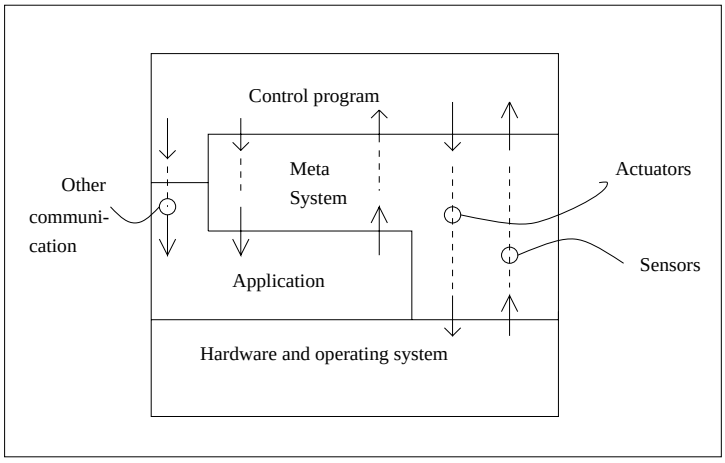

Abbildung 11. Meta Anwendungsmanagement

Aber diese Lösungen haben den Nachteil, daß die Managementaufgaben komponentenspezifisch aufgeteilt sind und daher das System an Granularität verliert. Desweitern entsprechen diese Methoden nicht den OSI-Netzwerkmanagement-Standards.(siehe Bild 12 [Sch95])

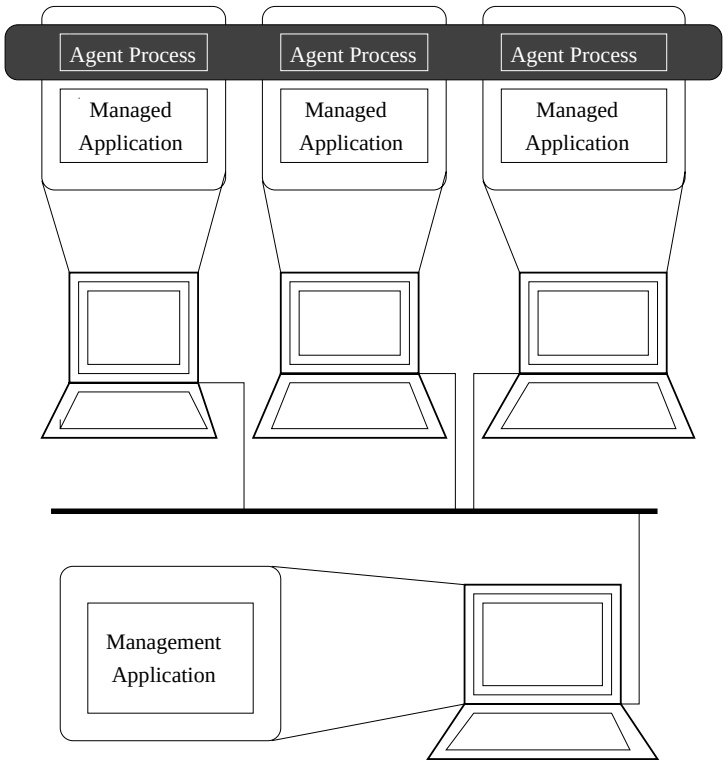

Abbildung 12. OSI-Management einer verteilten Anwendung

3 Der PRISMA-Ansatz

3.1 Erläuterung

PRISMA (A Plattform for Integrated Construction and Management of Distributed Applications) ist ein Teil eines Projektes, das in Zusammenarbeit mit Digital Equipment Corporation realisiert wurde. Die folgende Methode ist ein Teil des Systems PRISMA. Das Ziel dieser Methode ist es, ein integriertes Framework für Spezifikation und Implementierung der Anwendung, des Managements und kommunikationorientierter Aspekte zu unterstützen.

Verteilte Anwendungen sind hier definiert als ein Verbund von interaktiven Anwendungskomponenten. Eine Komponente setzt sich aus Kommunikationseinheiten, Schnittstellen und Interaktionsmodulen zusammen. Die Struktur einer verteilten Anwendung wird getrennt, durch eine Konfigurationsbeschreibung, bestimmt.

Dauerbetriebene Anwendungen stellen das Hauptproblem für das Management verteilter Anwendungen dar, da spezielle Mechanismen zur Rekonfiguration von Nöten sind. Während eines laufenden Programms sollen Änderungen vorgenommen und gleich eingebunden werden können, ohne größere Störungen bei den laufenden Anwendungen hervorzurufen, oder gar deren Konsistenz zu gefährden.

Bei dieser Technik, die ich Ihnen jetzt vorstelle, werden die Rekonfigurationsmöglichkeiten als ein integraler Bestandteil schon in der Anwendungsspezifikation formuliert.

Um Managementaufgaben darstellen zu können, muß ein Verbund von Funktionen zur Verfügung stehen, der es emöglicht, Konfigurationen einzurichten, zu überwachen, zu

kontrollieren und zu beenden.

Die Grundidee des Ansatzes ist, die benötigten ,,managed objects" (MOs) direkt aus den Anwendungsspezifikationen herzuleiten.

3.2 Spezifikation

Basisbausteine Die Basisbausteine eines Anwendungsentwurfs sind Kommunikationskontext, Schnittstellen und Komponenten. (siehe Bild 13)

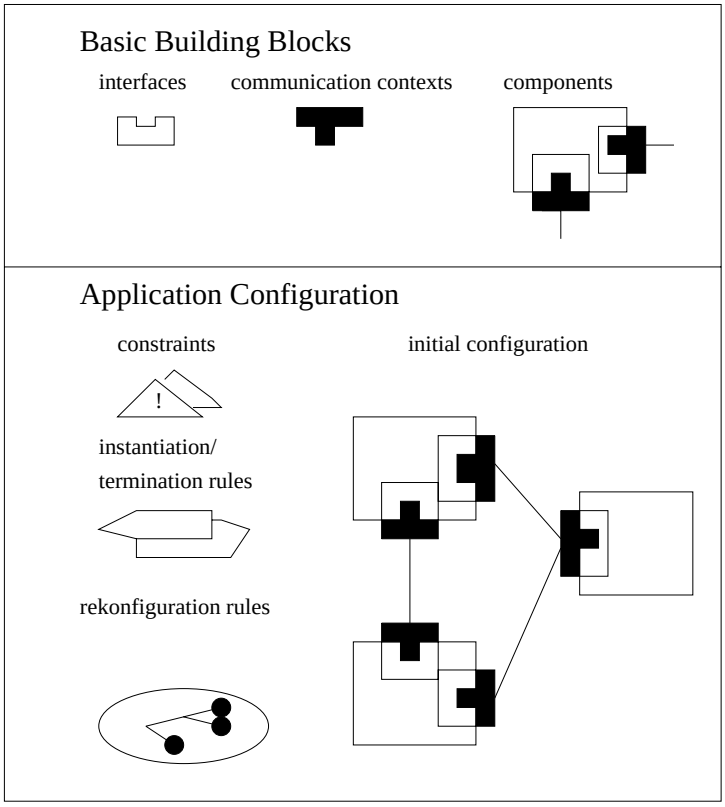

Abbildung 13. Verteiltes Anwendungsmodell

– Kommunikationskontext

  Ein Kommunikationskontext spezifiziert die Kommunikationsanforderung einer verteilten Anwendung. Dafür gibt es eine spezielle Sprache, die es ermöglicht, die kommunikationsorientierten Aspekte wie Kommunikationsrelationen, interaktive Typen und Eigenschaften der Transportdienste (Art und Menge der zu transportierenden Daten) zu spezifizieren.

– Schnittstelle

  Schnittstellen repräsentieren die Interaktionspunkte der Komponente und beschreiben alle möglichen Interaktionen zwischen kooperierenden Komponenten, d.h. die Dienste, die Komponenten anbieten und nutzen.

Nach der Interaktionsrichtung unterscheiden wir zwischen symmetrischen und asymmetrischen Schnittstellen. Eine symmetrische Schnittstelle ermöglicht eine duplexe Kommunikation. Eine asymmetrische Schnittstelle verdeutlicht man sich am besten am Beipspiel einer *client-server*-Beziehung.

Welche Operation auf welcher Schnittstelle und bei wem durchgeführt werden soll, muß festgelegt werden. Dazu benutzt man Rollen. Rollen stellen statische oder dynamische Teilnahme- und Interaktionsvorschriften dar. Jeder Komponente wird eine oder mehrere statische Rollen zugeordnet, die ihre spezifische Rechte beschreibt, die sich während der Laufzeit nicht ändern. Dynamische Rollen bestimmen allgemeine Interaktionsvorschriften, wie z.B. Synchronisation und exklusive Zugriffe auf gemeinsame Daten. Jede Komponente kann solche Rollen anfordern und zurückgeben.

- Komponenten

  Komponenten sind entweder interaktive Serviceanbieter oder sie führen irgendwelche Zwischenfunktionen durch. Sie sind bestimmt durch Schnittstellen und Kommunikationskontext.

  Die Schnittstellen können symmetrisch oder asymmetrisch sein. Bei asymmetrischer Schnittstellen soll das Verhalten der Komponente deterministisch festgelegt werden, d.h. entweder Dienstnehmer oder Dienstgeber. Bei symmetrischer Schnittstellen soll es sowohl Dienstgeber als auch Dienstnehmer sein.

  Die Kommunikationsanforderungen einer Komponente werden durch die Kommunikationskontexte zur Schnittstelle beschrieben.

Konfiguration Konfiguration einer verteilten Anwendung setzt sich aus Konfiguration und Rekonfigurationsaktivitäten zusammen.

- Konfiguration

  Eine Anwendungskonfiguration enthält die Komponenteninstanz und Bindungen zwischen ihren Schnittstellen. Außer bei einer ,,single" Komponenteninstanz ist es möglich, eine Komponentengruppe von variabler Größe zu bilden.

- Rekonfiguration

  Rekonfigurationsaktivitäten sind definiert als eine Reihe von ereignissteuerten Zustand-Aktion-Mustern. Rekonfigurationsaktivitäten können durch Instantiierungs-, Terminierungs- und Rekonfigurationsregeln spezifiziert werden. Instantiierungs- und Terminierungsregeln sorgen für die Konsistenzerhaltung.

  Rekonfigurationsregeln werden für die Managementaktivitäten benutzt. Solche Regeln sind für die Behandlung spezifischer Ereignisse (z.B. Ausfall eines Knotens, Überlastung einer Serverkomponente) notwendig.

Beispiel Am Beispiel eines vereinfachten *client-server*-Modells werde ich jetzt diese Spezifikationstechnik verdeutlichen. (siehe Bild 14)

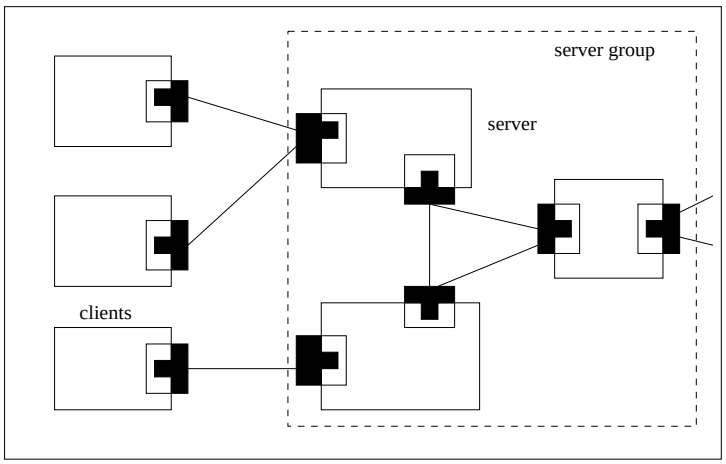

Abbildung 14. Beispiel einer verteilten Anwendung

```
Server COMPONENT
  APPLICATION PROPERTIES
    INTERFACES
      CONSUMER SUPPLIER AT groupAccess: GroupInt
                { MULTIPLE BINDING }
      CONSUMER AT fileAccess: fileAccessInt
                { MUTIPLE BINDING }
    RESOURCES
      mainMemory > 8 MB
              ....
  COMMUNICATION PROPERTIES
    groupCommContext AT groupAccess

              .....
```

Abbildung 15. Beispiel einer Komponentenspezifikation

1. Komponenten

   Das Bild 15 zeigt einen *file server*. Ein Server ist eine reagierende Komponente, ein Anbieter spezieller Operationen. Er setzt sich aus zwei Schnittstellen „*groupAccess*" und „*fileAccess*" zusammen. Über die erste läuft die Kooperation mit den anderen Gruppenmitgliedern, die zweite ermöglicht den *clients* Zugriffe auf ihre Ressourcen. Unter Ressourcen verstehen wir die Umgebungsanforderungen, wie benötigten Hauptspeicher, Prozessoren und die Existenz spezieller Einrichtungen.

   Die Eigenschaften der Bindungen legen die Kardinalität (*single/mutiple*) und die Komponentenbezüge der Bindungen beim Initialisierungsaufbau fest. Außerdem können auch Zeitpunkte für Initialisierung und Terminierung der Bindungen festgelegt werden. Im weiteren gehen wir davon aus, daß die Bindung *client-server* zum Zeitpunkt des Vorgangs, oder der Erzeugung der Komponente, existent ist.

Die Managementkomponente benutzt die Informationen des Abschnitts ***RESOURCES*** um die Anwendungskomponenten während der Konfigurations-/Rekonfigurationsaktionen, auf passende Computerknoten zu verteilen.

2. Initiale Anwendungskonfiguration

   Eine Konfigurationsspezifikation beschreibt die Anwendungskomponententypen, die Komponenteninstanzen, die Rollen sowie die Art der Verknüpfungen zwischen den Komponenten. Optional kann eine Menge von Konfigurationsvorschriften angegeben werden.

```
clientServerApplication DISTRIBUTED APPLICATION
  COMPONENTS
     clients[]: Client {
        INSTANTIATION BY { user1, user2, ...}
                }
     servers[]: Server {
        INSTANTIATION REQUIRES ROLE Manager
        INITIAL MEMBERS n1
                }
  BINDINGS
     FOR ALL c: clients
         c -- ANY OF s IN servers
  CONSTRAINTS
     NODES FOR servers { mars, saturn, ...}
```

Abbildung 16. Konfigurationsspezifikation

Betrachten wir Bild 16 :

Im Abschnitt „***COMPONENTS***" werden die initialen Komponenteninstanzen der *client-server*-Anwendung definiert.

Eine Komponentengruppe ist zur Spezifizierung einer Sammlunng von *client-* und *server*-Einheiten gedacht. So einer Gruppe wird eine Anfangsmenge von Elementen und Rollen zugeordnet. In unserem Beispiel sind die *clients* ein interaktiver Teil der interaktiven Anwendung. Wir können sogar vorbestimmen, wer eine *client*-Komponenteninstanz erzeugen darf.

Im Abschnitt „***BINDINGS***" können Bindungen explizit oder implizit spezifiziert werden. Im ersten Fall muß eine Bindungsdeklaration, die konkrete Komponenteninstanzen enthält, zur Spezifikationszeit angegeben werden. Im zweiten Fall sind die Bindungen implizit in den Komponenteneigenschaften definiert, was zur Folge hat, daß das Konfigurationsmanagement beim Bindungsaufbau eine passende Komponente dynamisch auswählen muß. Dies ermöglicht eine hohe Flexibilität, da die Bindungsspezifikation unabhängig von der momentanen Konfiguration ist.

Zur besseren Darstellung von Konsistenzanforderungen kann man ,,*constraints*" verwenden. Sie definieren die Invarianten der benötigten Komponenten und/oder Bindungen. Die Plazierungsdaten, also die Informationen, welche Komponenten auf welche Knoten gelegt werden, sind in den ,,*placement constraints*" spezifiziert. Bild 16: Ein einfaches *constraints* beschreibt die Plazierung von *server*-Komponenten.

collocation-Definition: Sie beschreiben plazierungsbezogene Relationen zwischen aufgerufenen Komponenten (*supplier polarity*) und Benutzer-Komponenten (*invoking component*).

Durch diese Methode können Plazierungen dynamisch vorgenommen werden. So erhält man mobile Komponenten, die während eines bestimmten Intervalls der Laufzeit, zum Benutzer wandern.

3. Rekonfigurationsregeln

   Als Teil der Konfigurationsspezifikation überwachen die Rekonfigurationsregeln Konfigurationsaspekte. Rekonfigurationsoperationen spielen die Rolle der Aktionen.

   Diese Architektur basiert auf der Idee, zwischen Managementkomponenten, Repräsentation und Manipulation der Wächer, zu trennen.

```
clientServerApplication DISTRIBUTED APPLICATION
  COMPONENTS ...

    ...
  RECONFIGURATION RULES
    CONDITIONS
        shutdown: prepareForShutdown (NODE n) AND
              // event from network management
        NOT EMPTY serverSet = { s1: servers WITH (
              s1 HAS LOCATION n AND
              s1 HAS PROPERTY migration AND
              NOT EMPTY nodeset= { m: NODES WITH (
                    memory(m) >= memory_needs(s1) AND
                    adminState(m) = active )})}
    TRANSACTIONS
       FOR ALL s: serverSet
             SELECT n1 FROM nodeset WITH
                        (memory(n1) = max(memory(nodeset))
             migrate s TO n1;
       readyForShutdown; //notification to network mgmt
```

Abbildung 17. Beispiel einer Rekonfigurationsregel

Bild 17 zeigt eine einfache Rekonfigurationsregel, für eine *client-server*-Anwendung, bei der ein "*shutdown*" vorbereitet wird. Der Leitgedanke hinter dieser Regel ist, daß die Komponente auf einen Ausweichknoten, der genug Ressourcen anbietet, wandert.

Im Abschnitt „***CONDITIONS***" werden zustandsabhängige Aktionen ausgewählt, die unter „***TRANSACTIONS***" beschrieben sind. Befinden sich *servers* auf dem betroffenen Knoten wird, wie oben beschrieben, ein „Exilknoten" ausgewählt, die Komponenten verschoben und dem Anwendungsmanagement, durch eine *ReadyForshutdown*"-Operation, Vollzug gemeldet.

3.3 Managed Objects

Um ein System von verteilten Anwendungen zu nutzen, benötigt man ein geeignetes Management, welches die Anwendungen errichtet, überwacht, kontrolliert, verändert und schließlich wieder beendet.

In unserem Anwendungsmodell betrachten wir Anwendungskomponenten, Komponentengruppen, Schnittstellen, Rollen und Kommunikationskontexte als *managed objects*.

Das Ziel ist ein maßgeschneidertes Managementsystem, das in jede verteilte Anwendung integriert werden kann.

Hierarchie Um die Managementeigenschaften der verschiedenen Anwendungs- und Kommunikationsbausteine darzustellen, entwickeln wir eine Klassenhierarchie. Dabei werden die für das Anwendungsmanagement benötigten Instantiierungs-, Status-, Bindungs- und Rollen-Managementfunktionen identifiziert und berücksichtigt. (siehe Bild 18)

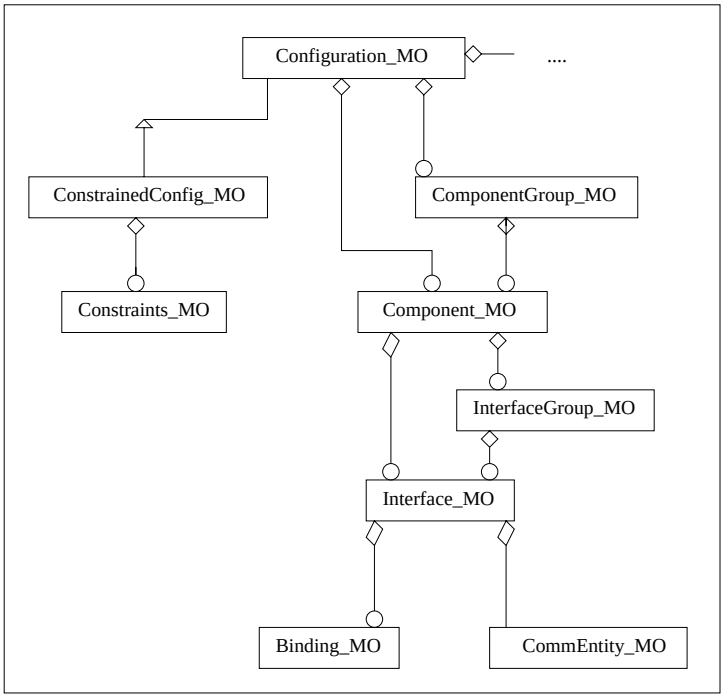

Abbildung 18. MO-Klassenhierachie

Die Generalklasse *Configuration_MO* erleicht die Verwaltung der Komponenten_MOs. *ConstrainedConfig_MO* repräsentiert eine *constraints*-bezogene Konfiguration.

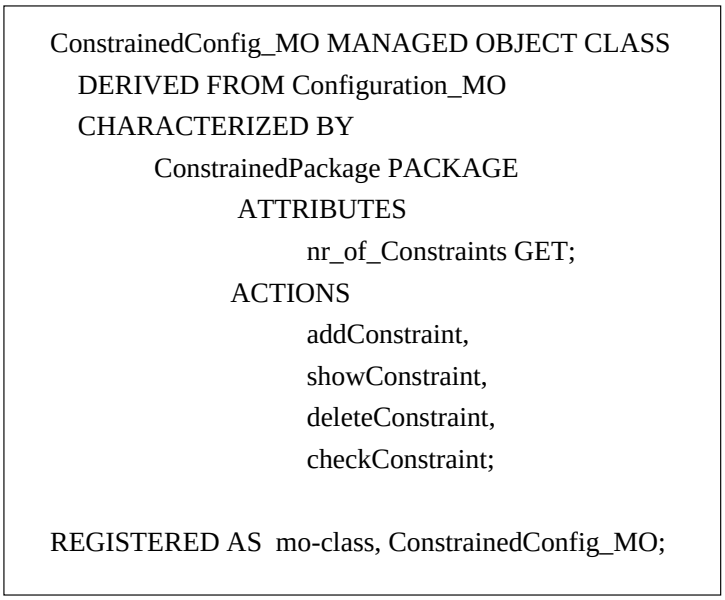

Abbildung 19. *Constrained configuration MO specification*

Bild 19 zeigt zusätzlich Operationen zum Einfügen und Löschen von *constraints* und eine Überprüfung der aktuellen Konfiguration auf Gültigkeit.

Auswahl und Einbinden von MOs Die Auswahl und Initialisierung eines *configuration_MO* für eine konkrete Anwendung ist abhängig von der Konfigurationsspezifikation. In unserem Beispiel benötigen wir eine Menge von *client* MOs, ein Servergruppen MO und ein MO für die Verwaltung der Rekonfigurationsregeln. Dazu kommt noch ein *placement constraint* MO für die Ausweichknoten.

Um Auswahl und Instantiierung zu automatisieren, werden momentan, basierend auf Analysen von Anwendungsspezifikationen, Mechanismen entwickelt. So kann z.B. man aus der Anwendungskonfiguration ablesen, ob ein einfaches *Configuration_MO* oder ein *ConstrainedConfig_MO* benötigt wird.

Grundzüge zur Bestimmung und Integration von MOs Kommunikationskontexte, Schnittstellen, Komponenten und Anfangskonfiguration werden bestimmt. Aus einer MO-Klassen-Bibliothek, die eine Sammlung von Grundbausteinen für Anwendungen (auch in C++ erhältlich) enthält, werden passende MO-Klassen ausgewählt und instantiiert.

Die Auswahl, Instantiierung und Initialisierung der passenden MOs wird durch eine Anzahl von Tools unterstützt.

3.4 Implementierung

C++ wird für die Implementierung verschiedener Bausteine benutzt.

Managementkomponente Sie bietet ihre Funktionen über eine graphische Benutzerschnittstelle an.

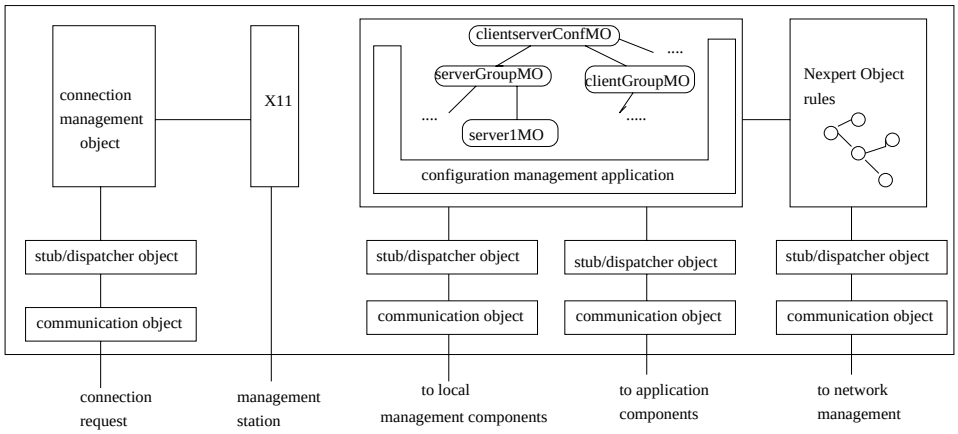

Abbildung 20. *Architektur einer managementkomponente*

Bild 20 erläutert grob die Architektur einer solchen Komponente:

- *connection management* Verbindungsmanagement
- *information base* Datenbanken für die Verwaltung der MO's
- *configuration management application* Konfigurationsmanagement
- Kommunikationsinfrastruktur für die Interaktion mit laufenden Anwendungen
- graphische Benutzerschnittstelle zur Ereignisverwaltung und graphischen Darstellung

1. Verbindungsmanagement

   Es wird benötigt, da die Kooperation zwischen Management und Anwendungskomponenten durch einen verbindungsorientierten Mechanismus realisiert wird.

2. *configuration management application*

   Sie reagiert auf Ereignisse, die vom Benutzer über das graphische Interface, oder von externen verteilten Anwendungen über Kommunikations- und Melder-Objekte ankommen.

   Es kann verbundene MOs modifizieren und/oder Zugriffsrechte vergeben.

   (a) Konfigurations-MO (*configuration MO*)

   Ein Konfigurations-MO wird in C++ als Containerklasse, die Komponenteninstanzen, Plazierungsinformationen und Regeln enthält, implementiert. Operationen wie Einfügen, Löschen und Zugriffe auf MOs sind angegeben.

   (b) Komponenten MO

   Ein Komponenten MO ist ein Teil eines Konfigurations MO's und besteht im wesentlichen aus drei Teilen:

   - Konfigurationsinformation
   - Zugang zur laufenden Anwendungskomponente

– graphische Repräsentation

Konfigurationsinformationen sind die Informationen der Schnittstellen, der Rollen und der entsprechenden Typen. Der Zugang zur laufenden Anwendungskomponente ist die Referenz zu einem Kommunikationsstack und einem Codierungs-/Decodierungsobjekt. Die graphische Repräsentationen stellen die Benutzerschnittstelle dar.

(c) Nexpert object

Die für Rekonfigurationsaktionen zuständige *management application* basiert größtenteils auf einer Expertensystem Shell. Wir nutzen in diesem Fall *Nexpert Object* als Oberfläche für ein regelbasiertes System. Zuerst werden die Rekonfigurarionsregeln für *Nexpert object* transformiert bevor sie durch den *built-in rule interpreter* ausgeführt werden.

Anwendungskomponenten Eine Anwendungskomponente setzt sich zusammen aus anwendungsorientierten, managementorientierten und kommunikationorientierten Klassen.

Anwendungorientierte Klassen enthalten Codierungsprotokoll- , Decodierungsprotokoll-, Kooperationsprotokoll- und Anwendungsklassen.

Um die Managementfunktionalität in die Anwendungskomponenten zu integrieren, werden die abstrakten MO-Spezifikationen in C++ Klassen abgelegt. Dazu gibt es zwei Verfahrensweisen:

1. MOs als selbständige Objekte

2. Managementfunktionalität wird in der anwedungs- und kommunikationorientierten Klassen integriert

Die zweite Alternative führt zu einer Vererbungshierarchie von Anwendungs- und Managementklassen.

Bild 21 zeigt eine anwendungtransparente Integration des Bindungsmanagements. Es ist realisiert durch die Integration eines Bindungsobjekt in der Archietektur der Anwendungskomponente. Operationen für das Bindungsmanagement sind *bind, unbind.* Die Anwendung kann auf Bindungsinformationen zugreifen (*Stub object*), um so Adressen von gesuchten Komponenten zu finden.

Ein *object-I/O* Subsystem ermöglicht die ,,*save*" und ,,*restore*" Operationen für Komponenteninstanzen.

3.5 Managementarchitektur

Es gibt verschiedenen Managementarchitekturen für verteilte Anwendungen. Wir unterscheiden:

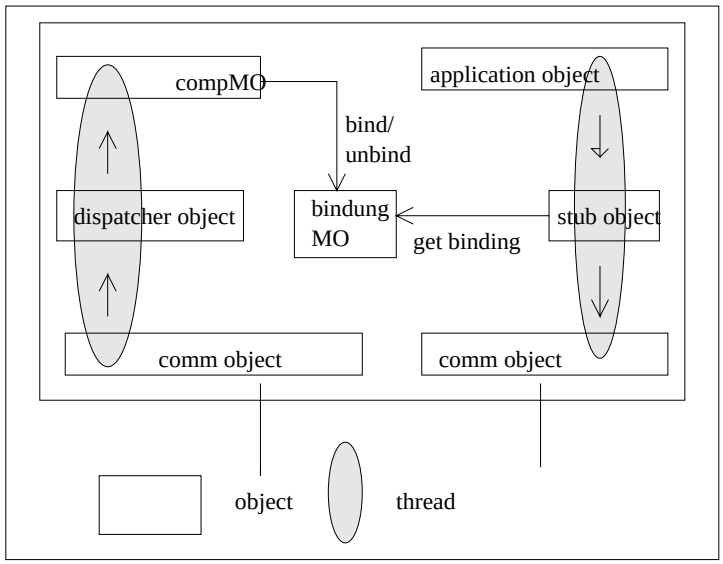

Abbildung 21. *Integration of management properties*

Zentralisierte Managementarchitektur

- Definition

  Eine global Managementkomponente verwaltet alle Anwendungskomponenten und ihre Verbindungen.

  Um eine Komponente auf anderen Rechneren zu erzeugen, brauchen wir zusätzlich einen lokalen Manager für jeden Rechner.

- Nachteile

  Für komplexe verteilte Anwendungen ist eine zentralisierte Managementarchitektur nicht geeignet.

  Begründung:

  Alle Managementinformationen verwaltet eine zentrale Komponente, die auf einem einzigen Rechner installiert ist.

  Mögliche Folgen:

  - *overload* auf dem lokalen Rechner
  - langsame Reaktionen von der Managementkomponente
  - Bei einem Ausfall der zentralen Managementkomponente, ist das ganze System nicht mehr funktionsfähig.

Dezentralisierte Managementarchitektur Die Architektur eines dezentralisierten Systems ist in drei Ebenen aufgeteilt:

- Anwendungsebene
- Managementebene

– Benutzeroberfläche

Die verteilten Anwendungen liegen auf der Anwendungsebene. Die Kernkomponenten bilden die Managementebene des ganzen Modells.

Innerhalb dezentralisierter Architekturen kann man noch zwischen zwei Methoden unterscheiden:

1. Alle Manager verwalten die gleiche Menge von Anwendungskomponenten. Es gibt keine direkte Verbindungen zwischen den Managementkomponenten.

2. Manager verwalten disjunkte Mengen von Anwendungskomponenten. Und es existieren direkte Verbindungen zwischen Managementkomponenten.

Die zweite Methode hat den Vorteil, daß jede Anwendungskomponente explizit einen Manager hat. Das erleicht die Integration. Deswegen bevorzugt man diese Variante.

– Kernkomponenten

  Jede Kernkomponente verfügt über eine interne Darstellung der Anwendungsebene. Die Menge der Anwendungskomponenten wird disjunkt auf die Kernkomponenten aufgeteilt.

– Verteilungssynchronisation

  Die erste Kernkomponente erhält eine Operation bestehend aus einer Liste von Teilaufgaben. Daraus wählt sie sich einige Teilaufgaben aus und streicht diese aus der Liste. Die Kernkomponente erzeugt daraufhin eine ,,Beschäftigt"-Liste und trägt sich in diese ein. Anschließend gibt sie die Liste mit den restlichen Teilaufgaben, sowie die ,,Beschäftigt"-Liste, an die nächste Kernkomponente weiter. Diese übernimmt weitere Teilaufgaben und trägt sich ebenfalls in die ,,Beschäftigt"-Liste ein. Dieser Vorgang wird fortgesetzt, bis entweder alle Kernkomponenten beschäftigt, bzw. keine Teilaufgabe mehr zu vergeben sind.

– Gegenseitiger Ausschluß

  Fordert eine Kernkomponente exklusiven Zugriff auf eine Menge von MOs an, fertigt sie zuerst Kopien der betroffenen MO's an, und verschickt diese an alle anderen Kernkomonenten. Diese führen durch einen Vergleich mit ihren Speicher, eine Überprüfung durch, ob sie diese Objekte benötigen. Nur wenn samtliche Kernkomponenten keinen Anspruch auf die besagten MOs erheben, wird der exklusive Zugriff genehmigt.

4 Zusammenfassung und Ausblick

In diesem Aufsatz ist eine neue Technik für den Aufbau eines maßgeschneiderten Managementsystems, das jede verteilte Anwendung repräsentieren kann, vorgestellt. Es basiert auf der Idee, zwischen kommunikations-, anwendungs-, und managementorientierten Aspekten zu trennen. Die Randbedingungen und Rekonfigurationsregeln garantieren die

Konsistenz. Der wesentliche Beitrag dieser Methode ist das integrierte Konstruktionsverfahren und die Formulierung der Rekonfigurationsaktivitäten. Die Implementierung bietet eine Basis für die Implementierung der dezentralierten Managementarchitektur.

In zukünftigen Arbeiten sollte eine Untersuchung durchgeführt werden, um den Informationsstrom des Managements in den Netzwerken zu reduzieren. Eine Aufteilung der Managementebene in mehreren Unterebenen, die unterschiedliche Stufen von Abstraktionen haben, ist empfehlenswert. Im weiteren sollen verschiedene Typen von Benutzerkomponenten entwickelt werden, die das verteilte Anwendungsmanagement automatisch unterstützen.

Management verteilter Anwendungen: Vergleich zweier Ansätze

Daniela Wichert

Kurzfassung

Verteilte Systeme profitieren wie alles andere auch von einer guten Verwaltung. Die ersten Untersuchungen in diese Richtung wurden erst vor kurzem gemacht, wobei die Notwendigkeit des Managements der verteilten Anwendungen immer deutlicher wurde. In diesem Beitrag werden zwei aktuelle Ansätze des Anwendungsmanagements vorgestellt und im Anschluß daran diese miteinander verglichen. In dem ersten aktuellen Ansatz wird untersucht, ob eine Benutzung des OSI NMF (Network Management Framework) für das Management verteilter Anwendungen möglich ist. Hierbei wird die Wechselwirkung zwischen dem Manager, Agenten und dem Managed Object erläutert, wobei auf die Realisierung nicht näher eingegangen wird. In dem zweiten aktuellen Ansatz werden die Fragen der Verwaltung verteilter Applikationen diskutiert und ein Werkzeug - das Meta System - vorgestellt. Der Meta Ansatz stellt auch Sprachmittel für die Managementanwendung bereit und ermöglicht somit eine integrierte Realisierung und Verwaltung verteilter Anwendungen.

1 Einleitung

Das Schreiben von verteilten Anwendungen, die zielgerecht funktionieren, ist besonders schwierig. Ein solches Programm kann in stark unterschiedlichen Konfigurationen laufen – von einem Einzelrechner bis zu 10 oder 100 Rechnern– und außerdem auf Rechnern unterschiedlicher Leistungsfähigkeit und verschiedener Hersteller. Oft muß das Programm auch weiterarbeiten, wenn einige Rechner ausgefallen sind.
Wir nennen den Prozeß, eine in einer gegebenen Umgebung gut funktionierende verteilte Anwendung bereitzustellen, die *Verwaltung einer verteilten Anwendung.* Hierzu gehört

- die Anpassung der Systemkomponenten an eine gegebene Hard- und Softwareumgebung,
- die geordnete Initialisierung der Anwendung,
- die Überwachung des Verhaltens und der Ablaufgeschwindigkeit der Anwendung und
- das effiziente Zuteilen von Aufgaben an die verschiedenen Komponenten.

Eine Anwendung muß während ihrer Ausführung überwacht werden, weil das System ständig auf wechselnde Auslastung, Änderungen der Umgebung oder Fehlersituationen reagieren muß.
Eine verteilte Rechnerumgebung verursacht mehr Probleme für die Verwaltung von Anwendungen als eine nichtverteilte Umgebung. Die Daten zur Beurteilung der Leistungsfähigkeit, welche für die Überwachung des Systems benötigt werden, sind im ganzen System verteilt und dadurch schwer zu erfassen.

Fehler sind eine Tatsache in verteilten Systemen und erschweren die Verwaltung erheblich. Die Fehlerhäufigkeit steht im direkten Verhältnis zur Anzahl der Hardwarekomponenten.
Da immer mehr auftragskritische Applikationen und Dienste in verteilten Anwendungen entwickelt werden, wird es immer wichtiger, diese Applikationen und Dienste mit wachsender Zuverlässigkeit und Leistung zu verwalten. Computer Umgebungen heutzutage werden immer größer und komplexer. Sie bestehen aus miteinander verbundenen Netzwerken, heterogener Computerhardware und Software Plattformen. Es steigt die Nachfrage der Benutzer für leistungsfähigere und höher entwickelte Applikationen und Dienste. Gleichzeitig verlangen die Benutzer, daß diese Applikationen und Dienste zuverlässiger und effizienter werden. Daraus folgt, daß die darunter liegenden Systeme und andere Subsysteme, die diese Aplikationen und Dienste unterstützen, immer größer und komplexer werden. Die Systemverwalter sehen sich schwierigen und herausfordernden Aufgaben gegenübergestellt, solch eine Computerumgebung zu verwalten. Sie müssen fähig sein die Betriebsmittel so gut wie möglich für ihren jeweiligen Einsatz zu planen, konfigurieren, entwickeln, anzeigen und kontrollieren. Sie müssen sich nicht nur um die Hardwarebetriebsmittel kümmern, sondern auch um die Menge der Softwarebetriebsmittel, die darauf laufen. Leider haben sie meistens keine Werkzeuge und Einrichtungen, die solche Managementaufgaben durchführen könnten. Noch schlimmer ist, daß die meisten verteilten Anwendungen und Dienste, die entwickelt werden, gar nicht darauf ausgerichtet sind, verwaltet zu werden. Dieser Beitrag konzentriert sich auf das Management verteilter Anwendungen und diejenigen Dienste, die eine Managementunterstützung von Applikationen brauchen.

2 OSI Management Framework

Im Rahmen einer Arbeit [HKB93] wurden Grundfragen der Eignung des OSI-Network-Management-Frameworks (**OSI-NMF**) für das Anwendungsmanagement untersucht. Dabei sind einige Fragen entstanden.
Können existierende NMF benutzt oder erweitert werden, um verteilte Anwendungen und Dienste anzuzeigen, zu koordinieren oder zu steuern?
Wenn Erweiterungen notwendig sind, dann was für welche?
Gibt es Beschränkungen in diesen Frameworks, die ihre Benutzung für die Verwaltung verteilter Anwendungen und Dienste limitieren würden?
Dieser Ansatz stellt eine anfängliche Untersuchung vor, um entscheiden zu können, ob eine Benutzung oder Erweiterung eines NMF für die Verwaltung verteilter Anwendungen durchführbar ist.

2.1 Beschreibung des OSI Management Framework

Das OSI Management bringt Aktivitäten mit sich, die gebraucht werden, um Betriebsmittel zu kontrollieren, koordinieren und anzuzeigen, und die erlauben, Kommunikation in der OSI Umgebung einzubetten. Das OSI MF definiert Dienste und Protokolle für den Austausch von Managementinformationen zwischen offenen Systemen. Das OSI

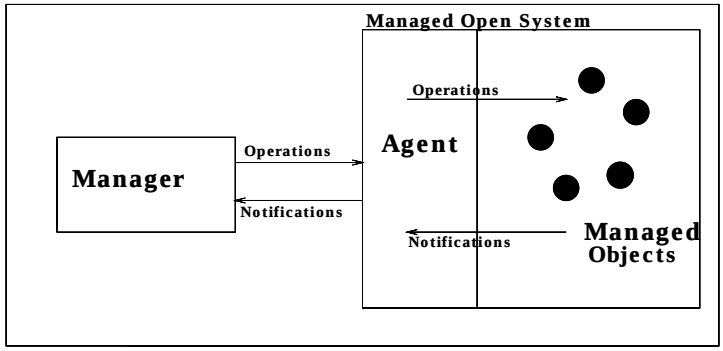

Abbildung 22. OSI Manager-Agent Interaktion Modell

MF benutzt eine objektorientierte Methodik für die Modellierung von offenen Systembetriebsmitteln, die verwaltet werden sollen. Diese Betriebsmittel können physikalisch oder logisch sein und werden durch den abstrakten Begriff des sogenannten *Managed Object*(**MO**) beschrieben. Ein MO ist definiert durch Begriffe von *Eigenschaften* die es besitzt, *Operationen* die mit ihm durchgeführt werden können, *Mitteilungen* die es ausgeben kann und *Beziehungen* mit anderen MOs. Die Menge von MOs innerhalb eines Systems, zusammen mit deren Eigenschaften, bildet die Management Information Base (**MIB**) des Systems. Auf MOs wird von Agenten (Management Anwendungen in Agenten-Rollen) und von Managern (Management Anwendungen in Manager-Rollen) aus zugegriffen, um Managementabsichten zu implementieren. Abbildung 22 zeigt die Interaktion zwischen einem Manager und einem Agenten im OSI MF. Ein *Manager* ist Teil einer Managementanwendung und bietet dieser eine Schnittstelle für das Manipulieren von Managementinformationen im Netz.

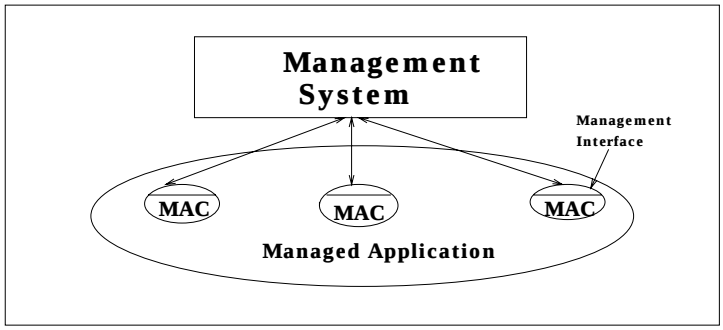

Abbildung 23. Architektur der verteilten Anwendungen

Der *Agent* ist der Kommunikationspartner des Managers und für die Manipulation der MOs zuständig. Der OSI Management Standard, der Richtlinien vorgibt wie MO Klassen definiert werden, beinhaltet:

- das Management Information Model (**MIM**), welches das Modell für MOs definiert,
- Definition von Management Information (**DMI**), die System MOs definiert,
- Muster, die in eine Vielzahl von MO Klassen Definitionen importiert werden können,

– Richtlinien für die Definition von MOs (**GDMO**), die Anleitungen, Methoden und Notationstechniken für die Spezifizierung von MO Klassen liefern.

OSI Management liefert den Common Management Information Service (**CMIS**), der aus einer Menge von Primitiven besteht, die von Managern und Agenten benutzt werden können, um Management Informationen auszutauschen. OSI Management liefert auch das Common Management Information Protocol (**CMIP**), was benutzt wird um CMIS zu implementieren.

2.2 Frühere und verwandte Arbeiten

Wie bereits erwähnt, benötigt man für die Verwaltung verteilter Anwendungen und Dienste:

1. *Anwendungen*, die mit einer geeigneten Managementschnittstelle ausgestattet wurden, damit sie verwaltet werden können, und

2. Ein *Management System*, das die Aktivitäten der bearbeiteten Applikationen anzeigen und ihr Verhalten soweit notwendig steuern kann.

In diesem Abschnitt werden kurz die früheren und anderen Arbeiten über die Vorbereitung verteilter Anwendungen für das Management und Arbeiten über die Verwaltung solcher bearbeiteter Anwendungen beschrieben.

Vorbereitung verteilter Anwendungen für das Management. Die meisten existierenden Anwendungen sind nicht mit der Absicht entworfen worden, sie zu verwalten. Entwickler von Anwendungen werden sich jedoch mehr und mehr der Notwendigkeit bewußt, Anwendungen zu verwalten, und daher müssen sie Managementaspekte in der Entwurfsphase berücksichtigen. Genauso müssen verteilte Anwendungen und Systeme mit Managementschnittstellen ausgestattet werden, damit sie verwaltet werden können. Eine typische verteilte Anwendung besteht aus mehreren Komponenten (z.B.Prozessen), die miteinander agieren, um bestimmte Aufgaben der Anwendung auszuführen. Das bedeutet, daß das Ausstatten einer verteilten Anwendung mit einer Managementschnittstelle grundsätzlich das Ausstatten seiner Komponenten zur Folge hat. Um Anwendungen mit einer Managementschnittstelle auszustatten, werden die folgenden vier Schritte ausgeführt.

1. Analyse der Anwendung

2. Spezifikation der Managementschnittstelle

3. Codegenerierung und

4. Errichten der bearbeiteten Anwendung

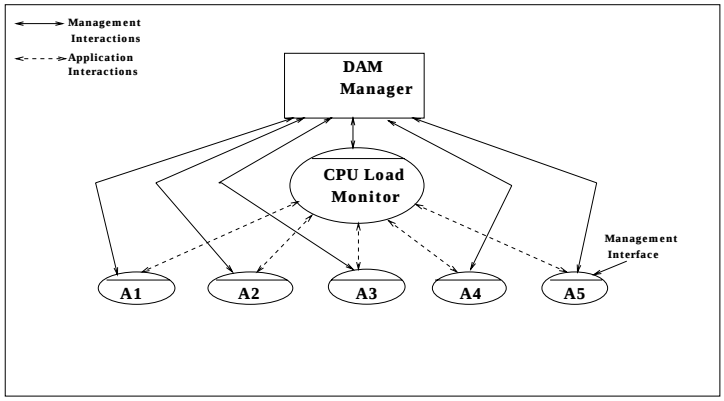

Abbildung 24. Interaktion zwischen dem DAM Manager und der bearbeiteten verteilten Anwendung

Wurden diese vier Schritte für die Anwendung erfolgreich durchgeführt, so kann sie verwaltet werden. (siehe Abbildung 23). Um die Arbeit, jedesmal die obigen Schritte für jede verschiedene Anwendung wiederholen zu müssen, zu erleichtern, wurde ein sogenanntes *Generic Management Interface* (**GMI**) definiert, das für die meisten Anwendungen benutzt werden kann und das für jeden Anwendungstyp spezialisiert werden kann, indem die anwendungsspezifischen Daten und Operationen hinzugefügt werden.

Verteiltes Anwendungs Management System. Frühere Arbeiten betrafen Entwurf und Implementierung eines Prototyps für eine verteilte Anwendung, genannt **DAM** Manager (Distributed Application Management). Abbildung 24 zeigt die Wechselwirkungen zwischen DAM Manager und den zu verwaltenden Anwendungskomponenten, in diesem Fall UNIX Prozesse. Die bearbeitete verteilte Anwendung zeigt die CPU-Auslastung individueller Rechner in einem Netzwerk auf. Die zu verwaltende Anwendung besteht aus einem CPU-Load-Monitor, der periodisch mit seinen Monitoragenten (A1 bis A5) interagiert, von denen jeder auf einem Rechner läuft, der angezeigt wird, und der die CPU-Load-Information sammelt. Die Interaktionen zwischen Anwendungs-Prozessen, die die CPU-Load-Informationen sammeln sollen (d.h. Anwendungs Interaktionen) sind durch gestrichelte Linien innerhalb des Prozesses in Abbildung 24 gekennzeichnet. Die Management Interaktionen zwischen DAM Manager und den zu verwaltenden Prozessen sind durch durchgezogene Linien gekennzeichnet. Das Management Protokoll, das für den Austausch von Management Informationen zwischen Manager und zu verwaltenden Prozessen benutzt wurde, ist kein standardisiertes Management Protokoll, sondern ein eigenes, einfaches Protokoll. Es ist ausreichend, um Managementanfragen und Informationen zwischen Manager und den zu verwaltenden Prozessen weiterzuleiten. Die Hauptmotivation für die aktuelle Arbeit stammte von Ergebnissen, die aus dieser Arbeit resultierten: das Protokoll war beschränkt; niemand glaubte daran, daß es ausreichend allgemeingültig war, um andere Anwendungen damit zu unterstützen; bei der Ausstattung eines allgemeinen verteilten Systems müßte solch ein Protokoll standardisiert werden. Folglich sollte die Durchführbarkeit untersucht werden, ein Standard Management Protokoll (wie OSI CMIP oder Internet SNMP: Simple Network Management Protocol) zu benutzen. Dies erforderte den Austausch des einfachen Management Protokolls mit

einem Standard Protokoll. Der Prototyp des DAM Managers besteht aus drei Hauptwerkzeugen: Das *Konfigurationswerkzeug* ist verantwortlich für die Konfigurierung der Applikation und das Starten und Beenden der Komponenten Prozesse. Das *Monitoringwerkzeug* ist verantwortlich für das Anzeigen des Status und des Betriebsmittelverbrauchs der Komponenten Prozesse. Das *Visualisierungswerkzeug* zeigt die Daten an, die vom Monitoringwerkzeug von den Prozessen gesammelt wurden.

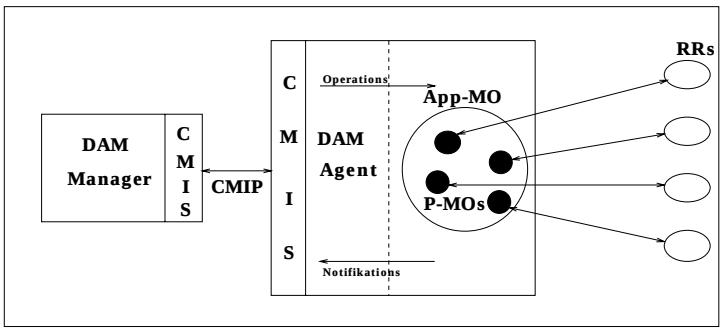

Abbildung 25. Auf OSI-basierende DAM Architektur

2.3 DAM Architektur auf OSI basierend

In diesem Abschnitt wird gezeigt, wie das OSI-Management Interaktionsmodell (siehe Abbildung 22) als Basis für die Verwaltung verteilter Anwendungen benutzt werden kann. Wie bereits erwähnt, sind die zu verwaltenden Betriebsmittel als MOs im OSI MF dargestellt. Diese MOs werden von Management Agenten gehandhabt und liefern Zugriffsmöglichkeiten zu realen Betriebsmitteln, deren Verhalten angezeigt und gesteuert werden soll. Zugriff auf reale Betriebsmittel kann, abhängig vom Standort der realen Betriebsmittel, verschiedene Formen annehmen, genauso wie die Art der Kommunikation, die von den realen Betriebsmittel unterstützt wird. Da die Betriebsmittel, die verwaltet werden sollen, verteilte Anwendungen und Dienste sind, müssen sie genauso von MOs verkörpert werden. Dies wird erreicht durch die Definition einer MO-Klasse oder durch Klassen die verteilte Anwendungen verkörpern und sie je nach Gebrauch einsetzen. Eine typische verteilte Anwendung besteht aus einer Vielzahl von Komponenten (z.B. Prozessen) die innerhalb der Computerumgebung laufen und miteinander agieren, um die Aufgaben der bestimmten Anwendung auszuführen. Somit kann eine verwaltete Anwendung durch ein Anwendungs-MO (**App-MO**) verkörpert werden, das ein oder mehrere Prozeß-MOs (**Pro-MO**) enthalten kann. Im nächsten Abschnitt wird gezeigt, daß die hier definierten MO-Klassen so gestaltet sind, daß sie für die meisten (wenn nicht sogar alle) Typen verteilter Anwendungen benutzt werden könnten. Das Ergebnis ist eine neue Managementarchitektur verteilter Anwendungen basierend auf dem OSI MF (siehe Abbildung 25). Einer der Hauptunterschiede zwischen dieser und früheren Architekturen (wie in Abbildung 23 gezeigt) ist das Einfügen des DAM Agenten zwischen dem DAM Manager und den zu verwaltenden Anwendungen (realen Betriebsmittel oder **RR**s). Ein DAM Agent ist wie jeder andere OSI Management Agent, da er im Grunde Management Anfragen von einem DAM Manager erhält und diese Anfragen auf MOs im Namen

vom Manager ausführt. Er erhält ebenso Mitteilungen von MOs und überträgt diese an den Manager. Ein DAM Agent und eine DAM Manager Anwendung können auf unterschiedlichen Rechnern ansässig sein. Dies wird durch Benutzung des OSI CMIS/P zwischen DAM Manager und DAM Agent erreicht. CMIS stellt eine Dienstschnittstelle zur Verfügung, die Management Anwendungen (Manager und Agent) für den Austausch von Management Informationen benutzen können. Der Austausch zwischen diesen Management Einheiten wird durch das CMIP Protokoll erreicht, das das eigene, einfache proprietäre Protokoll, das von den in Abschnitt 2.2 beschriebenen Arbeiten verwendet wurde, ersetzt.

2.4 Prototypische Implementierung

In diesem Abschnitt wird die Prototyp-Implementierung des DAM Manager und DAM Agenten in der OSI Management Umgebung vorgestellt. Diese Prototyp Implementierung benutzt den OSI Management Information Service (**OSIMIS**) als Entwicklungsplattform.

Überblick über OSIMIS. OSIMIS liefert eine Menge von Einrichtungen zur Entwicklung von Manageranwendungen, Agentenanwendungen und MOs. Es wurde unter Benutzung der ISO Entwicklungsumgebung (**ISODE**) entwickelt und hat generische und spezifische Teile. Die generischen Teile sind objektorientierte Infrastrukturen für die Entwicklung von OSI Agenten und Managern, und eine Menge von generischen Manageranwendungen. Die spezifischen Teile sind Agenten, die spezifische MIB's und damit verbundene Management Anwendungen unterstützen. OSIMIS stellt ein objektorientiertes API (Applikation Interface, in C++) zur Verfügung, welches die Einzelheiten der Zugriffe auf Management Informationen über das OSI CMIS/P versteckt. Ein OSI Management Anwendungsentwickler muß eine beachtliche Menge von Code erstellen. Zum einen für die Einheiten der Management Anwendung (Agenten und Manager), die über den OSI Protokoll Stack kommunizieren, und zum anderen für das Kodieren und Dekodieren der ausgetauschten Daten. OSIMIS stellt zusammen mit ISODE viel von diesem Code zur Verfügung, dadurch kann der Entwickler eine beachtliche Menge an Zeit und Mühe sparen.

Der OSI DAM Manager. Die erste Implementierung die hier vorgestellt wird, ist der OSI/Motif-based OSI DAM Manager. Der DAM Manager besteht aus vier Werkzeugsets: Konfiguration, Steuerung, Anzeige und Analyse. Das Konfigurationswerkzeugset erlaubt dem Anwender die zu verwaltende verteilte Anwendung zu konfigurieren. Das Anzeigewerkzeugset erlaubt dem Anwender eine oder mehrere MOs zum Anzeigen ihrer Aktivitäten auszuwählen. Das Analysewerkzeugset erlaubt dem Anwender die angezeigten Daten zu analysieren. Schließlich erlaubt das Steuerungswerkzeugset dem Anwender Aktionen auf MOs dynamisch aufzurufen, zu löschen, zu ändern und auszuführen. Der DAM Manager interagiert mit DAM Agenten um verteilte Anwendungen zu verwalten. Die Interaktion zwischen DAM Manager und DAM Agenten geschieht durch CMIS/P.

Der OSI DAM Agent und MOs. Da großes Interesse bestand, verteilte Anwendungen zu verwalten, wurde ein Management Agent für verteilte Anwendungen (DAM Agent) entwickelt, in dem MO Klassen definiert wurden, die den verwaltenden Anwendungen und ihren Komponenten entsprechen. Um dies zu erreichen, wurden OSIMIS Einrichtungen benutzt.
Die wichtigsten Komponenten jeder verteilten Anwendung sind ihre Prozesse. Es wurde eine GDMO Definition eines verteilten Anwendungs-MO und eine GDMO Definition eines Prozeß MOs definiert.
Objekt Instanzen der Objektklasse der verwaltenden Anwendung und der Prozeß Objektklasse sind App-MOs, bzw. P-MOs (siehe Abbildung 25). Es wurde der OSIMIS GDMO Compiler benutzt, um den Sourcecode (in C++) für den Zugriff auf diese MOs durch einen DAM Agenten zu generieren. Die realen Betriebsmittel, mit denen diese P-MOs interagieren, sind UNIX Laufzeit Prozesse, die Teil der zu verwaltenden verteilten Anwendung sind (wie im vorherigen Kapitel beschrieben). Interaktionsmethoden zwischen MOs und RRs (siehe Abbildung 25) hängen zum einem von dem Aufenthaltsort der RRs ab, und zum anderen von dem Typ des Kommunikationsmusters, den die behandelten RRs liefern. Wenn die RRs auf einer lokalen Maschine ansässig sind, dann kann die Interaktions Methode die Form lokaler Kommunikation annehmen (z.B. Systemaufrufe oder Prozeduraufrufe). Da die RRs mit denen die P-MOs interagieren, entfernte UNIX Prozesse sind, wurde irgendeine Form von Fernkommunikation gebraucht. Die frühere Implementierung (die das eigene Protokoll benutzte) beinhaltete Fernkommunikation (speziell die Internet Socket communication) zwischen der Manager Anwendung und den zu verwaltenden Prozessen. Daher wurde einfach derselbe Mechanismus zwischen MOs und RRs benutzt. Dies bewahrte die Entwickler davor, den Code für die Managementschnittstelle der zu verwaltenden Prozesse umarbeiteten, umschreiben oder modifizieren zu müssen.
Die Entfaltung einer verteilten Anwendung in diesem Framework ist equivalent mit der schnellen Einrichtung eines App-MOs mit der Konfigurations Information, die den Namen einer Applikation enthalten kann und der Information, wo die Komponenten Prozesse laufen sollen. Als Teil der App-MO Erzeugung wird eine Anzahl von P-MOs ebenso erzeugt. Umgekehrt erzeugt jedes P-MO einen Anwendungs Komponenten Prozeß innerhalb seiner eigenen Erzeugungsprozedur. Diese Prozesse können auf die Computer Umgebung verteilt sein, so wie es in der Prototyp-Implementierung der Fall ist.
Das Beenden einer Anwendung ist gleichbedeutend mit dem Löschen einer existierenden App-MO Instanz, die dann auch alle ihr untergeordneten P-MOs löschen würde. Jedes P-MO ist dann dafür verantwortlich, seinen zugeordneten realen Prozeß zu beenden. Der Manager ist so flexibel, daß er P-MOs individuell erzeugen und löschen kann. Dadurch kann der Manager Steueraktionen ausführen, die vom Status der Applikation abhängen und davon, ob sie vom Benutzer gebraucht werden.
In dem oben vorgestellten Ansatz wurde die Frage der Eignung des OSI NMF für das Anwendungsmanagement untersucht und bestätigt. Im Folgenden werden die Fragen der Verwaltung verteilter Applikationen diskutiert und ein weiterer Ansatz - das Meta System - vorgestellt.

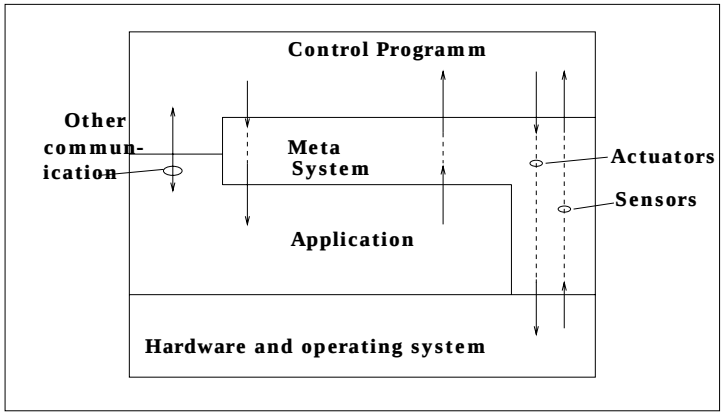

Abbildung 26. Meta Anwendungsarchitektur

3 Meta System

In der Praxis werden viele der Aspekte der Anwendungsverwaltung ignoriert, was zu schlecht organisierten Systemen führt, die zwar funktionieren, aber oft ein unvorhersehbares Laufzeitverhalten besitzen, inkonsistent werden, zu teilweisem Fehlerverhalten neigen und sich als sehr instabil erweisen, wenn an der Hard- oder Softwarebasis kleine Änderungen vorgenommen werden.
Es wird versucht, die Nachteile dieses Ansatzes zu vermeiden, indem ein Gerüst zur Verfügung gestellt wird, welches die Entwicklung von stabiler verteilter Verwaltungs- und Anwendungssoftware begünstigt. Es wurde ein Satz von Werkzeugen – das Meta System – entwickelt, der diesen Ansatz unmittelbar unterstützt.
Es wird der Ausdruck *Anwendungsprogramm* für eine verteilte Anwendung, die sich aus mehreren Prozessen zusammensetzt benutzt.
Ein *Prozeß* ist ein einzelner nichtverteilter Addressraum (wie in UNIX) mit einem oder mehreren Kontrollinstanzen.
Eine *Komponente* ist ein Teilsystem der Gesamtanwendung, welches einen oder mehrere Prozesse beinhaltet. Die Komponente wird gelegentlich als Bezug auf eine Umgebungskomponente wie ein Fileserver oder eine Workstation benutzt.

3.1 Architektur der Anwendung

Abbildung 26 zeigt das Meta-Modell einer verteilten Anwendung. Die Aspekte der Anwendungsverwaltung sind von denen der Hauptfunktionsbereiche getrennt, und die Schnittstelle zwischen beiden ist genau definiert. Diese Trennung von Verwaltung und Funktion erleichtert die Veränderung der Verwaltung einer Anwendung und vermindert die Wahrscheinlichkeit, daß die Korrektheit des Rests des Programms beeinflußt wird.
Die Verwaltungsschicht wird das Kontrollprogramm genannt. Während die unterlagerte Anwendung mit konventionellen Entwicklungsmittel erstellt wird, ist es am besten, das Kontrollprogramm in einem (reaktiven) regelbasierenden Stil zu schreiben. Das Metasystem ist zwischen dem Kontrollprogramm und der Anwendung plaziert und stattet das Kontrollprogramm mit einer abstrakten Sicht der Anwendung und ihrer Umgebung aus (Wie Abbildung 26 zeigt muß nicht jede Kommunikation zwischen Kontrollprogram

und Anwendung über Meta laufen). Die Struktur des Anwendungsprogramms – seine Bestandteile und deren Beziehungen – werden Meta in Form eines objektorientierten Datenmodells zugänglich gemacht.
Das Kontrollprogramm beobachtet das Verhalten der Applikation durch die Abfragesensoren, d.h. Funktionen, die Werte aus dem Applikationsstatus und der Umgebung liefern (siehe oben). Analog kann das Verhalten der unterlagerten Anwendung und ihrer Umgebung durch den Gebrauch von Funktionen names Aktoren geändert werden. Meta stellt eine einheitliche ortsunabhängige Schnittstelle sowohl für eingebaute als auch für benutzerdefinierte Sensoren und Aktoren zur Verfügung. Diese Schnittstelle stellt auch Möglichkeiten zur Kombination mehrerer Sensorwerte zur Verfügung, um daraus kompliziertere Sensoren zu bilden oder Fehlertoleranz zu ermöglichen. Die jeweils verwendeten Sensoren und Aktoren hängen von der zu kontrollierenden Anwendung ab.
Meta bietet Programmen mehrere Schnittstellen zur Abfrage von Sensoren und zum Aufruf von Aktoren. Das grundlegende Interface ist von der Anwendungs-Programmiersprache abhängig. Andere Schnittstellen sind z.B. die Basissprache NPL in Postfix-Notation, die von einem fehlertoleranten verteilten Interpreter ausgeführt wird, und die Kontroll-Hochsprache Lomita, die in NPL-Ausdrücke übersetzt wird. Lomita kombiniert Echtzeit-Intervall-Logik mit einer regelbasierenden Syntax zur Sensorabfrage. Die Ausdrücke in Lomita stellen ein klares Abbild der zeitlichen Natur von komplizierten Ereignissen in der verteilten Anwendung dar.
Meta benutzt das Isis Toolkit für die verteilte Programmierung. Isis stellt Grundfunktionen für verläßliche Programmierung zur Verfügung, worin Prozeßgruppen ebenso wie geordnete Mehrfachbenachrichtigung (Senden einer Nachricht an mehrere Ziele) vorkommen. Basierend auf diesen Grundfunktionen stellt Isis Lösungen für grundlegende Probleme verteilter Systeme zur Verfügung, wie verteilte Synchronisation, flexible Verarbeitung, Protokollierung und Wiederaufsetzen. Meta und Isis laufen auf dem Unix Betriebssystem.

3.2 Verwirklichung einer verteilten Anwendung

Um die Diskussion von Sensoren und Aktoren zu konkretisieren, wird eine hypothetische Anwendung präsentiert und gezeigt, wie sie innerhalb des Meta-Gerüsts verwaltet wird. NuMon ist ein seismologisches Analysesystem, das die Einhaltung von Verträgen zum Verbot von Nukleartests überwacht. Science Applications International Corporation entwickelte ein echtes Nuklearüberwachungssystem, auf dem dieses vereinfachte Beispiel basiert, unter Zuhilfenahme von Isis und einer früheren Version von Meta, in der die Sprachen NPL und Lomita noch fehlten. Aus Erfahrung mit diesem Projekt wuchs die Motivation für die Arbeit an höheren Schichten von Meta.
Die Verwaltung einer Anwendung wie NuMon durch Meta erfordert 3 Schritte. Der Entwickler versieht seine Anwendung und deren Umgebung mit Sensoren und Aktoren. Diese Funktionen zusammen mit den eingebauten Sensoren und Aktoren bilden die Schnittstelle zwischen Kontrollprogramm und der eigentlichen Anwendung.
Als nächstes beschreibt der Entwickler die Struktur der Anwendung mit Hilfe von Lomitas Einrichtungen zur objektorientierten Datenmodellierung. (Meta kann betrachtet werden als Lieferant einer temporären objektorientierten Datenbank, in der die Anwen-

dung und deren Umgebung die Datenwerte zur Verfügung stellen.)
Zum Abschluß schreibt der Entwickler ein Kontrollprogramm, das das Datenmodell referenziert. Das Kontrollprogramm kann als Lomita-Script oder in einer konventionellen Programmiersprache mit integrierten Aufrufen an den Lomita-Übersetzer geschrieben sein. Das Kontrollprogramm kann direkte Aufrufe an Sensoren, Aktoren und andere Funktionen im Datenmodell machen. Es kann auch höherwertige Verwaltungsregeln nutzen, in denen eine Reihe von Bedingungen an Sensoren und die auszuführenden Aktionen bei Eintreten einer bestimmten Bedingung spezifiziert sind. Abbildung 27 zeigt die funktionalen Schichten im Meta-Modell.

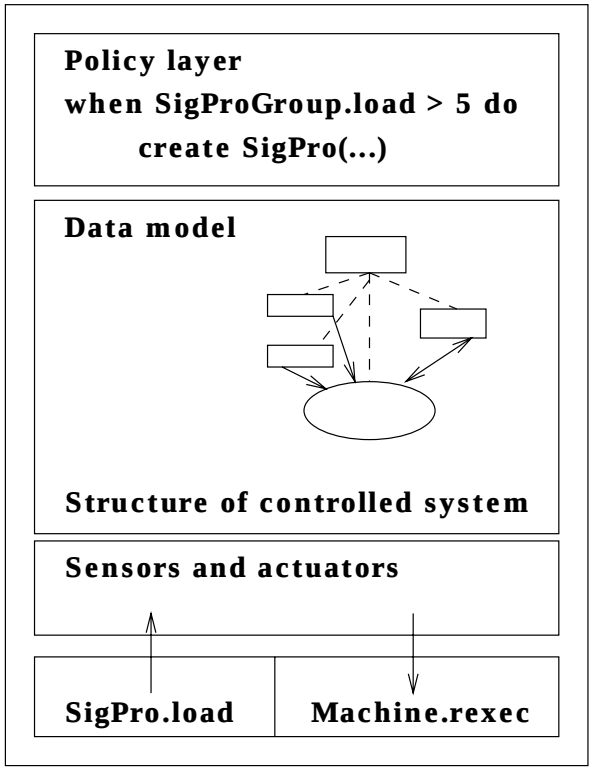

Abbildung 27. Funktionale Schicht im Meta

Abbildung 28 zeigt diese funktionale Architektur am Beispiel der NuMon-Anwendung. Die meisten Meta-Funktionen werden durch die Meta-Bibliothek (eine Kopie der Bibliothek wird zu jedem Anwendungsprozeß gebunden) zur Verfügung gestellt. Die Bibliothek enthält Routinen, mit deren Hilfe eine Anwendungskomponente ihre Primitiv-Sensoren und -Aktoren bekanntgibt. Neben den Anwendungsprozessen wie SigPro und Assess stellen die von Meta gelieferten Machinen-Prozesse eingebaute Sensoren und Aktoren zur Verfügung.
Der zweite Hauptbestandteil von Meta betrifft die Übersetzung und Ausführung von Lomita. Der Lomita-Compiler übernimmt Quellanweisungen, die aus Dateien gelesen oder dynamisch von Anwendungen wie AppManager generiert werden, und übersetzt sie in NPL-Anweisungen. Jede Kopie der Meta-Bibliothek enthält einen NPL-Interpreter. Das Objektprogramm wird in die Anwendungsprozesse transferiert und von diesen Interpretern verteilt ausgeführt.

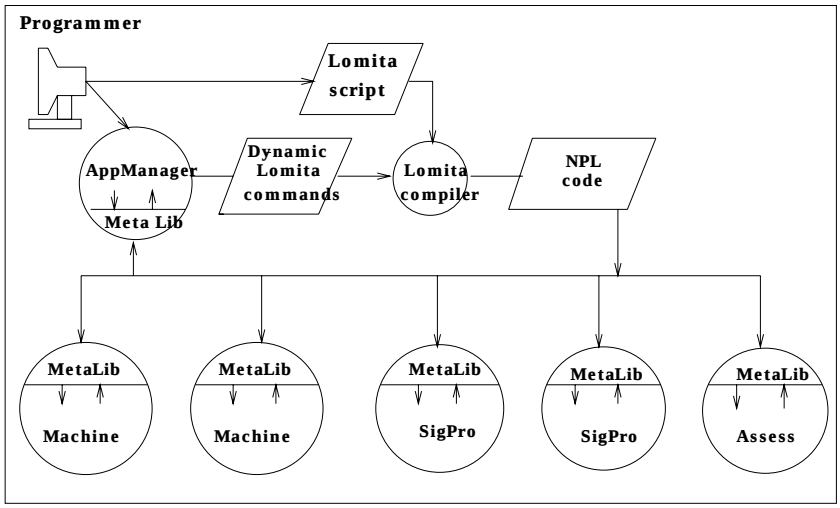

Abbildung 28. Funktionale Architektur von Meta

Sensoren. Ein Meta-Sensor repräsentiert einen Teil der Statusinformation der überwachten Anwendung. Jeder Sensor wird identifiziert durch die Art der Anwendungskomponente, die er überwacht (z.B. SigPro), die Art des Wertes, die er überwacht (z.B. backlog), und die Instanz der Komponente, die er überwacht (z.B. SigPro1).

Eingebaute Sensoren. Meta stellt einen Satz von eingebauten Sensoren zur Verfügung, die direkt zu Werten korrespondieren, die der Umgebung entnommen werden. Beispiele sind Sensoren, die statistische Daten wie Prozessor- und Speicherauslastung von Unix-Prozessen liefern. Außerdem enthält Meta den readvar Sensor, der das Lesen der Werte bestimmter Arten von globalen Variablen eines aktiven Prozesses gestattet. Dies ist implementiert mit einem Unix-Systemaufruf, der zu Debug-Zwecken den Zugriff auf den Addressraum eines anderen Prozesses erlaubt. Der eingebaute Lebenszeichensensor liefert TRUE solange er eine fehlerfreie Komponente überwacht.

Benutzerdefinierte Sensoren. Meta erlaubt es dem Entwickler, Primitivsensoren zu definieren und zu implementieren. Solche Sensoren beziehen sich auf dynamische Eigenschaften der Anwendung, deren Werte nicht durch simples Abfragen des Status des unterlagerten Betriebssystems erfragt werden können. Sensoren dieser Kategorie werden Meta zur Laufzeit bekannt gegeben.

Aktoren. Meta stellt eine Reihe von eingebauten Aktoren zur Verfügung, die in der gleichen Weise benannt und referenziert werden wie Sensoren. Hierzu zählen ein Aktor, der einen Prozeß mit einer gegebenen Argumenteliste startet und ein writevar Aktor, der die Veränderung globaler Variablen ermöglicht.

Nebenläufigkeit. Um inkosistente Werte oder Aktionen zu vermeiden, sollten benutzerdefinierte Sensoren und Aktoren nur zu wohldefinierten Zeitpunkten während der Ausführung eines Prozesses aufgerufen werden. Um Inkonsistenzen zu vermeiden werden

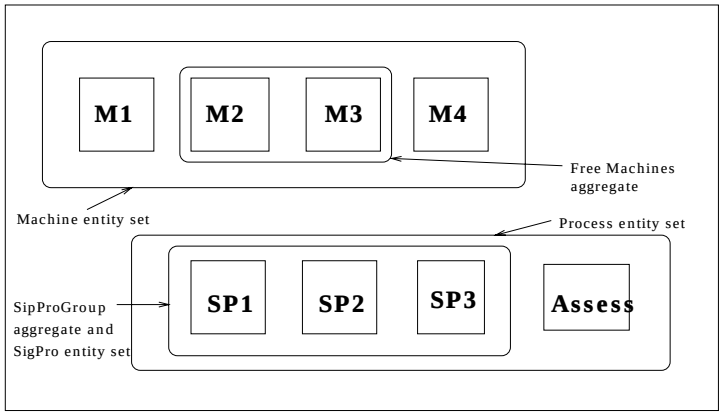

Abbildung 29. Lomita Gruppenstruktur

Meta und der unterlagerte Prozeß ausschließlich als Koroutinen ausgeführt. Der Anwendungsprozeß muß metanotify (möglicherweise mit einer leeren Sensorliste) mindestens so oft wie das kleinste Sensorabfrageintervall aufrufen. Bei readvar und writevar jedoch ist der einzige Synchronisationsmechanismus (wenn überhaupt) die atomare Einheit des Lesens oder Schreibens von einem Speicherwort. Deshalb sollten die Werte von globalen Variablen durch ein Speicherwort (oder weniger) repräsentiert werden.

3.3 Beschreibung der Applikation

Der nächste Schritt ist, Meta die Applikationsstruktur bekanntzugeben.

Lomita Datenmodell. Der Entwickler erzeugt mit Hilfe der Lomita-Datenmodellsprache eine Beschreibung der Anwendung. Komponenten in der Applikation und in der Umgebung werden als Einheiten modelliert, entsprechend der Namensgebung der Entity-Relationship-Datenbank-Theorie. Lomita erlaubt die Spezifikation einer großen Auswahl von Verbindungen und Gruppierungen zwischen Komponenten, um die Struktur der Anwendung zu beschreiben.

Abbildung von Anwendungskomponenten auf Prozessgruppen. Meta beinhaltet die Möglichkeit, Aggregationen, also Sammlungen von Objekten gleichen Typs, zu bilden.
Eine Isis-Gruppe ist ein einfacher Weg, um Komponenten zu organisieren und mit ihnen zu kommunizieren. Isis Mehrfachbenachrichtigung greift simultan auf alle Instanzen eines Sensors oder Aktors eines bestimmten Komponententyps zu. Mehrfachbenachrichtigung ist unteilbar: der Aufruf eines Aktors wird von allen Gruppenmitgliedern empfangen oder von keinem. Außerdem werden nebenläufige Mehrfachbenachrichtigungen bei allen Gruppenmitgliedern konsistent angeordnet, und der Gruppenbegriff von Isis stellt sicher, daß Meta eine genaue Kenntnis davon besitzt, welcher Gruppe eine Aggregation oder ein Typ im Moment angehört. Änderungen der Gruppenmitgliedschaft, ob geplant (z.B. durch Beitreten einer neuen Komponente zu einer Gruppe) oder ungeplant (z.B. durch einen Prozeßfehler), werden mit der Gruppenkommunikation synchronisiert (serialized). Abbildung 29 zeigt die Gruppenstruktur.

Abgeleitete Sensoren. Die Werte primitiver Sensoren, die von der Anwendung erfragt wurden, können in Form von abgeleiteten Sensoren kombiniert werden, was eine höherwertige Sicht auf das Verhalten eines Programms ermöglicht. Ein abgeleiteter Sensor kann die Werte mehrerer primitiver Sensoren oder den Wert eines Einzelsensors über einen gewissen Zeitraum kombinieren. Abgeleitete Sensoren werden definiert durch einfache arithmetische Ausdrücke, ergänzt um leistungsfähigere Kombinationsoperationen.

3.4 Beschreiben von Verwaltungs-Regeln

Es wird jetzt die oberste Schicht von Meta, die regelbasierende Kontrollsprache, beschrieben. Der Entwickler erstellt eine Beschreibung des beabsichtigten Verhaltens des Systems, bestehend aus Lomita Regeln der Form:

when Bedingung **do** Aktion

Dieses Statement bedeutet, daß beim Beobachten der spezifischen Bedingung die Aktion ausgeführt werden soll. Der Bedingungsteil jeder Regel ist eine Eigenschaft (ein Prädikat), die durch das unterlagerte Datenmodell ausgedrückt wird. Die Aktionskomponente ist lediglich eine Folge von Ausdrücken, die Sensoren und Aktoren miteinbezieht.

Bedingungen. Die Form der Bedingungen ist begrenzt auf einfache Prädikate (Eigenschaften), die optional in eine zeitliche Logik eingebunden sein können.

Aktionen. Der Rumpf der when Anweisung spezifiziert eine lineare Sequenz von Aktionen. Eine Kette von Regeln, bei der jede Aktion die Bedingung der nächsten Regel auslöst, verwirklicht einen komplexeren Kontrollfluß.
Diese Regel ist für eine echte Anwendung viel zu einfach. Zum Beispiel würde ein SigPro-Prozeß, der wiederholt beim Starten durch einen Softwarefehler wieder abstürzen würde, durch diese Regel immer wieder gestartet werden. In der Realität würde eine solche Regel eine komplexere Bedingung benötigen, die das wiederholte Starten innerhalb einer kurzen Zeit, z.B. 5 Minuten, erfassen würde, und auch andere Regeln, die bei wiederholten Fehlern das Bedienpersonal benachrichtigen würden.

Der Regelinterpreter von Lomita. Ein Lomita Kontrollprogramm wird in NPL übersetzt. Interpreter, die in der Meta-Bibliothek vorhanden sind, führen das übersetzte Programm verteilt aus. Das grundlegende Programmkonstrukt in NPL ist wie in Lomita eine Menge von Bedingungs-Aktions-Regeln. Wo Lomita jedoch einen großen Befehlsvorrat für die Objekte im Datenmodell unterstützt, stellt NPL nur einfache Postfix-Ausdrücke für primitive Sensoren und Aktoren zur Verfügung. Außerdem ist ein gegebener NPL-Ausdruck fest mit einem Prozeß verbunden und muß nichtlokale Sensoren explizit aufführen. Der Lomita-Compiler verteilt Teile des übersetzten Kontrollprogramms auf die Interpreter in den Anwendungskomponenten. Hierbei bemüht er sich, die Referenzen auf nichtlokale Sensoren und Aktoren zu minimieren, um die Antwortzeiten zu

verbessern. Das heißt, daß eine Regel, die nur Sensoren und Aktoren einer einzelnen Komponenten anspricht, auch lokal vom Interpreter dieser Komponenten ausgeführt wird.

Interpretation der Regeln. Die Ausführung einer Lomita when-do Regel kann als Ausführung eines endlichen Automaten gesehen werden. Einfache Bedingungen, die keine Zeitbeziehungen enthalten, werden auf einen einzelnen Zustandsübergang im Automaten abgebildet. Ein Ausdruck, der sich auf ein Zeitintervall bezieht, wird auf mehrere Zustandsübergänge abgebildet.

Fehlertoleranz der NPL Implementation. Mehreren Interpretern wird die Aufgabe zugewiesen, die Sensoren und Aktoren eines Komponententyps oder Aggregats zu bearbeiten. Einer der Interpreter trägt die Hauptverantwortung, die anderen Prozesse sind Backups.
Trotz dieser Redundanz können alle Interpreter gleichzeitig versagen. Um dem vorzubeugen kann jede Anwendung das Komplettversagen der Interpretergruppe überwachen. Wenn der Fall eintritt, beendet sich die Anwendung normalerweise. Das einfache Neustarten von Lomita mit Hilfe seiner Initialisierungsdateien startet auch die Anwendung in geordneter Weise neu. Da sich die Anwendung selbst beendet, können keine Waisen-Prozesse den Fehler des Kontrollprogramms überleben. Derartige Prozesse könnten für viel Verwirrung beim Neustart der Anwendung sorgen.
Eine zweite Option ist, überlebende Anwendungsprozesse laufen zu lassen und das Kontrollprogramm anzuweisen, beim Start nach existierenden Prozessen zu suchen. Eine dritte Option wäre, daß die Interpreter ihren Zustand auf Platte sichern.

Unteilbarkeit von Aktionen. Für einfache Aktionen, die nur aus einem Aktor-Aufruf bestehen (zu einem Prozeß oder zu einer Menge von Prozessen in einer Aggregation), stellt Isis mit der unteilbaren Benachrichtigung (atomic broadcast) die notwendige Nebenläufigkeitskontrolle zur Verfügung.
Wenn jedoch Metaoperationen mehrere Aktoren ansprechen, sind Nebenläufigkeit und Fehlerunteilbarkeit weit komplizierter. Wenn man sich zum Beispiel vorstellt, daß eine Metaregel auf einen Fehler reagiert, indem sie eine wenig ausgelastete Maschine auswählt, für sich reserviert und auf der Maschine ein Programm startet. Wenn mehrere solcher Regeln gleichzeitig angesprochen werden, darf die Maschine nur einmal reserviert werden. Der gleiche Fall tritt auf, wenn ein Mitglied einer Aggregation nach einem Fehler eines anderen Mitglieds dessen Aufgaben übernimmt.

4 Vergleich der oben vorgestellten Ansätze

Beide hier vorgestellte Ansätze versuchen ein Gerüst zur Verfügung zu stellen, welches die Entwicklung von stabiler verteilter Verwaltungs- und Anwendungssoftware begünstigt.

4.1 Gemeinsamkeiten

Architektur. Die Architekturen der oben vorgestellten Ansätze werden in drei Schichten gegliedert (auf OSI basierende Architektur/Meta Architektur):

1. Verwaltungsschicht (DAM Manager/Kontrollprogramm)
2. Kommunikationsschicht (DAM Agent/Meta System) und
3. Anwendungsschicht (App-MO/Application)

Übernahme existierender Software. In der OSI Management Umgebung wurde zur Implementierung eines Prototyps die Entwicklungplattform OSIMIS benutzt. OSIMIS liefert eine Menge von Einrichtungen zur Entwicklung von Manager Anwendungen, Agenten Anwendungen und MO.
Obwohl die Verwaltung verteilter Anwendungen verhältnismäßig neu ist, verwendet Meta Techniken, die von exitierenden Arbeiten in der verteilten Programmierung, darunter Überwachung des Laufzeitverhaltens, Debugging und Betriebssysteme, übernommen worden sind.

Objektorientierung. OSI MF benutzt eine objektorientierte Methodik für die Modellierung von offenen Systembetriebsmittel, die verwaltet werden sollen. Meta kann betrachtet werden als Lieferant einer temporären objektorientierten Datenbank, in der die Anwendung und deren Umgebung die Datenwerte zur Verfügung stellen.

4.2 Unterschiede

Der Hauptunterschied der beiden Ansätze liegt in der zweiten Schicht der Architektur, der Kommunikationsschicht. Während bei der DAM Architektur, die auf OSI basiert, die Verwaltungsschicht und die Anwendungsschicht nur über den DAM Agenten kommunizieren können, ist dagegen beim Meta System auch eine direkte Kommunikation zwischen der Verwaltungs- und der Anwendungsschicht möglich.

4.3 Leistungsfähigkeit

OSI MF. Die Leistungsfähigkeit einer verteilten Anwendung hängt vom aktuellen Zustand des zugrundeliegenden Netzwerks ab. Leider liegen hierzu keine aktuellen Messungen vor.

Meta System. Meta muß eine vorhersagbare kurze Zeitverzögerung zwischen dem Auftreten einer Bedingung und ihrer Meldung an das Kontrollprogramm gewährleisten. Es wurden die Ausführungszeiten der Regel:

when S **do** A;

gemessen, wobei S ein einfacher Sensor und A ein trivialer Aktor ist. Die Zeiten wurden auf zwei Sun Microsystems 4/60s mit einem 10 Megabit Ethernet und Isis Version 3.0 gemessen. Die Varianz war kleiner als 2 Prozent. Die lokale Ausführungszeit betrug 84 Millisekunden. Hierzu muß die Zeit addiert werden, die benötigt wird, um das Vorhandensein von externen Ereignissen abzuprüfen (was den Unix select Aufruf miteinbezieht), die beträchtlich sein kann. Wenn S und A auf einer Maschine lokalisiert waren und die Regel auf einer anderen Maschine ausgeführt wurde, betrug die Ausführungszeit 17.1 Millisekunden. Es ist offensichtlich, daß die Leistungsfähigkeit des Kontrollprogramms stark von der Örtlichkeit der NPL Ausdrücke abhängt. Referenzen zu lokalen, also im aktuellen Prozeß existierenden Sensoren und Aktoren sind sehr schnell. Das Ansprechen von Sensoren und Aktoren von Prozessen auf anderen Maschinen ist zeitintensiver, aber durchaus ausreichend bei den Anwendungen, die wir bis jetzt betrachtet haben.

4.4 Echtzeitverhalten

Meta System. Mit Meta wird ein Kontrollprogramm für die Verwaltung einer verteilten Anwendung zu einem sanften, in Echtzeit reagierenden System. Die Erzeugung eines robusten Kontrollprogramms erfordert die Bearbeitung von Punkten wie die Genauigkeit der Sensorintervalle oder die Verzögerung, mit der Aktoren wirken. Die Echtzeitanforderungen an Verwaltungssoftware für verteilte Anwendungen sind minimal, zumindest für alle Anwendungen, die bis heute in [MCD93] betrachtet wurden. Trotzdem ist es möglich, das Meta-Modell zu erweitern, damit es Echtzeitumgebungen bearbeiten kann. Das Haupthindernis hierbei ist die Echtzeitbenachrichtigung und ein Echtzeitbetriebssystem.
Die Vorhersagbarkeit der Meta Leistungsfähigkeit ist mindestens genauso wichtig wie die absolute Ausführungsgeschwindigkeit. Die oben aufgeführten Zahlen zeigten eine geringe Streuung, da sie unter kontrollierten Bedingungen erbracht wurden. Die aktuelle Version von Isis und das Unix Betriebssystem stellen keine vorhersagbare Leistungsfähigkeit zur Verfügung. Man könnte vorsichtig schätzen, daß die Ende-zu-Ende Sensor Verzögerung bis zu mehreren Sekunden sein kann. Es bleibt dem Anwendungsentwickler überlassen, die Genauigkeitsintervalle für Sensoren und die größte sinnvolle Abfrageperiode festzulegen. Für die Verwaltung der meisten Anwendungen werden Abfrageintervalle von einigen Sekunden oder Minuten sinnvoll sein.

4.5 Vorteile

OSI MF. Als Hauptvorteil ist hier zu nennen, daß auf der Managerseite wahrscheinlich keine neuen Konzepte benötigt werden und somit auch bereits entwickelte Managementplattformen und -oberflächen genutzt werden können. Dies bedeutet zum einen weniger Entwicklungs- aber auch weniger Einarbeitungsaufwand bei einer Einführung des Anwendungsmanagements in eine bestehende Netzwerkinstallation.

Meta System. Die Aspekte der Anwendungsverwaltung sind von denen der Hauptfunktionsbereiche getrennt, und die Schnittstelle zwischen beiden ist genau definiert.

Diese Trennung von Verwaltung und Funktion erleichtert die Veränderung der Verwaltung einer Anwendung und vermindert die Wahrscheinlichkeit, daß die Korrektheit des Rests des Programms beeinflußt wird.

4.6 Nachteile

OSI MF. Während der Entwicklung des Prototypen wurde auch deutlich, daß es in einer großen, verteilten Umgebung mit vielen Anwendungen, Agenten, Managern etc. notwendig sein wird, die Management Information zu verwalten. In einer verteilten Computer Umgebung, müssen die MOs verschiedenen Managern und Agenten zugänglich gemacht werden. Zur Zeit ist es innerhalb des OSIMIS Management Systems nicht möglich, da MOs nur dem Management Agenten bekannt sind, der sie erzeugt hat (d.h. innerhalb des Laufzeit Adreßraums des Agenten).

Meta System. Alle Meta Funktionen wurden eigens entwickelt, sogar mit eigener Kontroll-Hochsprache Lomita, deren Implementierung noch nicht abgeschlossen ist. Dies bedeutet einen, im Gegensatz zu OSI MF, hohen Entwicklungsaufwand. Der Meta-Ansatz stellt außerdem Sprachmittel für die Managementanwendungen bereit, während dies bei dem DAM-Ansatz gar nicht betrachtet wird.

5 Zusammenfassung und Ausblick

Nach bisherigen Erfahrungen stellt das OSI MF eine Menge mächtiger Mechanismen zur Verfügung, die ausreichend für das Management verteilter Anwendungen zu sein scheinen. Es hat sich herausgestellt, daß es einigermaßen einfach ist, OSI Management Anwendungen (sowohl Manager als auch Agenten) zu entwickeln. Dies hat wahrscheinlich viel mit den mächtigen und generischen Einrichtungen zu tun, die OSIMIS bietet. Die Benutzung schon existierender Protokolle und Frameworks wurde als vorteilhaft empfunden. Zuerst gibt es, egal ob CMIP oder SNMP, eine sauber definierte Menge von Diensten und Framework. Dies vereinfacht die konzeptionelle Gestaltung des Managementproblems. Zum zweiten gibt es oft existierende Werkzeuge, die die Gestaltung und Implementierung von Managern und Agenten stark vereinfachen können. Zum Dritten deutet der Prototyp an, daß es viele wiederverwendbare Komponenten gibt, die man zur Implementierung anderer Manager und Agenten verwenden kann, was die Entwicklung von verwaltbaren verteilten Anwendungen erleichtet. Obwohl das OSI MF als Basis für das Management verteilter Anwendungen erfolgreich verwendet wurde, sind diese Erfahrungen noch begrenzt. Es ist notwendig, die Rolle des OSI MF und die des SNMP in noch komplexeren verteilten Anwendungsuntersuchungen zu erforschen.
Die verteilte Programmierung verspricht größere Flexibilität, Verläßlichkeit und Leistungsfähigkeit. Viele Systementwickler stellen jedoch fest, daß die Versprechen unerfüllt bleiben, weil die erforderlichen Hilfsmittel fehlen. Die Verwaltung verteilter Systeme ist eines der wichtigsten – aber vernachlässigten – Gebiete. Das Meta System füllt diese Lücke und macht es einfacher, robuste verteilte Anwendungen unter Verwendung nicht eigenständig fehlertoleranter Komponenten zu erstellen. Designer können leichter planen

und die Richtigkeit der entstehenden Kontrollstrukturen besser überprüfen. Dies sind wichtige Schritte in Richtung eines offenen, verteilten Betriebssystems.
Bis zu einer gut funktionierenden Verwaltung verteilter Anwendung werden noch einige Forschungs- und Entwicklungsarbeiten zu leisten sein. Aber aufgrund der vorliegenden Tatsachen ist zu erkennen, daß auf diesem Gebiet einige Fortschritte gemacht worden sind, und auf baldige Ergebnisse gehofft werden kann.

Management by Delegation

Christian Krakovszky

Kurzfassung

An bestehenden Netzwerkmanagementimplementierungen wird kritisiert, daß zu viel Funktionalität in den Manager verlegt wird. Agenten spielen nur eine eingeschränkte, unterstützende Rolle. Diese ineffiziente Verteilung der Management–Verantwortung führt zu ernsthaften Schwierigkeiten in komplexen und Hochgeschwindigkeitsnetzwerken, wo eine Verteilung der Verwaltung obligatorisch ist.

Das in diesem Beitrag vorgestellte MAD Modell (manager agent delegation) erweitert das OSI–Netzwerkmanagementmodell bedeutend und ermöglicht eine flexible Verteilung und Koordination von Managementfunktionalität zwischen Manager und Agenten. Die Managementprogramme beinhalten Funktionen, die es den Agenten erlauben die Objekte lokal effizient zu verwalten, ohne den Manager unnötigerweise in Anspruch zu nehmen.

Die meisten Ideen in diesem Beitrag stammen aus dem Artikel "Network Management By Delegation" von Yechiam Yemini, German Goldszmidt und Shaula Yemini.

Der Beitrag beinhaltet auch die Beschreibung des SMARTS Operations Servers der Firma System Management ARTS, Inc., eine Implementierung der vorgestellen Konzepte.

1 Einleitung

Die Verwaltung von Netzwerken gewinnt zunehmend an Wichtigkeit. Umso größer und komplexer Netzwerksysteme werden, umso häufiger treten Fehler auf und umso teurer wird die Verwaltung des Systems. Die effiziente Verwaltung von Netzwerken und die Garantierung von Fehlerfreiheit sind zu einer Notwendigkeit für die Anwender geworden. Das Verwaltungsproblem beinhaltet eine Reihe nichttrivialer technischer Herausforderungen. Softwareanbieter, Standardisierungskomitees und Forschungsinstitute versuchen verwaltbare Systeme zu entwickeln. Leider hat man beim Entwurf von typischen Netzwerksystemen nicht ausreichend an die Verwaltung gedacht.

Ein typisches Netzwerkverwaltungssystem besteht aus *Agenten*, die für die Überwachung und Steuerung der *Verwaltungsobjekte* verantwortlich sind, und aus *Managern*. Letztere sammeln dynamische Zustandsinformation von den Agenten, werten diese aus und steuern die Agenten (z.B. wie sie eine Fehlerbehandlung durchführen sollen). Um ihre Aktivitäten zu koordinieren benutzen Manager und Agenten ein Managementprotokoll. Für die Beschreibung der Überwachungs– und Steuerungsoperationen, die auf die Verwaltungsobjekte angewendet werden, benutzt das Managementprotokoll eine spezielle Sprache *(management scripting language, MSL)*. Ein in dieser Sprache geschriebenes Programm heißt ein Managementskript. MSL beinhaltet Funktionen für die Strukturierung der Verwaltungsinformation (structure of management information, SMI) und für den Zugriff und die Bearbeitung dieser Information.

Der Kern der ganzen Problematik ist: Wie kann man die Verantwortung so auf Manager und Agenten verteilen, um eine effiziente Verwaltbarkeit des Systems zu erreichen? Um diese Verteilung zu erreichen benutzt man im OSI–CMIS (common management information service) Modell folgende Interaktionsprimitive:

- **M–CREATE**: Damit wird eine neue Instanz eines Verwaltungsobjektes erzeugt.
- **M–DELETE**: Damit wird eine Instanz eines Managementobjektes gelöscht.
- **M–GET**: Damit wird ein bestimmtes Attribut eines Verwaltungsobjektes ausgelesen (lesender Zugriff).
- **M–SET**: Damit wird ein bestimmtes Attribut eines Managementobjektes verändert (schreibender Zugriff).
- **M–ACTION**: Damit wird eine Operation auf einem Verwaltungsobjekt ausgelöst.
- **M–EVENT–REPORT**: Damit wird ein Ereignis gemeldet. Diese Ereignisse werden für asynchrone Meldungen des Agenten an den Manager eingesetzt.

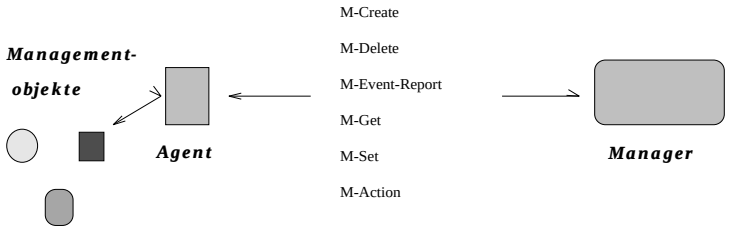

Abbildung 30. Die Struktur des CMIP Protokolls

Manager und Agenten können für ihre Interaktion auch nur eine Teilmenge der oben genannten Primitive benutzen. In diesem Fall müssen sie sich aber vorher darüber einigen, um welche Teilmenge es sich dabei handelt.

Bisherige Standardisierungsbemühungen konzentrieren sich auf das Managementprotokoll und auf das SMI–Modell (structure of management information). Das Managementmodell und die Manager/Agenten–Software werden implizit vorausgesetzt. Fragen wie 'Wie können Agenten und Manager entworfen und implementiert werden, um ein effizientes Management zu erreichen? Wie soll die Verantwortung auf Manager und Agenten verteilt werden? Wie können Agenten das Verhalten von Managementobjekten überwachen und koordinieren?' sind noch nicht vollständig geklärt.

Dieser Beitrag untersucht grundlegende Probleme beim Entwurf von Netzwerkmanagementsystemen. Der 2. Abschnitt beschreibt das Problem der Mikroverwaltung in bestehenden Protokollen. Das neue Delegationsmodell wird im nächsten Abschnitt eingeführt. Der 4. Abschnitt beschäftigt sich mit dem Entwurf von Skriptsprachen, die eine effiziente Überwachung und Steuerung ermöglichen. Die Koordination zwischen Manager und Agenten wird dann im nächsten Abschnitt betrachtet. Der 6. Abschnitt beschreibt einige

wichtige Aspekte beim Entwurf von Agenten. Im 7. Abschnitt wird eine Implementierung der neu eingeführten Konzepte vorgestellt. Der letzte Abschnitt beinhaltet eine Bewertung und einen Ausblick dieses neuen Modells.

2 Das Problem des Mikromanagements

2.1 Die Ausdruckskraft eines Protokolls kann das Management eines Systems in hohem Maße begrenzen

Gegeben sei ein Agent, der für die Überwachung und Steuerung eines Verbindungsobjektes verantwortlich ist. Eine typische Situation besteht zum Beispiel aus der Erkennung von ungewöhnlichen Verbindungsbedingungen, der Ausführung eines Tests für die Identifizierung der Fehlerursache und der Ausführung von Prozeduren, die das Problem beseitigen.

Beispiel 1: Managementskript für Verbindungsfehler

```
On (link.control_stat > normal_stat) & (link.q_length > normal_q);
Begin
link.handle_congestion;
link.test;
...
If link.failure then
    Begin
       recover(link.failure_type);
       ...
       notify(Manager, link.failure_params);
    End
End
```

Sowohl die Erkennung des Fehlers als auch die Ausführung von Aktionen benötigen Daten, die sich lokal beim Agenten und beim Managementobjekt (in diesem Fall: Verbindungsobjekt) befinden. Der Manager muß das Managementprotokoll benutzen, um die Ausführungsverantwortung an den Agenten zu übertragen. Als nächstes werden die Schritte, die für die Ausführung des Skripts unter Benutzung des CMIP (common management information protocol) Protokolls durchgeführt werden, näher betrachtet.

Als erstes muß das ungewöhnliche Ereignis entdeckt werden. Grundsätzlich gibt es hierfür zwei Möglichkeiten: Polling und Traps. Beide sollen im weiteren genauer betrachtet werden.

Ereignisse treten selten auf und haben meistens eine sehr kurze Dauer. Die Polling–Frequenz des Managers muß also groß sein, damit keine Ereignisse verloren gehen. Das kann zur Erschöpfung der Kommunikationsbandbreite führen. Ein weiteres Problem beim Polling ist, daß intermittierende Ereignisse nicht erkannt werden. Zusammenfassend kann man also sagen, daß der abfragende Ansatz eine sehr schlechte Methode zum Aufspüren von Ereignissen in einem Netzwerksystem ist.

Diese wesentlichen Beschränkungen haben die Entwerfer des CMIP Protokolls veranlaßt, einen Ereigniserkennungsmechanismus zu implementieren. Wenn ein Agent die M–EVENT–REPORT Primitive benutzt, dann kann er mittels eines Traps benachrichtigt werden, sobald ein Fehler eintritt. Das hat den Vorteil, daß sofort eine Benachrichtigung erfolgt. Leider ist der Manager nicht fähig zusammengesetzte Ereignisse zu erkennen. So wird er im Beispiel die Ereignisse `''(link.control_stat > normal_stat)''` und `''(link.q_length > normal_q)''` unabhängig voneinander empfangen und dann entscheiden, daß diese zusammengehören. Diese Teilung der Ereigniserkennung führt zu Ineffizienz und Risiken. Traps treten zufällig auf und so kann es vorkommen, daß ein Ereignis unbemerkt bleibt, auch wenn die einzelnen Komponenten richtig erkannt wurden. Manager und Agenten verbrauchen teure Bandbreite und Ausführungszeit, um sich die Erkennungsverantwortung zu teilen. [1]

Sobald das Ereignis entdeckt wurde, werden Handlungen zur Feststellung der Fehlerursache eingeleitet. Solche Handlungen sind zum Beispiel `''link.test, If link.failure, recover(link.failure_type)''`. Das Primitiv M–ACTION wird für das Testen benötigt. Den Wert von `link.failure` kann man mit Hilfe von M–GET herausfinden. Weder SNMP (simple network monitoring protocol, das im *Internet* benutzte Managementprotokoll) noch CMIP bieten brauchbare Mechanismen für die Kombination von Primitiven zur Handhabung von zusammengesetzten Skripten. Deshalb muß der Manager den Agenten durch jeden einzelnen Schritt des Skripts führen. Diesen Vorgang bezeichnet man als Mikroverwaltung.

Mikromanagement führt zu ineffizienten und teuren Managementsystemen. Teure Kommunikationsbandbreite und Rechenzeit sind sogar für einfache Aufgaben notwendig. Dies führt zu großen Einschränkungen bzgl. des Managements. Es können nur triviale Managementskripten ausgeführt werden. Mikromanagement bedeutet auch, daß alle Managementfunktionen im Manager zentralisiert werden. Fällt ein Manager aus, so können die Agenten den Fehler nicht beheben, weil sie ohne den Manager nichts tun können. So können auch kleine Fehler zum Zusammenbruch des gesamten Systems führen.

Die Zentralisierung der Verwaltung ist nicht vereinbar mit dem Bedarf von komplexen Hochgeschwindigkeitsnetzwerken. In solchen Systemen führt Zentralisierung zu Ineffizienz. Zentralisierung steht auch im Gegensatz zu gegenwärtigen und zukünftigen Entwicklungen. Agenten besitzen immer mehr Rechenkapazität, die leider nicht ausgenutzt wird. In zukünftigen Systemen wird man berücksichtigen müssen, daß Agenten sehr wohl in der Lage sind, Managementkompetenz zu übernehmen. Die Beteiligung des Managers in der Ausführung ist weder notwendig noch nützlich. Sie ist lediglich eine Folge der Einschränkungen gegenwärtiger Protokolle.

2.2 Weitere Modelle für die Interaktion zwischen Manager und Agenten

Der entfernte Prozeduraufruf (remote procedure call, RPC) ist ein zentrales Modell für die Strukturierung verteilter Systeme. In diesem Modell exportiert der Server eine Anzahl von festgelegten Prozeduren, die dann von entfernten Clients aufgerufen werden können.

[1] Dieser Abschnitt aus dem Artikel von Yemini, Goldzsmidt und Yemini ist falsch. Mit CMIP können sehr wohl zusammengesetzte Ereignisse erkannt und an den Manager weitergeleitet werden.

Bei einem RPC Aufruf wird der Aufrufer suspendiert, die entfernte Prozedur wird ausgeführt, und die Ergebnisse werden zum Aufrufer zurückgegeben, der seine Ausführung dann wiederaufnimmt.

Nun stellt sich die Frage, ob der RPC–Mechanismus das Problem des Mikromanagements lösen kann. Angenommen das ganze Managerskript aus dem Beispiel wird unter dem Namen `link_handler` im Agentencode integriert. Ein Manager wird dann einen RPC durchführen müssen, um die Ausführung des Skripts beim Agenten auszulösen.

Leider bringt diese Lösung zwei erhebliche Schwierigkeiten mit sich. Der Grund des ersten Problems liegt in der synchronen Aufrufsemantik des RPC–Mechanismus. Ein typisches Management Skript beinhaltet auch Aktionen, die von unabhängigen Ereignissen im Agenten ausgelöst werden. Dies geschieht asynchron mit dem RPC–Aufruf des Managers. Die eigene Ausführung des Managers darf nicht mit der Ausführung des Managementskripts verbunden werden. Ein RPC Aufruf würde den Manager bis zur endgültigen Ausführung des Skripts blockieren.

Die zweite Beschränkung beim RPC Modell ist die statische Natur der exportierten Prozeduren. Dies hat zur Folge, daß man beim Entwurf eines Agenten die ganze Palette von Managementskripten codieren müßte, an die der Agent beteiligt sein könnte. Es ist aber unmöglich alle Situationen zu berücksichtigen, die auftreten können. Auch ist das nicht wünschenswert weil es die Komplexität des Agenten erhöhen, und seine Leistungsfähigkeit beschränken würde.

Eine andere Interaktionsmöglichkeit in verteilten Systemen ist die entfernte Ausführung REV (remote evaluation). Diese erlaubt den Transfer einer Prozedur von einem Clienten zu einem Server, wo sie von einem Interpreter ausgeführt wird. Im Gegensatz zum RPC kann die Programmierung eines Managementskripts zur Zeit des Managerentwurfs gemacht werden. Leider bleibt immer noch das Problem, daß eine REV–Prozedur synchron aufgerufen wird. Somit ist weder der RPC noch der REV für das Netzwerkmanagement geeignet.

3 Die entfernte Delegation

Das Problem des Mikromanagements kann durch die *Delegation* von ganzen Skripten vom Manager an die Agenten gelöst werden. Der Agent würde das Skript ohne das Eingreifen des Managers ausführen. Er ist für die Ausführung aller Primitive von der Ereigniserkennung bis zur Ausführung von fehlerbeseitigenden Aktionen verantwortlich. Die Managementprogramme werden als Teil einer spezifischen Verwaltungsplattform entworfen und codiert. Delegation ist dynamisch und erlaubt es, flexibel einen Teil der Verantwortung vom Manager an den Agenten zu übertragen.

3.1 Das delegierte Programm

Ein delegiertes Programm ist ein in MSL (management scripting language) geschriebenes Managementskript, das von einem Manager an einen Agenten übertragen wird.

MSL beinhaltet Primitive für die Überwachung und Steuerung von Managementobjekten und für das Zusammenfügen dieser Primitive zu Programmen. Zusätzlich beinhaltet die Sprache Primitive für die Unterstützung einer Interaktion zwischen Managementprogrammen und *Agentendiensten*. Ein Agentendienst ist z.B. die Benachrichtigung des Managers beim Auftreten von ungewöhnlichen Ereignissen. Agentendienste unterstützen auch eine Koordination mit anderen delegierten Programmen. Das delegierte Programm kann solche Konstrukte benutzen, um dem Agenten Verwaltungsfähigkeiten bereitzustellen.

Das Beispiel aus dem letzten Abschnitt könnte wie folgt behandelt werden. Das Managementskript wird in einer geeigneten MSL codiert. Dann wird das Programm an den Agenten übertragen und für die Ausführung eingerichtet. Als erstes fordert der Manager einen Agenten an, der das Ereignis `''(link.control_stat > normal_stat)''` `& ''(link.q_length > normal_q)''` behandeln kann. Der Agent überwacht das Auftreten des Ereignisses, und wenn er es entdeckt hat, benachrichtigt er das delegierte Programm. Das delegierte Programm wartet auf die Nachricht und macht dann mit den restlichen Aktionen wie `''link.handle_congestiion'', ...` weiter.

Delegation ermöglicht es, daß ein Manager die Verantwortung für die Ausführung der Funktionen aus dem delegierten Skript an einen entfernten Agenten übertragen kann. Der Agent kann diese Funktionen unabhängig vom Manager ausführen. Dadurch kann die Funktionalität des Agenten zur *Laufzeit* erweitert werden.

In Zusammenhang mit der Delegation von Programmen bleiben noch einige Fragen offen:

1. Welche Programmiersprache ist am besten für die Codierung geeignet?
2. Wie kann man die Lebensdauer eines Managementprogramms steuern?
3. Wie kann ein Manager die Ausführung von Programmen steuern, die er delegiert hat?
4. Wie bekommt ein Manager Zugang zu Managementobjekten?
5. Wie kann ein Agent den nebenläufigen Zugriff auf Verwaltungsobjekten steuern und überwachen?
6. Wie kann ein Managementprogramm seine Ausführung mit anderen Programmen koordinieren?

Einige dieser Fragen werden im folgenden geklärt.

3.2 Die Lebensdauer eines delegierten Programms

Manager müssen fähig sein, unabhängig von den Managementprogrammen, die von ihnen delegiert wurden, weiterzulaufen. D.h. daß ein delegiertes Programm eine unabhängige Lebensdauer hat. Der Manager muß die Kontrolle über das delegierte Programm behalten, um ihm die Terminierung, Suspendierung und Wiederaufnahme des Programms zu erlauben. Gegeben sei z.B. ein Manager, der ein Programm delegiert, das sämtliche Verwaltungsvariablen überwacht und den Zustand periodisch dem Manager meldet. Zu

gewissen Zeitpunkten wird der Manager die Meldung zurückstellen, weil er sich wichtigeren Problemen zuwenden möchte. Sobald diese gelöst sind kann die Meldung behandelt werden. Diese Funktionalität muß in speziellen Prozeduren implementiert werden.

Konzeptionell kann die Lebensdauer von Managementprogrammen als ein Leichtgewichtsprozess (thread) in der Umgebung des Agenten betrachtet werden. Im Unterschied zu einem Prozeß verfügt ein thread nicht über einem eigenen Adreßraum. In diesem Fall ist ein thread eine Ausführungseinheit, über die die Umgebung des Agenten Kontrolle hat: Er kann initialisiert, suspendiert, wiederaufgenommen und beendet werden. Delegation kann als die Erzeugung eines Leichtgewichtsprozesses in Zusammenhang mit einem Managementskript in der Umgebung des Agenten betrachtet werden. Die Lebensdauer dieser delegierten Managementthreads (DMT) kann vom Manager gesteuert werden. Aus der Perspektive eines Protokolls erreicht man die Steuerung der Lebensdauer mittels Generalisierung von CMIS Primitiven wie M–CREATE und M–DELETE. Solche Konstrukte benutzt man typischerweise zur Lebensdauersteuerung von inaktiven Objekten. Analog dazu kann ein DMT als ein "aktives Managementobjekt" betrachtet werden. Aus der Perspektive einer Implementierung muß ein DMT nebenläufig mit dem Agentenprozeß ausgeführt werden und benötigt so eine Laufzeitumgebung. Die Lebensdauer eines DMTs muß mit der Lebensdauer der von ihm benötigten Verwaltungsobjekte synchronisiert werden. Der Agent oder die darunterliegende Ausführungsumgebung muß dafür sorgen, daß Managementobjekte von einem DMT zugreifbar sind.

3.3 Delegation und Zugriffsrechte

Ein DMT benötigt eine Möglichkeit an Informationen des Agenten heranzukommen und mit dessen Umgebung zu interagieren. Der DMT benötigt einen *kontrollierten Zugriff* auf die Objekte des Agenten. Ein willkürlicher Zugriff eines DMT auf den Adreßraum des Agenten muß verhindert werden, um Konflikte zu vermeiden. Das Verhalten der Agenten darf von den delegierten threads nicht gestört werden. Der DMT sollte nur auf diejenigen Attribute der Managementobjekte Zugriff haben, die explizit exportiert werden.

Der DMT sollte fähig sein, lokal alle oder eine Teilmenge der Aktionen aufzurufen, die sein Erzeuger (Manager) entfernt aufrufen kann. Das delegierte Programm erbt so Zugriffsfähigkeiten von seinem Erzeuger. Zusätzlich sollte der DMT die Möglichkeit haben, Nachrichten an seinen Erzeuger zu senden und sogar Aktionen im Erzeuger auslösen können. Ein DMT Programm sollte auch Ereignisse empfangen können, die ursprünglich an seinen Erzeuger gerichtet waren.

4 Management–Skriptsprachen

Eine Management–Skriptsprache wird benutzt, um ein delegiertes Programm zu codieren. Die Sprache muß die Delegation und Koordination von Managementprogrammen unterstützen. Außerdem sollte sie Möglichkeiten zur Spezifikation von Ereignissen als boolsche Ausdrücke und Mechanismen zur Bearbeitung von Ereignissen besitzen. Der nächste Abschnitt beschreibt Ansprüche, die an moderne Skriptsprachen gestellt werden.

4.1 Die Abstufung von Managementsprachen

Delegierte Managementsprachen müssen mit den rechnerspezifischen Fähigkeiten der Agenten kompatibel sein. Netzwerksysteme bestehen aus sehr unterschiedlichen Hardwarekomponenten. Agenten, die Modems oder Multiplexer überwachen, bieten nur sehr eingeschränkte Managementfähigkeiten, während solche, die Rechner steuern, sehr gute Verwaltungsmöglichkeiten bieten. Ein Managementsystem muß also ein ganzes Spektrum an Managementmöglichkeiten bieten. Bei OSI Managementprotokollen erreicht man dies mit dem Konzept des *association type*. Ein assotiation type definiert eine Teilmenge des CMIP Protokolls. Man bezeichnet Teilsprachen von Management–Skriptsprachen als Stufen (grades) einer Hierarchie von Sprachen. Das Abstufen definiert eine Ordnung auf der Menge der Teilsprachen von Management–Skriptsprachen. Am unteren Ende dieser Ordnung befindet sich die leere Sprache (keine Verwaltbarkeit). Darüber befinden sich vier Stufen, die alle verschiedene Sprachen aus Managementprimitiven definieren. Zusätzliche Primitiven und Kompositionsoperatoren erweitern diese Sprachen und bilden eine höhere Stufe in der Hierarchie. Am oberen Ende der Ordnung befindet sich eine allgemeine Programmiersprache, die zur Unterstützung von Management voll ausgebaut wurde.

Die Kosten des Einsatzes von Managementfähigkeiten, die von verschiedenen Stufen angeboten werden, spiegeln sich in der Komplexität der Agenten wider. Untere Stufen sind für ganz einfache Agenten geeignet, die über wenig Speicher und Rechenkapazität verfügen. Ein Agent der nur für die Steuerung eines einfachen Geräts (Modem, Multiplexer) verantwortlich ist, wird eine sehr einfache Sprache benutzen. Agenten, die für komplexere Systeme (Rechner) verantwortlich sind, benutzen eine Sprache aus einer höheren Stufe. Mikromanagement entsteht dann, wenn Manager und Agenten, die für komplexe Verwaltungsfunktionen verantwortlich, sind nur eine sehr einfache Sprache benutzen können.

Das MAD Modell erweitert die OSI assotiation types und bietet eine höhere Programmiersprache an, die für die Verwaltung von komplexen Systemen eingesetzt werden kann. Diese Erweiterungen enthalten neue Primitive für Überwachung, Koordinierung und Steurung, sowie Möglichkeiten für die Komposition dieser Primitive und für die Delegation von Skripten an Agenten.

5 Das Problem der Koordinierung

Die Verteilung von Managementfunktionen setzt einen effizienten Mechanismus für die Koordinierung von Managementaktivitäten voraus.

Beispiel 2: Der Nichtdeterminismus von Managementaktivitäten

Gegeben sei eine X.25 Zugangspunkt mit einem Agenten, der für die Verwaltung der Steuerungsoperationen verantwortlich ist. An den Agenten werden nun drei Skripten von drei verschiedenen Managern delegiert. Der Grund für die Auslösung der Delegation ist ein Ereignis E=" buffers full".

- A_1 = Ausführung von lokalen Kontrollroutinen für die Feststellung von möglichen Verbindungs- oder Steuerungsfehlern und Neubooten des Systems im Fehlerfall.

- A_2 = Berechnung und Mitteilung von Leistungsparametern, Abbruch einiger Steuerungsoperationen und Einführung einer Begrenzung für die Benutzung der lokalen Resourcen.

- A_3 = Zurücksetzen aller Bufferinhalte.

Dies ist ein typisches Beispiel, wo verteilte Managementaktivitäten in einem Netzwerksystem zu unvorhersehbaren Managementaktionen führen können. Die Reihenfolge, in der die Skripten ausgeführt werden, führt zu ganz unterschiedlichen Ergebnissen. Wenn A_1 zuerst ausgeführt wird, dann wird der X.25 Zugagnspunkt neugebootet und A_2 meldet ganz andere Ergebnisse als im Falle daß A_1 nach A_2 ausgeführt wird. In diesem Fall wäre es günstiger, wenn A_2 vor A_1 ausgeführt wird, weil das Neubooten des X.25 Zugangpunktes den Fehler nicht beheben kann. Offensichtlich behindern sich die Manager gegenseitig.

Ein globaler Netzwerkmanager ist nicht der einzige Manager in einem Netzwerksystem. Meistens werden Geräte mit Softwarepaketen ausgestattet, die auch Aufgaben von (lokalen) Managern ausführen. Außerdem spielt jede Art von Überwachungs- oder Prüfungsroutine im System auch die Rolle eines Managers. Beim Entwurf eines globalen Managementsystems ist es sehr schwierig alle Managementaktivitäten und die Beziehungen zwischen diesen zu berücksichtigen. Die Verteilung von Managementfunktionen auf verschiedene Geräte geschieht oft zufällig und führt so zu erheblichen Koordinierungsproblemen.

Nichtdeterminismus kann aber vermieden werden. Man muß nur Interaktionen zwischen nebenläufigen Managementprogrammen unterbinden. Zwei Managementprogramme heißen *nebenläufig*, wenn sie gleichzeitig ausgefürt werden können; sie heißen assoziativ, wenn sie unabhängig von der Ausführungsreihenfolge zum gleichen Ergebnis führen. Beim Entwurf eines Managementsystems muß man darauf achten, daß nebenläufige Managementprogramme auch assoziativ sind. Die Ausführungsreihenfolge von nichtassoziativen nebenläufigen Programmen muß sehr sorgfältig gesteuert werden, um sicherzustellen, daß die Ergebnisse deterministisch sind. Hier gibt es einige Ähnlichkeiten mit der Serialisierung von Transaktionen bei Datenbanksystemen. [BHG87] Ein Ansatz für die Lösung des Problems ist die Einführung von Prioritäten für nichtassoziative Programme. Es könnten aber auch Manager mit Dringlichkeiten ausgestattet werden. Zusätzlich müßte man die Skriptsprache um Mechanismen erweitern, die es erlauben mit nichtassoziativen Programmen korrekt umzugehen.

6 Anforderungen an Agenten

Agenten führen Managementprogramme aus, sie müssen DMTs aufrechterhalten und koordinieren diese Funktionen mit den Eigenschaften der Verwaltungsobjekte. Dieser Abschnitt beschreibt einige grundlegenden Fragen beim Entwurf von Agenten.

6.1 Der Zugriff auf Verwaltungsobjekte

Agenten müssen den Zugriff von Managementprogrammen auf die Verwaltungsobjekte steuern. Agentenfunktionen sollten beim Entwurf von Managementobjekten integriert werden. Agenten müssen eine Schnittstelle für Verwaltungsobjekte beinhalten. Diese Schnittstellen können mit ähnlichen Konstrukten wie die privaten Attribute einer C++ Klasse implementiert werden.

Steuerungsaktionen wirken sich meistens in Änderungen der Managementobjekte oder den Aufruf von Prozeduren, die von diesen exportiert werden, aus. Auch für diese Zugriffsart sollte man die gleichen Schnittstellentechniken benutzen. Auf jeden Fall muß jeder unerwünschte Zugriff auf ein Verwaltungsobjekt unterbunden werden.

6.2 Ereignisbehandlung

Managementereignisse können als boolsche Ausdrücke über die Attribute von Verwaltungsobjekten definiert werden. Die DMTs müssen sehr schnell auf das Auftreten von Ereignissen reagieren können. Das heißt, daß die Agenten fähig sein müssen Ereignisbedingungen auszuwerten und so schnell wie möglich die Ausführung eines DMTs auslösen. Zusätzlich sollten Agenten Mechanismen für die Koordinierung von DMTs beinhalten, so wie diese in den Abschnitten 3. und 5. vorgestellt wurden.

Die Auswertung von Ereignissen ist, relativ zu deren Entstehung im realen Objekt, zeitlich verschoben. Diese Verzögerungen entstehen durch den Erkennungsmechanismus, der von den Agenten benutzt wird. Benutzt der Agent Polling, um sich Informationen zu beschaffen, so können diese Verzögerungen sogar erheblich sein. Zusätzliche Verzögerungen entstehen bei der Mitteilung von Ereignissen an das delegierte Programm. Insgesamt kann man also sagen, daß die Erkennung von Ereignissen nur eine Annäherung an das reale Auftauchen ist. Das kann unter gewissen Umständen zu Schwierigkeiten führen, vor allem in Systemen, wo die Antwortzeiten auf Ereignissen wichtig sind. Eine effiziente Ereigniserkennung ist in allen Systemen, vor allem aber in Realzeitsystemen, eine sehr wichtige Komponente beim Entwurf von Agenten.

6.3 Die Struktur von Agenten

Bild 2 zeigt die Beziehungen zwischen einem Manager, einem Agenten, einem delegierten Programm und einigen Verwaltungsobjekten. Der Manager benutzt das Delegationsprotokoll, um die Ausführung des Programms an den Agenten weiterzuleiten. Das Programm wird in der Umgebung des Agenten ausgeführt und erhält Zugang zu den Managementobjekten. Der Agent hat vor allem eine *auslösende* und *koordinierende* Aufgabe. Der private Adreßraum muß vor einem unkontrollierten Zugriff durch den DMT geschützt werden.

Der nächste Abschnitt beschreibt eine komerzielle Implementierung des Delegationsmodells.

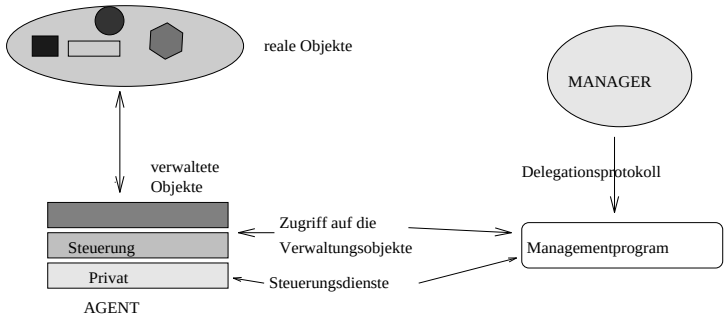

Abbildung 31. Die Beziehung zwischen Manager und Agent mittels DMT

7 SMARTS Operations Server (SOS)

SOS enthält eine Implementierung der Konzepte, die in den bisherigen Abschnitten vorgestellt wurden. Allerdings darf man SOS nicht als ein neues, eigenständiges Protokoll verstehen. SOS ist vor allem ein Versuch, standardisierte Protokolle wie SNMP und CMIP mit der Idee der Delegation zusammenzubringen.

7.1 Einführung

Der SMARTS Operations Server ist eine Entwicklungs- und Laufzeitumgebung für Management durch Delegation. Aus der Sicht von SOS ist Delegation ein verteiltes Paradigma, das Programme zu den Daten bringt und nicht Daten zu Programmen. Was können nun delegierte Programme?

- Sie erweitern die Menge der unterstützten Anwendungen und Protokolle. Auf SOS basierte Manager und Agenten können zur Laufzeit dynamisch um neue Funktionen erweitert werden.

- Sie automatisieren die Verwaltung eines Netzwerksystems ganz oder teilweise. Netzwerksysteme, die sich selbst verwalten, verringern die Komplexität von Endbenutzeroperationen und verbessern so die Robustheit dieser Systeme.

- Sie führen datenintensive Berechnungen direkt bei der Quelle aus. Verteilte Managementfunktionen, die sich nahe bei den Daten befinden, erlauben Realzeitüberwachung. Dies wäre von einer entfernten Managementstation nicht möglich.

SOS ist also ein ideales Werkzeug für die Entwicklung von skalierbaren intelligenten Netzwerken, die sich kontinuierlich neuen Benutzeranforderungen und neuen Standards anpassen müssen. Ein großer Vorteil der Delegation ist die Möglichkeit des Hinzufügens von Modulen zur Laufzeit. Delegation erleichtert auch das Schreiben von plattformunabhängigem Code, was Portierungen wesentlich erleichtert. Außerdem trägt Delegation in hohem Maße zur Entlastung der Kommunikationsbandbreite in Netzwerksystemen bei.

7.2 Management–Skriptsprachen

In SMARTS können delegierte Programme in TCL codiert werden. TCL ist eine sehr flexible Skriptsprache, die an der Berkeley–Universität entwickelt wurde. Obwohl nicht speziell für Netzwerkmanagement entworfen, erfüllt TCL die im 4. Abschnitt an Management–Skriptsprachen gestellen Anforderungen. Außerdem kann man delegierte Module auch in C oder C++ codieren. In diesem Fall muß das übersetzte Programm delegiert werden.

TCL bietet gegenüber C und C++ zwei Vorteile:

1. Plattformunabhängigkeit: Das delegierte Skript kann überall, wo ein TCL–Interpreter installiert ist, ausgeführt werden.
2. Geringe Belastung der Bandbreite: Ein TCL–Programm benötigt wesentlich weniger Bits als ein übersetztes C oder C++ Programm.

7.3 Die Architektur von SOS

Der SMARTS Operations Server besteht aus folgenden vier Schichten:

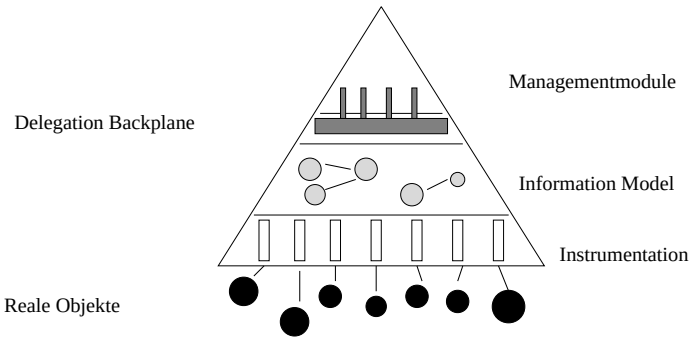

Abbildung 32. Die Architektur von SMARTS

Managementmodule: Das sind Sofwaremodule, die zu einem gegebenen Zeitpunkt in einem SO–Server geladen werden. Diese können Protokollmodule wie SNMPv1, SNMPv2, CMIP, RPC–Agenten- und/oder Managerschnittstellen, Informationsmodule und Anwendungsmodule für Überwachung, Analyse und Steuerung enthalten.

Delegation Backplane: Ein backplane für die Unterbringung der Managementmodule. Dieser backplane unterstützt das dynamische Laden von Modulen und Multitasking von Programmen. Die Systemschnittstelle ist POSIX–konform, was eine Portierung auf unterschiedliche Systeme ermöglicht.

Information Model: Diese Schicht enthält eine Modellierung der realen Verwaltungsobjekte im SOS–Bereich. Das Informationsmodell exportiert zwei Schnittstellen:

1. Die Schnittstelle nach oben bietet eine lokale API mit Zugriff auf den modellierten Objekten. Diese Objekte können in einer SNMP MIB (Management Information Base) organisiert sein. Objekte können aber auch mit dem objektorientierten SMARTS–Modeler dargestellt werden.

2. Die Schnittstelle nach unten begünstigt durch ihre Einfachheit die Erweiterung der darunterliegenden Schicht.

Model Instrumentation: Diese Schicht enthält Funktionen für die Steuerung und Kontrolle der realen Verwaltungsobjekte. Diese sogenannten Instrumentatiosfunktionen benutzen Standardprotokolle oder DMA (direct memory access) für den Zugriff auf die realen Objekte.

7.4 Zusammenfassung von SMARTS–Vorteilen

Der SMARTS Operations Server ist ein gelungener Kompromiß zwischen dem neuen Konzept der Delegation und Protokollen wie SNMP und CMIP. Das Zusammenwirken unterschiedlicher Konzepte erhöht die Flexibilität des Produkts und bietet einige wesentliche Vorteile gegenüber bisherigen Netzwerkmanagementlösungen:

- Erweiterbarkeit
- Bessere Leistung
- Erhöhte Zuverlässigkeit
- Einfache Entwicklung von Managementlösungen
- Automatisierung der Verwaltung

8 Bewertung und Ausblick

Dieser Beitrag zeigt, wie man Managementfunktionalität auf Manager und Agenten verteilen kann. Das Konzept der Delegation erlaubt es, daß Manager entlastet werden und daß Agenten selbständig auch komplexe Managementaufgaben durchführen können.

Die Entwickler von SNMP (das im Internet benutzte Managementprotokoll) verfolgen eine ganz andere Philosophie:

Das Hinzufügen von Netzverwaltung in einem Netzwerksystem sollte möglichst geringe Auswirkungen auf die verwalteten Knoten (Agenten) haben.

Aus der Sicht der SNMP–Entwickler sprechen folgende Gründe für eine geringe Belastung der Agenten:

- Manager–Software wird meistens auf einer Workstation installiert. Die Erfahrung zeigt, daß die meisten Rechner sehr wohl in der Lage sind, die benötigten Betriebsmittel für eine effektive Managementstation bereitzustellen.
- Da es viel mehr Agenten als Manager gibt, paßt dieser Ansatz auch sehr gut für große Netze. Es ist besser, nennenswerte Funktionalität von einem Bruchteil der Geräte zu verlangen anstatt von der großen Mehrzahl von Geräten.

- In einigen Fällen bringt eine Verlagerung von mehr Managementfunktionalität auf die Agenten das Hinzufügen einer weiteren Speicherkarte oder eines schnelleren Prozessors zu diesen Agenten mit sich. Die Akzeptanz für eine derartige Aufrüstung ist bei den Benutzern nicht sehr groß.

Gegenüber diesem und ähnlichen Ansätzen bietet Delegation einige Vorteile:

- Entlastung der Kommunikationsbandbreite durch einen geringeren Datenfluß zwischen Manager und Agenten.
- Höhere Flexibilität: Die Funktionalität von Managern und Agenten kann zur Laufzeit erweitert werden.
- Automatisierung der Verwaltung.

Ein Vorteil der Delegation ist auch die Tatsache, daß sie durchaus mit gegenwärtigen Ansätzen zusammenleben kann. Das zeigt uns auch die Implementierung des SMARTS Operations Servers. Delegation ist also kein Muß. In einigen Fällen, in denen Agenten wirklich zu wenig Rechenkapazität und Speicher besitzten, können bisherige Methoden eingesetzt werden. Man hat also die Möglichkeit, Delegation nur in den Fällen einzusetzen, wo ein Vorteil dieses neuen Konzepts klar ersichtlich ist. Somit ist Delegation ein Konzept, das im Bereich des Netzwerkmanagements eine reelle Chance hat.

Teil II

Hochgeschwindigkeitskommunikation

Vorwärtsfehlerkorrektur in ATM-Netzen

Steffen Schlachter

Kurzfassung

Das gängigste Verfahren bei der Korrektur von Übertragungsfehlern bei der Datenübertragung via Netzwerke ist die Übertragungswiederholung (ARQ, automatic repeat request), wie sie heute weitgehend in den Netzprotokollen (ISO/OSI, TCP/IP, usw.) angewendet wird. Verschiedene Anwendungen aber erfordern oftmals isochrone Datenströme. Hier kommt beispielsweise die Vorwärtsfehlerkorrektur (auch vorausschauende Fehlerkorrektur genannt) nutzbringend zum Einsatz.

Kapitel 1 beschreibt das grundlegende Prinzip der Vorwärtsfehlerkorrektur und zeigt einige Anwendungsgebiete auf. Kapitel 2 gibt eine kurze Einführung in die mathematischen Grundlagen, soweit sie zum grundlegenden Verständnis des Verfahrens notwendig sind. Eine ausführlichere Beschreibung der Methode der Wiederherstellung findet sich in Kapitel 3 wieder. Kapitel 4 diskutiert kurz die Spezifikation der FEC-Service Specific Convergence Sublayer für die AAL-Typ 5, und Kapitel 5 legt eine Leistungsanalyse des Verfahrens der Vorwärtsfehlerkorrektur bei der Übertragung großer Datenmengen und Videosequenzen dar.

1 Einsatz und Prinzip der FEC

Ist die Datenverlustrate in einem Netzwerk größer, als es die entsprechenden Anwendungen erlauben, so ist es die Aufgabe des Transportprotokolls, diesen Unterschied wieder wettzumachen. Multimediale Anwendungen wie die Übertragung von Videodaten, die Audioübertragung oder Videokonferenzen erfordern einen isochronen Datenfluß. Die Verzögerungen, die durch Fehlerkorrekturmechanismen auftreten, die mittels Übertragungswiederholung der verlorengegangenen oder zerstörten Datenpakete arbeiten, können zu so großen Verzögerungen führen, daß sie den Anforderungen bestimmter Applikationen, die die Übertragungszeit verbergen müssen, nicht mehr genügen. Hier kommt die Vorwärtsfehlerkorrektur (FEC, forward error correction) zum Einsatz. Die Vorwärtsfehlerkorrektur erlaubt (bis zu einer gewissen Grenze) die Wiederherstellung fehlerhafter bzw. verlorengegangener Daten ohne Übertragungswiederholung.

1.1 Was ist FEC?

ATM (asynchronous transfer mode) ist ein weithin anerkannter Übertragungsmodus für Breitband-Netzwerke. Die Grundeinheit beim ATM ist die Zelle. Sie besteht aus insgesamt 53 Bytes, 48 Datenbytes plus 5 Bytes an Kontroll- und Steuerungsinformation. Der Vorteil von ATM liegt in der Tatsache, daß unterschiedliche Anwendungen mit unterschiedlichen Anforderungen an die Qualität der Übertragung diesen Dienst nutzen können. Man spricht dabei von Quality of Service (QoS). Eine Anforderung an die QoS ist die Verläßlichkeit der Übertragung.

Man kann grundsätzlich unter drei Arten von Übertragungsfehlern unterscheiden: Bit-Fehler, die den Zelleninhalt verfälschen, Übertragungsfehler durch fehlgeleitete Zellen

und Zellverluste durch zu hohe Belastung des Netzes. Letztere sind am wahrscheinlichsten. Ist die Verläßlichkeit in einem Netzwerk nun kleiner, als es die Anwendungen erlauben, so ist es Sache des Senders sowie des Empfängers, diesen Mißstand zu beheben.

Nun gibt es zwei verschiedene Ansätze zur Korrektur von Übertragungsfehlern in Rechnernetzen: Zum einen die Erkennung und Wiederholung der fehlerhaft übertragenen Pakete und zum anderen die Korrektur von Bitfehlern mittels redundanter Informationen (Vorwärtsfehlerkorrektur).

Bei der Übertragungswiederholung (ARQ, automatic repeat request) sendet der Sender die auf Empfängerseite verlorengegangenen Daten erneut ab. Diese einfache Methode bringt allerdings einige Nachteile mit sich: Bei jeder einzelnen Nachricht müssen der Empfänger und der Sender Statusinformationen über die Korrektheit der versandten Nachrichten austauschen. Insbesondere bei größeren Entfernungen fällt die Zeit ins Gewicht, die eine erneute Übertragung benötigt. Somit ergeben sich Probleme bei Anwendungen, die keine Verzögerungen erlauben, wie etwa Video, Audio, Prozeßkontrolle, usw. Zu spät eingetroffene Daten sind gegebenfalls für solche Anwendungen wertlos [Bie93].

Die Alternative zu ARQ ist die Vorwärtsfehlerkorrektur. Hierbei wird vor der Übertragung beim Sender redundanter Code erzeugt, der mit den Nutzdaten versandt wird, und der es gestattet, möglicherweise verlorengegangene Originaldaten mit Hilfe der redundanten Daten direkt beim Empfänger wiederherzustellen, ohne daß eine erneute Übertragung notwendig wird. Die Menge der anfallenden redundanten Daten muß relativ klein sein im Vergleich zur Menge der Nutzdaten, damit das Verfahren effektiv bleibt. Im Vergleich zum Verfahren der Übertragungswiederholung liegt ein großer Vorteil der Vorwärtsfehlerkorrektur im Erhalt der Isochronität. Diese ist insbesondere bei multimedialen Datenströmen in DQDB- (dual queue dual bus) und ATM-Netzwerken von Bedeutung. Die langwierigen Übertragungswiederholungen entfallen, da Fehler sofort korrigiert werden können.

FEC-Verfahren, die sich auf die Bitfehler-Korrektur auf dem Übertragungsmedium beziehen, sind allerdings heutzutage nicht mehr sinnvoll, da die hauptsächliche Fehlerquelle verlorengegangene Pakete sind. Im weiteren wird also ein FEC-Verfahren behandelt, das sich mit der Restaurierung ganzer Pakete beschäftigt.

1.2 Wo wird FEC benutzt?

Die wohl bekannteste Anwendung der Vorwärtsfehlerkorrektur ist in CD-Spielern vorzufinden, wegen ihrer hohen Fehlerrate beim Lesen und den bei der Produktion verursachten Fehlern. Auch für Computerspeicher und neuerdings auch Video-Kodierung verwendet man FEC [CEG+95b].

Die Leistung in einem ATM-Netzwerk verringert sich mit einer steigenden Zellverlust- und Bitfehlerrate. Der Durchsatz sinkt zusätzlich bei höheren Paketgrößen. Wird ein Multicast-Service (interaktive Spiele, Konferenzen übers Internet) benutzt, so verringert sich der Durchsatz zusätzlich mit einer ansteigenden Zahl an Empfängern. Abhilfe in all diesen Fällen schafft ein FEC-Schema. FEC läßt sich überall, wo eine Übertragungswiederholung eine geringe Quality of Service zur Folge hat, gewinnbringend einsetzen. Da die einzelne Zelle in einem ATM-Netzwerk verhältnismäßig klein im Vergleich zu den

Dateneinheiten der Anwendungsschicht ist, sinkt die QoS einerseits mit steigender Verlustrate der Zellen, andererseits mit steigender Paketlänge. Die Verlustrate bei Paketen höherer Schichten steigt sogar ungefähr linear mit der Anzahl der Zellen, aus denen ein Paket zusammengesetzt ist.

2 Mathematische Grundlagen

In diesem Kapitel wird die grundlegende Idee zur Erzeugung redundanter Daten erklärt. Nach einer mathematischen Vorschrift werden beim Sender vor dem Senden neben den Nutzdaten zusätzliche Pakete erstellt, so daß im Falle eines Datenverlusts bei der Übertragung die originalen Nutzdaten wiederhergestellt werden können.
Beispiel 1: Es sollen zwei Pakete P_1 und P_2 verschickt werden, wobei 100 % Redundanz in Kauf genommen wird. Das bedeutet, daß zwei zusätzliche Pakete mit den ursprünglichen zwei Paketen verschickt werden. Dabei sollen die ursprünglichen zwei Pakete bei Verlust zweier beliebiger Pakete restauriert werden können (siehe Abbildung 33).

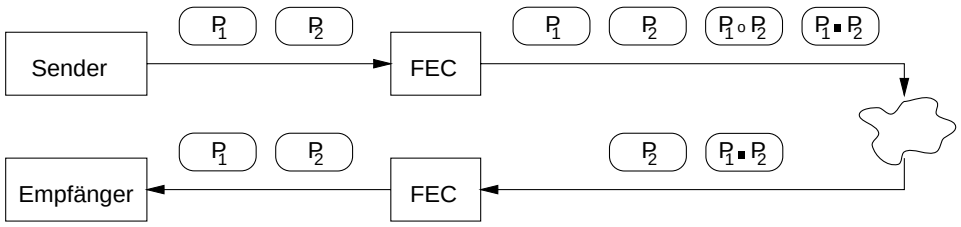

Abbildung 33. Prinzip der Forward Error Correction

Hierbei bezeichnen $\circ$ und $\bullet$ zwei Operationen zum Erzeugen der redundanten Information. Im Beispiel 1 sind die Pakete P_1 und $P_1 \circ P_2$ verlorengegangen.

Formal besteht ein Paket aus einer beliebig großen (endlichen) Bitfolge. Gegeben sind im allgemeinen n Originalpakete $p_1, \ldots, p_n$ und gesucht sind $n+m$ Pakete $q_1, \ldots, q_{n+m}$, so daß bei Verlust von höchstens m Paketen alle n Originalpakete restauriert werden können.

Beispiel 2: In diesem Beispiel seien zwei Pakete gegeben und es soll ein zusätzliches Paket erzeugt werden ($n = 2$ und $m = 1$). Hierbei wählt man nun etwa $q_1 = p_1, q_2 = p_2$ und $q_3 = p_1 \oplus p_2$, wobei $\oplus$ den binären Operator XOR darstellt. Geht eines der beiden Originalpakete verloren, zum Beispiel p_1, dann läßt es sich durch $p_1 = q_2 \oplus q_3$ wiederherstellen. Es ist darauf zu achten, daß man bei der Codierung einen Operator verwendet, der eine eindeutige Rekonstruktion zuläßt. Dazu eignet sich nur der XOR-Operator bzw. seine Negation, die Äquivalenz.

Man betrachte nun die Operationen $+$ und $*$ auf einem Körper mit vier Elementen (siehe Tabelle 1).

Sind zwei Originalpakete a und b vorhanden und man möchte zwei redundante Pakete c und d erzeugen, so lassen sich die beiden Operatoren $\circ$ und $\bullet$ wie folgt definieren:

+	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

*	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Tabelle 1. Operationen + und $*$ auf einen Körper mit vier Elementen

$$a \circ b = a + b$$
$$a \bullet b = a + 10 * b$$

und somit

$$c = a + b$$
$$d = a + 10 * b$$

In Tabelle 2 ist aufgezeigt, welche Berechnungen durchgeführt werden müssen, wenn zwei Datenpakete verlorengehen. Somit lassen sich bei einem Verlust zweier beliebiger

angekommene Pakete	Berechnungen zur Rekonstruktion	
	a	b
a, b	-	-
a, c	-	$a + c$
a, d	-	$11 * (a + d)$
b, c	$b + c$	-
b, d	$d + 10 * b$	-
c, d	$11 * c + 10 * d$	$10 * (c + d)$

Tabelle 2. Berechnungen zur Rekonstruktion

Pakete die Nutzpakete a und b rekonstruieren. Allgemein können Körper mit p^n (p eine Primzahl und $n > 0$) hergeleitet werden. Zur Herleitung wird auf [LBE93] verwiesen.

Um das Maß an benötigter Redundanz beurteilen zu können, muß man die gegebenen Anforderungen betrachten. So ist etwa das menschliche Ohr wesentlich sensibler gegenüber Tonschwankungen als das Auge gegenüber leichten Schwankungen bei der Anzeige von Bildern. Einzelne fehlende Bilder werden in einem Bildablauf normalerweise nur als wenig störend empfunden. Zusätzlich gibt es aber Informationen, die keinesfalls verloren gehen dürfen, wie zum Beispiel Steuerdaten, Formatdaten oder die Farbtafel. So läßt sich nun eine mögliche Einteilung in verschiedene Prioritätsklassen finden:

Priorität 1: Teile, die auf keinen Fall verloren gehen dürfen. Dabei werden zu je zwei Paketen a und b zwei FEC-Pakete $a \circ b$ und $a \bullet b$ erzeugt und übertragen.

Priorität 2: Teile, deren Verlust der Qualität deutlich abträglich ist. Es wird zu je zwei Datenpaketen ein FEC-Paket erzeugt.

Priorität 3: Teile, deren Verlust der Qualität wenig schadet und die ohne Redundanz übertragen werden.

Es ist also eine Methode nötig, mit der sich die Redundanz je nach Anforderung beliebig einstellen läßt. In größeren Körpern ist die Erstellung unabhängiger Redundanzpakete allerdings nicht mehr ganz trivial.

3 Die Methode der FEC

Kapitel 3 stellt eine Wiederherstellungsmethode beispielsweise für Zellverlust vor. Wegen der hohen Qualitätsansprüche, die an heutige Übertragungssysteme (z. Bsp. B-ISDN) gestellt werden, ist eine kleine Zellverlustrate äußerst wichtig. Hier wird eine Methode der Vorwärtsfehlerkorrektur vorgestellt, die auf virtuelle Pfade (VP, virtual path) anstatt auf virtuelle Kanäle (VC, virtual channel) angewendet wird, genannt CREG-VP (cell regeneration on virtual paths). Diese Methode bringt mehrere Vorteile mit sich, unter anderem:

- Sie kann auf Zellverluste in fortlaufenden Zellen angewendet werden.
- Die Kodierungs- und Dekodierungszeiten sind relativ gering.
- Die Originalzellen bleiben bei der Kodierung unverändert.
- Zellverluste werden nicht in jedem Durchgangsknoten behandelt, sondern nur in Endknoten der virtuellen Pfade.
- Bit-Fehler werden nicht hier, sondern in einer höheren Schicht behandelt.

Zuerst werden die zwei Hauptursachen für Zellverluste betrachtet.

3.1 Mögliche Fehlerursachen

Grundsätzlich gibt es zwei Ursachen für Zellverlust in ATM-Netzwerken: zum einen Zellen, die den Empfänger nicht erreichen, weil sie fehlgeleitet wurden, und zum anderen Pufferüberläufe beim Multiplexen in einem Durchgangsknoten.

Die Weiterleitung von Zellen in ATM-Netzwerken geschieht anhand des fünf Byte langen Header-Feldes, so daß Bitfehler im Header-Feld Zellverluste verursachen. Das fünfte Byte des Header-Feldes, das Header-Prüfsummen-Feld (HEC, header error control), enthält eine Prüfsumme, die über den gesamten ATM-Header berechnet wird. Da Bitfehler bei der Übertragung relativ selten sind und einzelne Bits anhand eines CRC-Schemas (cyclic redundancy check code) mit Hilfe des Header-Prüfsummen-Feldes korrigiert werden können, ist die Wahrscheinlichkeit eines Zellverlustes durch einen Header-Fehler sehr

klein. Dagegen bringen Fehler beim Zellentwurf, auch wenn sie nur sehr gering sind, eine Folge von fehlgeleiteten Zellen mit sich. Die Standard-Zellentwurfsmethode erkennt einen Entwurfsfehler nach sieben Zellen, und sie benötigt weitere sechs Zellen, bis der Fehler behoben ist [OK91].

Wenn ein Puffer-Überlauf in einem Knoten entsteht, so ist damit zu rechnen, daß weitere Überläufe folgen können, da der Puffer fast voll ist. Die durchschnittliche Zellverlustrate kann durch das sogenannte Gilbert-Modell beschrieben werden (Abbildung 34) und beträgt $P_D = \frac{P}{P+p}$.

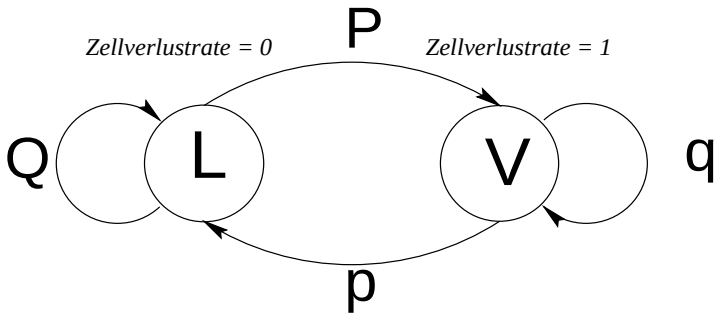

Abbildung 34. Gilbert-Modell

In dem Modell bezeichnen L den Zustand, in dem der Puffer noch nicht voll ist, und V den Zustand, bei dem Zellen wegen eines überfüllten Puffers verlorengehen. P, Q, p und q bezeichnen die Übergangswahrscheinlichkeiten. Insbesondere bezeichnet dabei q die Wahrscheinlichkeit, daß einem Zellverlust ein erneuter Zellverlust folgt. Versuche ergaben bei geringer Ankunftsrate einen Wert von $\leq 0,1$ und bei starkem Datenverkehr einen Wert von bis zu $0,3$ für q.

Die Wahrscheinlichkeit der Fehlleitung einer Zelle ist so gering, daß sie gegenüber dem Pufferüberlauf vernachlässigt werden kann.

3.2 Die Methode der Wiederherstellung verlorengegangener Zellen

Zwei mögliche Ansätze der Vorwärtsfehlerkorrektur in ATM-Netzwerken lassen sich unterscheiden, die end-to-end-Methode und die node-to-node-Methode.

Die end-to-end-Methode basiert auf der Behandlung der Datenpakete innerhalb der virtuellen Kanäle. Anhand eines Sequenznummern-Feldes lassen sich in den ATM-Adaptionsschichten (AAL, ATM adaptation layer) leicht Zellverluste aufspüren. Bei dieser Methode ist der Kodierungs- und Dekodierungsaufwand sehr groß.

Bei der node-to-node-Methode muß ein Zellverlust an Endknoten des virtuellen Pfades festgestellt werden. Die Übertragung der Verlustinformationen ist zur Bestimmung der Position der verlorengegangenen Zellen notwendig. Diese Methode wird im weiteren näher beschrieben.

Die Struktur Die CREG-VP-Methode wird bei jedem Endknoten angewendet. Dabei werden an den Absenderknoten Zellen mit gleicher Pfadidentifikationsnummer (VPI, vir-

tual path identifier) gesammelt und nach der CREG-VP-Methode kodiert. Die zusätzlichen Zellen, die für die Erkennung und Restauration verlorengegangener Zellen notwendig sind, werden erzeugt und mit den anderen Zellen, die die Nutzdaten erhalten, zusammen verschickt. Am Empfängerknoten werden die verlorengegangenen Zellen wiederhergestellt und die zusätzlichen Zellen wieder entfernt.

Abbildung 35 zeigt den Aufbau einer Zellkodierungsmatrix. Alle Zellen mit der selben Pfadidentifikationsnummer wurden in einer Matrix gruppiert und nach der CREG-VP-Methode kodiert. Diese Matrix enthält $n-1$ Spalten und $m-1$ Zeilen an Datenzellen. Am Ende jeder Zeile steht eine CLD-Zelle (cell loss detection cell), mit deren Hilfe verlorengegangene Zellen entdeckt werden. Am Fuße jeder Spalte steht eine Paritätszelle. Sie dient zur Regenerierung von Zellen. Eine CLD-Paritätszelle zur Wiederherstellung verlorengegangener CLD-Zellen befindet sich rechts unten in der Zellkodierungsmatrix.

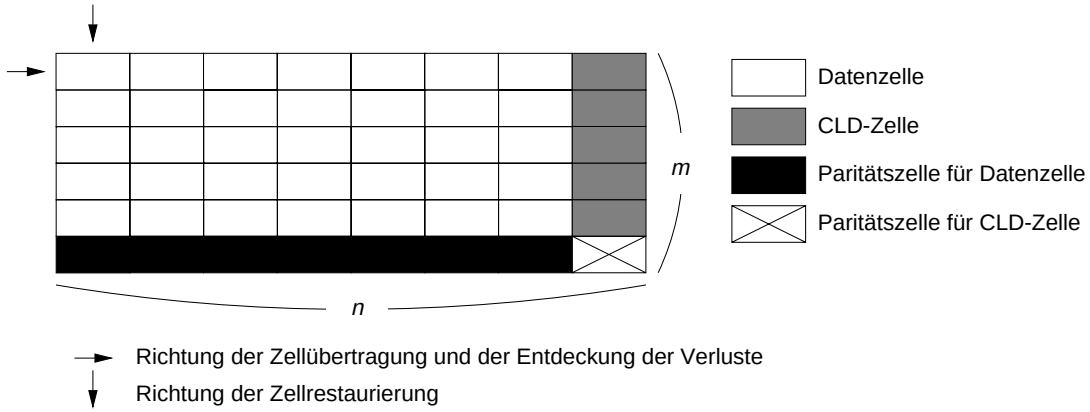

Abbildung 35. Zellkodierungsmatrix

Alle Zellen, auch die CLD- und Paritätszellen, haben den selben VPI und die selbe Länge wie die Datenzellen. Somit können Datenzellen, CLD-Zellen und Paritätszellen an den Durchgangsknoten auf die gleiche Weise behandelt und weitergeleitet werden wie nicht-kodierte Zellen.

Nach der CREG-VP-Methode kann eine Zelle pro Spalte wiederhergestellt werden, also insgesamt n aufeinanderfolgende Zellen.

Die Methode Zuerst wird eine eventuell fehlende CLD-Zelle mit Hilfe der CLD-Paritätszelle wiederhergestellt. Dies geschieht nach einem einfachen Verfahren, beispielsweise Addition modulo zwei. Anhand der nun vollständigen CLD-Zellen kann der Verlust von Datenzellen erkannt werden. An den entsprechenden Stellen werden Dummy-Zellen eingefügt, die anschließend durch die restaurierten Datenzellen ersetzt werden.

Im weiteren wird die Endeckung und Restaurierung verlorengegangener Datenzellen beschrieben. Die Restaurierung von CLD-Zellen verläuft auf ähnliche Weise und ist in [OK91] beschrieben.

In Abbildung 36 ist eine CLD-Zelle dargestellt. Aus jeder Datenzelle einer Zeile in der Matrix werden das VCI-Feld (VCI, virtual channel identifier) und die ersten sechs Bit aus

dem Datenfeld, zusammen 22 Bit, in das Datenfeld der CLD-Zelle kopiert. Sie bilden das CRP-Feld (cell recognition pattern). Das SN-CLD-Feld (sequence number) ist acht Bit lang und dient der Identifikation der CLD-Zellen als solche und der Erkennung fehlender CLD-Zellen. Bitfehler in einer CLD-Zelle können anhand des CRC-Feldes korrigiert werden. Der Header einer CLD-Zelle ist gleich dem Header einer Datenzelle.

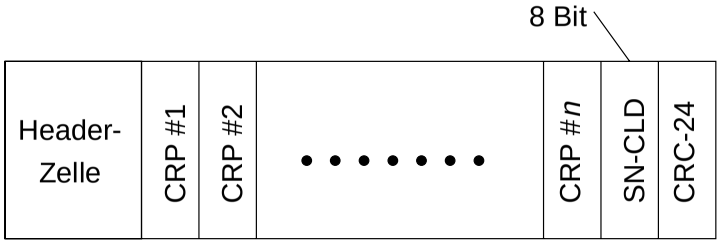

Abbildung 36. CLD-Zelle

CLD-Zellen werden immer gleich im Anschluß an die Datenzellen berechnet und verschickt, so daß die Erkennung verlorengegangener Datenzellen Zeile für Zeile durchgeführt wird. Die CRPs sind zum Zwecke der Eindeutigkeit immer so berechnet, daß mit großer Wahrscheinlichkeit keine zwei Zellen in einer Zeile das gleich CRP haben. Ist eine Zeile beim Empfänger angekommen, so werden die CRP in der CLD-Zelle mit den CRPs in den Datenzellen verglichen und somit fehlende Zellen sofort lokalisiert. Verlorene Zellen werden anschließend mit Dummy-Zellen (sie enthalten nur Nullen) aufgefüllt, so daß die weiteren Vergleiche wieder stimmen. Stimmen alle CRP-Paare überein, so kann man annehmen, daß keine Zellverluste aufgetreten sind. Im Zellrestaurierungsprozeß werden die Dummy-Zellen wieder durch regenerierte Zellen ersetzt.

Verlorengegangene Zellen werden mit Hilfe der Paritätszellen wiederhergestellt. Abbildung 37 verdeutlicht die Generierung der Paritätszellen. Dabei werden alle Zellen in einer Reihe modulo zwei addiert. Jedoch werden nicht alle Bits einer Zelle benötigt, so zum Beispiel die VCIs. An ihrer Stelle werden andere Informationen gespeichert, etwa eine Sequenznummer (SN) der Paritätszellen, anhand derer ein Verlust einer Paritätszelle erkannt wird.

Zuerst wird die Anzahl der Dummy-Zellen in einer Spalte festgestellt. Ist sie größer als eins, so wird der Restaurierungsprozeß gestoppt. Ansonsten erfolgt die Restaurierung Spalte für Spalte. Alle Zellen einer Spalte einschließlich der Paritätszelle werden dabei wieder modulo zwei addiert. Ist das Ergebnis gleich Null, so gibt es keine fehlenden Zellen. Ansonsten ist das Ergebnis gleich dem Wert der fehlenden Zelle. Durch ihn wird die Dummy-Zelle ersetzt. Sind nur Paritätszellen verlorengegangen, so ist keine Restaurierung nötig.

Andere Kodierungsfunktionen außer der einfachen Paritätskontrolle sind möglich. Durch sie können gegebenenfalls mehrere fehlende Zellen pro Spalte wiederhergestellt werden.

Die CREG-VP-Methode wurde auf ihre Leistung untersucht. Durch diese Methode wurde die Zell-Verlustrate in ATM-Netzwerken deutlich verringert. Das Ergebnis der Leistungsanalyse sah folgendermaßen aus:

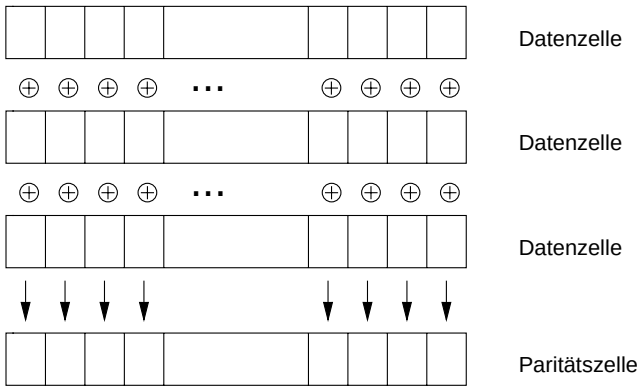

Abbildung 37. Generierung der Paritätszelle

Bei großen Matrixgrößen (zum Beispiel $n = 16$ und $m = 20$) wurde die Zell-Verlustrate um 88 % verbessert [OK91].

4 Die FEC-Service Specific Convergence Sublayer (SSCS)

In diesem Kapitel wird kurz eine Spezifikation einer FEC-Service Specific Convergence Sublayer (FEC-SSCS) für die AAL-Typ 5 beschrieben [CEG+95a].

4.1 Anforderungen

Um eine sichere Datenübertragung zu erreichen, werden u. a. folgende Anforderungen gestellt:

- Auf jeden Fall ist die Kompatibilität mit existierenden AAL-Typ 5 Schichten zu gewährleisten.
- Die Parameter des FEC-Algorithmus sollen zu jedem Zeitpunkt einstellbar sein. Damit soll ermöglicht werden, die Redundanz jeweils den gestellten Anforderungen und Umgebungen anzupassen.
- Die Größe der Übertragungspakete soll variabel sein.
- Tritt kein Fehler auf, so soll der Aufwand, der durch die FEC-Methode dazukommt, minimal sein.
- Die FEC-Methode soll drei Operationsmodi anbieten
  - Fehlerbehandlung für Zellverlust und Bitfehler,
  - Fehlerbehandlung nur für Zellverlust,
  - Fehlerbehandlung nur für Bitfehler.

4.2 Die drei verschiedenen Fehlerkorrektur-Modi

Wird von der Empfänger-CPCS (common part convergence sublayer) ein Fehler entdeckt, so versucht die FEC-SSCS-Einheit den Fehler zu beheben. Abbildung 38 verdeutlicht die Struktur des gegebenen Protokolls.

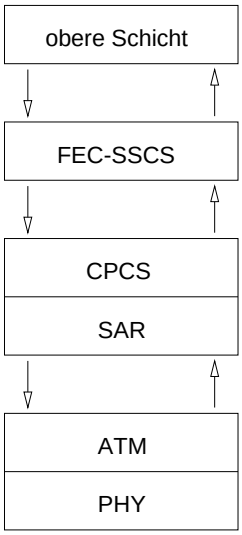

Abbildung 38. Protokoll-Referenz-Modell

Darin werden die drei schon oben erwähnten Modi angeboten:

SEAL (symbol error and loss correction mode): Im SEAL-Modus werden sowohl Bitfehler behoben als auch verlorengegangene Zellen wiederhergestellt.

SLC (symbol loss correction mode): Hierbei werden nur verlorengegangene Zellen wiederhergestellt. Bitfehler werden nicht behoben.

SEC (symbol error correction mode): Der SEC-Modus versucht, Bitfehler in den Datenpaketen zu beheben. Fehlende Zellen werden hier nicht berücksichtigt.

Eine genauere Beschreibung dieser drei Modi zusammen mit einigen Beispielalgorithmen ist in [CEG+95a] gegeben. Zusätzlich werden verschiedene weitere Methoden und Funktionen der FEC-SSCS vorgestellt.

5 Experimentelle Leistungsanalyse der FEC

Im folgenden wird eine Leistungsanalyse der Vorwärtsfehlerkorrektur in einer ATM-Umgebung betrachtet. Aufgezeigt wird die Verlustrate eines gepufferten Multiplexers bei drei unterschiedlichen Anforderungen [Bie93].

Bei Verwendung der Vorwärtsfehlerkorrektur-Methode stehen sich zwei gegenläufige Effekte gegenüber:

– die von der FEC erzeugte redundante Information wird benötigt, um verlorengegangene Zellen wiederherzustellen,

– diese zusätzliche Information erhöht die Last bei der Datenübertragung und somit auch die Zellverlustrate.

Die FEC ist nur effektiv, wenn der erste Effekt überwiegt.

Der Erfolg der FEC hängt in hohem Maße von der Art der auftretenden Fehler ab. Seltene, einzeln auftretende Zellverluste lassen sich viel eher beseitigen als Verluste mehrerer aufeinanderfolgender Zellen. Mehrere aufeinanderfolgende Zellverluste entstehen, wenn plötzlich eine große Belastung des Netzes eintritt. Damit die FEC effektiv arbeitet, ist es wichtig, daß auch solche Fehler größtenteils behoben werden können.

5.1 Vorbemerkungen

In der vorgestellten Versuchsanordnung wird ein Reed-Solomon-Korrektur-Code verwendet. Dabei werden zu je k Datenzellen h redundante Zellen erzeugt. Es werden $k + h$ Zellen über das Netz übertragen, wobei der Empfänger die Datenzellen nur wiederherstellen kann, solange nicht mehr als h Zellen verlorengegangen sind.

Diese Funktion kann auf einem einzigen Chip implementiert werden und erreicht dank seiner Einfachheit (es können nur verlorengegangene Zellen wiederhergestellt werden, keine Bitfehlerkorrektur) einen Durchsatz von 400 Mb/s oder 1Gb/s. Da keine Kodierung der Datenzellen stattfindet, entstehen keine Verzögerungen beim Kodieren und Dekodieren, falls alle k Datenzellen korrekt übertragen wurden. Bei $k = 50$ und $h = 10$ ergibt sich bei einer Arbeitsgeschwindigkeit von 400 MHz eine Verzögerungszeit von $d = 8,005\mu s$ für das Kodieren. Dazu kommt eine Wartezeit w, bis mindestens k Zellen übertragen wurden, so daß sich die Gesamtzeit für das Kodieren und Dekodieren auf $d + (w + d)$ beläuft, was im Vergleich zur Übertragungszeit vernachlässigt werden kann.

Versteht man unter der Last λ die Gesamtlast, die durch alle Datenquellen verursacht wird, so ergibt sich für die *effektive* Last λ_{eff} die Last, wenn alle Datenquellen die FEC verwenden, $\lambda_{\text{eff}} = \lambda(1 + h/k)$ mit Werten zwischen 0 und 1 für λ und λ_{eff}, wobei 1 eine Auslastung des Systems zu 100 % meint.

5.2 Versuchsanordnung

Das ATM-Netzwerk wird als Multiplexer mit N Eingängen und einem Puffer für B Zellen realisiert. Zwei verschiedene Arten von Anwendungen machen Eingaben: 1) *Burst Sources*, große Datenmengen, die in Schüben auftreten, und 2) *Video Sources*, wie sie bei Videokonferenzen oder Filmen vorkommen.

Drei verschiedene Szenarien wurden getestet:

V32 32 Video Sources,

B32 32 Burst Sources,

VB24-8 24 Video Sources und 8 Burst Sources,

jeweils mit $k + h = 50$ und einer Puffergröße von 100 Zellen.

5.3 Vergleich verschiedener Anforderungen

Die Blockverlustrate $\mathrm{P}_{\mathrm{Data}}(\lambda) := \mathrm{P}(\textit{mind. eine Zelle wurde verloren} | \textit{Last} = \lambda)$ und das Auftreten der Verluste bestimmen die Effektivität der FEC. Da die Last nach der FEC-Kodierung von λ auf λ_{eff} steigt, erhöht sich damit auch $\mathrm{P}_{\mathrm{Data}}(\lambda)$ auf $\mathrm{P}_{\mathrm{Data}}(\lambda_{\mathrm{eff}})$. Sei CL eine diskrete Zufallsvariable, die den Zellverlust angibt, so läßt sich die Zellverlustrate beschreiben durch $F_{CL} := \mathrm{P}(CL \leq x)$, $CL \in [0, 100]$. $F_{CL} = p$ bedeutet, daß mit der Wahrscheinlichkeit p maximal x Zellen verlorengingen.

Nun wird die Blockverlustrate eines Multiplexers innerhalb dieser drei Versuchsanordnungen betrachtet. Bei der Anordnung B32 beginnt der Blockverlust im Multiplexer schon bei einer Last von $\lambda = 0,3$ und steigt gleichmäßig an. Bei V32 treten Verluste erst bei $\lambda \geq 0,9$ und wachsen stufenweise. Die Blockverlustrate der Versuchsanordnung V24-8 liegt etwa in der Mitte zwischen den zwei vorhergehenden Werten, wobei hier der Verlust der Burst Sources und der Video Sources ungefähr gleich ist. Daraus läßt sich rückschließen, daß die Mischung der beiden Sources beim Verlust von Datenblöcken eine entscheidende Rolle spielt.

Die Burst Sources treten schubweise auf. Die Video Sources dagegen schicken ihre Daten in gleichmäßigen Abständen an den Multiplexer, wobei das Intervall nicht kleiner als $\frac{1}{24}$ Sekunde ist, also viel seltener als die Burst Sources. Das Ergebnis ist, daß der Multiplexer für die Anordnung B32 mit einer mehr als doppelt so hohen Wahrscheinlichkeit zu über 90 % gefüllt ist als bei VB24-8. Für V32 dagegen ist der Mutliplexer nie mehr als 10 % gefüllt.

Damit die Vorwärtsfehlerkorrektur effizient arbeitet, ist es nötig, daß die meisten zerstörten Blöcke wiederhergestellt werden können, was nichts anderes bedeutet, als daß nur so viele Zellen verlorengehen dürfen, daß sie sich per FEC rekonstruieren lassen. Betrachtet man nun F_{CL}, so zeigt sich, daß im Durchschnitt mehr Zellen bei der Versuchsanordnung B32 verlorengehen als bei der Anordnung V32. Die Werte zwischen für VB24-8 liegen wieder zwischen denen für B32 und denen für V32. Wieder läßt sich folgern, daß der Wert F_{CL} von der Art und der Zusammensetzung der übertragenen Daten abhängt, je „glatter“ eine Datenquelle ist, desto größer ist der Wert von F_{CL}.
Tabelle 3 gibt die Verluste für F_{CL} =90,0 , 99,0 und 99,9 % mit $\lambda = 0,9$ wieder. Beispielsweise verlieren im Fall von V32 99 % aller defekten Blöcke nur weniger als 6 % ihrer Zellen. Die Tabelle zeigt, daß für eine Video Source bei einer bestimmten Menge an Redundanz wesentlich mehr Blöcke rekonstruiert werden können als für eine Burst Source.

Um die Leistung dieses FEC-Schemas evaluieren zu können, definiert man nun drei Maßzahlen:

Anordnung	90,0 %	99,0 %	99,9 %
V32	4	6	6
B32	40	72	86
VB24-8, Video	8	18	26
VB24-8, Burst	14	34	54

Tabelle 3. Zellverluste in Prozent für die verschiedenen Anordnungen

- Der *Gewinn* G wird definiert als $G := \log_{10}(\mathrm{P}_{\mathrm{Data}}(\lambda)/\mathrm{P}_{\mathrm{FEC}}(\lambda_{\mathrm{eff}}))$ und beschreibt die Blockverlustrate.
- Der *Verlust* D mißt für eine Nicht-FEC-Quelle die relative Zunahme der Blockverlustrate, $D := \log_{10}(\mathrm{P}_{\mathrm{Data}}(\lambda_{\mathrm{eff}})/\mathrm{P}_{\mathrm{Data}}(\lambda))$.
- Der *Netto-Gewinn* AG ist eine Kombination aus Gewinn und Verlust und zeigt die Effektivität eines FEC-Schemas auf. Er wird definiert als $AG :=$ *(Anzahl der FEC-Quellen)* $*$ *Gewinn - (Anzahl der Nicht-FEC-Quellen)* $*$ *Verlust.*

Hierbei ist $\mathrm{P}_{\mathrm{FEC}}(\lambda_{\mathrm{eff}})$ die Blockverlustrate nach Anwendung der Vorwärtsfehlerkorrektur.

Wie man aus Tablelle 3 erkennen kann, gehen für die Versuchsanordnung V32 bei den meisten zerstörten Blöcken nur wenige Zellen verloren. Messungen ergaben, daß für eine einzelne Datenquelle der Gewinn proportional mit der Erzeugung redundanten Codes wächst. Steigt die Anzahl der Datenquellen, so geht der Gewinn gegen Null oder wird sogar negativ. Die Abhängigkeit der Blockverlustrate von der Zunahme der Last schlägt sich in einer hohen Verlustrate nieder. Ein stets negativer Netto-Gewinn verdeutlicht, daß FEC für V32 nicht effektiv ist.

Für die Anordnung B32 ist die FEC schon bei einer kleineren Last noch uneffektiver als für V32. Hier wäre demnach ein ARQ-Verfahren für die Fehlerkorrektur angebracht, das nur die defekten Nachrichten wiederherstellt.

Allein für die Anordnung VB24-8 und eine Last $\lambda = 0,7$ ist der Netto-Gewinn immer positiv, was ein Zeichen dafür ist, daß ein FEC-Schema sehr effektiv angewendet werden kann.

Zusammenfassend läßt sich sagen, daß die Effektivität der Vorwärtsfehlerkorrektur sowohl von der Last, der Übertragungsart der Daten, als auch der Menge der erzeugten Redundanz abhängt. FEC eignet sich vor allem zum Einsatz in Systemen, die die Übertragungszeiten verbergen müssen, aber einige verlorengegangene Zellen tolerieren können, etwa bei Videoübertragungen. Das Ergebnis der oben beschriebenen Untersuchung ergab, daß FEC insbesondere bei heterogenen Datenmengen effektiv arbeitet. Eine graphische Darstellung der Versuchsergebnisse findet man in [Bie93].

Signalisierung in ATM-Netzen

Markus Bauer

Kurzfassung

ATM stellt einen verbindungsorientierten Hochgeschwindigkeitskommunikationsdienst bereit. Dabei werden kleine Datenpakete über ein Netz aus Vermittlungsstellen transportiert. Durch das Einrichten von virtuellen Pfaden und Kanälen und entsprechendes Kennzeichnen der Pakete wird dazu eine virtuelle Verbindung aufgebaut. Das Aufnehmen und Beenden solcher Verbindungen ist Aufgabe der Signalisierung.

Aufgrund der Struktur von ATM-Netzen existieren zwei Arten von Signalisierungsprotokollen — die Protokolle der UNI-Schnittstelle zur Signalisierung zwischen Netzteilnehmer und dem Netz, sowie die der NNI-Schnittstelle zwischen den Vermittlungsstellen innerhalb des Netzes. Der folgende Text behandelt die grundsätzlichen Merkmale zweier UNI-Protokolle — DSS2 und CMAP — und geht dann noch kurz auf eine Realisierung der NNI-Schnittstelle ein.

1 Einführung

1.1 Logischer Aufbau von ATM-Netzen

ATM (Asynchronous Transfer Mode) stellt einen modernen verbindungsorientierten Hochgeschwindigkeitskommunikationsdienst zur Verfügung. Nutzdaten werden dabei in kleinen Zellen (Paketen) von 53 Bytes über ein Netz versandt. Ein solches ATM-Netz besteht aus verschiedenen logischen Einheiten (vgl. Abb. 39):

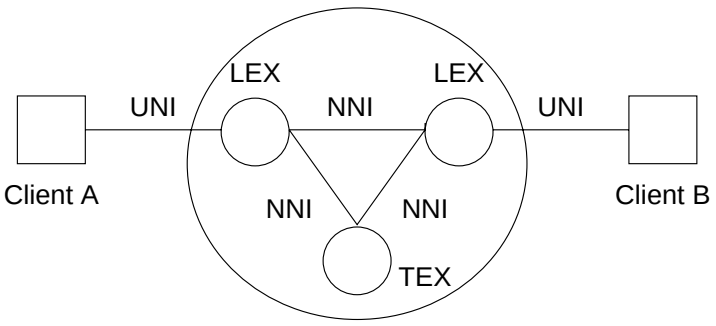

Abbildung 39. Logischer Aufbau eines ATM-Netzes.

- den Netzteilnehmern *(Clients)*; dies können einzelne Endgeräte wie beispielsweise intelligente Bildtelefone oder auch komplette LANs sein,
- den Teilnehmervermittlungsstellen (äußere Netzknoten, *Local EXchange Nodes, LEX*) zur Anbindung der Teilnehmer ans Netz,

– den Transitvermittlungsstellen (innere Netzknoten, *Transit EXchange Nodes, TEX*); diese ermöglichen Verbindungen zwischen einzelnen Teilnehmervermittlungsstellen. An Transitvermittlungsstellen sind nie Endeinrichtungen geschaltet.

Die Verbindungsstrecken zwischen den einzelnen Einheiten heißen im folgenden Übertragungsteilstrecken.

1.2 Die Schnittstellen UNI/NNI

Zwischen den einzelnen Netzbestandteilen sind Schnittstellen definiert, die es ermöglichen, Verbindungen aufzubauen oder zu beenden. In ATM-Netzen unterscheidet man zwei Arten von Schnittstellen, die beide getrennt vom ITU-T[2] festgelegt wurden.

Wie aus Abb. 39 ersichtlich, befindet sich zwischen Teilnehmer und der zugehörigen Vermittlungsstelle das *User Network Interface (UNI)*, während zwischen den eigentlichen Netzknoten (Teilnehmer- und Transitvermittlungsstellen) das *Network Node Interface (NNI)* eingesetzt wird.

1.3 ATM-Vermittlungstechnik — virtuelle Pfade, virtuelle Kanäle

Auf jedem Übertragungsabschnitt im Netz werden permanent Zellen übertragen — liegen keine zu übertragenden Informationen vor, so werden einfach speziell ausgezeichnete Leerzellen übertragen.

ATM ermöglicht durch ein Multiplexing-Verfahren das Übertragen mehrerer Nutzdatenströme mit jeweils unterschiedlichen Geschwindigkeiten. Dabei werden aus den Nutzdatenströmen unabhängig voneinander ATM-Zellen gebildet und auf die gemeinsame Übertragungsstrecke gesandt. Den Zellen der einzelnen Ströme wird kein fester Platz im Zellenstrom zugeordnet, sondern sie belegen je nach geforderter Bitrate des zugehörigen Teilstromes bei Bedarf freie Zeitschlitze. Dies illustriert Abb. 40. Ein solches Verfahren bezeichnet man als *asynchron*.

Diese Fähigkeit von ATM spiegelt sich im Konzept der virtuellen Pfade und Kanäle wieder. Die verfügbare Kapazität einer physikalischen Übertragungsstrecke wird zunächst in sogenannte virtuelle Pfade *(Virtual Paths, VP)* aufgeteilt. Zudem kann jeder Pfad weiter in virtuelle Kanäle *(Virtual Channels, VC)* zerlegt werden. Jeder virtuelle Kanal kann dann einen Bitstrom repräsentieren (vgl. Abb. 41).

Da, wie oben erläutert, ATM jedoch grundsätzlich paketorientiert (zellenorientiert) arbeitet, bestehen diese Pfade und Kanäle nicht wirklich (deshalb nennt man sie virtuell), sondern sie entstehen auf einer abstrakten Ebene durch Kennzeichnen der Zellen mit einer Pfad- und einer Kanalnummer (*Virtual Path Identifier, VPI* sowie *Virtual Channel Identifier, VCI*).

Jede ATM-Zelle besteht aus zwei Komponenten: dem 5 Bytes großen Kopf *(Header)* und der eigentlichen Nutzinformation *(Body, Payload)* von 48 Bytes. Im Header sind

[2] ITU steht für International Telecommunication Union; ITU-T bezeichnet deren Standardisierungsausschuß. (Neben einer ausführlichen Einführung in die ATM-Technik bietet [MS94] eine Übersicht über die am Entwurf von ATM-Standards beiteiligten Organisationen.)

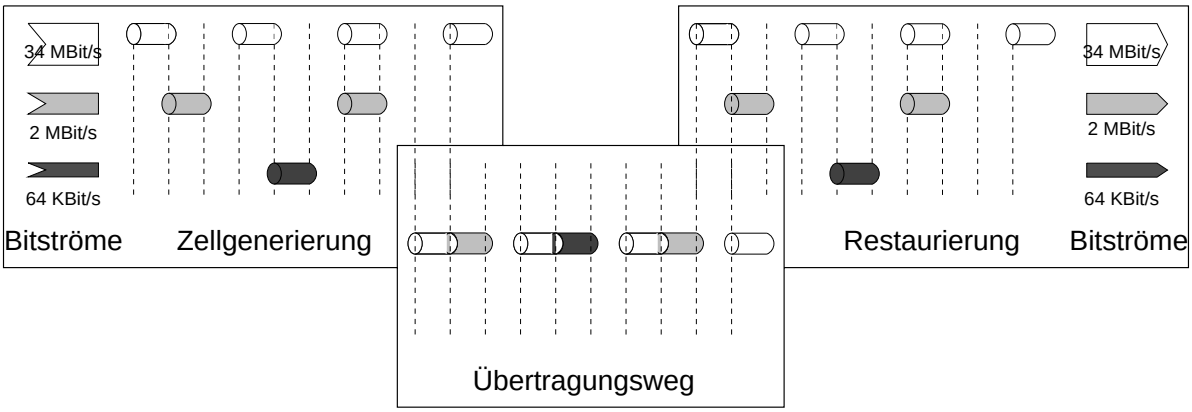

Abbildung 40. Asynchrones Zeitmultiplexing.

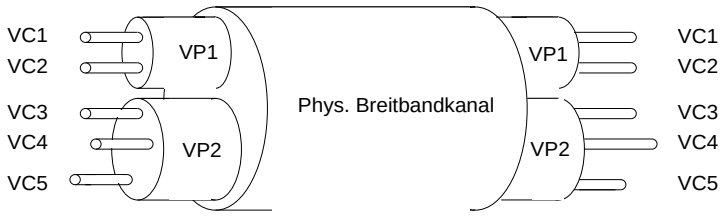

Abbildung 41. Virtuelle Pfade und virtuelle Kanäle.

zunächst die Pfad- und Kanalinformationen VPI und VCI gespeichert, zudem noch weitere Angaben. Das Aussehen des Zellkopfes hängt von der Art der Schnittstelle ab, über die die Zelle versandt wird (vgl. Abb. 42) — UNI-Zellen und NNI-Zellen unterscheiden sich leicht.

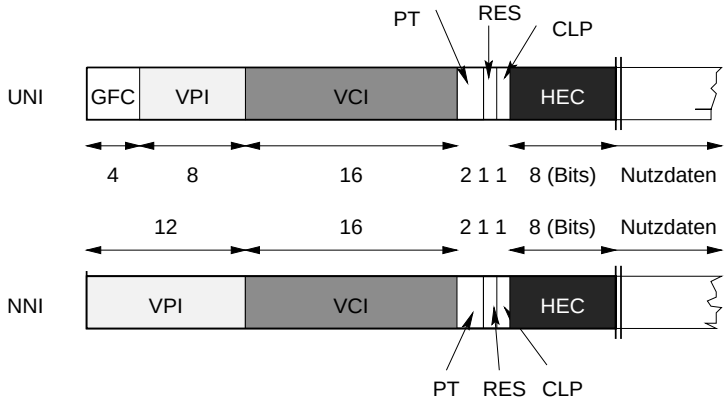

Abbildung 42. UNI- und NNI-Zellheader.

Die Bedeutung der einzelnen Elemente des Kopfes kann Tabelle 4 entnommen werden.

Die Knoten im Netz können nun mit Hilfe der VPI- und VCI-Kennung jede Zelle einem logischen Nutzdatenstrom zwischen zwei (oder mehr) Teilnehmern zuordnen und auf geeignete Teilübertragungsstrecken leiten. Auf diese Weise entsteht eine scheinbar feste Verbindung zwischen den Teilnehmern. Man spricht hierbei von der Vermittlung *(Routing)* der Zellen im Netz.

Feld	Name	Bedeutung
GFC	Generic Flow Control	Traffic Shaping, Netzlastparameter
VPI	Virtual Path Identifier	s.o.
VCI	Virtual Channel Identifier	s.o.
PT	Payload Type	Art der Nutzdaten: Benutzerdaten oder Signalisierungsinformationen
RES	Reserved	—
CLP	Cell Loss Priority	Markierung für verlustempfindliche Zellen. Diese werden dann bei Leistungsengpässen bevorzugt übertragen.
HEC	Header Error Control	Bitfehlerkontrolle, Feststellen fehlerhaft übertragener Zellen, so daß diese verworfen werden können.

Tabelle 4. Felder im ATM-Zellheader.

Grundsätzlich sind zwei Vermittlungstechniken von großer praktischer Bedeutung:

Virtual-Path-Vermittlung: Beim Aufbau einer Verbindung zwischen zwei Netzteilnehmern A und B wird hierbei ein virtueller Pfad von A zu dessen Vermittlungsstelle eingerichtet. Ebenso werden entlang des Übertragungsweges durch das Netz für jede Einzelübertragungsstrecke von Knoten zu Knoten weitere virtuelle Pfade belegt, bis die Vermittlungsstelle von B und schließlich B selbst erreicht werden. A markiert alle zu dieser Verbindung gehörenden Zellen beim Senden mit dem VPI-Wert des ersten Abschnittes. In der Teilnehmervermittlungsstelle wird mit Hilfe dieses Wertes und einer gespeicherten Routingtabelle durch ein sogenanntes Koppelnetz die passende Folgeübertragungsstrecke bzw. der passende virtuelle Pfad zum nächsten Netzknoten ausgewählt. Die Teilnehmervermittlungsstelle überschreibt dann das VPI-Feld in den Zellköpfen mit dem Wert dieses Pfades und schickt die Zelle auf den Weg. Im nächsten Knoten wiederholt sich dieser Vorgang, ebenso in allen weiteren Knoten. Wenn die Zelle schließlich in B anlangt, ist ihr VPI-Wert mehrfach umgesetzt worden (vgl. Abb. 43).

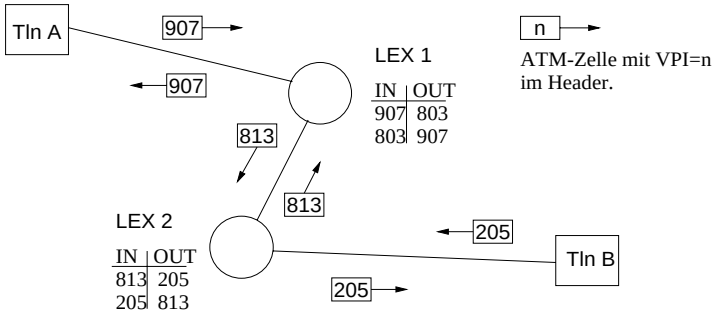

Abbildung 43. VP-Vermittlung

Die VCI-Information der Zellen dagegen bleibt bei der Virtual-Path-Vermittlung unberührt. Dies hat den Vorteil, daß diese zur Unterscheidung von Einzelströmen einer Verbindung genützt werden kann. So kann man z.B. Sprache und Bildsignal in getrennten virtuellen Kanälen übertragen oder bei Mehrpunktverbindungen die Quelle identifizieren. Abb. 44 zeigt dies an einem Beispiel mit drei Teilnehmern.

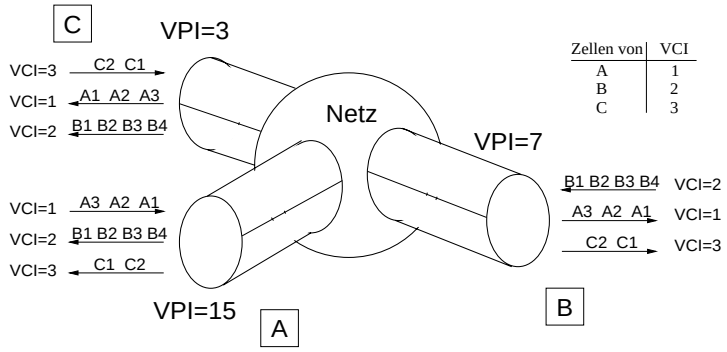

Abbildung 44. Mehrpunktverbindung, Nutzung des VCI-Feldes zur Quellenidentifikation.

VC- und VP-Vermittlung: Bei dieser Vermittlungstechnik werden in den Netzknoten sowohl die VCI- als auch die VPI-Werte umgesetzt, d.h. das VCI-Feld kann nicht für benutzereigene Zwecke genutzt werden.

1.4 Signalisierung

Um Verbindungen zwischen den Netzknoten aufzubauen, abzubauen oder zu modifizieren, müssen *Signalisierungsinformationen* zwischen diesen ausgetauscht werden. So muß z.B. dem Zielknoten der Wunsch nach einem Verbindungsaufbau mitgeteilt werden, der Initiator muß einen freien VPI übermittelt bekommen, usw.

Je nach Schnittstellenart (UNI oder NNI) müssen unterschiedliche Informationen standardisiert ausgetauscht werden. Hierzu definiert man einheitliche Signalisierungsprotokolle. Im UNI-Bereich sind dies beispielsweise DSS2 oder CMAP, im NNI-Bereich B-ISUP.

Der Austausch der Signalinformationen findet in Form von ATM-Zellen statt, die über gesonderte Signalisierungskanäle mit festgelgten VPI-/VCI-Werten vermittelt werden. Diese Kanäle sind entweder vom Netzbetreiber fest vorgegeben oder werden durch sogenannte *Metasignalisierung* bestimmt.

1.5 Grundform des ATM-Schichtenmodells

Wie im Kommunikationsbereich üblich, existiert auch für ATM ein Schichtenmodell (vgl. Abb. 45), welches die Dienste und Funktionen in ein übersichtliches Gerüst einpaßt. Das ATM-Schichtenmodell besteht aus drei Säulen, der Signalisierungssäule *(C-Plane)*, der Benutzersäule *(U-Plane)*, sowie der Managementsäule *(M-Plane)*.

Die Signalisierungssäule und die Benutzersäule bauen im wesentlichen auf drei Grundschichten auf: der physikalischen Schicht, der ATM-Schicht und einer Adaptionsschicht

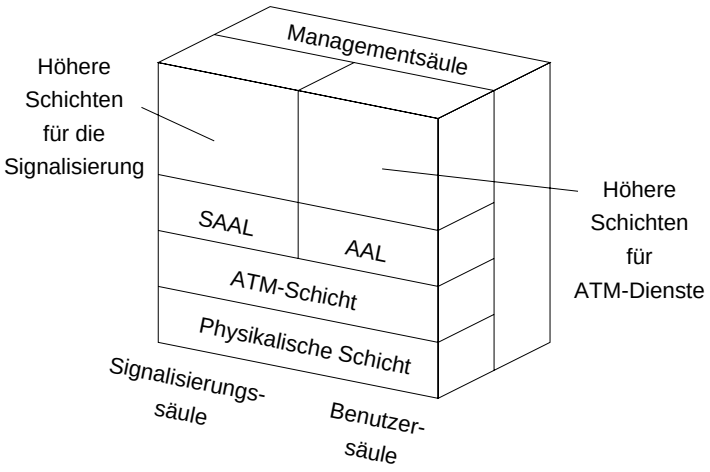

Abbildung 45. Das ATM-Schichtenmodell

(*SAAL* bzw. *AAL*). Dabei ist die AAL/SAAL-Schicht dienstabhängig und stellt für verschiedene ATM-Übertragungsarten unterschiedliche Protokolle zur Verfügung (z.B. AAL-5 für einfache Datenübertragung) und verpackt die Protokollinformationen in ATM-Pakete, während die ATM-Schicht als dienstunabhängige Schicht für den Umgang mit den Zellen selbst verantwortlich ist. Im folgenden interessiert uns nur die Signalisierungssäule.

2 UNI-Signalisierung nach DSS2 (Q.2931)

2.1 Grundprinzip

Für die Signalisierung an der Teilnehmer-Netz-Schnittstelle (UNI) kann das *Digital Subscriber Signaling No. 2* (*DSS2*, festgelegt als ITU-T-Empfehlung *Q.2931*) verwendet werden. Die DSS2-Signalisierung ist Bestandteil des internationalen Standards *B-ISDN*. Die folgende Bescheibung des DSS2-Standards richtet sich nach den Ausführungen in [Sie94].

DSS2 legt Signalisierungsnachrichten *(Messages)* fest, die in Form von ATM-Zellen über einen speziellen virtuellen Kanal ausgetauscht werden. Die DSS2-Signalisierung wird von einer sogenannten *Schicht 3* ausgeführt, die in der Signalisierungssäule über der physikalischen, der ATM- und der SAAL-Schicht angesiedelt ist[3].

2.2 Aufbau einer DSS2-Nachricht

Jede DSS2-Nachricht besteht aus einem Kopf und zusätzlichen Informationselementen (vgl. Abb. 46). Die einzelnen Komponenten einer solchen Nachricht sind die folgenden:

Protocol ID (1 Byte) kennzeichnet das verwendete Protokoll, hier DSS2 (Q.2931).

[3] Die physikalische Schicht wird bisweilen auch als *Schicht 1* bezeichnet; ATM-Schicht und SAAL zusammen bilden die zweite Schicht.

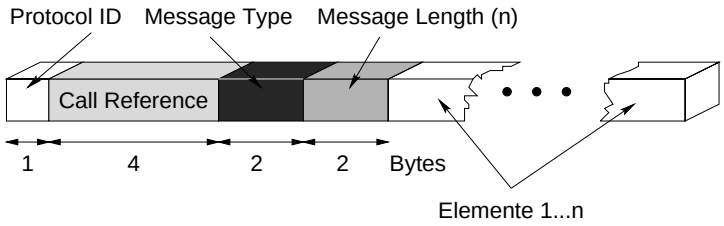

Abbildung 46. DSS2-Nachricht

Call Reference (4 Bytes): Für jede Verbindung, die ein Teilnehmer unterhält, liegt eine getrennte Signalisierungsaktivität vor, die über Call Reference = 1,2,... identifiziert wird.

Message Type (2 Bytes) bezeichnet die verwendete Signalisierungsnachricht. So steht beispielsweise `00000101` für die SETUP-Nachricht. Zusätzlich kann im zweiten Byte festgelegt werden, wie nicht unterstützte Nachrichten zu behandeln sind.

Message Length (2 Bytes): Anzahl der nachfolgenden Informationselemente.

Information Elements : Hier werden die Parameter der einzelnen Nachrichten eingetragen, bei einer SETUP-Nachricht sind dies unter anderem: ATM-Adresse, benötigte (Spitzen-) Zellrate der Verbindung, usw.

2.3 Signalisierungsbeispiel: Point-to-Point-Verbindung

Wir wollen nun ein einfaches Signalisierungsbeispiel betrachten. Dazu nehmen wir an, ein Netzteilnehmer A möchte zu einem Netzteilnehmer B eine Verbindung aufbauen, um Daten zu übertragen. Eine solche Verbindung mit zwei Endpunkten nennt man Punkt-zu-Punkt-Verbindung *(Point-to-Point-Connection)*. Der dafür nötige Protokollablauf ist in Abb. 47 dargestellt und soll im folgenden genauer analysiert werden.

Der Verbindungsaufbau durch A wird begonnen, indem dem Netz (respektive der Vermittlungsstelle von A) eine SETUP-Nachricht gesandt wird. Diese enthält in der Regel alle für die Verbindung erforderlichen Parameter — dies sind:

- AAL-Parameter und Broadband Bearer Capability: gewünschter AAL-Diensttyp und Merkmale wie feste/variable Übertragungsrate.
- ATM-Traffic-Descriptor: benötigte (Spitzen-) Zellrate der Verbindung.
- Called Party Number: Nummer (Adresse)[4] von Teilnehmer B.
- Called Party Subaddress: Subadresse von B.
- Calling Party Number: Nummer von Teilnehmer A.

[4] Der DSS2-Standard verwendet ISDN-Adressen nach der ITU-T-Norm E.164, die ähnlich Telefonnummern aufgebaut sind, d.h. sie bestehen aus einer Rufnummer mit Länder- und Ortsnetzkennzahl sowie Teilnehmernummer und einer Subadresse, um einzelne Komponenten einer Endeinrichtung zu adressieren.

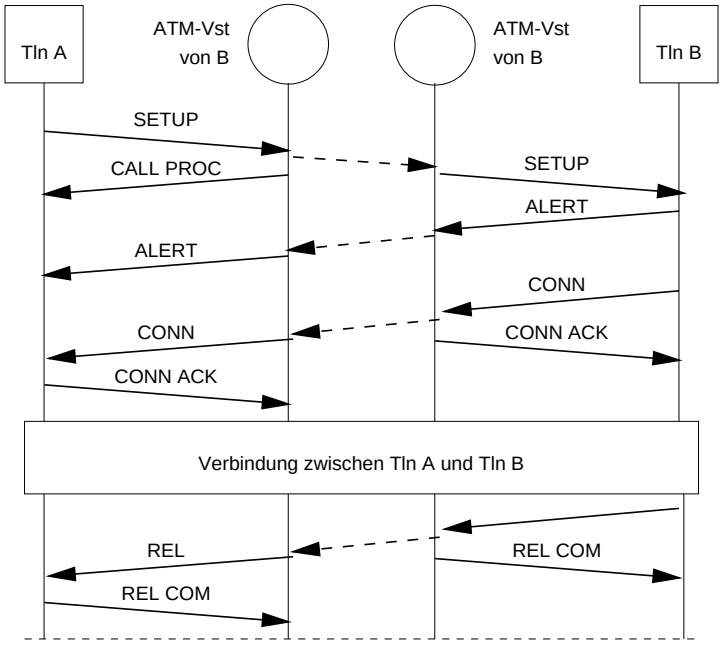

Abbildung 47. Protokollablauf für eine Punkt-zu-Punktverbindung

- Calling Party Subaddress: Subadresse von A.
- Quality-of-Service-Parameter: Qualitätsanforderungen an die Verbindung (z.B. geforderte Zellenverlustwahrscheinlichkeit).
- sowie eine Reihe von optionalen Parametern zur Beschreibung der Verbindung.

Die Teilnehmervermittlungsstelle von A reagiert auf eine unvollständige SETUP-Nachricht mit SETUP ACKnowledge, darauf hat A die Möglichkeit, mit der INFOrmation-Nachricht zusätzliche Parameter nachzuliefern[5]. Sind nicht genügend Ressourcen (z.B. VP) frei, so wird die Verbindung mit RELease COMplete abgewiesen. Im Erfolgsfall bekommt A eine CALL PROCeeding-Nachricht. Mit dieser wird A je nach Vermittlungsart die zu verwendende VPI/VCI-Information mitgeteilt.

Durch interne Netzsignalisierung (NNI) wird die Teilnehmervermittlungsstelle von B dazu veranlaßt, eine erweiterte SETUP-Nachricht an B zu senden[6]. B bekommt mit der SETUP-Nachricht ebenfalls die zu verwendende VPI/VCI-Information — durch diese wurde die SETUP-Nachricht erweitert — und überprüft die Verbindungsparameter. B reagiert mit einer ALERTing-Nachricht, die A vom Netz übermittelt wird. Mit CONNect akzeptiert B die Verbindung endgültig. Sowohl die Vermittlungsstelle von B als auch A quittieren dies jeweils mit CONNect ACKnowledge — die Verbindung ist dann eingerichtet und es können Daten im gewählten Diensttyp (z.B. AAL-5) übertragen werden.

Der Verbindungsabbau beginnt auf einer der beiden Seiten (hier B) durch das Senden der RELease-Nachricht. Daraufhin reagiert die Vermittlungsstelle von B mit RELease

[5] Dies ist vor allem bei Verbindungen mit Schmalband-ISDN-Anlagen der Fall, die stets die Rufnummern „nachliefern".

[6] Im folgenden lassen wir die NNI-Signalisierung außer acht und stellen uns vor, daß die Signalisierungsinformationen direkt zugestellt werden.

COMplete und gibt die entsprechenden VP und VC frei. Schließlich sendet die Vermittlungsstelle von A an A ebenfalls ein RELease und gibt auch hier die verwendeten Ressourcen frei. A quittiert mit RELease COMplete.

2.4 Übersicht über die DSS2-Nachrichten

Grundsätzlich existieren DSS2-Nachrichten immer in zwei Varianten, je nach Senderichtung:

- vom Teilnehmer zur Teilnehmervermittlungsstelle (zum Netz), und
- von der Teilnehmervermittlungsstelle (vom Netz) zum Teilnehmer.

Dabei unterscheiden sich die enthaltenen Informationselemente geringfügig. So enthält die SETUP-Nachricht in der Richtung vom Teilnehmer zum Netz Verbindungsparameter, und Adreßinformationen, in der Richtung vom Netz zum Teilnehmer zusätzlich noch die zu verwendenden VPI/VCI-Werte für den Übertragungsabschnitt vom Netz zum Teilnehmer.
Es folgt nun eine Übersicht über die DSS2-Nachrichten, wobei jeweils nur die wesentliche Funktion beschrieben wird. Auf Unterschiede zwischen den beiden Richtungsvarianten wird nicht eingegangen.

- Nachrichten zum Verbindungsaufbau
  - SETUP (s.o.)
  - SETUP ACKnowledge (s.o.)
  - INFOrmation (s.o.)
  - CALL PROCeeding (s.o.)
  - ALERTING (s.o.)
  - CONNect (s.o.)
  - CONNect ACKnowledge (s.o.)
  - RESTART: Neustart einer Verbindung
- Nachrichten zum Verbindungsabbau
  - RELease (s.o.)
  - RELease COMplete (s.o.)
- Nachrichten zur Statusabfrage
  - STATus: Information über den Zustand einer Verbindung
  - STATus ENQuiry: Anforderung eines Zustandsberichtes (STATus)

- NOTIFY: Übermittlung verbindungsbezogener Daten

– Nachrichten aus dem ISDN-Bereich

(Diese ermöglichen es, Verbindungen zu parken und Endgeräte umzustecken.)

- SUSPend: Anforderung, eine Verbindung zu parken
- SUSPend ACKnowledge: Bestätigung von SUSPend und Parken einer Verbindung
- SUSPend REJect: Abweisen einer SUSPend-Nachricht
- RESume: Aufforderung, eine Verbindung wieder aufzunehmen
- RESume ACKnowledge: Bestätigung von RESume und Wiederaufnahme einer Verbindung
- RESume REJect: Abweisen einer RESume-Nachricht

2.5 Übertragung der Signalisierung durch die SAAL-Schicht

Die SAAL-Schicht (vgl. Abb. 45) stellt der Signalisierungsschicht (Schicht 3) einen Dienst zur gesicherten Übertragung von DSS2-Nachrichten zur Verfügung.

Die SAAL besteht aus einem dienstspezifischen *(SSCS)* und einem dienstunabhängigen Teil *(CPCS)*. Wie aus Abb. 48 ersichtlich, ist der dienstspezifische Teil durch die beiden Teilschichten *SSCF (Service Specific Coordination Function)* und *SSCOP (Service Specific Connection Oriented Protocol)* gegeben, der dienstunabhängige besteht im wesentlichen aus dem *AAL-5* (*ATM-Adaption Layer Type 5* zur einfachen Datenübertragung).

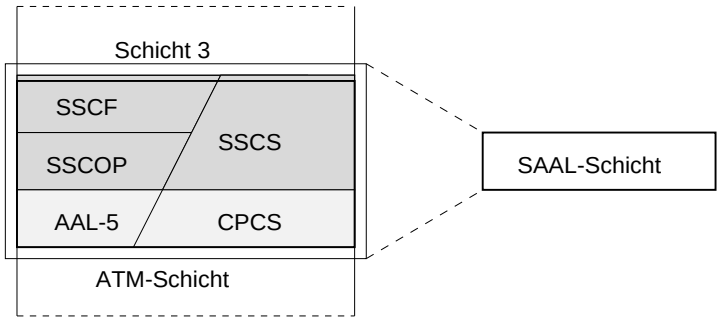

Abbildung 48. Feingliederung der SAAL-Schicht

Die SSCF-Schicht realisiert den Zugangspunkt *(Service Access Point)* für die Dienstaufträge aus der Schicht 3, während die SSCOP-Schicht das Versenden der Signalisierungsnachrichten durch die dienstunabhängige AAL-5-Schicht koordiniert.

Die SSCOP-Schicht sorgt für

– den Auf- und Abbau der Signalverbindung,

- die Einhaltung der Sendereihenfolge.
- selektives Wiederholen verlorengegangener Nachrichten (fehlerhaft übertragene Nachrichten werden von der AAL-5-Schicht einfach verworfen),
- Flußkontrolle und Synchronisation.

Die Schicht-3-Nachrichten werden dabei in numerierten Blöcken übertragen. In gewissen Abständen, die durch einen *Credit*-Parameter vorgegeben sind, führt der Sender eine *Poll-Abfrage* beim Empfänger durch, in der er die laufende Nummer des nächsten Blockes übermittelt. Der Empfänger muß dann die zuvor empfangenen Blöcke quittieren.

2.6 Metasignalisierung

Wenn keine festen Signalisierungskanäle für die Übertragung der Schicht-3-Nachrichten (respektive der ATM-Zellen, die von den Schichten SSCF, SSCOP, AAL-5 und ATM daraus erzeugt wurden) reserviert sind, müssen diese durch *Metasignalisierung* festgelegt werden. Die Instanzen der Metasignalisierung befinden sich innerhalb der ATM-Schicht und sind durch die ITU-T-Norm Q.1420 festgelegt.

Vereinfacht dargestellt läuft die Metasignalisierung ab wie folgt: Es sei angenommen, daß die Signalisierung zwischen einer Endeinrichtung und einer Vermittlungsstelle im virtuellen Pfad VPI=XX ablaufen soll. Dann sendet die Endeinrichtung im Kanal VPI=XX, VCI=1 eine genormte Signalisierungsanforderung (Assign-Request), auf die die zuständige Vermittlungsstelle mit einer Assign-Nachricht antwortet, in der sie den zur Signalisierung zu verwendenden VCI mitteilt. Steht kein Signalisierungskanal mehr zur Verfügung, so antwortet die Vermittlungsstelle mit Denied.

3 UNI-Signalisierung nach CMAP

3.1 Konzepte des CMAP-Protokolls

CMAP (Connection Management Access Protocol) wurde als UNI-Signalisierungsprotokoll ab 1989 an der Washington University, St. Louis, USA, entwickelt. Grundlage der folgenden Betrachtung ist die CMAP-Spezifikation 3.0 [CD94].

CMAP basiert auf dem sogenannten *Call*-Konzept. Ein Call umfaßt in der Regel mehrere (Einzel-) Verbindungen *(Connections)* zwischen mehreren Teilnehmern *(Clients)*. Diese Clients werden als sogenannte Endpunkte *(Endpoints)* in die Calls aufgenommen. Dabei ist jeder Endpunkt an allen Connections des Calls beteiligt.

Jedem Call werden bestimmte Parameter zugeordnet (vgl. Tabelle 5).

Von besonderer Wichtigkeit sind natürlich die Felder Endpoint List und Connection List, da hier alle Verbindungen des Calls und alle beteiligten Teilnehmer zusammen mit den jeweiligen Parametern (vgl. Tabellen 6 und 7) aufgeführt werden.

Mit Hilfe der Endpoint List wird ausgedrückt, in welcher Beziehung die Endpunkte zu den einzelnen Connections des Calls stehen. Dies wird durch sogenannte Mappings bewirkt. Dabei handelt es sich um Tripel der Form

Parameter	Beschreibung
Owner	Eigentümer[7] des Calls; er übernimmt die Verwaltung des Calls.
Root	weiterer Client; zu ihm werden die ATM-Zellen zunächst geroutet. Meist handelt es sich bei Owner und Root um denselben Client (siehe aber Abschnitt 3.2).
Local Call Identifier	bildet zusammen mit der Root-Adresse eine eindeutige Referenz auf den Call
Type	Multipoint- oder Point-to-Point-Call
Accessability	gibt an, ob und wie Clients als Endpunkte in den Call aufgenommen werden dürfen.
Modifiability	gibt an, ob neben dem Owner auch andere Clients dem Call weitere Connections hinzufügen dürfen.
Traceability	gibt an, welche Clients Informationen über den Call abfragen dürfen.
Monitoring	gibt an, welche Clients bei Veränderung der Parameter des Calls informiert werden.
Priority	Einstufung der Priorität des Calls.
Endpoint List	Liste mit allen Endpunkten des Calls und deren Parametern
Connection List	Liste mit allen Connections des Calls und deren Parametern

Tabelle 5. Parameter eines CMAP-Calls

$$(\text{receive, transmit, echo}) \in \{\text{ON, HOLD, OFF}\} \times \{\text{ON, HOLD, OFF}\} \times \{\text{ON, OFF}\},$$

die angeben ob ein Endpunkt in der jeweiligen Connection empfangs- und/oder sendeberechtigt ist, und ob er die von ihm selbst gesandten Zellen inspizieren darf. ON bedeutet dabei jeweils, daß der Endpunkt die entsprechende Erlaubnis besitzt, OFF dagegen, daß er sie nicht hat. HOLD bedeutet, daß der Endpunkt im Moment nicht empfangen und/oder senden darf, daß aber Bandbreite im Netz für den Fall einer späteren Genehmigung reserviert ist.

Alle Parameter eines Calls können mit CMAP-Operationen beeinflußt werden[8]. CMAP unterscheidet vier Klassen von Operationen: Befehle *(Commands)*, Aufforderungen *(Prompts)*, Nachfragen *(Queries)* und Mitteilungen *(Notifications)*. Diese Operationen

[7] Gespeichert wird die Adresse eines Clients. Gültige Adressen müssen einer CMAP-Adressschablone genügen, in die sich IP-, E.164- und OSI NSAP-Adressen leicht einpassen lassen.

[8] Neben diesen Operationen zur Manipulation eines Calls (den *Call Operations*) gibt es auch die sogenannten *Maintenance Operations*, die jedoch hier nicht weiter interessieren sollen. Sie dienen beispielsweise zu Statusabfragen oder um Fehler anzuzeigen.

Parameter	Beschreibung
Address	Client mit dem dieser Endpunkt assoziiert wird.
Local Identifier	für den Client eindeutiger Bezeichner
und zusätzlich für jede Connection des Calls individuell:	
Mapping	Angabe, ob der Endpunkt in der jeweiligen Connection empfangen und/oder senden darf.
Defaults	Default Mapping, also das Mapping, das ein Endpunkt beim Anlegen der Connection zugewiesen bekommen hat (vgl. Tabelle 7).
Permissions	Angabe, ob der Endpunkt sein Mapping für die jeweilige Connection selbst ändern darf.
Transmit Pair	VPI/VCI-Paar für das Senden von ATM-Zellen durch die jeweilige Connection.
Receive Pair	VPI/VCI-Paar für das Empfangen von ATM-Zellen durch die jeweilige Verbindung.

Tabelle 6. Parameter eines CMAP-Endpoints

Parameter	Beschreibung
Identifier	innerhalb des Calls eindeutiger Bezeichner für die Verbindung
Type	Art der Vermittlung (VP- oder VC-Vermittlung), QoS-Service Parameter, sowie Markierung ob fixe/variable Übertragungsrate benötigt wird.
Bandwidth	Reservierte Bandbreite der Verbindung
Defaults	Default Mapping, das jedem neuen Endpunkt der Connection angeboten wird.
Permissions	Rechte zur Änderung des Mappings, die ein neuer Endpunkt für diese Connection angeboten bekommt.
User Type	Benutzerdefinierter Typ für die Connection.

Tabelle 7. Parameter einer CMAP-Connection

werden durch den Austausch von CMAP-Nachrichten *(Messages)* zwischen den Clients realisiert.

Zur vollständigen Manipulation eines Calls (z.B. zum Erzeugen oder zum Zufügen eines weiteren Endpunktes) ist eine ganze Folge von Operationen notwendig, die jeweils von einer Command-Operation eingeleitet wird. Auf diese können der Reihe nach Prompts, Queries und Notifications folgen. Jede dieser Einzeloperationen besteht wiederum aus bis zu drei festgelegten Phasen:

1. Phase *(Request)*: Ein Client beginnt eine Operation mit einer REQ-Nachricht.

2. Phase *(Response)*: Das Netz reagiert mit einer Antwortnachricht: ACKnowledge, Ne-

gative ACKnowledge oder NEGotiate.

3. **Phase** *(Confirmation)*: Der initiierende Client erklärt sich mit der Response aus Phase 2 einverstanden und sendet im Falle eines ACK ein COMmit und im Falle eines NACK ein ABORT.

   (War die Response ein NEG, so ist weitere Kommunikation nötig, d.h. es muß gegebenenfalls eine weitere Operation durchgeführt werden — eine Confirmation findet dann nicht statt.)

In Tabelle 8 sind die wichtigsten CMAP-Befehle zusammengestellt.

Befehl	Beschreibung
open_call	Erzeugen eines neuen Calls
mod_call	Ändern der Parameter eines Calls
close_call	Beenden eines Calls
add_con[9]	Hinzufügen einer neuen Connection zum Call
mod_con	Ändern der Parameter einer Connection
drop_con	Beenden einer Connection
add_ep	Hinzufügen eines weiteren Endpunktes
mod_ep	Ändern der Parameter eines Endpunktes
drop_ep	Entfernen eines Endpunktes
trace_call	Abfragen der Parameter eines Calls, der beteiligten Endpunkte und der erzeugten Connection
trace_ep	Abfragen der Parameter eines Endpunktes
change_owner	Weitergabe des Eigentumsrechtes an einen anderen Client
change_root	Bestimmen eines anderen Clients zum Root-Client

Tabelle 8. Die wichtigsten CMAP-Befehle

Um die Signalisierungsabläufe nach CMAP und die Bedeutung des Call-Konzeptes besser verstehen zu können, betrachten wir in den folgenden Abschnitten zwei Beispiele. Beide stellen typische Multipoint-Anwendungen dar, die in komplizierterer Form in Zukunft im Alltag häufig anzutreffen sein werden.

3.2 Anwendungsbeispiel: Video/Audio-Server

Wir nehmen die folgende Situation an: An ein ATM-Netz (vgl. Abb. 49) ist eine Audio/-Video-Quelle (A) angeschlossen. Diese A/V-Quelle soll von mehreren Clients genutzt werden. Die Verwaltung und die Signalisierung der A/V-Quelle übernimmt ein weiterer Client C. Man bezeichnet A als stummen Client, der dem Netz als solcher bekannt ist: alle Signalisierungsinformationen an A werden zu C umgeleitet *(Surrogate Signaling)*.

[9] In den Namen der CMAP-Operationen steht das Kürzel *ep* für Endpoint und *con* für Connection.

Da A keine Signale erhält, spielt A permanent Daten ins Netz ein. Diese werden vom Netz solange verworfen, bis A in einen Call aufgenommen worden ist[10].

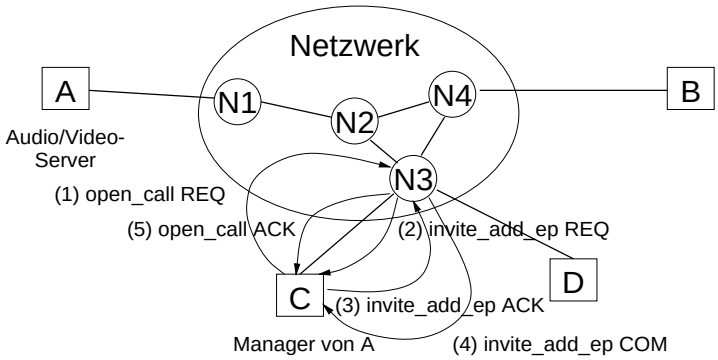

Abbildung 49. Aufbau eines Calls für einen Audio/Video-Server.

C richtet zunächst einen Call ein, indem er ein open_call REQ an das Netz sendet (1). Dabei fordert C für den Call die folgenden Parameter (siehe auch Tabelle 5):

- Owner: C wird Eigentümer des Calls.
- Root: A.
- Type: Der Call ist ein Multipoint-Call.
- Accessability: VERIFY, d.h. alle Clients können am Call teilnehmen, allerdings nur nach Genehmigung durch den Eigentümer C.
- Monitoring: OWNER, d.h. alle Veränderungen (Hinzunahme, Ausscheiden eines Clients) werden C angezeigt.
- Connections: Es werden zwei Connections vorgesehen, eine für die Übertragung des Videobildes und eine für das Audiosignal.

Für die beiden Connections fordert C eine konstante Bitrate und eine hohe Qualität; die Connection Defaults (ON, OFF, OFF) und die Connection Permissions (OFF, OFF, OFF) werden so gesetzt, daß ein neuer Client nur empfangen darf und dies auch nicht ändern kann.

Nachdem das Netzwerk die open_call-Nachricht erhalten hat, fordert es Client A mit invite_add_ep REQ auf, am Call teilzunehmen. Da ja C die Signalisierung für A übernimmt, wird diese Meldung an C umgeleitet (2), worauf C für A mit invite_add_ep ACK antwortet (3). Daraufhin signalisiert das Netz an A (respektive C), daß A in den Call als (bis jetzt einziger) Endpunkt aufgenommen worden ist (invite_add_ep COM, 4), und an C, daß der Call erfolgreich eingerichtet worden ist (open_call ACK, 5).

[10] Einen solchen stummen Client dem Netz bekannt zu machen, ist nicht Aufgabe des CMAP-Protokolls, sondern anderer Einrichtungen des Netzes.

Nun möchte ein Client B (Abb. 50) von A A/V-Daten empfangen[11]. Dazu beantragt er mit add_ep REQ (1) die Aufnahme in den Call. Da der Accessability-Parameter für den Call vorschreibt, daß zunächst bei C die Genehmigung für die Teilnahme eingeholt werden muß, sendet das Netzwerk eine entsprechende Anfrage an C (verify_add_ep REQ, 2). Beantwortet dies C mit verify_add_ep ACK (3), so darf B teilnehmen und bekommt somit vom Netz ein add_ep ACK (4). Wegen des Monitoring-Parameters (s.o.) erhält C noch zum Abschluß vom Netz ein announce_add_ep REQ (5), das anzeigt, daß B aufgenommen wurde. So kann C beispielsweise eine Art Gebührenzähler aktivieren, um B die genützten Videodaten später in Rechnung zu stellen. Darf B nicht am Call teilnehmen — beispielsweise, weil er seine Video-Rechnung nicht bezahlt hat — so erhält B ein add_ep NACK.

Im Erfolgsfall nimmt B nun als Empfänger — Mapping: (ON. OFF, OFF) — an beiden Connections des Calls teil und erhält durch sie die A/V-Signale. Somit sind nun zwei Endpunkte am Call beteiligt: der Video-Server A und Client B; C ist nur Eigner des Calls und nicht als Endpunkt in die Datenverbindung eingebunden.

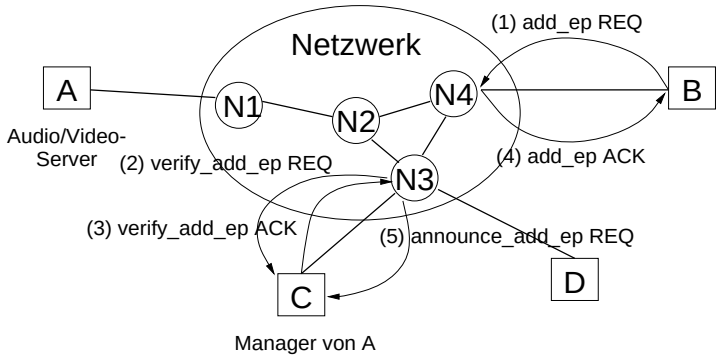

Abbildung 50. Der Audio/Video-Server bekommt Kunden.

Das Hinzunehmen weiterer Clients (z.B. D) verläuft absolut identisch; es werden dieselben Signalbotschaften ausgetauscht wie für Client B. Folgende Punkte sind dabei beachtenswert:

- Der Übergang vom Punkt-zu-Punkt-Call zum Mehrpunkt-Call ist nahtlos: sowohl der zweite Endpunkt als auch alle weiteren (D,...) werden durch ein- und denselben Befehl eingebunden: add_ep.
- B erhält weiter ein ungestörtes Videobild; es ist vom Anfügen neuer Clients nicht betroffen — vorausgesetzt es stehen genügend Netzwerkressourcen zur Verfügung.

Möchte ein Client den Bezug der Videodaten einstellen, so sendet er ein drop_ep REQ. Das Netzwerk antwortet mit drop_ep ACK und die Verbindung wird getrennt. Wegen des

[11] Woher weiß Client B, daß eine solche A/V-Quelle existiert, und daß er am oben eingerichteten Call teilnehmen muß, um A/V-Daten zu empfangen? — Dazu müßte man ein Verzeichnis aller Video-Quellen und der damit verbundenen Calls einrichten, welches die Clients dann abfragen können. Dies ist aber nicht Aufgabe des CMAP-Protokolls, sondern einer höheren Netzwerkschicht.

Monitoring-Parameters wird C davon mit announce_drop_ep REQ in Kenntnis gesetzt. C kann nun den Gebührenzähler stoppen.

3.3 Anwendungsbeispiel: Multimediakonferenz

Wir betrachten folgende Anwendung (stark vereinfacht): Mehrere Benutzer möchten eine Videokonferenz abhalten. Jeder der Benutzer hat an seinem Arbeitsplatz eine leistungsstarke, grafikfähige Workstation mit Multimediaausrüstung (d.h. Videokamera, Mikrophon, Lautsprecher oder Kopfhörer, Videokompressionskarte, ...) zur Verfügung.

Für die Durchführung der Konferenz sollen gewisse (einschränkende) Merkmale gelten:

- alle Benutzer sollen gleichzeitig senden dürfen
- ein Benutzer kann die Audio/Video-Daten genau eines anderen Benutzers empfangen, er kann aber jederzeit zwischen den Benutzern hin und herschalten.
- ein Benutzer kann sich jederzeit aus der Konferenz ausklinken
- eine grafische Benutzeroberfläche soll den Anwender übersichtlich über den Status der Konferenz informieren (welche Benutzer sind an der Konferenz beteiligt, welche möchten wichtige Tonmitteilungen machen, ...) und ihm die komfortable Auswahl der aktiverbaren Benutzer ermöglichen
- die Workstation-Software sollte irgendwelche Mechanismen bieten, um gezielte Diskussionen zu ermöglichen (beispielsweise durch Vergabe von Rederechten oder Einführung eines Moderators)

Mit CMAP kann man ein solches System realisieren, indem man einen Call mit den folgenden Eigenschaften definiert:

- Jeder Benutzer wird als Endpunkt in den Call aufgenommen.
- Eine Connection (con_0) wird zur Koordination vorgesehen.
- Jedem Benutzer wird eine Connection zugeordnet, die alle anderen Teilnehmer erreicht (con_1,...,con_n). Auf diese Weise können alle anderen Benutzer seine Daten empfangen. Dies wird erreicht, indem ein Benutzer A für seine eigene Connection das Senderecht erhält: Mapping (OFF, ON , ON). Die Connections, die ihn von anderen Benutzern erreichen, werden auf seiner Seite mit dem Mapping (ON, OFF, OFF) bzw. (HOLD, OFF, OFF) versehen — je nach dem, ob A den betreffenden Partner aktiviert hat oder nicht. (Für eine Konferenz mit drei Teilnehmern sind diese Mappings in Abb. 51 dargestellt.)

Die Konferenz beginnt damit, daß ein Benutzer A den zugehörigen Call eröffnet: Er sendet ein open_call REQ ans Netz und definiert mit dessen Parametern die für die Koordinierung bestimmte Connection con_0 und sich selbst als einzigen Endpunkt. Er schreibt ferner vor, daß neue Endpunkte nur mit seiner Genehmigung (Accessability=VERIFY)

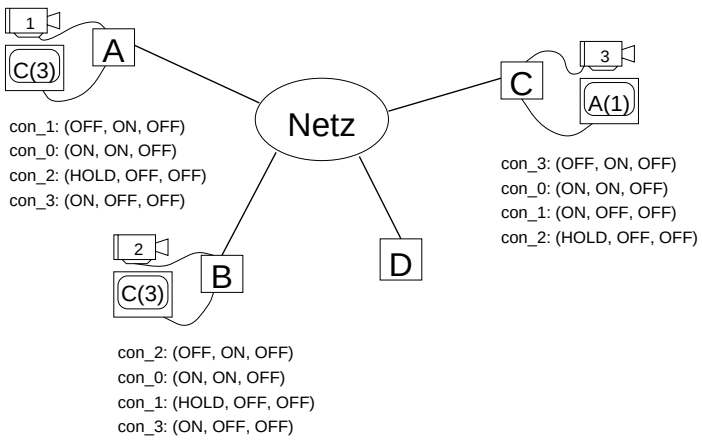

Abbildung 51. CMAP: Connections für eine Multimediakonferenz.

und neue Connections beliebig (Modifiability=OPEN) eingerichtet werden dürfen. Weiter sollen alle Mitglieder des Calls (d.h. alle Teilnehmer der Konferenz) über neue Endpunkte einer Connection informiert werden (Monitoring=MEMBERS). Für con_0 sieht A das Default Mapping (ON, ON, OFF) vor — jede Client-Software soll ja schließlich Kontrollinformationen senden und empfangen dürfen.

Dann definiert A seine Connection (con_1). Dazu sendet er ein add_con REQ ans Netz und setzt für sie das Default Mapping (also die Rechte für die anderen Teilnehmer) auf (HOLD, OFF, OFF). A erlaubt den anderen Endpunkten aber durch die Permisson Defaults von (ON, OFF , OFF), dieses Mapping in (ON OFF, OFF) abzuändern, um ihn empfangen zu können. Für sich selbst benutzt A das Mapping (OFF, ON, OFF), da er ja durch diese Connection senden will.

Ein neuer Benutzer B kann mit add_ep REQ die Aufnahme in die Konferenz beantragen. Gibt der Eigentümer des Calls (A) seine Genehmigung, so muß B wie vorher A seine Connection (z.B. con_2) dem Call hinzufügen (add_con REQ). Automatisch nimmt B auch an allen anderen im Call definierten Connections teil, wobei er das jeweilige Default-Mapping zugeordnet bekommt. Den anderen Clients wird der neue Benutzer wegen des Monitoring-Parameters MEMBERS mit announce_add_ep angekündigt.

Um einen bestimmten Kommunikationspartner einzublenden, muß die Workstation-Software eines Nutzers dessen Connection auf Empfang (ON, OFF, OFF) setzen, alle anderen Connections dagegen auf (HOLD, OFF, OFF). Dies kann mit dem mod_ep-Befehl bewirkt werden.

Möchte ein Benutzer C die Konferenz verlassen, so muß er der Reihe nach die folgenden Schritte durchführen (lassen):

1. Ist C Eigentümer des Calls, so muß er diese Eigenschaft mit change_owner an einen anderen Teilnehmer weitergeben.

2. Er muß seinen Wunsch, aus der Konferenz auszuscheiden, dem (neuen) Eigner mitteilen. (Hier kann con_0 genutzt werden.) Der Eigner muß dann C's Connection löschen (drop_con). Alle anderen Clients werden durch announce_drop_con davon informiert, so daß die Konferenzsoftware gegebenenfalls auf einen neuen Kanal wechseln kann.

3. Mit drop_ep löscht der Eigentümer dann auch noch den C repräsentierenden Endpunkt. Auch dies wird den anderen Teilnehmern angezeigt (announce_drop_ep).

Die anderen Teilnehmer können ihre Konferenz ungestört weiterführen.

Bemerkung: Dies ist nur eine grobe, vereinfachte Skizze eines solchen Systems und eine Beschreibung der Möglichkeiten, die CMAP zu seiner Realisierung bietet. Es sind selbstverständlich zahlreiche (nicht triviale !) planerische und technische Punkte zu beachten, wenn man ein solches System tatsächlich erstellen möchte.

4 NNI-Signalisierung

4.1 Grundprinzip und Schichtenmodell

Zwischen den Vermittlungsstellen müssen ebenfalls Signalisierungsinformationen ausgetauscht werden. Diese Aufgabe bezeichnet man als NNI-Signalisierung. Es soll im folgenden kurz auf deren (geplante) Realisierung im ATM-basierten B-ISDN eingegangen werden. (Genauere Informationen zu diesem Themenkomplex finden sich beispielsweise in [Sta94]).

Die Signalisierung erfolgt über einen netzintern vorgegebenen virtuellen Kanal, den zentralen *Zeichengabekanal*[12], durch den Austausch von Signalisierungsnachrichten nach dem ITU-T-Zeichengabesystem Nr. 7 *(SS7, Signaling System 7)*.

Der NNI-Signalisierung im B-ISDN liegt das Schichtenmodell aus Abbildung 52 zugrunde.

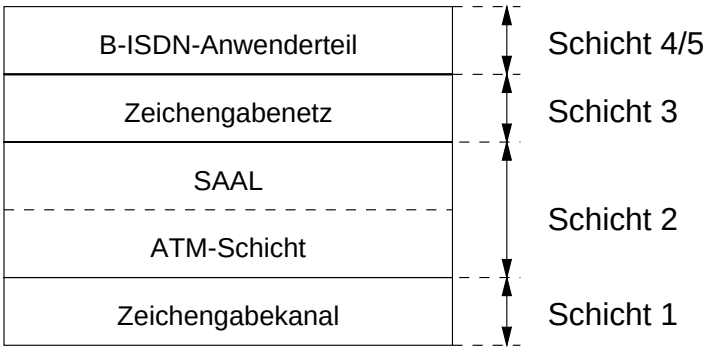

Abbildung 52. Schichtenmodell für die NNI-Signalisierung.

Die einzelnen Schichten dienen der Bearbeitung folgender Aufgaben:

Schicht 4/5: Der B-ISDN-Anwenderteil ist zuständig für den Auf- und Abbau von Nutzkanalverbindungen. Mit Hilfe einer *abschnittsweisen* Signalisierung von Vermittlungsstelle zu Vermittlungsstelle wird anhand der ISDN-Adresse der Weg für eine Nutzkanalverbindung gesucht und eine *Ende-zu-Ende*-Signalverbindung zur Realisierung der Nutzverbindung aufgebaut. Hierbei werden sogenannte B-ISUP-Nachrichten verwendet.

[12] Zeichengabe ist der aus der ISDN-Welt entnommene Begriff für Signalisierung.

Schicht 3: Das Zeichengabenetz dient zur Beförderung der Signalisierungsnachrichten. Die Schicht 3 sorgt dafür, daß die B-ISUP-Nachrichten die geeignete Schicht-4/5-Instanz erreichen — abhängig davon, ob es sich um eine abschnittsweise oder eine Ende-zu-Ende-Signalisierung handelt. Ferner unterstützt Schicht 3 die Schicht 4/5 durch die Übernahme von Managementfunktionen für das Zeichengabenetz (Zustandserfassung von Signalisierungsteilstrecken, Ersatzschaltung von Teilstrecken,...)

Schicht 2: Die Schicht 2 übernimmt, wie bei der UNI-Signalisierung, die gesicherte Übertragung der Nachrichten auf den Teilstrecken.

4.2 Netzmanagementfunktionen

Eng mit den Aufgaben der Schicht 4/5 verknüpft ist das Management des ATM-Kommunikationsnetzes im B-ISDN. Hierzu gehören Funktionen wie

- Verkehrsverwaltung
- Gebührenverwaltung
- Verwaltung der Teilnehmeranschlüsse
- Steuerung und Konfiguration der Vermittlungsstellen

Es ist hierzu geplant, ein physikalisch und logisch getrenntes Managementnetz *(TMN, Telecommunication-Management Net)* einzuführen, welches über eine genormte Schnittstelle (ITU-T Q.3) mit dem B-ISDN-Anwenderteil kommuniziert.

5 Abschließende Betrachtung

Zum Abschluß sollen die beiden UNI-Signalisierungsprotokolle DSS2 und CMAP bezüglich ihrer wesentlichen Eigenschaften kurz miteinander verglichen werden.

Die primäre Zielsetzung beim Entwurf des DSS2-Signalisierungsschemas war die Kompatibilität zum Signalisierungsstandard Q.931 des konventionellen Schmalband-ISDN[13]. Dies ermöglicht den Einsatz von ATM als Basistechnologie für das Breitband-ISDN und eine leichtere Integration von älteren ISDN-Anlagen. Da ISDN bisher nur Punkt-zu-Punkt-Verbindungen kennt, stellt auch die Release 1 (1994) des DSS2 nur solche zur Verfügung.

CMAP dagegen ermöglicht durch sein flexibles Call-Konzept in natürlicher Weise Mehrpunktverbindungen, wie sie für moderne Multimediaanwendungen erforderlich sind (vgl. Abschnitte 3.2 und 3.3). Weitere Gesichtspunkte (offene Architektur, Verwendung von TCP/IP-Elementen) wurden beim Entwurf des CMAP-Protokolls ebenfalls berücksichtigt. Die CMAP-Architektur sieht zusätzlich Mechanismen vor, die es ermöglichen, Übergänge zu Q.2931-basierenden Anlagen zu schaffen.

[13] ISDN verwendet allerdings keine virtuellen Kanäle zur Signalisierung, sondern einen dazu vorgesehenen physikalischen Kanal (D-Kanal).

Diese Flexibilität und konzeptionelle Überlegenheit von CMAP muß sich aber zunächst in einer konkreten Implementierung erweisen, die jedoch noch nicht vollständig erfolgt ist.

DSS2 weist gegenüber CMAP den Vorteil auf, ein international anerkannter Standard zu sein und im Gespann mit B-ISDN vermutlich weltweite Verbreitung zu erfahren. Ferner sind Erweiterungen für DSS2 — Release 2 (1995): Call-Konzept; Release 3 (frühestens 1996): Multimediadienste — geplant, die DSS2 gründlich modernisieren werden.

Bereitstellung eines verbindungslosen Dienstes in ATM-Netzen

Frank Müller

Kurzfassung

ATM als Hochgeschwindigkeitsnetz der Zukunft bietet zunächst nur einen verbindungsorientierten Dienst an. Unter Berücksichtigung der Anforderungen, die besonders im LAN Bereich an ATM gestellt werden, wird die Notwendigkeit der Existenz eines verbindungslosen Dienstes auf der Basis von ATM deutlich. Die Bereitstellung eines solchen Dienstes erfordert die Etablierung eines virtuellen, verbindungslosen Netzes auf dem verbindungsorientierten ATM Netz. Dabei ist vor allem darauf zu achten, daß die wesentlichen Vorteile der ATM-Technik – Skalierbarkeit und Zeittransparenz – beim aufgesetzten verbindungslosen Netz nicht verlorengehen. Zur Lösung dieser Aufgabe wurden von verschiedenen Normungsgremien Vorschläge veröffentlicht. Der Schwerpunkt dieses Artikels liegt in der Vorstellung der *Connectionless Server* Technik, wie sie auch bei B-ISDN Pilotprojekten angewendet wird.

1 Einleitung

Das ATM-Netz ist als Weiterentwicklung des X.25-Konzepts ein verbindungsorientiertes Netz mit Vermittlungsfunktion [BHK94a]. ATM Endgeräte können nur miteinander kommunizieren, nachdem zwischen ihnen eine Verbindung aufgebaut wurde. Eine ATM-Verbindung besteht aus einer permanenten Reservierung von virtuellen Kanälen (VCs, *Virtual Channels*), virtuellen Pfaden (VPs, *Virtual Paths*) und der festen Zuordnung ihrer VPI/VCI (*Virtual Path Identifier/Virtual Channel Identifier*) Identifikationspaare zur Verbindung. Ein ATM-Vermittlungssystem (ATM-*Switch*) hat dabei nur noch die Aufgabe, die VPI/VCI Werte einer einkommenden Verbindung auf die entsprechenden VPI/VCI Ausgangswerte umzusetzen. Eine Unterstützung von logischen Punkt-zu-Mehrpunkt-Verbindungen, wie sie in klassischen LANs benötigt wird, ist damit zunächst nicht möglich.

Im Gegensatz dazu arbeiten die klassischen LANs, wie CSMA/CD (Ethernet) oder FDDI verbindungslos. Hierbei können zwei Endsysteme miteinander kommunizieren, ohne daß zuvor ein Kontext über den Datenaustausch der beiden auf dem Netz etabliert wird.

Das ATM als Hochgeschwindigkeitsnetz der Zukunft kann es sich nicht leisten, seine Kompatibilität zu den klassischen LANs zu vernachlässigen. LANs sind heute weit verbreitet und haben sich als eine geeignete Plattform für viele existierende verteilte Anwendungen erwiesen. Außerdem muß ein Weitverkehrsnetz immer auch für die Verbindung von einzelnen LANs über große Entfernungen bereit stehen. Eine Integration der beiden Konzepte ATM und LAN ist also dringend geboten.

Diese nötige Integration wird nun von der Bereitstellung verbindungsloser Dienste über ATM erwartet. So soll die Brücke zwischen ATM und den klassischen LANs geschlagen

werden. Die verbindungslosen Dienste müssen den herkömmlichen Endgeräten das ATM-Netz als klassisches Netz darstellen.

Zur Lösung dieser Aufgabe wurden von den verschiedenen Normungsgremien 3 Ansätze entwickelt:

1. Im WAN Bereich wurde von der ITU (früher CCITT) die Verwendung von Servern für verbindungslose Dienste (*Connectionless Server*) vorgeschlagen. Dieses Verfahren wird bei den B-ISDN Pilotversuchen zwischen Berlin, Hamburg und Köln und zwischen Paris, Lyon, Karlsruhe, Stuttgart, Heidelberg und Ulm erprobt.

2. Die IETF (Internet Engineering Task Force) beschäftigt sich mit einem Konzept, das einer Station mit Internet Protokollstack erlauben soll, ATM als Transportmedium zu verwenden. Dieses Konzept wird auch kurz *IP über ATM* genannt.

3. Im lokalen Bereich heißt die Lösung *LAN-Emulation* und stammt vom ATM-Forum. Diese Technik soll herkömmlichen Anwendungen ATM als virtuelles LAN anbieten, ohne daß die Anwendungen auf gewohnte Funktionalität wie Broad-/Multicast verzichten müssen.

Der Rest dieses Kapitels wird sich ausführlich mit der ersten Lösung befassen. Obwohl sich diese Lösung für den WAN-Bereich aus den ersten Vorschlägen für die Integration verbindungsloser Dienste entwickelt hat, beinhaltet sie immer noch viele offene Fragen, besonders wenn es um Detaillösungen geht, bei denen stets mehrere Vorschläge sorgfältig untersucht werden müssen.

Die Behandlung der Thematik wird dabei im Folgenden stufenweise verfeinert. Am Anfang bildet das gesamte Netz den Rahmen der Betrachtung. Danach wird der Verbindungslose Server (*Connectionless Server, CLS*) als Bestandteil des Netzes zum Mittelpunkt. Schließlich werden dann die internen Mechanismen des Verbindungslosen Servers untersucht. Eine konkretes Beispiel soll zum Abschluß noch einige der bis dahin vorgestellten Aspekte der CLS-Technik veranschaulichen.

2 Das Netz wird aktiv

Wenn wir unser Ziel erreichen wollen, das ATM-Netz herkömmlichen Endgeräten als ein klassisches Netz darzustellen, so werden wir nicht darum herum kommen, die Aufgabenverteilung zwischen Endsystem und Netz neu vorzunehmen. ATM hat einen Großteil seiner Funktionalität in die Endsysteme verlagert. Dort sorgt in Hardware realisierte Verarbeitungslogik für die bestmögliche Geschwindigkeitsausbeute. Das Netz ist von komplexen Aufgaben weitgehend befreit. Im Gegensatz dazu stellen verbindungslose Netzwerke keine großen Anforderungen an die Endsysteme. Diese müssen sich nicht um Verbindungen und Routing kümmern. Ein verbindungsloser Dienst über ATM, der dieses Verhalten nachahmen soll, muß also zwangsläufig einen Teil seiner Funktionalität ins Netz zurückverlagern. Das ATM-Netz wird aktiv.

Für die Überlagerung eines virtuellen verbindungslosen Netzes über ein ATM-Netz wurden von der ITU (CCITT) zwei Konzepte vorgeschlagen: der Indirekte Ansatz und der Direkte Ansatz.

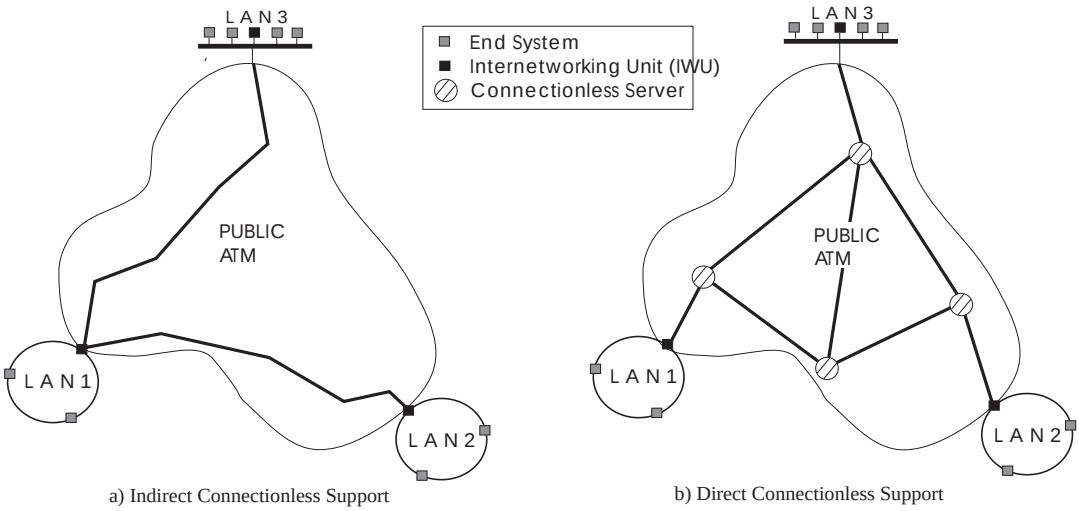

Abbildung 53. Indirekter und Direkter Ansatz.

2.1 Der Indirekte Ansatz

Der Indirekte Ansatz realisiert den verbindungslosen Dienst auf der Grundlage von virtuellen Verbindungen zwischen ATM-IWUs[14]. Dabei werden die IWUs jeweils paarweise miteinander verbunden (Abbildung 53a). Es entsteht so ein dichtes Gewebe von virtuellen Verbindungen auf dem öffentlichen Netz. Bei diesen Verbindungen unterscheidet man zwischen PVCs (*Permanent Virtual Connection*) und SVCs (*switched virtual connection*). Eine PVC-Verbindung bleibt für die gesamte Dauer der Bereitstellung des verbindungslosen Dienstes aktiv, während eine SVC-Verbindung bei Bedarf aufgebaut wird und nach einer inaktiven Phase gewisser Länge wieder verschwindet.

Zunächst fällt auf, daß hier die Notwendigkeit, Funktionalität ins Netz zurückzuverlagern, wie sie sich aus den einführenden Bemerkungen ergab, umgangen wurde. Jedem IWU-IWU Paar wird eine Art eigene Standleitung im ATM-Netz bereitgestellt. Der Datentransport erfolgt auf der ATM-Schicht mit allen Vorteilen, die sich daraus ergeben. So können die Daten auf dem kürzesten Weg, mit garantierter Geschwindigkeit und reihenfolgetreu ausgetauscht werden. Besonders bei der Verwendung von PVC-Verbindungen fallen die benötigten IWUs einfach und kostengünstig aus.

Das größte Problem beim Indirekten Ansatz stellt sein verschwenderischer Umgang mit Bandbreite dar. Permanente Verbindungen verbrauchen Netzresourcen, auch wenn sie gerade unbenutzt sind. Andere Reservierungsanforderungen müssen so zurückgewiesen werden, obwohl genügend unbenutzte Bandbreite im Netz vorhanden wäre. Verfügen

[14] Eine ATM-IWU (internetworking unit) zwischen einem LAN und dem öffentlichen ATM-Netzwerk ist entweder eine Bridge oder ein Gateway. Sie regelt die Transformation des Datenstroms beim Übergang zwischen den Netzen.

die PVCs jedoch über zu wenig vorreservierte Bandbreite, so kommt es bei Hochlast zu Datenverlusten. Ein weiterer Kritikpunkt ist die mangelhafte Skalierbarkeit. Wird eine neue IWU in ein Netz mit `n` IWUs gefügt, so sind `n` neue Verbindungen auf dem Netz zu etablieren. Teilweise kann man diese Probleme durch die Verwendung von SVCs anstatt der PVC-Verbindungen lösen. Dabei handelt man sich allerdings für jeden Verbindungsaufbau hohe Verzögerungszeiten ein. In den meisten Fällen ergeben sich dadurch noch größere Probleme.

2.2 Der Direkte Ansatz

Beim Direkten Ansatz unterstützt das ATM-Netz den Transport verbindungsloser Daten durch die Bereitstellung *verbindungsloser Server, CLS* (Abbildung 53b). Die verschiedenen CLSs bilden ein virtuelles, verbindungsloses Netz auf dem verbindungsorientierten ATM-Netz. Verbindungen zwischen CLS sind dabei permanent, während IWU-CLS Verbindungen auch nach Bedarf aufgebaut werden können. Ein CLS muß für jedes einkommende Paket den nächsten CLS ermitteln, der auf dem Weg des Pakets zu seinem Zielknoten liegt, und dieses dann dorthin weiterleiten. Müssen mehrere einkommende Pakete auf die gleiche Ausgangsverbindung gegeben werden, so übernimmt der CLS das Multiplexen.

Dieser Ansatz löst nun die größten Probleme des Indirekten Ansatzes. Bandbreite wird auch in den permanenten Verbindungen gut genutzt, da jede Verbindung durch die Versorgung mehrerer Kommunikationsbeziehungen ausgelastet werden kann. Da es im virtuellen CLS-Netz viel weniger Verbindungen gibt, kann man davon ausgehen, daß diese Auslastung auch erreicht wird. Die Skalierbarkeit ist dadurch gegeben, daß jede IWU nur eine Verbindung mit dem verbindungslosen Netz unterhalten muß, nämlich die zu ihrem nächsten CLS.

Daß der Direkte Ansatz die Probleme des Indirekten Ansatzes löst, bedeutet jedoch nicht, daß er auch dessen Vorteile aufrechterhält. Der Datentransport erfolgt jetzt auf einer höheren, auf die ATM-Schicht aufgesetzten Schicht. Dadurch ist mehr Verarbeitung im Netz nötig, und wir können uns nicht mehr auf die Qualitätsmerkmale verlassen, die uns die ATM-Schicht bieten kann. Für Übertragungen, die sehr lange dauern und hohe Anforderungen an die Bandbreite stellen, ist eine permanente IWU-IWU Verbindung, wie sie der Indirekte Ansatz benutzen würde, auf jeden Fall die bessere Wahl. Die Rahmenverarbeitung, die jeder CLS vornimmt, führt zudem an den Netzzugangspunkten zu einer Ende-zu-Ende-Verzögerung. Diese kann nur durch den Einsatz von sehr schnellen Servern vermindert werden. Dadurch steigen allerdings die Kosten pro CLS. Den Hauptanteil an der Rahmenverarbeitung im CLS hat die Wegewahl (Routing), die beim Direkten Ansatz von den Endsystemen ins Netz verlagert wurde. Da nun jeder CLS einen Teil der Wegewahl für ein Datenpaket durchführt, wird insgesamt nie der direkte Weg von IWU zu IWU eingeschlagen werden. Pakete, die mehr Links durchlaufen als nötig sind, belegen jedoch auch das Netz länger und erhöhen damit die Staugefahr. Außerdem geht durch unterschiedliches Routing die Reihenfolgetreue der Pakete verloren. Betrachtet man in Abbildung 53 die Plazierung der Verbindungslosen Server im Netz, so erkennt man, daß diese als zentrale Medien leicht zum Flaschenhals des Systems werden können.

Eine wichtige Voraussetzung für die Eindämmung dieser Gefahr ist die Fähigkeit der Server, auf einer Verbindung mehrere Kommunikationsanforderungen zu multiplexen. Dazu sollte der CLS allerdings auf der AAL 3/4[15] Dienstklasse aufbauen, was zu einer Inkompatibilität mit ATM-LANs führt, die AAL 5 benutzen.

Welchem der beiden Ansätze soll man nun den Vorrang geben? Für den Einsatz in WAN-Netzen hat sich die Verwendung des Direkten Ansatzes als vorteilhaft erwiesen. Wenn man annimmt, daß alle CLS-CLS Verbindungen permanent sind und jeweils wenigstens eine IWU-CLS Verbindung pro IWU besteht, so ist dieser Ansatz auch in der Verzögerung besser. Beim Indirekten Ansatz sind weder die Aufrechterhaltung permanenter IWU-IWU Verbindungen für jedes Paar von IWUs als auch der bedarfsgesteuerte Auf- und Abbau von SVC-Verbindungen in einem WAN-Netz tragbar. Zu diesen technischen Gründen kommt aus der Sicht der Netzbetreiber noch ein politischer Grund: Sie haben durch den Einsatz eines verbindungslosen Netzes auf der Basis des Direkten Ansatzes die Möglichkeit der Kontrolle des Datenaufkommens und der Daten. Das wird ihnen dadurch ermöglicht, daß das Netz selbst an der Verarbeitung der Kommunikationsdaten beteiligt ist, während diese beim Indirekten Ansatz transparent durch das Netz befördert werden.

3 Der Connectionless Server

In diesem Abschnitt rückt der verbindungslose Server als Bestandteil des Netzes in den Mittelpunkt der Betrachtung. Dabei werden zuerst zwei Ansätze zum Aufbau eines solchen Servers betrachtet, und danach werden Empfehlungen zur Plazierung der Server im Netz gegeben.

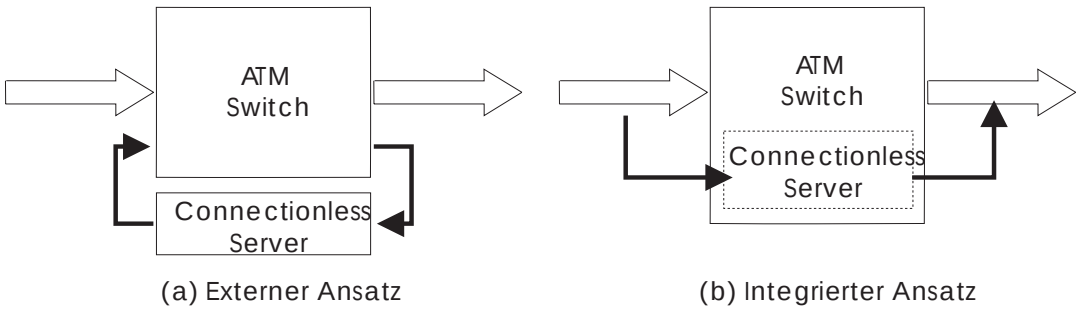

Abbildung 54. Aufbau eines Verbindungslosen Servers.

3.1 Externer Ansatz und Interner Ansatz

Bei die Realisierung des Connectionless Server als Bestandteil des Netzes werden zwei Arten unterschieden (Abbildung 54). Ist die Funktion zur Bereitstellung des verbindungslosen Dienstes in den ATM Switch integriert, so spricht man vom *Integrierten Ansatz.*

[15] AAL bedeutet ATM-Adaption-Layer. Diese Begriffe werden in späteren Abschnitten näher erläutert.

Wird dagegen eine Connectionless Server Einheit extern an den ATM Switch angeschlossen, bezeichnet man das als *Externen Ansatz.*

Beim externen Ansatz werden verbindungslose Daten von einem oder mehreren Switch-Ausgängen im externen CLS verarbeitet und dann zu einem oder mehreren Eingängen desselben Switches zurückgeleitet. Der Datenstrom wird also zweimal durch den Switch geschickt. Dabei entsteht eine Verzögerung und damit eine potentielle Staugefahr am Switch, was als Nachteil des externen Ansatzes zu sehen ist. Der integrierte Ansatz ist ohne Zweifel der elegantere der beiden. Hier fließen die verbindungslosen Daten zu einer speziell zu deren Verarbeitung vorgesehenen Einheit, die im Switch integriert ist. Von dort aus können die Daten dann direkt zum richtigen Switch-Ausgang weitergeleitet werden. Die Daten durchlaufen den Switch also nur einmal. Diese Lösung ist so einfach, elegant und leistungsfähig, daß sie eigentlich keine Nachteile haben sollte. Doch Nachteile müssen nicht immer technischer Natur sein. In diesem Fall sprechen politische Gründe gegen die Benutzung des externen Ansatzes. Die Integration der verbindungslosen Serverfunktion in den Switch ist nämlich Sache der Hardwarehersteller. Diese stehen der Unterstützung neuer und unerprobter Technologien aber naturgemäß zögernd gegenüber. Selbst wenn in diesem Fall die Hardwarehersteller die Vorstellungen der Normungsgremien sofort in Produkte umsetzten und integrierte Switches anböten, so bliebe immer noch der Nachteil der fehlenden Nachrüstmöglichkeit bestehender Switches. Für den Netzbetreiber bedeutet dies, daß er alle Switches, die als verbindungslose Server dienen sollen, gegen neue, gemäß dem integrierten Ansatz aufgebaute, austauschen müßte. Dies ist ein nicht unwesentlicher Kostenaspekt.

Der große Vorteil des externen Ansatzes ist also seine Flexibilität. CLS-Funktionalität kann leicht nachträglich zum Switch dazugefügt werden. Dieser Vorteil wiegt aus den genannten Gründen und trotz der technischen Eleganz des integrierten Ansatzes sehr stark.

3.2 Plazierung der Server

Die Plazierung eines verbindungslosen Servers kann nicht an jedem Switch des ATM-Netzes erfolgen; deshalb muß eine geeignete Auswahl getroffen werden. Diese kann einen wichtigen Einfluß auf die Leistung des verbindungslosen Dienstes haben.

In den meisten Fällen ist die Plazierung eines CLS am jeweils ersten Switch, der im Netz auf eine IWU folgt, zu empfehlen. Dadurch befinden sich mehr Verbindungen des Netzes zwischen verbindungslosen Servern, wo sie besser ausgelastet werden können. Auch das Management der CLS wird dadurch vereinfacht, da jeder Server mit der Adreßauflösung und der Kostenkalkulation für die an ihm angeschlossenen IWUs beauftragt werden kann. In Einzelfällen kann jedoch die Plazierung von Servern an zentralen Punkten innerhalb des Netzes bei gleicher Leistung deren Anzahl verringern und dadurch Kosten sparen.

4 Die Interna des Connectionless Server

Bis jetzt haben wir gesehen, wie ein Dienst aussehen muß, der uns, auf dem verbindungsorientierten ATM-Netz aufbauend, ein virtuelles verbindungsloses Netz zur Verfügung

stellt. In diesem Abschnitt geht es nun darum zu untersuchen, wie und wo dieser Dienst erbracht wird. Wenn es um die Betrachtung von internen Abläufen eines Kommunikationssystems geht, so ist damit immer die Untersuchung der verwendeten Protokolle und deren Einordnung in eine Schichtenarchitektur verbunden. In diesem Fall stehen die ATM-Schichten im Mittelpunkt. Um nicht zu viel voraussetzen zu müssen, wird die Funktionalität der ATM-Adaptionsschicht (AAL) hier nochmal soweit erklärt, wie sie für diesen Kontext relevant ist.

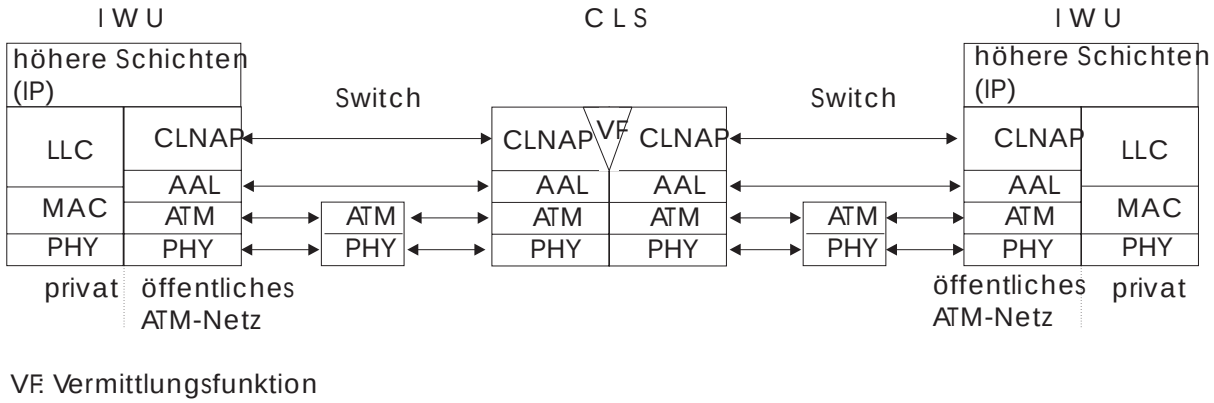

Abbildung 55. Protokollarchitektur.

4.1 Die Protokollarchitektur

In (Abbildung 55) ist die Protokollarchitektur dargestellt. Man beachte die drei Bestandteile, die nötig sind, um einen verbindungslosen Dienst bereitzustellen:

1. Verbindungslose Server, CLS
2. Endsysteme, IWUs
3. ein Protokoll, CLNAP (*Connectionless Network Access Protocol*)

Die IWUs beinhalten eine Ergänzung der Funktionalität der Connectionless Server. Sie müssen sowohl Zellen innerhalb des ATM-Netzes als auch MAC-Rahmen eines privaten Netzes versenden und empfangen können. Dazu unterhalten sie zwei Protokollstacks. Auf der einen Seite ist der Protokollstack des privaten LANs bis hin zur Netzwerkschicht implementiert; auf der anderen Seite, zum ATM-Netz hin, setzt sich ein zweiter Protokollstack aus den Schichten ATM, AAL, CLNAP zusammen. Sollen nun verbindungslose Daten aus dem privaten Netzwerk über unseren verbindungslosen ATM Dienst verschickt werden, so verwendet die IWU eine CLNAP-SDU zum Verpacken dieser Daten. Ein existierendes Protokoll der Netzwerkschicht (hier ist beispielhaft, aber nicht ganz realitätsfern, IP angegeben) sorgt dabei für die Umsetzung der Adresse des Zielhosts auf die CLNAP-Adresse der Ziel-IWU. Wie ein CLNAP in der Praxis aussehen könnte, wird am Ende dieses Kapitels an einem Beispiel demonstriert.

4.2 AAL Alternativen

Die AAL-Schicht *ATM Adaption Layer* ist in einem ATM System neben dienstspezifischen Aufgaben vor allem für die Segmentierung von Nutzdaten in die Länge des Nutzfeldes einer ATM Zelle sowie die Reassemblierung auf der Empfangsseite zuständig. Diese Mechanismen werden beim Netzzugang in den IWUs, aber auch, wie der nächste Abschnitt zeigen wird, innerhalb des CLS benötigt. Der dienstspezifische Teil der AAL-Schicht hat zur Unterscheidung von 4 AAL-Typen geführt. In diesem Zusammenhang interessieren wir uns nur für die Typen *AAL 3/4* sowie *AAL 5*. Ein wichtiges Unterscheidungsmerkmal der beiden ist der MID-Wert (*Multiplexing ID*), den nur AAL 3/4 in seiner PDU unterstützt. Wie der Name schon andeutet, unterstützt AAL 3/4 mithilfe dieses Wertes das Multiplexen von Zellen verschiedener Pakete über eine VPI/VCI-Verbindung. Dabei haben jeweils die Zellen, die zu einem bestimmten Paket gehören, auch denselben MID-Wert. AAL 5 unterstützt im Gegensatz dazu kein Multiplexing auf Zellenbasis (*cell interleaving*) über eine VC-Verbindung. Will man ein solches Verfahren hier anwenden, so muß jede Zelle eines Pakets auf einem eigenen VC-Kanal verschickt werden, wobei dann der VCI-Wert als Ersatz des fehlenden MID-Wertes betrachtet werden kann. Stellt man sich die einzelnen VCs bildlich als kleine Rohre innerhalb eines größeren VP-Rohres vor, so gleicht diese Methode also einem Raummultiplex. In öffentlichen WAN-Netzen ist das nicht anwendbar. Man müßte dort viele VPs permanent reservieren, um jederzeit genügend VCs für Multiplexing bereitstellen zu können, oder man müßte dauernd bedarfsgesteuert Verbindungen auf- und abbauen. Für WANs sind beide Optionen aus den Gründen, die schon in Abschnitt 2.1 gegen die Indirekte Lösung sprachen, ungeeignet. AAL 5 kann natürlich trotzdem mehrere Datenströme über einen VC-Kanal multiplexen, aber nur auf Rahmenbasis (frame interleaved). Wie wir in Abschnitt 4.3 sehen werden, ist diese Einschränkung manchmal nicht hinnehmbar. Der Vorteil von AAL 5 ist seine Effizienz und die Kompatibilität einer auf ihr basierender Lösung. Eine gegenüber AAL 3/4 größere Effizienz erzielt AAL 5 dabei einerseits aus seiner besseren Nutzdatenrate. Während sie bei AAL 3/4 44 Byte beträgt, kann AAL 5 hier 48 Byte transportieren. Andererseits sind die Nutzdaten bei AAL 5 auch besser plaziert. Die Ausrichtung erfolgt hier an einer 4 Byte Grenze gegenüber der 2 Byte Grenze, an der bei AAL 3/4 ausgerichtet wird. Wenn es um Kompatibilität geht, ist AAL 5 klar im Vorteil. Es wird in ATM-LANs verwendet und von vielen neuen Entwicklungen unterstützt. Ein Beispiel dafür ist die neue Fibre Channel Technik, die SCSI ablösen soll.

4.3 Vermittlungsmethoden

Neben den verschiedenen Arten des Aufbaus eines verbindungslosen Servers, wie sie in Abschnitt 3.1 besprochen wurden, ist das Vorgehen bei der Nachrichtenweiterleitung das wichtigste Unterscheidungsmerkmal bei der Betrachtung einer Server-Implementierung. Ein verbindungsloser Server kann im allgemeinen zwei Betriebsarten unterstützen:

- Nachrichtenvermittlungs-Modus (*Reassembly Mode* oder *Message Mode*)
- ATM-Zellenvermittlungs-Modus (*Streaming Mode*)

Im Nachrichtenvermittlungs-Modus werden ankommende Zellen zunächst zu kompletten Rahmen zusammengesetzt. Erst wenn alle Zellen einer Nachricht korrekt empfangen sind, erfolgt die erneute Segmentierung und Weiterleitung. Dagegen werden im Zellenvermittlungsmodus alle korrekt empfangenen Zellen direkt weiter an das Zielsystem verschickt.

Zellenvermittlung, Streaming-Mode Da man im Zellenvermittlungsmodus einerseits kaum erwarten kann, daß die Zellen eines Rahmens immer direkt hintereinander bleiben, andererseits aber auch keine Möglichkeit besteht, diese Ordnung in einem Zwischensystem wiederherzustellen, ist man dort in hohem Maße auf Multiplexing angewiesen. Zieht man noch einmal die Ausführungen aus Abschnitt 4.2 in Betracht, so wird klar, daß man beim Zellenvermittlungsmodus die AAL 3/4 Schicht verwenden wird. Die Funktion des Servers läßt sich dann wie folgt beschreiben: Sobald die erste Zelle eines neuen Rahmens eintrifft, wird deren VPI/VCI/MID Umsetzung durch Routing ermittelt und intern im Server abgelegt. Nachfolgende Zellen des gleichen Rahmens können nun ohne Routing-Verzögerung auf diese Umsetzungsinformation zugreifen. Für diese nachfolgenden Zellen ist der Durchlauf durch den CLS mit dem Durchlauf durch einen ATM-Switch vergleichbar - mit dem Unterschied, daß die interne Adreßumsetzungsinformation im CLS durch die Information in der ersten Zelle eines Rahmens festgelegt wurde und nicht durch die Verbindungsaufbauphase. Außerdem ist bei dieser Analogie noch darauf zu achten, daß im CLS auf der AAL 3/4-Schicht und im ATM-Switch auf der ATM-Schicht gearbeitet wird. Trotzdem wird durch den Vergleich deutlich, daß die Weiterleitung in einem CLS kaum schneller durchgeführt werden kann als mit dem Zellenvermittlungsmodus. Es mag absurd klingen, aber diese hohe Geschwindigkeit wird hier sogar zum Problemfaktor. Die Zeit, die ein verbindungsloser Server im Zellenvermittlungsmodus für die Adreßumsetzung aus den Daten der ersten Zelle hat, entspricht nämlich gerade der Zellübertragungszeit. Diese beträgt im *günstigsten* Fall (beim Einsatz von SONET STS-3c mit 155,52 Mbit/sec) jedoch nur 2,7 μsec. Hier ergeben sich Geschwindigkeitsanforderungen, die schon beim Entwurf berücksichtigt werden müssen und den Einsatz von Hochgeschwindigkeits-Schaltungen sowie Parallelisierung nötig machen. Diese Probleme kann man, wenn auch mit hohem Kostenaufwand, in den Griff bekommen. Ein Nachteil des Zellenvermittlungsmodus bleibt aber. Der Modus hat keine Möglichkeit der Fehlerkontrolle auf Rahmenbasis. Ist ein Rahmen fehlerhaft, so stellt das erst das Endsystem fest. Dies führt zur unnötigen Belastung des ATM-Systems durch die Übertragung von vornherein unbrauchbaren Zellen. Schließlich ergibt sich aus dem AAL 3/4 PDU-Format noch ein weiteres Problem. Dort sind nur 10 bit für den MID-Wert vorgesehen. Das entspricht der Möglichkeit, 1024 Rahmen gleichzeitig zwischen zwei Knoten befördern zu können. Da nun aber beim Zellenvermittlungsmodus der gesamte Rahmen zwischendurch nie zusammengebaut wird, können dessen Zellen sehr weit auseinandergezogen werden. Ein einzelner Rahmen ist also länger unterwegs, bis er komplett am Ziel ist und belegt damit auch länger MID-Werte. Diese Werte können deshalb, bei entsprechender Entfernung der CLS-Knoten voneinander, sehr schnell knapp werden. Steht einem Rahmen für eine Verbindung kein MID-Wert mehr zur Verfügung, so muß er verworfen werden. Das geschieht ungeachtet der Bandbreite, die vielleicht noch in ausreichendem Maße zur Verfügung steht.

Nachrichtenvermittlung, Reassembly-Mode Die Fehlererkennung auf der Ebene von ganzen Rahmen ist die Stärke der Nachrichtenvermittlung. Beschädigte oder verlorene Zellen können sofort die Verwerfung des gesamten Rahmens auslösen, wodurch unnötige Übertragungen eingespart werden. Es ergeben sich aber noch weitere Vorteile. So kann jetzt zur Übertragung AAL 5 eingesetzt werden, woraus sich die aus Abschnitt 4.2 bekannten Vorteile ergeben, besonders in Verbindung mit dem Einsatz in ATM-LANs. Bei der Leistung stellt der Nachrichtenvermittlungsmodus vergleichsweise geringe Anforderungen an den Connectionless Server. Die Adressauflösung und das Routing kann hier mit dem Empfang der ersten Zelle eines Rahmens beginnen und dann parallel zum Empfang der restlichen Zellen durchgeführt werden. Da die Rahmen zusammenhängend übertragen werden[16] wirken sich Störungen des Übertragungskanals, die meistens mehrere aufeinanderfolgende Zellen betreffen, nur auf wenige Rahmen aus. Ungeachtet dessen sind bei der Nachrichtenvermittlung inhärent Nachteile vorhanden. Diese betreffen vor allem die dem Datenfluß auferlegte Verzögerung. Man unterscheidet drei Arten:

- Die *Empfangsverzögerung* kommt dadurch zustande, daß die erste Zelle eines Rahmens erst nach dem Empfang der letzten weitergeleitet werden kann. Hier ist die Verzögerung also *mindestens* so groß, wie die Übertragungsdauer des gesamten Rahmens.

- Dazu kann sich noch die *Warteschlangenverzögerung* addieren. Sie entsteht, wenn ein übertragungsbereiter Rahmen warten muß, bis die Warteschlange seines Ausgangskanals leer ist.

- Die *Verarbeitungsverzögerung* schließlich wird durch die Verarbeitungsschritte verursacht, die der Server auf den Daten durchführt. Dazu gehören: Puffer-Verwaltung, Adreßauflösung und Fehlerbehandlung. Die dabei verursachte Verzögerung ist von der Speicherbandbreite und Verarbeitungsleistung des Servers abhängig und kann, wie oben bereits erwähnt, durch Nebenläufigkeit von Verarbeitung und Empfang stark minimiert werden.

Zu beachten ist dabei, daß *jede* dieser Verzögerungen an *jedem* CLS-Knoten erneut auftritt. Auf einem Pfad durchs Netz wächst die Verzögerung somit proportional zur Anzahl der verbindungslosen Server darauf. In den beiden letzten Punkten obiger Aufzählung klingt auch schon die Bedeutung der Pufferverwaltung an. Sowohl die Größe der Puffer als auch deren Verwaltung kann im Nachrichtenvermittlungsmodus zum entscheidenden Leistungs- und Kostenfaktor werden. Bei ungenügender Beachtung dieses Aspektes kann es zu Pufferüberläufen und damit zum Verlust von Zellen kommen. Dann hat man sich mit dem verbindungslosen Server eine weitere potentielle Fehlerquelle ins Netz geholt.

[16] Die zusammenhängende Übertragung von Rahmen muß bei der Nachrichtenvermittlung nicht immer angewendet werden. Es gibt auch andere Interleaving-Techniken, bei denen dann aber die Vorteile der zusammenhängenden Übertragung verloren gehen. Die Interleaving-Techniken werden in [BHS93] vorgestellt.

4.4 Adreß-Auflösung

Die Adreß-Auflösung macht die Bereitstellung eines verbindungslosen Dienstes in einem Weitverkehrs-ATM-Netz noch komplizierter. Das liegt daran, daß für die Kommunikation von zwei Hostrechnern, die sich jeweils in einem lokalen Netz befinden, erst die IWUs gefunden werden müssen, über die die lokalen Netze und mit ihnen die Hostrechner an das ATM-Netz angebunden sind. Diese IWUs können dann im ATM-Netz als Stellvertreter für ihre angeschlossenen Hostrechner dienen.

Es gibt zwei Techniken für die Adress-Auflösung unter diesen Umständen: *ARP Multicast* und *ARP Server*. Beide sind dem klassischen ARP-Protokoll, wie es im Internet benutzt wird, ähnlich.

Das ARP Multicast kommt aus dem LAN-Bereich. Die Adresse des Zielsystems wird hier an alle IWUs im ATM-Netz verschickt, dann wartet das Protokoll auf eine Antwort. Hier wird bereits die Existenz eines ATM-Multicast Dienstes benötigt. (Dieses Verfahren nutzt also bereits den verbindungslosen Dienst, zu dessen vollständiger Realisierung es beiträgt.) Eine IWU, die eine an sie gerichtete Anfrage nicht beantworten kann, schickt ihrerseits eine lokale ARP Anfrage an alle bei ihr angeschlossenen LANs.

Beim ARP Server Verfahren werden zentralisierte oder verteilte Datenbanken in Servern zur Adreß-Auflösung befragt. In diesen Datenbanken müssen sich neu ans Netz kommende IWUs anmelden. Jede IWU muß also mindestens einen ARP Server kennen.

Vergleicht man nun beide Verfahren miteinander, so fällt auf, daß beim ARP Multicast das Versenden von Anfragedaten an eine große Anzahl von IWUs, von denen nur *eine* die Daten beantworten kann, sehr ineffizient ist. Beim Einsatz der ARP Server reduziert sich zwar das Verkehrsaufkommen auf dem Netz, aber die Server müssen sich um Speicherung und Verwaltung der Adressen kümmern und stellen obendrein ein großes Sicherheitsrisiko dar. So kann der Ausfall eines Servers einen ganzen Netzzweig lahmlegen.

4.5 CLIP, ein Beispiel für ein CLNAP-Protokoll

Schaut man sich nochmal unseren verbindungslosen Dienst anhand der Protokollarchitektur in Abbildung 55 an, so gibt es noch eine Lücke in den bisherigen Betrachtungen. Dies ist das CLNAP-Protokoll. Wir wissen zwar, daß es zur Einkapselung von verbindungslosen Daten benutzt wird, haben aber über den internen Ablauf noch nichts erfahren. In diesem Abschnitt soll zu diesem Zweck ein Beispielprotokoll vorgestellt werden, das als CLNAP-Protokoll dienen kann: das CLIP-Protokoll (*Connectionless Interworking Protocol*). Dabei soll sich die Betrachtung an dieser Stelle darauf beschränken, anhand bestimmter Bestandteile von CLIP allgemeine Entwurfsgrundsätze für ein solches Protokoll darzustellen. CLIP wird ausführlich in [BHS93] behandelt.

CLIP wird von IWUs und CLS benutzt, um verbindungslose Datenpakete effizient einzukapseln und zu übertragen. Wie aus der Abbildung 55 ersichtlich ist, setzt CLIP auf der AAL-Schicht auf. Damit fügt es also zwangsläufig einen größeren Protokolloverhead zum ATM-Netz dazu, da dieses gewöhnlich direkt über die AAL-Schicht angesprochen wird. Das Protokoll muß somit von Anfang an für die hohen Geschwindigkeitsanforderungen in einem ATM-Netz entworfen sein. Bei CLIP wurden dazu folgende Festlegungen gemacht:

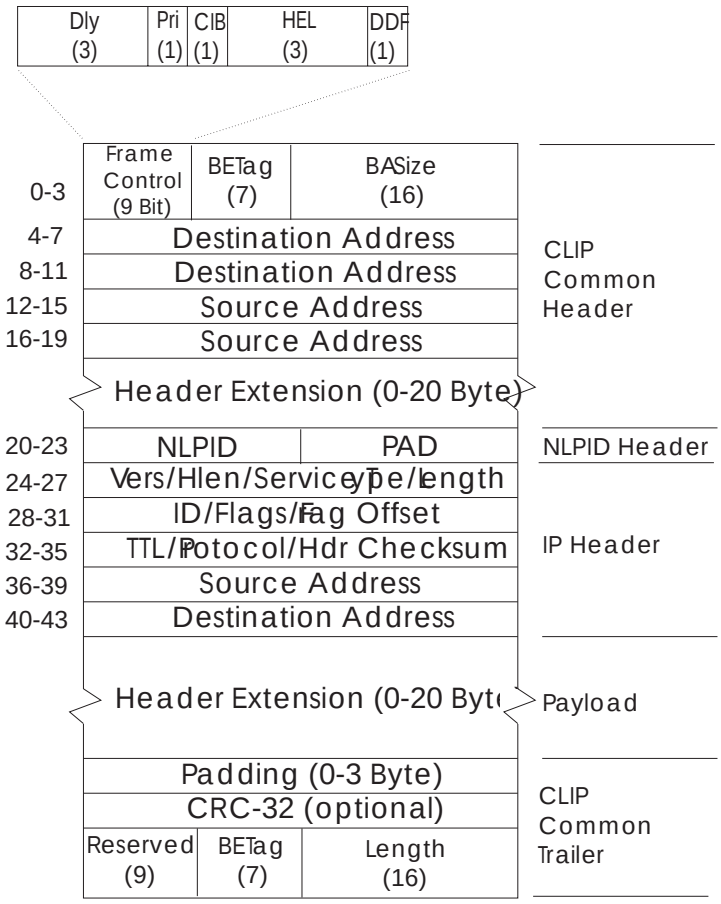

Abbildung 56. CLIP SDU-Format.

- festes Paketformat
- Protokolle mit einfachen Zustandsautomaten
- Protokollverarbeitung muß komplett in Hardware möglich sein, ohne die Notwendigkeit zusätzlicher Softwareverarbeitung

CLIP SDU-Format Dieses Format ist in Abbildung 56 dargestellt. Man beachte, daß hier *wichtige Felder des eingekapselten Protokolls in der ersten Zelle eines Rahmens plaziert* werden können. In Abbildung sieht man das am Beispiel eines eingebetteten IP-Rahmens, wo die komplette Adressinformation des Absenders und des Empfängers in der ersten Zelle auftauchen. Diese Tatsache kann von allen Komponenten, die das CLIP-Protokoll verwenden, mit Gewinn genutzt werden. Zwischensystemknoten können Zellen bei Bedarf aufgrund von Informationen aus höheren Protokollschichten verarbeiten, und ein CLS im Zellenvermittlungsmodus würde ohne die kompletten Adreß-Informationen in der ersten Zelle gar nicht funktionieren. Aber auch CLSs im Nachrichtenvermittlungsmodus profitieren davon, da sofort nach dem Empfang der ersten Zelle die Adreß-Auflösung und Umsetzung parallel zum Empfang weiterer Zellen beginnen kann. Dieser Prozeß ist dann mit großer Sicherheit beim Empfang der letzten Zelle beendet, und es tritt nicht die aus Abschnitt 4.3 bekannte Verarbeitungsverzögerung auf. Die Plazierung der kompletten Adreß-Information und anderer Felder des eingekapselten Protokolls ist also ein

sehr wirksames Mittel zur Leistungssteigerung des gesamten verbindungslosen Servers.

Von den einzelnen Feldern des Paketformats soll hier nur noch die Funktion des DDF (*Deliver Damaged Frame*) Feldes erwähnt werden. Dieses Feld wird auf 1 gesetzt, um Rahmen zu kennzeichnen, die auch mit vermißten oder beschädigten Zellen noch transportiert werden sollen. Wie sich dieses Feld auf den Protokollablauf auswirkt, wird weiter unten gezeigt. Für die Bedeutung der restlichen Felder kann auf die Korrespondenz zwischen dem CLIP-Format und dem 802.6 Rahmenformat verwiesen werden.

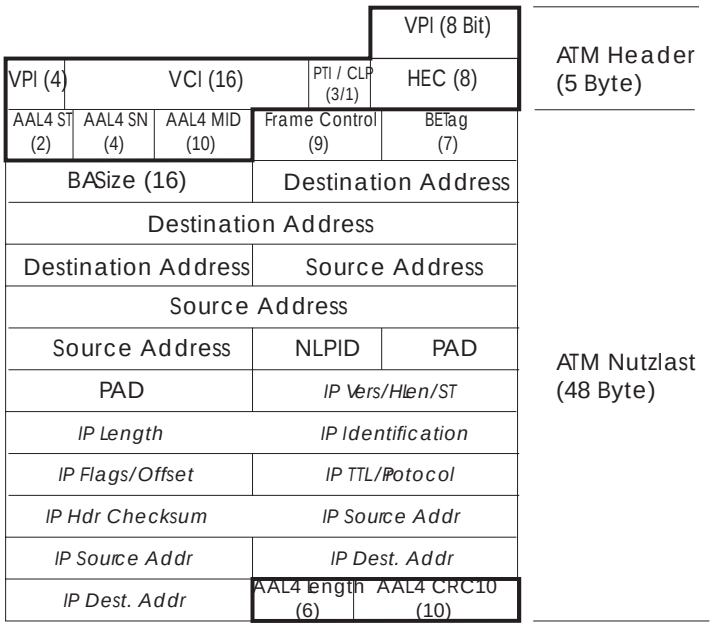

Abbildung 57. CLIP PDU-Format.

CLIP PDU-Format CLIP benutzt bei der Generierung seiner PDU die AAL 3/4 Schnittstelle zum ATM-Netz. Abbildung 57 zeigt außerdem, daß hier trotz der geringeren Nutzdatenrate von AAL 3/4 gegenüber AAL 5 die komplette Adressierungsinformation wie gewünscht in der ersten Zelle untergebracht werden kann.

CLIP Protokoll-Ablauf Abbildung 58 zeigt den Protokoll-Ablauf am Beispiel des Rahmenempfangs. Den eingangs erwähnten Entwurfsentscheidungen folgend ist dies ein einfacher endlicher Automat. Die Übergänge bestimmen sich aufgrund der Untersuchung von zwei festen Positionen im Kopf des AAL-Rahmens und des DDF-Feldes in der ersten Zelle eines CLIP-Rahmens. Einer Realisierung in Hardware steht bei dieser Einfachheit nichts im Wege. Der Protokollablauf läßt sich folgendermaßen zusammenfassen: Der verbindungslose Server gewinnt die Adreß-Information eines Rahmens aus dessen BOF (*Begin of Frame*) Zelle. Der Inhalt des DDF-Feldes dieser Zelle entscheidet nun darüber, ob aus dem Ruhezustand `IDLE` in den Zustand `RECV_DDF` oder `RECV` übergegangen wird. Diese und darauffolgende Zellen mit dem selben MID-Wert werden nun an einen Ausgang weitergeleitet. Sobald ein Fehler in Form eines empfangenen COF (*Continuation*

of Frame) mit falscher Sequenznummer eintritt, hängt das weitere Vorgehen des Protokolls vom Empfangszustand ab. Befindet es sich im `RECV` Zustand, führt eine falsche Sequenznummer zur Verwerfung des gesamten Rahmens, während im `RECV_DDF` Zustand dieser Fehler ignoriert wird. (Eine fehlende Zelle wird später im Endsystem durch eine Null-Zelle ersetzt.)

Dieser endliche Automat ist nur ein Teil des gesamten Protokollgeschehens, zeigt aber trotzdem, wie einfach die Teile aufgebaut werden müssen, um die Leistungsanforderungen zu erfüllen.

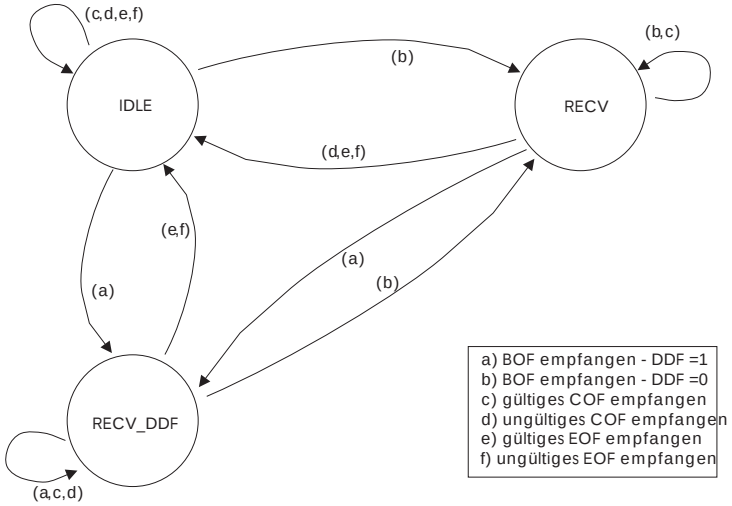

Abbildung 58. CLIP Empfangs-FSM.

Zusammenfassung Beim CLIP-Protokoll sind die Leistungsanforderungen durch die Plazierung wichtiger Routing-Informationen in der ersten Zelle eines Rahmens, durch Benutzung der AAL 3/4-Multiplexingklasse und durch einfache Protokollmechanismen berücksichtigt.

5 Schlußbetrachtung

Bei dem Versuch, verbindungslose Dienste in ein Weitverkehrs-ATM-Netz zu integrieren, stößt man sehr schnell an Grenzen. Man muß erkennen, daß viel von der Eleganz, mit der die beeindruckende Leistungsfähigkeit erreicht wird, durch die verbindungsorientierte Auslegung des ATM möglich wird.

Ein verbindungsloser Dienst wird hier immer nur ein Aufsatz sein, der nicht so recht ins Bild passen will. Die statische Zuordnung und Reservierung von virtuellen Kanälen (VCs) oder ganzen virtuellen Pfaden (VPs) für eine Verbindung ermöglicht Dienstgütemerkmale, die spielend Anforderungen moderner Multimedia-Anwendungen befriedigen können. Der effiziente Aufbau eines verbindungslosen Dienstes auf der Grundlage dieser virtuellen Kanäle und virtuellen Pfade gestaltet sich allerdings sehr schwierig. Es wird immer um das Problem gehen: entweder man reserviert reichlich Kanäle, um die

Dienstgütemerkmale gewährleisten zu können, oder man geht mit Kanälen sparsam um, um die Skalierbarkeit des ATM nicht zu gefährden. Einmal bedeutet das den verschwenderischen Umgang mit Bandbreite, das andere Mal den zu spärlichen Umgang damit. Dies kann wiederum zu Staugefahr und Datenverlusten aufgrund von Pufferüberläufen führen. Will man beide Gefahren etwas mindern, entscheidet man sich meist für die Verwendung von AAL 3/4 und damit die Ausnutzung der zellenbasierten Multiplexing-Funktionalität von ATM. Doch auch hier setzt schon wieder heftige Kritik an. In nahezu allen modernen verbindungslosen Erscheinungsformen der ATM-Technik wird diese auf der Grundlage der AAL 5-Dienstklasse benutzt. Eine Technik, die auf AAL 3/4 aufbaut, befindet sich damit von vornherein in einer Sonderrolle, die leicht zur Außenseiterrolle werden kann, auf jeden Fall aber viel Umsetzungsaufwand zwischen den Techniken mit sich bringt.

Ganz wird man die Kritiker wohl nie befriedigen können. Wenn man sich mit den hier vorgestellten Techniken einen verbindungslosen Dienst zusammensetzt, so kann es immer nur darum gehen, die Vor- und Nachteile der verschiedenen Entwurfsentscheidungen abzuwägen. Die ideale Technik, die nur Vorteile in sich vereint, kann daraus nicht abgeleitet werden. Der Grund für dieses Dilemma liegt in der Sache selbst begründet: Ein Benutzer eines verbindungslosen Dienstes ist immer gezwungen, seine Daten ab einem gewissen Punkt auf der Übertragungsstrecke sich selbst zu überlassen. Von dort an müssen sie sich ihren Weg zu Ziel eigenständig suchen. Das macht den Dienst eben gerade verbindungslos. In den klassischen Netzen stellt dieses Verfahren eine Erleichterung für die Endsysteme dar, bei ATM führt es dagegen ins Dilemma. Einerseits möchte das Endsystem die Routingaufgabe ans Netz übertragen, andererseits möchte es aber weiterhin die Kontrolle über die Dienstgütemerkmale der Übertragung (z.B. isochron) behalten.

Die Schwierigkeiten bei der Auswahl der geeigneten Technik für die Bereitstellung eines verbindungslosen Dienstes über ATM-WANs schlagen sich auch in der zähen Art und Weise nieder, mit der die Standardisierung in manchen Bereichen voranschreitet. So gibt es bis jetzt noch keine Einigung über das Adreß-Auflösungsverfahren, den Vermittlungsmodus im CLS und dessen geeignete Plazierung im Netz. Trotzdem ist die Unterstützung für verbindungslose Kommunikation entscheidend für den Erfolg von ATM.

Kreditbasierte Flußkontrolle in ATM-Netzen

Markus Mäder

Kurzfassung

Dieser Artikel faßt die technischen Grundlagen der kreditbasierten Flußkontrolle zusammen, inklusive der Vor- und Nachteile dieser Technik. Im zweiten Teil wird die flußkontrollierte Link zu Link Verbindung als Lösungsansatz zur Adaptierung dieser Technik an die Anforderungen von ATM-Netzwerken vorgestellt.

1 Prolog

Der Übergang zu den Hochgeschwindigkeitsnetzen stellt enorme Forderungen an die zur Übertragung zuständigen Kommunikationssubsysteme. Geht man von einer mittleren Paketlänge von 256 Byte aus und stehen bei den derzeit im Weitverkehrsnetz üblichen Datenrate von 64 kBit/s zur Bearbeitung jedes Pakets noch ca. 32 ms zur Verfügung, so sind es bei einer projektierten Datenrate von 500 Mbit/s nur noch $4\mu s$. Diese erheblich geringere Bearbeitungszeit hat Konsequenzen, die das ganze Systemprinzip betreffen.
Mit der enormen Leistungsfähigkeit glasfaserbasierter Kommunikationsnetze verschiebt sich der Engpaß vom Übertragungsmedium hin zu den an der Datenübertragung beteiligten aktiven Elementen. Um diese hohen Datenraten auch oberhalb der Schichten eins bis vier im OSI-Basisreferenzmodell verfügbar zu machen, gibt es noch viel zu tun. Primär werden derzeit folgende Ansätze verfolgt:

1. schnellere Erbringung der Protokollfunktionen durch den Einsatz hochintegrierter Spezialbausteine [ABBY89, KKS87, BY92] sowie die Verwendung paralleler Architekturen zur Ausnutzung potentiell paralleler Protokollstrukturen bzw. Piplinestrukturen zur zeitlich überlappten Bearbeitung von Paketen in einem Protokollturm
2. Entwurf spezieller, auf die Umgebung von Hochgeschwindigkeitsnetzen zugeschnittener Protokolle [Che89, SN89, DDK+90]

Die nachfolgenden Kapitel beschäftigen sich mit dem Problemkreis 2, der Entwicklung eines kreditbasierten Flußkontrollmechanismusses für den Einsatz in ATM-Netzwerken.

2 Die Flußkontrolle

Zur Illustration der Aufgabe einer Flußkontrolle betrachte man die Minimalkonfiguration einer Datenübertragung, zwei direkt miteinander verbundene Endsysteme. Ohne die Kontrolle während der Datenübertragung besteht die Möglichkeit, daß der Sender Daten mit einer deutlich höheren Geschwindigkeit überträgt, als sie vom Empfänger verarbeitet

werden können. Die Puffer beim Empfänger würden überlaufen, was zum Verwerfen von neu ankommenden Paketen führt. Solche Verluste sind durch geeignete Maßnahmen zu kompensieren (d.h. die betroffenen Paketen werden nochmals übertragen), was zu einer zusätzlichen Belastung des Kommunikationssystems führt.
Die **Flußkontrolle** dient dem Schutz der beim Empfänger vorhandenen Ressourcen vor Überlastung. Sie stellt damit einen bilateralen Kontrakt dar, wobei beide Teilnehmer in der Regel zur Kooperation bereit sind. Aus der Beschränkung auf zwei Partner wird auch unmittelbar ersichtlich, das jedes Paar im Prinzip unterschiedliche Verfahren zur Flußkontrolle verwenden kann. Die zur Zeit zum Einsatz kommende Realisierung ist zumeist die Fenstertechnik. Der Vollständigkeit halber sei an dieser Stelle noch die **Lastkontrolle**[17] erwähnt. Ihre Aufgabe ist es, die in den Netzwerkkomponenten vorhandenen Ressourcen vor einer Überlastung schützen. Es handelt sich damit um ein multilaterales Abkommen, daß alle Teilnehmer des Netzes umfaßt.

2.1 Grundkonzept der Fenstertechnik

Jedes zu verschickende Paket erhält eine Nummer und wird in dieser Reihenfolge auch abgeschickt[18]. Wenn der Adressat die Pakete fehlerfrei empfangen hat, schickt er eine positive Rückmeldung, die die Nummer des zuletzt fehlerfrei empfangenen Pakets enthält. Die Anzahl der Pakete, die verschickt werden dürfen, ohne daß eine Quittung (**ACKnowledgement**) zurückgekommen ist, bezeichnet man als die Größe des Flußkontrollfensters oder einfach als **Kredit** (siehe Bild 59). Ein Fenster ist eine geordnete Folge von aufeinanderfolgenden Sendelaufnummern N(S). Die Ordnung der Sendelaufnummer ergibt sich aus der Modulonumerierung[19] (Modulo n). Der Wert des sogenannten **unteren Fensterrands** entspricht dem Wert der zuletzt empfangenen Paketlaufnummer beim Empfänger bzw. die zuletzt vom Empfänger quittierte Paketlaufnummer beim Sender, abhängig davon, welches Fenster man betrachtet. Die Abbildung 59 macht dies deutlich. Mit der Empfangsfolgenummer N(R) werden alle Datenpakete bis zur Sendelaufnummer $N(S) = N(R) - 1 \bmod n$ quittiert.
Der Fenstermechanismus legt nun fest, daß der Sender nur die Datenpakete bis Sendelaufnummer $N(S) = N(R) + W - 1 \bmod n$ senden darf. Er dürfen also maximal W Datenpakete unquittiert ausstehen (Sendekredit).
In der Abbildung 59 wird dargestellt, wie sich das Sender-Fenster durch Empfang einer Quittung (ACK) öffnet und durch das Senden von $W = 4$ Paketen wieder schließt (da keine neuen Quittungen eingehen). Um das Fenster wieder zu öffnen, und damit dem Sender zu ermöglichen, weitere Datenpakete zu senden, muß der Empfänger den unteren

[17] Hier ist es wichtig anzumerken, daß diese Definition nicht überall so eindeutig und sauber verwendet wird. Vor allem in den englischen Artikel aus dem ATM-Forum fiel mir auf, daß der Begriff Lastkontrolle nicht benutzt wurde (siehe dazu [KM95, KC94a, KBC94c, KC94c, Cal87] ...). Die Aufgabe der Flußkontrolle schlossen dort auch die Aufgaben der Lastkontrolle mit ein. In den deutschen Veröffentlichungen zog man dagegen eine klare Grenze zwischen Last- und Flußkontrolle (siehe die Dissertation [Ros95]).

[18] ATM sichert die Zellreihenfolge in VCC.

[19] Dabei ist zu beachten, daß beispielsweise bei Modulo $n = 8$ auf Sendelaufnummer $n - 1 = 7$ als nächste Laufnummer die 0 folgt, und daß der maximal mögliche Kredit $n - 1 = 7$ Paket beträgt.

Fensterrand durch die Übermittlung einer Quittung (ACK) mit einer Empfangsfolgenummer verschieben (hier mit $N(R) = 3$ geschehen).

Zur Reduzierung der Anzahl der Quittungen (ACK) kann dabei eine kumulative Quittierung verwendet werden. Der Erhalt einer Quittung mit der Folgenummer j bestätigt in diesem Fall den korrekten Empfang aller Pakete bis einschließlich des Paketes j (**Inclusive AcKnowledgement/Implizite Bestätigung**). Durch die Zurückhaltung von Quittungen kann der Empfänger schließlich dafür Sorge tragen, daß der Sender keinen neuen Kredit erhält. Dieser darf nur noch den bereits gewährten Kredit ausschöpfen und muß danach die Übertragung weiterer Daten unterbrechen. Auf diese Weise kann der Empfänger Paketverluste aufgrund fehlender lokaler Pufferspeicher unterbinden. Die Quittung (ACK) kann geschickt werden nach jedem empfangenen Paket, nachdem der komplette Kredit verbraucht wurde oder nach einem beliebigen Wert (der allerdings kleiner sein muß als der Kredit). Der Kredit sollte ausreichend bemessen sein, so daß der Sender kontinuierlich bis zum Eintreffen der ersten Quittung Daten übertragen kann. Diese Zeitspanne, auch als **Umlaufzeit** bezeichnet, setzt sich aus folgenden drei Komponenten zusammen:

1. der **Verarbeitungszeit** der Pakete innerhalb der an der Übertragung beteiligten Systeme,

2. der **Wartezeit** in eventuell vorhandenen Warteschlangen, die sich bei temporären Überlastungen eines Systems ausgebildet haben können, und

3. der physikalischen **Signallaufzeit**.

Die Größe des notwendigen Kredits (und somit die Pfadkapazität) bestimmt sich dann zu:

$$Pfadkapazität \approx Umlaufzeit * Übertragungsrate \quad (1)$$

Die Übertragungsrate ist dabei gleich dem langsamsten System zu setzen, das auf dem Pfad vorhanden ist. Eine Datenübertragung mit höherer Rate würde lediglich zum Stau von Paketen an diesem System führen und so eine unnötige Netzbelastung verursachen. Noch ein Wort zur Fehlerbehebung: Fehlerhafte Pakete werden verworfen und dadurch genauso behandelt wie nicht empfangene Pakete. Je nach festgelegten Fehlerbehebungsmechanismus werden Pakete entweder gar nicht, nur nach Aufforderung oder nach einem Time Out wiederholt.

2.2 Sliding Window

Sliding Window ist der Fensterkontrollmechanismus, bei dem nach jedem bzw. jedes Paket quittiert wird, das Fenster wird also immer um eine Einheit weiter geschoben [Teb93]. Allgemein versteht man aber unter Slidung Window den Oberbegriff für die Familie der kreditbasierten Flußkontrolltechnik.

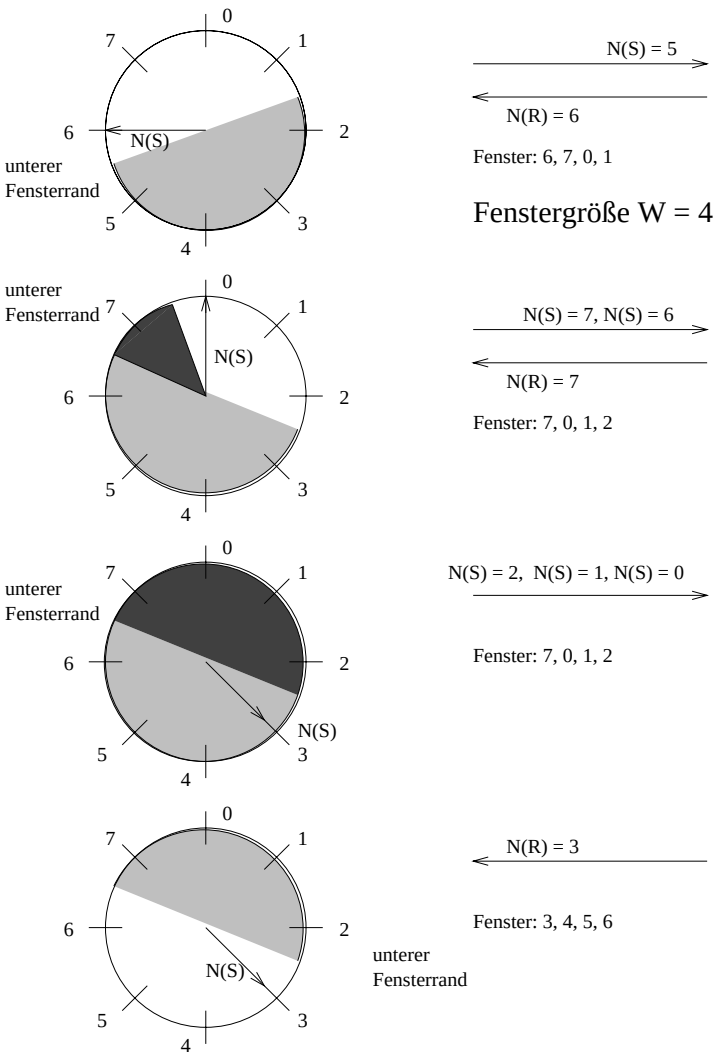

Abbildung 59. Flußkontrolle mit Sliding Window-Technik

2.3 Acknowledge at end of window

Der Kredit wird hier erst erneuert, wenn er vollständig verbraucht (at end of window) wurde. Es werden weniger Rückmeldungspakete verlangt als beim Sliding Window Mechanismus, was den Nutzdaten-Durchsatz vergrößert.

2.4 Probleme der kreditbasierten Flußkontrolle

Aufgrund des Fehlens einer maximalen Senderate entstehen im Rahmen der Fenstertechnik sehr leicht plötzliche Lastspitzen durch die Freigabe einer großen Datenmenge, was unter Umständen zum Stau großer Datenmengen an Engpässen führen kann. Gründe für diese Lastspitzen sind:

- beim Start einer Verbindung steht das gesamte Fenster als Kredit zur Verfügung oder
- eine Fehlerbehebung (z.B. mit dem Go-back-N Verfahren).

Zum besseren Verständnis dieser Problematik betrachte man folgende Situation (vergleiche dazu Bild 60):

Eine Datenübertragung findet über einen Pfad mit einer Kapazität von 100 Paketen statt (der Kredit beträgt also 100 Pakete). Beim Starten der Verbindung wird das gesamte Fenster schlagartig übertragen, wobei das 30. Paket verloren geht. Durch den Empfang der ersten Quittung erhält der Sender wieder einen Kredit von einem Paket, auf das er mit dem Aussenden des 101. Paketes reagiert. Die weiteren Quittungen erreichen den Empfänger genau in dem Abstand, in dem die Datenpakete zuvor einen eventuelle vorhandenen Engpaß durchlaufen haben. Das Senden weiterer Pakete auf diese Quittung erfolgt daher genau mit der Geschwindigkeit des Engpasses, d.h. ***die selbsttaktende Phase*** *ist erreicht. Nach Empfang der Quittung für das 29. Paket und der zugehörigen Übertragung eines weiteren Paketes wartet der Sender eine gewisse Zeitspanne auf das Eintreffen der Quittung für das 30. Paket. Da diese Quittung ausbleibt, beginnt er mit der Fehlerbehebung, die idealerweise hier mit dem Selective Repeat Technik erfolgen sollte. Der Sender wiederholt also nur das Paket mit der Nummer 30 und wartet die Quittung dafür ab. Bis das wiederholte Paket den Empfänger erreicht, sind dort alle anderen Pakete bis einschließlich 129 eingetroffen. Da diese vom Empfänger gepuffert wurden, füllt Paket Nr. 30 eine Sequenz von 130 korrekt empfangen Paketen. Der Empfänger quittiert also und eröffnet damit den Kredit im Umfang von erneut 100 Paketen - das Fenster klappt um. Dies führt wieder zu einer Lastspitze wie beim Start der Verbindung. Das geschieht sogar unabhängig vom verwendeten Mechanismus der Fehlerbehebung. Diese plötzliche Lastspitze trifft auf die ohnehin schon überlastete Engpaßressource. Somit ist die Wahrscheinlichkeit sehr hoch, daß erneute Pakete wegen eines Pufferüberlaufs verloren gehen. Dieses zyklische Verhalten führt zu einer permanenten Reduktion der Nutzdatenrate. Das System generiert so intern selbst eine zusätzliche Auftragslast, die für den Benutzer sichtbare Nutzdatenrate fällt daher auf einen Bruchteil der verfügbaren Datenrate ab (man hat im Internet schon Reduktionen bis auf 0,1 Prozent festgestellt).*

Dynamische Seiteneffekte einer fensterüberwachten Datenübertragung sind fast ausschließlich auf die Tatsache zurückzuführen, daß diese Form der Kontrolle keine Beschränkung hinsichtlich der Geschwindigkeit, mit der die einzelnen Pakete zum Senden übertragen werden, auferlegt. In der einfachsten Form werden hieraus entstehende Effekte beim Start einer Verbindung ersichtlich:
Zu Beginn einer Datenübertragung besitzt der Sender einen Kredit, der bei optimaler Konfiguration etwa der Pfadkapazität entspricht. Dieser Kredit erlaubt die volle Nutzung der unterliegenden Bandbreite (da so im eingeschwungenen Zustand keine Wartezeiten des Senders auftreten), ohne daß es zu erheblichen Stausituationen im Netz kommt. Beim Start der Verbindung wird der Sender nun den kompletten Kredit verbrauchen, wobei die Datenrate der Geschwindigkeit der Netzankopplung des Senders entspricht. Aufgrund der hohen Dynamik der im Netz transportierten Verkehrsströme kann jedoch jederzeit ein Engpaß auftreten, der diese Datenrate nicht unterstützen kann. Die Pakete werden daher an diesem Punkt (z.B. einem Router) mit einer höheren Rate ankommen, als sie weitergeleitet werden können, d.h. es wird eine gewisse Anzahl von Paketen gestaut. Es müssen daher im Zuge des Startvorganges Pakete im Umfang der Fenstergröße vom Router gepuffert werden, obwohl der Sender mit der für die Konfiguration optimalen Fenstergröße arbeitet! Da der dem Sender zur Verfügung stehende Kredit begrenzt ist, wird er nach Aussendung des Fensters eine längere Zeit auf die Ankunft der ersten Quittung

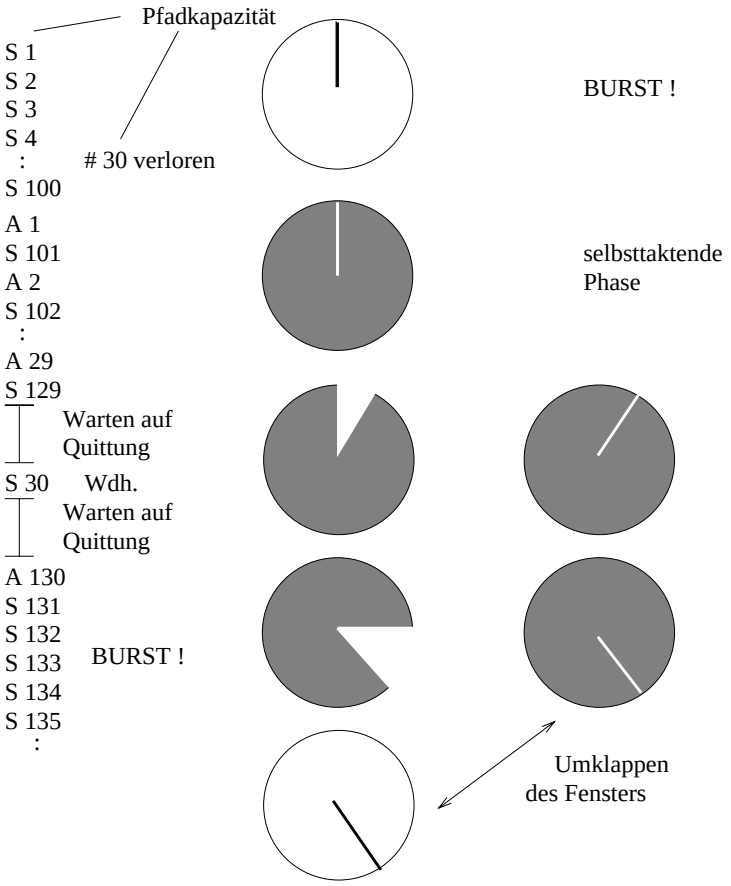

Abbildung 60. Phasen einer fensterüberwachten Datenübertragung

warten müssen. Während dieser Zeitspanne ist das Engpaßsystem möglicherweise in der Lage, die entstandene Warteschlange abzubauen, so daß der eingeschwungene Zustand dennoch erreicht werden kann (sofern der Router ausreichend Puffer zur Aufnahme der überzähligen Pakete besitzt). So muß selbst bei optimaler Konfiguration des Senders im Zuge von Lastspitzen mit einer Pufferbelastung im Umfang der Pfadkapazität (und damit einiger hundert Kilobyte bis zu einigen Megabyte) gerechnet werden.
Zusammenfassend erwachsen aus diesen Lastspitzen zwei unterschiedliche Probleme:

1. Die entstehenden Warteschlangen sind im Umfang proportional zur Pfadkapazität. Ist ein Zwischensystem nicht in der Lage, diese Datenmengen zu puffern, so kommt es zu Paketverlusten. Diese führt in einen stabilen Zustand, in dem überwiegend Fehlerbehebungsmaßnahmen durchgeführt werden, was eine erheblich verringerter Nutzdatenrate nach sich zieht.

2. Die innerhalb eines Fensters übertragenen Pakete erfahren sehr unterschiedliche Laufzeiten. Während die ersten noch auf ein unbelastetes System treffen und daher sofort weitergeleitet werden, müssen sich nachfolgende Pakete zunächst in einer Warteschlange einreihen. Zu ihrer Umlaufzeit addiert sich somit die Zeitspanne, die sie innerhalb der Warteschlange zubringen (in der Größenordnung der normalen Umlaufzeit!).

Die letztgenannte Tatsache hat Auswirkungen sowohl auf traditionelle als auch auf isochrone Datenströme. Bei der Überwachung eines Datentransfers mit Hilfe von Timern muß darauf geachtet werden, daß Paketverluste nicht irrtümlich durch einen zu niedrigen Timer angezeigt und so unnötig Fehlerbehebungsmaßnahmen eingeleitet werden. Um die Schwankungen der Umlaufzeit zu kompensieren, wird daher ein relativ großer Wert für diese Überwachungszeit benötigt, was dann aber tatsächliche Paketverluste zusätzlich verzögert, so daß die Verluste an der Nutzdatenrate weiter wachsen. Schließlich haben erhebliche Warteschlangengrößen einen destabilisierenden Einfluß auf den Betrieb des Netzes, da es in dieser Situation sehr leicht zu Paketverlusten auch anderer die Engpaßressource nutzender Verbindungen kommt. Der Verlust von Paketen einer Verbindung überträgt sich so auf andere Verbindungen, die ebenfalls im Zuge der Fehlerbehebung kurzfristige Lastspitzen produzieren und weitere Verbindungen infizieren. Es kann auf diese Weise zu synergetischen Effekten kommen, die unter Umständen sogar im Zusammenbruch des Netzes münden.
Aus Sicht isochroner Datenströme erfordert eine hohe Varianz der Paketlaufzeit einen sehr großen Pufferspeicher zur Kompensation. Dieser wiederum erhöht die mittlere Laufzeit eines Pakets, was sich insbesondere bei interaktiven Anwendungen (z.B. bei Videodaten, oder auch Telnet) zumindest sehr störend auswirkt oder eine solche Interaktion sogar vollständig unmöglich macht.

3 Fenstermechanismen bei ATM

Die klassischen Fenstermechanismen sind in ihrer ursprünglichen Form ungeeignet für den Einsatz in ATM-Netzen. Ein Problem liegt z.B. in der unidirektionalen Natur von ATM-Verbindungen. So ist also nicht ohne weiteres möglich für den Empfänger, eine Rückmeldung an den Sender zu schicken. Zusätzlich steht ATM für eine hohe Datenübertragunggeschwindigkeit, aber alle Fenstermechanismen verlangen Zeit[20]:

1. die Rückmeldung vorzubereiten,
2. die Rückmeldung zu übertragen und
3. die Rückmeldung zu bearbeiten.

In Kapitel 2.4 offenbarten sich die Probleme der Ende-zu-Ende Kontrolle. Aufgrund der hohen Datenraten in ATM-Netzen ist der Ansatz der Ende-zu-Ende[21] Kontrolle vollkommen unbrauchbar.
Etliche Simulationen, Analysen und Experimente[22] haben gezeigt, daß adaptierte Fenstertechniken, jedoch als Link-zu-Link[23] Kontrolle eingesetzt, durchaus in der Lage sind, ein großes Spektrum an ATM-Verkehrsarten zu unterstützen. Dazu zählt vor allem auch der burstartigen Verkehr, der den ratenbasierten Techniken noch Probleme bereitet.

[20] Diese Anforderungen machen eine effiziente Hardware Umsetzung relativ teuer.
[21] Flußkontrolle ist der Definition nach eine Ende-zu-Ende Kontrolle.
[22] siehe dazu auch [KC94b, KC94a, KBC94b, KBC94c, KBC94a, HKM93, OON88]
[23] Die Link-zu-Link Kontrolle wird als Lastkontrolle bezeichnet (siehe Kap. 2).

In der Diskussion des ATM-Forums zur Standardisierung der Verkehrskontrollmechanismen in ATM-Netzen stand als weiterer Lösungsansatz die ratenbasierte Technik zu Debatte. Ein Vorteil der ratenbasierten Technik ist ihrer preiswerte Umsetzbarkeit in Hardware. Weiterhin verlangen Ratenmechanismen[24] keine Rückmeldungen (zumindest solange die Datenübertragung fehlerfrei bleibt).
Die Abstimmung unter den offiziellen Mitgliedern des ATM-Forums[25] am Ende des Jahres 1994 ergab dann ein klares Votum für die ratenbasierte Technik.
Doch damit zog noch lange keine Ruhe zu diesem Thema in das Forum ein. Die Befürworter[26] der kreditbasierten Verkehrskontrolle gaben sich nicht so einfach geschlagen. Es kamen einige sehr interessante Vorschläge[27] zur Adaption der Fenstermechanismen zum Vorschein. Zum besseren Verständnis muß ich aber hier den Pfad der Flußkontrolle verlassen und zur Lastkontrolle wechseln:
Jedes Netzwerk hat sogenannte Flaschenhälse: Punkte, an denen mehr Daten ankommen, als das Netzwerk abtransportieren kann. Solche Punkt können z.B. Switches mit vielen Ports sein. Staus treten z.B. immer dann auf, sobald Daten für einen Output-Port auf mehreren Input-Ports ankommen. Ein kurzfristige Lösung dieses Problems bilden Puffer, in denen eine geringe Anzahl von Daten überlasteter Output Ports gepuffert werden können. Da aber dieser Speicherplatz begrenzt ist, muß jede durch diesen Flaschenhals sendende Quelle dazu veranlaßt werden, nicht mehr Daten zu senden, als es die faire Verteilung der Ressourcen zuläßt. Zur Lösung dieses Feedback-Problems gibt es zwar mehrere Vorschläge, das Prinzip ist aber nahezu gleich: Jeder Netzwerk-Switch sammelt Informationen über Daten-Staus und informiert direkt oder indirekt die Quelle der Daten. Die Ansprüche an einen Kontrollmechanismus sind hoch:

- Die Verzögerungszeit der Rückkopplung muß so klein wie möglich sein, um die Ausmaße der Pufferspeicher entsprechend klein zu halten, welche die vor dem Eintreffen des Stausignals bei der Quelle zwischenzeitlich eintreffenden Pakete aufnehmen müssen.

- Pakete dürfen so wenig wie möglich wegen Pufferüberläufen verworfen werden.

- Verbindungen zwischen Switches sollen immer mit der maximal zur Verfügung stehenden Kapazität genutzt werden. D.h. wenn eine Verbindung ihre Datenrate verringert, soll es anderen möglich sein, ihre Rate um diesen Betrag zu erhöhen.

- Die Verteilung der Ressourcen im Staufall an einem Switch soll fair unter den konkurrierenden Verbindungen geschehen.

- Die Flußkontrolle soll robust gegen Verluste oder Verzögerung von Kontrollinformationen sein. Der Verlust soll nicht zur Vergrößerung eines möglichen Datenstaus führen.

[24] Für weiter Details: siehe Ausarbeitung von Matthias Jacob und [Teb93, DM91].
[25] `http://www.atm-forum.com/`
[26] Dazu gehören einige renomierte Unternehmen wie z.B. DEC, Intel und Northern Telecom.
[27] Da das Votum der Entscheidung ziemlich eindeutig war, werden diese Vorschläge daran nichts ändern können. Die Artikel sind aber aus der Sicht interessant, da die Vorteile des Kreditsystems durchaus von anderen Mechanismen verwendet werden könnten.

– Eine Anpassung durch komplexe Parameter an die jeweiligen Gegebenheiten soll nicht notwendig sein.

– Und zum guten Schluß: Der Aufwand für den Mechanismus soll in einem gesunden Verhältnis zum Nutzen stehen.

3.1 Modellierung von charakteristischen Datenquellen

Zur Beurteilung der Leistungsfähigkeit eines Kontrollmechanismusses ist ein Modell des Datenverkehrs in einem Netzwerk erforderlich. Für unsere Zwecke ist vollkommen ausreichend zwischen ruhigen und burstartigen Datenquellen zu unterscheiden:

– Der Datenstrom einer *"ruhigen"* Datenquelle ist relativ konstant und vorhersehbar bezogen auf Zeitintervalle, die wesentlich größer sind als die Reaktionszeiten des Kontrollmechanismusses. Die Bearbeitung solchen Datenströme ist relativ simpel und unproblematisch. Die Ressourcen an Krisenpunkten lassen sich fair an diese Datenquellen verteilen, ohne daß die Gefahr von plötzlichen Datenstaus gegeben ist. Die erforderlichen Puffergrößen sind relativ klein. Zu diesen Datenquellen gehören z.B. Video- und Audioübertragungen mit einer festen Kompressionsrate. Ratenbasierte Verfahren arbeiten ausgezeichnet mit diesen Datenquellen zusammen.

– Ein *burstartiger* Datenstrom verändern unvorhersehbar sein Datenaufkommen. Das Klicken beispielsweise eines Mosaic-Benutzers auf einen Link kann das Netzwerk nicht vorhersehen, und trotzdem will der Benutzer natürlich so schnell wie möglich die aufgerufene Seite sehen. Weitere mögliche Quellen sind Netzwerkprotokolle, die den Datenverkehr in Pakete oder Rahmen zerlegen, die in irregulären Zeitintervallen vergeschickt werden. Diese Bursts treten sporadisch auf und/oder sind kürzer als der Round Trip Delay[28]. Häufige Pufferüberläufe sind ein Zeichen dafür, daß ein Netzwerk durch burstartigen Verkehr überlastet wird. Kreditbasierte Verfahren eignen sich hervorragend zum Umgang mit burstartigen Datenquellen, da sie die Pufferverteilung direkt beeinflussen können.

Noch eine Bemerkung zur ATM-Serviceklasse **A**vailable **B**it **R**ate (**ABR**):
Einigen Anwendung ist es möglich, ihr Datenaufkommen zu reduzieren, wenn es das Netzwerk erforderlich macht. Andererseits würden sie aber auch gerne ihre Datenrate vergrößern, wenn freie Kapazitäten vorhanden sind. Um einen solchen Verkehr zu unterstützen, wurde die Serviceklasse ABR[29] eingeführt und standardisiert. Weitere ATM-Service-Klassen sind *Unspecified Bit Rate (UBR), Constant Bit Rate (CBR)* und *Variable Bit Rate (VBR)* (siehe dazu [Sat95]).

Im nächsten Kapitel soll der Vorschlag "FCVC - Flow-Controlled Virtual Channels" etwas näher vorgestellt werden.

[28] Die Laufzeit eines Signals von der Quelle zur Senke und zurück.
[29] Wie sich später herausstellen sollte, hatte man sich damit den *Teufel* ins Haus geholt.

4 Der FCVC Vorschlag aus dem ATM-Forum

Der Kern aller Artikel[30], die hier in Auszügen zitiert werden, ist die Idee einer kreditbasierten Flußkontrolle zwischen jedem einzelnen Link einer Verbindung [KM95, KBC94b, KC94c, Com94a]. Man bezeichnet diese Verbindung auch als Flow Controlled Virtual Connection, auch abgekürzt mit: **FCVC**.
Der Mechanismus erfüllt einige der Forderungen aus Abschnitt 3:

1. Durch die Link zu Link Überwachung ist die Verzögerungszeit der Rückkopplung minimal.

2. Flexible Möglichkeiten zur Überwachung der Fairneß von ABR VCs[31]. Die Verteilung der Ressourcen im Krisenfall ist fair.

3. Kein Zellverlust während auftretender Staues im Netz.

4. Keine Reduzierung des erreichbaren Datendurchsatzes.

5. Die Stauüberwachung sorgt dafür, daß entstandene Staues lokal begrenzt bleiben und keine negativen Auswirkungen auf das restliche Netzwerk haben.

6. Keine Anpassungen an das jeweilige Netz notwendig.

Die Vorteile dieses System erwachsen hauptsächlich aus der Idee, die Flußkontrolle nicht zwischen den kommunizierenden Endsystemen agieren zu lassen, sondern jeden Link einzeln mit dieser Technik zu überwachen. In einer Link-zu-Link Kontrolle ist die Rückkopplungszeit bedeutend kürzer als in einer Ende-zu-Ende Kontrolle. Aus diesem Grund sind viel kleinere Puffergrößen erforderlich, was wesentlich zur Leistungssteigerung, Fairneß und Verkürzung der Übertragungszeiten beiträgt.
Häufig entstehen Staus dadurch, daß mehrere Datenströme sich einen gemeinsamen Link teilen müssen. Daher macht sich an dieser Stelle ein Mechanismus erforderlich, der die vorhandenen Ressourcen fair auf die miteinander konkurrierenden Datenströme verteilt. Betroffen davon ist sowohl die Aufteilung der Puffer unter den einzelnen Verbindungen (Puffer-Management) als auch die Kreditvergabe jeder einzelnen Verbindung (Kreditmanagement).

4.1 Begriffsdefinition und prinzipielle Funktionsweise

Ein FCVC besteht aus einer oder mehreren per Flußkontrolle überwachten Verbindungen, sogenannten FC Links. Jeder dieser Links verbindet 2 Knoten miteinander. Das können sowohl Adapter als auch Switches sein. Die Abbildung 61 zeigt 2 FCVCs (VC1 und VC2), für jeden einzelnen Link existiert eine eigenständig kreditbasierte Flußkontrolle. Betrachtet man einen einzelnen Link, so fließen die Daten vom Upstream-Knoten

[30] Artikel zum Thema der kreditbasierten Flußkontrolle sind auf dem FTP-Server der Harvard-Universität: `virtual.harvard.edu:/pub/htk/atm` zu finden.
[31] VC: Virtual Channel, Virtueller Kanal

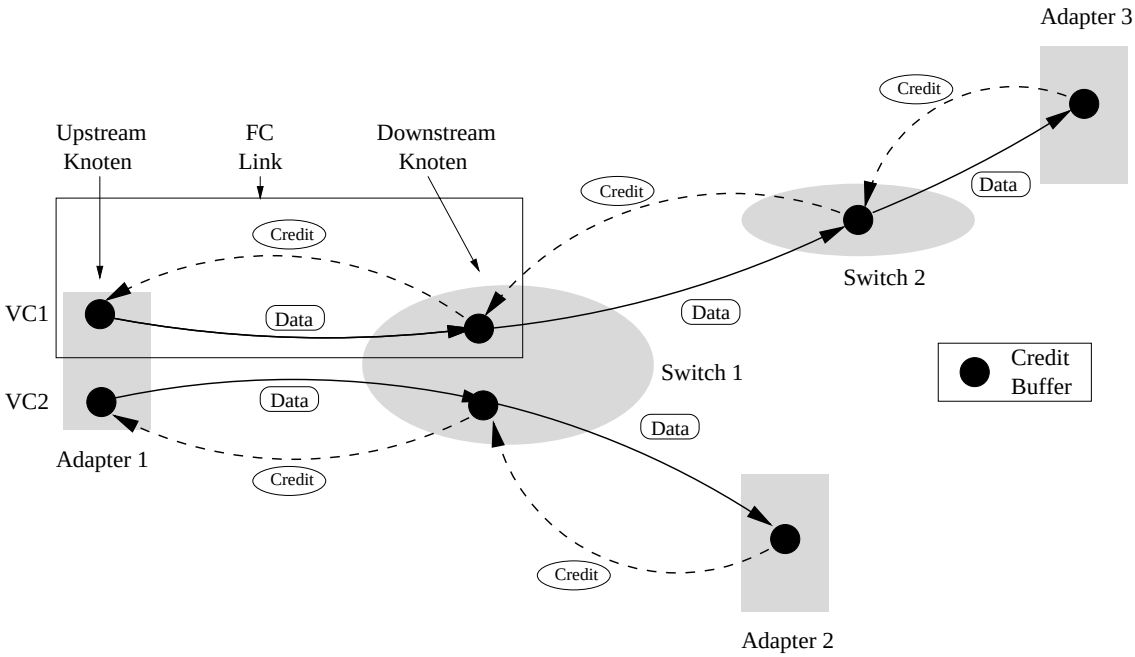

Abbildung 61. Kreditbasierte Flußkontrolle zwischen jedem Link

zum Downstream-Knoten und die Kreditzellen in die entgegengesetzte Richtung. In diesem Zusammenhang gibt es dann auch jeweils einen Sendepuffer (im Upstream-Knoten) und einen Empfangspuffer (im Downstream-Knoten). Meist ist der Empfangspuffer des vorhergehenden Links der Sendepuffer des folgenden Links. Um den Überlauf des jeweiligen Empfangspuffers zu verhindern, ist jeder Link flußkontrolliert (wie im Kapitel 2.1 beschrieben). Die Aufgabe der Kreditzellen[32] ist der Transport aller zur Flußkontrolle benötigten Informationen in entgegengesetzter Richtung zum Datenfluß (Upstream). Das Ankommen einer Kreditzelle signalisiert dem Upstream Knoten, daß der Downstream Knoten freien Pufferspeicher zur Verfügung hat, zum Empfang von Daten über die VC. Der Upstream Knoten darf nun entsprechend dem für ihn bestimmten Kredit-Element aus der Kreditzelle (siehe Abbildung 62) eine Anzahl von Datenzellen über die VC abschicken.

Die FCVC zerfällt prinzipiell in 2 Phasen :

1. **Puffer Allocation** (Puffer Bereitstellung):
   Der Upstream Knoten gibt das Dienstprimitiv Buf_Alloc an den Downstream Knoten.

2. **Credit Control** (Kredit Kontrolle):
   Der Upstream Knoten verwendet den Zähler: Crd_Bal zur Überwachung der Verbindung (unterer Fensterrand + geschickte Pakete).

Es folgt nun die Beschreibung eines Protokolls für die Kredit-Kontroll-Phase.
Anschließend folgt ein Überblick über zwei mögliche Realisierungsvarianten der Kreditkontrolle.

[32] Aufbau einer Kreditzelle: siehe Abbildung 62

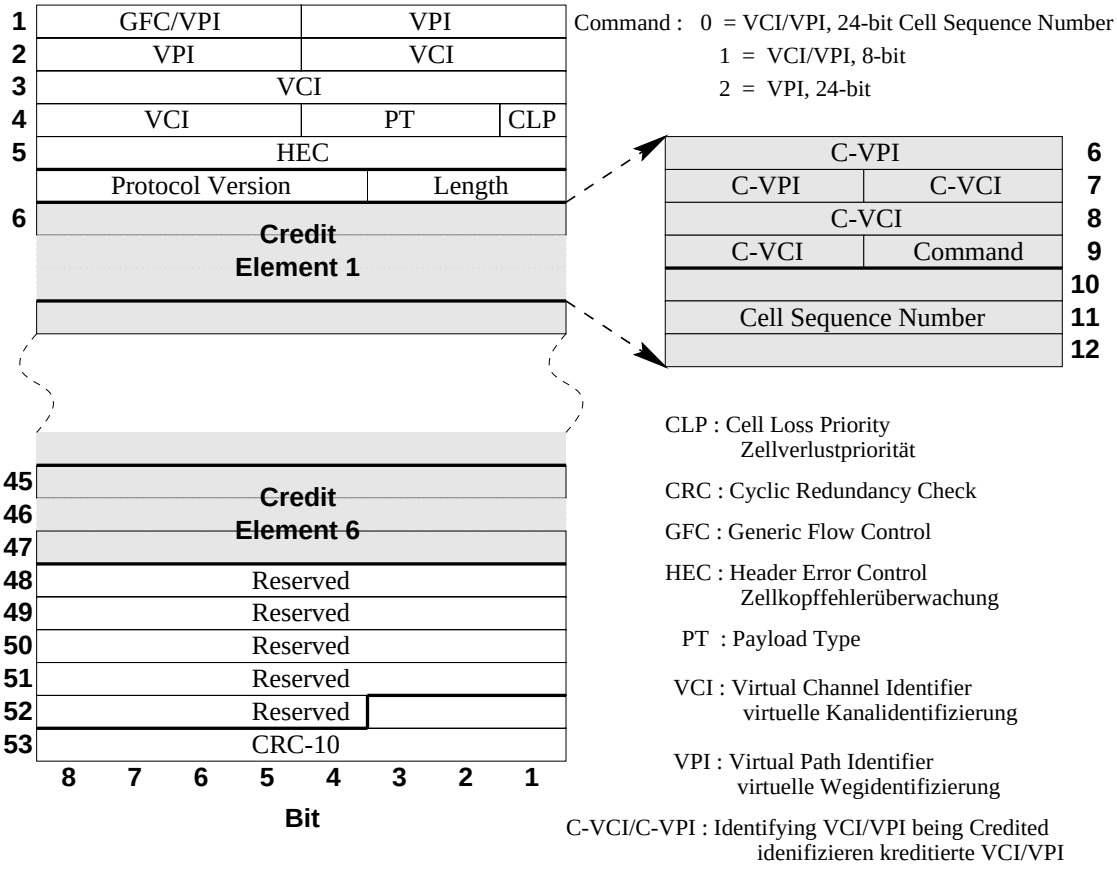

Abbildung 62. Credit Update Cell Format

4.2 Credit Update Protocol (CUP)

Das Credit Update Protocol ist ein effizientes und robustes Protokoll zur Implementierung einer Flußkontrolle über einen Link[33]. Die Aufgabe ist die Steuerung der Kommunikation zwischen zwei an einem Link beteiligter Knoten. Dafür sind folgende Kontrollzellen erforderlich:

1. **Credit Update**: Sie werden vom Downstream Knoten erzeugt und übernehmen die Funktion der Bestätigung (ACK). Die enthaltene Nummer bestätigt dem Upstream Knoten den Empfang des Paketes gleicher Nummer durch den Downstream-Knoten und erhöht damit den dem Sender zur Verfügung stehenden Kredit (siehe Abschnitt 2.1). Eine einzelne Credit Update Zelle (siehe Bild 62) kann für maximal 6 unterschiedliche Verbindungen mit ein und demselben Upstream Knoten die Kredite erneuern (durch das jeweilig zugehörige Kreditelement).

2. **Credit Check**: Der Upstream-Knoten erzeugt diese Zellart um zu protokollieren welche Zellen er gerade abgeschickt hat. Falls keine Zellen unterwegs verloren gegangen sind, stimmt die enthaltene Nummer mit der Nummer des zuletzt von Downstream Knoten empfangenen Pakets überein. Ansonsten wird errechnet wieviel Zellen verloren gegangen sind, und zu den aktuellen Zählerständen des Empfängers hinzu addiert.

[33] Unter einem Link versteht man hier sowohl die physikalische Verbindung zwei benachbarter Knoten als auch die virtuelle Verbindung zweier nicht unmittelbar benachbarter Knoten.

Wie in der Abbildung 63 zu sehen ist, besitzt der Sender für jede VC einen eigenen Tx_Cnt[34] zum totalen Zählen aller gesendeten Pakete über diesen Link und der Empfänger einen Fwd_Cnt[35] zum totalen Zählen aller von ihm weitergeleiteten Pakete aus diesem Link. Der Emfänger fügt periodisch den aktuellen Fwd_Cnt der Verbindung in das jeweilige *Credit Element* der *Credit Update Cell* ein.

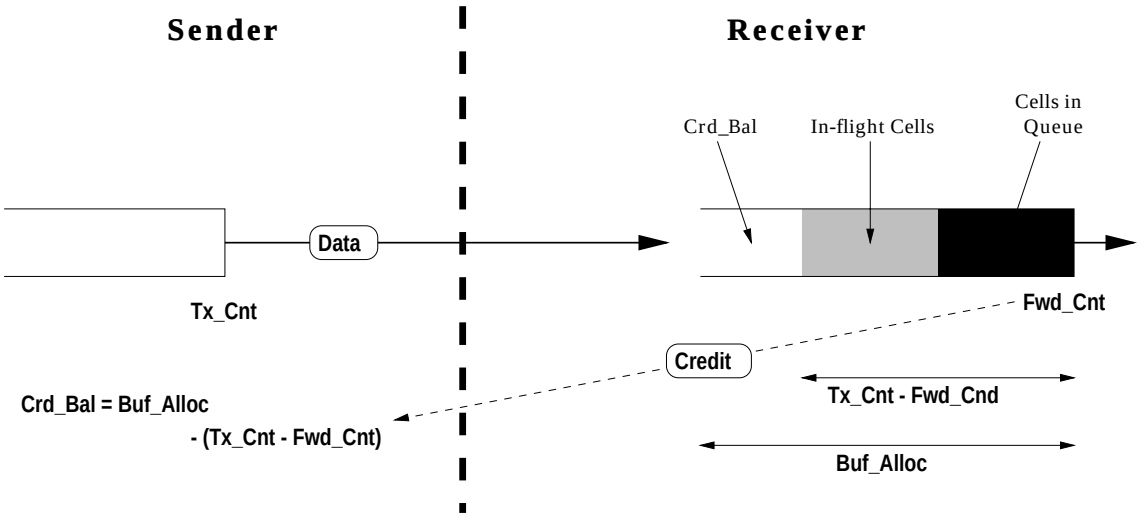

Abbildung 63. Credit Update Protocol (CUP)

Der Sender erhöht nachdem Empfang der Credit Update Cell seinen Crd_Bal[36] für diese Verbindung:

$$Crd\_Bal = Buf\_Alloc - (Tx\_Cnt - Fwd\_Cnt) \tag{2}$$

wobei Buf_Alloc die Größe des Empfangspuffer angibt. Der letzte Teil der Gleichung $(Tx\_Cnt - Fwd\_Cnt)$ repräsentiert die Anzahl der Pakete die noch nicht weiter geschickt wurden, was der Summe der Pakete im Empfangspuffer und der Pakete, die noch unterwegs zum Empfänger sind, entspricht (siehe Bild 63). Solange der mit Hilfe der Gleichung (2) beim Sender neu berechnete Kredit nicht überschritten wird, kommt es zu keinem Pufferüberlauf beim Empfänger. Der Sender generiert regelmäßig Credit Check Pakete um Paketverluste zu erkennen (zur genaueren Erläuterung siehe [KBC94c]). Nach welcher Anzahl (im weiteren mit $N2$ bezeichnet) der Empfänger ein Credit Update Paket erzeugt, hängt von der VC ab. Genauer gesagt, die Zahl $N2$ kann statisch als auch dynamisch festgelegt werden.
Das Credit Update Protocol ist einfacher und auf einer tieferen Schicht angesiedelt als üblicherweise Sliding Window Protokolle z.B. in X.25 oder TCP-Netzen. Insbesondere ist z.B. kein Mechanismus, vorgesehen um verlorene Pakete zu wiederholen. Kommt es bei X.25 oder TCP zu einem Paketverlust, wird das Kreditfenster so lange angehalten, bis das verlorengegangene Paket erfolgreich wiederholt wurde. Um dies zu implementieren

[34] Transmission Counter
[35] Forward Counter
[36] Credit Balance

müßte jedes Paket noch eine Sequenznummer tragen. Doch hier wiederholte der Sender keine Pakete, der Empfänger fordert keine an und die Zellen tragen keine Sequenznummer.

4.3 Statische oder dynamische Kreditkontrolle

Statische und dynamische Kreditkontrolle unterscheiden sich in ihrer Pufferbereitstellung. Sie ist entweder statisch oder dynamisch:

1. Bei der statischen Kreditkontrolle bleibt die Puffergröße konstant über die Lebenszeit der Verbindung hinweg. Die Realisierung des Protokolls ist extrem einfach, aber in den meisten Fällen uneffizient. Statische Puffergrößen verringern den Management-Aufwand. Sie finden vor allem Verwendung in LANs[37] als auch in Knoten mit relativ wenigen, existierenden Verbindungen. Die statische Puffergröße für eine Verbindung entspricht mindestens der Anzahl der Datenzellen zwischen zwei Credit-Update Zellen.

2. Die dynamische Kreditkontrolle teilt den VC je nach Bedarf dynamisch Pufferspeicherplatz zu. Falls der Datenstrom einer Verbindung nachläßt, wird die Größe ihres Pufferspeichers verringert und damit anderen Verbindungen zur Verfügung gestellt. Der Algorithmus zur dynamischen Pufferverteilung kann entweder auf der Senderseite oder auf der Empfängerseite sitzen (Einzelheiten dazu siehe [KC94a]).

4.4 Resümee

Transparenz: Im Initialisierungsprozeß eines Links kann ermittelt werden, ob beide Partner mit dem Credit Update Protokoll zusammenarbeiten. Ist das nicht der Fall, kommt kein FCVC zustande. Falls der Upstream Knoten nicht mit dem Credit Update Protokoll arbeitet, schützt sich der Downstream Konten gegen eine Ressourcenüberlastung durch den Upstream Knoten, indem er, sobald der jeweilige Empfangspuffer voll ist, einfach aller weiteren Pakete dieser Verbindung verwirft.

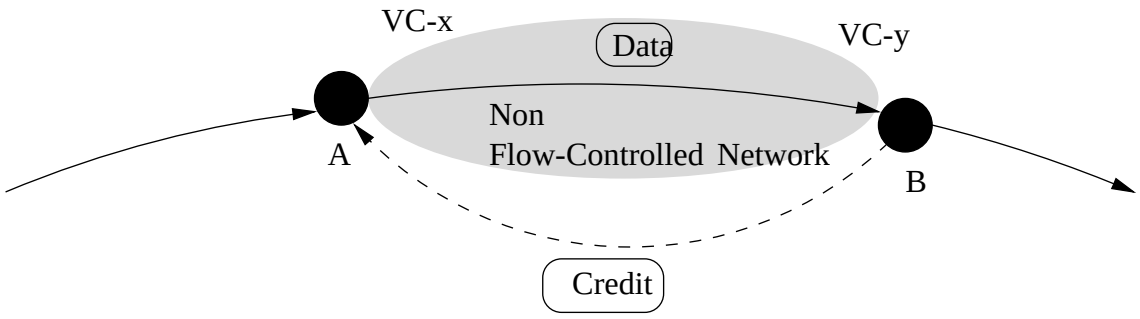

Abbildung 64. FC VC Link von A nach B via Tunneln

Ferner ist es möglich, FCVCs auch über Netzwerke hinweg aufzubauen, die keine Flußkontrolle unterstützen (siehe Bild 64). Man bezeichnet diesen Vorgang auch als Tunneln

[37] Local Area Network

(engl. **tunneling**). Das "getunnelte" Netzwerk behandelte den Up- und den Downstream wie zwei normale VC. Somit ist es möglich, CUP-Systeme in bereits existierende Systeme gefahrlos zu integrieren.
Falls also alle Links einer Verbindung die Flußkontrolle unterstützen kommt es zu keinerlei Zellverlusten. Die Gefahr des Verlustes ist aber sofort gegeben, sobald auch nur ein beteiligter Link keine Flußkontrolle zuläßt (mehr dazu in [Com94b]).

Puffergröße: Das Credit Update Protokoll unterstützt sowohl statische als auch dynamische Puffergrößen. Das führt zu einer höheren Effizienze bezüglich der Ressourcenauslastung und erzielt damit eine verbesserte Stauvermeidung.

maximale Bandbreite: Das Verfahren stellt die maximal nutzbare Bandbreite einer Verbindung zur Verfügung. Dadurch wird die Nutzdatenreduzierung durch Wiederholungen von Paketen, die auf Grund von Pufferüberläufe verworfen wurden, gesenkt.

Fairneß : Der FCVC-Vorschlag ermöglicht eine Aushandlung der Ressourcenverteilung zwischen allen beteiligten Systemen in der Initialisierungsphase. Innerhalb der gleichen Serviceklasse wird allen beteiligten Systemen ein fairer Zugriff auf die zur Verfügung stehenden Ressourcen ermöglicht. Im Falle eines Staus sorgt das Protokoll dafür, daß keine Interaktionen zwischen Datenströmen in einem Knoten auftreten, der entstandene Stau also keine weitere Verbindungen infiziert.

Netzkollaps: Da die Empfangspuffer in einem Knoten untereinander unabhängig sind, bleiben Datenstaues lokal begrenzt und haben keinerlei Einfluß auf das restliche Netzwerk.

Netzanpassung: Keine Datenzelle verläßt den Sender, wenn nicht das Netzwerk garantieren kann, die Datenzelle nicht zu verwerfen (Ausnahmefälle sind unvorhersehbare, plötzliche Fehler von Netzwerkelementen).

Einsetzbarkeit: FCVC ist vollkommen unabhängig von den jeweils implementierten Protokollen auf den höheren Schichten und unterstützt somit die komplette Bandbreite der vorkommenden Verkehrsarten. Zudem ist FCVC dazu ausgelegt, für einige Verkehrsarten auch Punkt-zu-Mehrpunktverbindungen zu unterstützen. Im Mehrpunktfall gibt es im Netz einen oder auch mehrere Verästlungspunkte, Punkte also mit einem Input Link und mehreren Output Links. Im allgemeinen werden mehrere davon FC Links sein (aber nicht unbedingt erforderlich). Wenn der Input Link ein FC Link ist, wird er nur Datenzellen weitergeben, sofern er noch Kredit von allen FC Output Links hat. Man hat somit ein großes Feld an Möglichkeiten, Mehrpunktverbindungen[38] zu kreieren. Sind z.B. bei einer Punkt-zu-Mehrpunkt-Verbindung[39] alle Output Links flußkontrolliert, regelt der langsamste Empfänger den Sender.

[38] Broadcast/Multicast
[39] Broadcast

5 Epilog

Die Entscheidung der Forum-Mitglieder über die Standardisierung der ATM-Flußkontrolle fiel zu Gunsten der ratenbasierten Techniken aus. Über die Beweggründe jedes einzelnen kann ich nur spekulieren, aber sicherlich hat die kostengünstigere Hardware-Realisierbarkeit der ratenbasierten Mechanismen letztendlich den Auschlag gegeben. Im Hinblick auf die hohen Datenraten in einem Hochgeschwindigkeitsnetz müssen z.B. die Adaptionsmechanismen sehr einfach gehalten werden. Lokal zu einem einzelnen beteiligten System stellt sich vor allem die Forderung, Pakete überhaupt mit dieser hohen Datenrate behandeln zu können. Hierfür gibt es unter anderen zwei Ansätze:

1. Die Protokollfunktionen werden durch Spezialhardware und parallele Systemarchitekturen übernommen. Dabei werden einzelne Protokollfunktionen durch VLSI-Schaltkreise realisiert. Man macht sich dabei die Tatsache zunutze, daß ein Teil der notwendigen Protokollfunktionen parallel realisiert werden kann und die Abarbeitung der Pakete durchaus in Pipelines möglich ist.

2. Um diese genannten Vorteile nutzen zu können, sind jedoch gewisse Anforderungen an die Protokolle zu stellen. Die einzelnen im Rahmen der Protokolle zu realisierenden Funktionen sind auf die Umsetzbarkeit durch Hardware auszurichten [Che89] und eine weitgehende Parallelisierbarkeit der Funktionen sollte bereits bei der Definition der Protokolle mit berücksichtigt werden [SN89]. Diese Zielsetzung ist bei allen modernen Protokollen zu beobachten, wobei sich insbesondere im Bereich der Transportprotokolle zu ihrer Beschreibung der Begriff des "Lightweight Protocols" etabliert hat [DDK+90].

Die letztgenannte Forderung nach einer leichten Umsetzbarkeit in Hardware hat vor allem auf die primär an der Übertragung von Daten zwischen Endsystemen beteiligten Schichten (1) bis (3) des OSI-Basisreferenzmodells erhebliche Auswirkung. Da sie den aggregierten Verkehr aller über das Netz kommunizierender Instanzen zu transportieren haben, ist ihre effiziente Realisierung von besonderer Bedeutung. Zur Erreichung dieses Ziels macht man sich speziell die veränderten, in modernen im allgemeinen auf Glasfaser basierenden Netzen vorzufindenden physikalischen Eigenschaften des Übertragungsmediums zunutze. Aufgrund der extrem niedrigen zu erwartenden Bitfehlerrate (im Bereich von 10^{-11} bis 10^{-13}) verzichtet man auf eine Fehlerbehebung über Teilstrecken und überläßt diese Aufgabe den Endsystemen.

Integrated Layer Processing

Oliver Krämer

Kurzfassung

In Hochgeschwindigkeitsnetzwerken mit Übertragungsraten im Gigabit-Bereich wirkt sich ein Protokollturm mit mehreren Schichten, in denen die Daten unabhängig voneinander bearbeitet werden, als Flaschenhals. Der Ansatz des Integrated Layer Processing stellt eine Möglichkeit dar, existierende Protokolle zu optimieren, und kann auch beim Design neuer Protokolle angewandt werden. Im folgenden werden zunächst allgemeine Möglichkeiten der Optimierung in Protokollen diskutiert und darauf aufbauend die Technik des Integrated Layer Processing vorgestellt. Weiterhin wird anschließend kurz auf die Einschränkungen des ILP eingegangen.

1 Einleitung

Bei den z. Z. im LAN-Bereich verbreiteten Netzwerktechniken wie Ethernet (mit Übertragungsraten von ca. 10 Mbit/s) stellt die Abarbeitung von Kommunikationsprotokollen wie TCP/IP oder den ISO/OSI-Protokollen kein Problem dar, da die Protokolloperationen, die in den voneinander unabhängigen Schichten des Protokollturms sequentiell ausgeführt werden, im Vergleich zur eigentlichen Datenübertragung vernachlässigt werden können. Beim Einsatz in Hochgeschwindigkeitsnetzwerken mit Übertragungsraten im Gigabit-Bereich kommt es jedoch beim Einsatz dieser Protokolle zu Leistungseinbußen aufgrund zu hoher Verarbeitungszeiten im Protokollturm. Die Technik des „Integrated Layer Processing“ beim Entwurf neuer Protokolle führt durch Optimierungen im Protokollturm zu einem deutlich höherem Durchsatz. Im folgenden sollen zunächst allgemeine Eigenschaften und der Aufbau von Protokollen beschrieben werden, um anschließend Bereiche für Optimierungsmaßnahmen zu identifizieren, die auf das Integrated Layer Processing führen.

Der vorliegende Beitrag faßt im wesentlichen die in [AP93], [CT90] erläuterten Überlegungen zur Technik des Integrated Layer Processing zusammen. Des weiteren wird in diesem Zusammenhang das Architekturkonzept des „Application Level Framing“ [CT90] als eine der Grundlagen des ILP vorgestellt, die allerdings auch unabhängig vom ILP eingesetzt werden kann.

2 Funktionen und Aufgaben von Protokollen

Da die Datenübertragung die eigentliche Kernfunktion eines Protokolls darstellt, sollen bei den folgenden Betrachtungen die Funktionen des Verbindungsauf- und Verbindungsabbaus sowie der Wegewahl nicht berücksichtigt werden.

Die Datenübertragungsfunktionen eines Protokolls können in zwei Klassen eingeteilt werden, den Funktionen zur *Datenmanipulation* (data manipulation) und denen zur *Übertragungskontrolle* (transfer control).

2.1 Datenmanipulationsfunktionen

Zu dieser Klasse zählen Funktionen, die unmittelbar auf die zu sendenden Daten zugreifen und diese ggf. verändern. Dies sind

- das eigentliche Schreiben und Lesen von Daten
- Funktionen zur Fehlererkennung bei falsch übertragenen Datenpaketen
- Pufferung von Daten für eventuelle Wiederholung der Übertragung
- Verschlüsselung von Daten
- Kopieren der Daten in bzw. aus dem Anwendungsadreßraum
- Änderung der Darstellung von Daten, um Unterschiede in der Repräsentation von Datentypen auf unterschiedlichen Rechnerarchitekturen zu verdecken

2.2 Funktionen zur Übertragungskontrolle

In diese Klasse fallen Funktionen, die den Datenfluß kontrollieren und steuern. Dies sind im einzelnen

- Funktionen zur Fluß- bzw. Staukontrolle
- Funktionen zur Erkennung von Übertragungsfehlern, z. B. bei Verlust, Vertauschung oder Verdopplung von Datenpaketen
- das Senden von Empfangsbestätigungen für erhaltene Pakete
- das Multiplexen bzw. Demultiplexen von Verbindungen über eine Verbindung der darunterliegenden Schicht
- die Segmentierung/Reasssemblierung der Datenpakete
- das Versehen der Datenpakete mit einem Zeitstempel

3 Ansatzpunkte zur Optimierung

In diesem Abschnitt sollen die oben genannten Funktionen eines Protokolls auf ihre Eignung zur Optimierung untersucht werden. Für Optimierungsmaßnahmen bieten sich diejenigen Elemente an, die am meisten CPU-Zeit benötigen. Die in Abschnitt 2 nicht berücksichtigten Funktionen zur Verbindungsinitiierung können auch hier außer acht gelassen werden, da sie keinen Einfluß auf den Datendurchsatz haben.

3.1 Aufwand für Übertragungskontrolle und Datenmanipulation

Ein eingehendes Datenpaket muß zunächst einer logischen Verbindung zugeordnet und nach Ausführung der Datenmanipulationsfunktionen an die übergeordnete Protokollinstanz weitergereicht werden, d. h. eine Kontrollfunktion greift hierzu auf einige Felder des Pakets zu. Die Prüfung auf Übertragungsfehler stellt eine Datenmanipulationsfunktion dar, die nur im relativ seltenen Fall eines tatsächlichen Fehlers den Aufruf einer Kontrollfunktion bewirkt. Der Test auf Reihenfolgetreue kann schließlich durch einen einfachen Vergleich erfolgen. Zuletzt wird von einer Kontrollfunktion noch die Bestätigung abgeschickt.

Alle diese Operationen können mit relativ geringem Aufwand an Rechenzeit durchgeführt werden und erfordern nur wenige Speicherzugriffe.

Im Gegensatz dazu benötigen Datenmanipulationsfunktionen, die beispielsweise zur Prüfsummenberechnung oder zum Kopieren eines Datenpakets auf alle Bytes des Pakets zugreifen müssen, einen wesentlich höheren Zeitbedarf und führen mehr Speicherzugriffe aus als die oben beschriebenen Funktionen zur Übertragungskontrolle.

3.2 Darstellungsschicht

Unter den Datenmanipulationsfunktionen nehmen die der Darstellungsschicht eine besondere Rolle ein, da die Formatanpassung der Daten in dieser Schicht einen vergleichsweise großen Anteil an Prozessorzeit erfordert.

In [AP93] werden als konkrete Durchsatzwerte bei einem MIPS R2000-Prozessor für eine reine Kopieroperation 115 Mbit/s angegeben. Dieser Wert ist also als das theoretische Maximum des Protokolldurchsatzes anzusehen. Bei einer Umwandlung eines Integer-Arrays in ASN.1 Format werden dagegen nur 28 Mbit/s erreicht. Daraus läßt sich leicht erkennen, daß sich insbesondere die Darstellungsschicht zur Optimierung in Hochgeschwindigkeitsprotokollen eignet.

Hierbei muß jedoch berücksichtigt werden, daß auf Daten aus dem Anwendungsadreßraum zugegriffen wird. Daher kann auch die Anwendung selbst einen Flaschenhals darstellen, wenn die Daten nicht schnell genug bereit gestellt bzw. entgegengenommen werden können.

Weiterhin kann die Geschwindigkeit der Datenkonvertierung in der Darstellungsschicht durch verlorene oder falsch geordnete Datenpakete auf unteren Schichten gemindert werden. In diesem Fall wird auf der Darstellungsschicht die Verarbeitung solange unterbrochen, bis das Transportprotokoll den aufgetretenen Fehler behoben hat, z. B. durch erneute Übertragung der verlorengegangen Datenpakete.

3.3 Application Level Framing

Bei einem Datenverlust hat die Applikation keinen Einfluß auf die Behandlung des Fehlers. Das Transportprotokoll puffert zu sendende Datenpakete und schickt sie ggf. erneut, falls ein Übertragungsfehler aufgetreten ist. Dies ist jedoch nicht in allen Fällen die optimale Verfahrensweise, da z. B. der Verlust bei Übertragung von Audio- oder Videodaten

toleriert werden kann. In anderen Fällen besitzt die sendende Anwendung inzwischen aktualisierte Daten, die anstelle der vom Transportprotokoll gepufferten Daten verschickt werden könnten.

Weiterhin besteht eine Abhängigkeit der Darstellungskonvertierung von der Bereitstellung der Daten der darunterliegenden Schichten, d. h. eine höhere Schicht erhält keine Datenpakete mehr, wenn auf einer darunterliegenden Schicht ein Übertragungsfehler aufgetreten ist und beispielsweise eine erneute Übertragung notwendig wird. Wenn bezüglich der Datenmanipulation einer Schicht aufeinanderfolgende Datenpakete voneinander unabhängig sind, könnte die empfangende Protokollinstanz bei einem Paketverlust bereits nachfolgend eintreffende Pakete außerhalb der Reihenfolge bearbeiten. Dies ist z. B. bei der Prüfsummenberechnung der Fall, die für jedes Datenpaket einzeln durchzuführen ist. Das Einfügen von Synchronisationspunkten stellt eine Möglichkeit dar, voneinander unabhängige Datenportionen zu kennzeichnen. Die Synchronisationspunkte werden allerdings vom Transportprotokoll festgelegt und sind für die Anwendung transparent und nicht von ihr beeinflußbar. Es werden also u. U. Daten, die für die Anwendung eine logische Einheit bilden und die sie unabhängig von vorausgegangen Dateneinheiten verarbeiten könnte, durch die Synchronisationspunkte getrennt.

Dies kann durch das Konzept des „Application Level Framing“ verhindert werden. Bei diesem Ansatz legt die Anwendung selbst die Länge der Datenpakete fest, wobei sie eine für ihren Kontext sinnvolle Größe wählt. Eine solche *Application Data Unit* (ADU) sollte von der Anwendung unabhängig von anderen Paketen gehandhabt werden können, um die Darstellungskonvertierung auch bei Paketen durchführen zu können, deren Reihenfolge evtl. vertauscht ist.

Bei diesem Ansatz verliert allerdings die Datenübertragung einen Teil der oft gewünschten Transparenz, da die Anwendung jetzt selbst die Größe der Datenpakete frei bestimmen muß. Wenn ihr dabei keine Informationen über das zugrundeliegende Transportsystem vorliegen, wird die Paketlänge u. U. ungünstig für die Transportschicht gewählt. Ist die ADU-Länge z. B. nur um wenige Bytes größer als die Paketlänge der Transportschicht, werden immer zwei Pakete der Transportschicht benötigt, um eine ADU zu übertragen.

Die Länge der Applikation Data Units wird so gewählt, daß in darunterliegenden Schichten keine Segmentierung bzw. Reassemblierung erfolgen muß.

3.4 Serielle und integrierte Implementierung von Datenmanipulationen

Eine allgemein anwendbare Technik zur Optimierung ist die integrierte Implementierung von Datenmanipulationsfunktionen. Als Beispiel werde ein Feld von 10.000 Integerwerten betrachtet, auf dem nacheinander folgende Operationen ausgeführt werden sollen:

1. Addition von 1 zu jedem Wert
2. Bilden des 1er-Komplements jedes Werts

Sollen diese Operationen unabhängig voneinander ausgeführt werden, wird man zwei Schleifen programmieren, die jeweils von 1 bis 10.000 laufen und die Operationen nach-

einander abarbeiten. Dies bedeutet bei Ausführung auf einem RISC-Prozessor, daß in beiden Schleifen pro Schleifendurchlauf zwei Lade- und Speicheroperationen durchgeführt werden müssen, d. h. anstelle von Registerzugriffen muß auf den Cache oder auf den noch langsameren Hauptspeicher zugegriffen werden.

Eine naheliegende Optimierungsmaßnahme besteht in der Integration beider Schleifen, d. h. die Datenmanipulationen werden unmittelbar hintereinander ausgeführt. Dadurch werden die kostspieligen Speicherzugriffe um die Hälfte reduziert, da nur zwei Zugriffe auf den Hauptspeicher notwendig sind, und der 2nd-Level-Cache wird effizienter genutzt, weil ein Cache-Miss wesentlich seltener auftritt.

4 Integrated Layer Processing

Bei der konkreten Anwendung des Integrated Layer Processing auf einen existierenden Protokollturm gibt es einige Problembereiche, die besonderer Betrachtung bedürfen:

- Die Technik der Schleifenintegration kann bei komplexeren Datenmanipulationen nicht immer angewandt werden, insbesondere dann nicht, wenn auf einer Protokollschicht die Länge der Daten verändert wird (z. B. durch Komprimierung).
- Die einzelnen Schichten haben verschiedene Sichtweisen auf ein zu übertragendes Datenpaket. Die Schicht N führt ihre Operationen auf anderen Daten aus als Schicht N+1, weil zu den Daten, auf denen Schicht N arbeitet, auch die von der oberen Schicht hinzugefügten Protokollkontrollinformationen gehören. Eine Voraussetzung für die Anwendung des ILP ist allerdings eine gemeinsame Sicht benachbarter Schichten auf die zu verarbeitenden Daten.
- Neben den eigentlichen Datenmanipulationsfunktionen gehören zu den Aufgaben einer Protokollinstanz auch die Generierung der Protokollkontrollinformationen und die Flußkontrolle. Diese Funktionen müssen in einer bestimmten Reihenfolge zu den Datenmanipulationsfunktionen ausgeführt werden, d. h. bei Integration der Datenmanipulationsfunktionen zweier benachbarter Schichten ist diese Ordnung beizubehalten.
- Durch die Integration eigentlich unabhängiger Protokollschichten wird die Modularität des Protokolls beeinträchtigt, was das Design, die Implementierung, die Fehlersuche und die spätere Pflege erschwert. Dies kann durch eine automatisierte Synthese des integrierten Protokolls vermieden werden.

In den folgenden Abschnitten werden Lösungsansätze aus [AP93] für obige Probleme vorgestellt.

4.1 Word Filter

Wenn zwei Schleifen (wie im obigen Beispiel) auf Dateneinheiten derselben Größe und Anzahl sequentiell und nach dem First-In-First-Out Verfahren zugreifen, können diese Schleifen integriert, also die Datenmanipulationen in einer Schleife zusammengefaßt

werden. Allerdings wird beispielsweise durch Kompression oder Verschlüsselung die Datenmenge verändert. Die Datenrate des Ausgabedatenstrom einer Datenmanipulationsoperation kann sich also von der Datenrate des Eingabedatenstroms unterscheiden. Der Ansatz der Schleifenintegration kann daher in diesem Fall nicht verwendet werden. Betrachtet man z. B. eine Schleife, in der ein Array von 10.000 Bytes komprimiert wird, und eine anschließend auf diesen Daten in einer zweiten Schleife auszuführende Operation, beispielweise eine Invertierung, so können diese Schleifen nicht integriert werden, da nach der Komprimierung der Array wesentlich kleiner geworden sein kann.

„Word filter" stellen eine Möglichkeit dar, dieses Problem zu umgehen. Ein word filter führt eine Operation für je ein Maschinenwort (32 Bit) aus. Dabei wird in der Regel für jedes eingegebene Wort eines ausgegeben. Bei Daten, die nicht auf dieser Wortbasis behandelt werden können, kann durch Einführung eines Kontext, z. B. in Form eines Statusbits, auch eine flexiblere Handhabung erreicht werden. Erwartet ein Filter beispielsweise zwei Maschinenworte als Eingabe, erfolgt nach dem ersten Wort keine Ausgabe, sondern erst nach Erhalt des folgenden Worts erfolgt die Verarbeitung und die Ausgabe des Resultats. Anschließend wird der nächste Filter aufgerufen, d. h. die aufeinanderfolgenden Filter sind in einer Fließbandstruktur (Pipeline) angeordnet.

Nach [AP93] stellt ein Maschinenwort eine gut handhabare Einheit dar, da es die Schnittstelle zwischen zwei Protokollschichten einfach hält und außerdem Prozessoren in der Regel auf 32-Bit-Worte optimiert sind. Diese Stückelung wird unabhängig von der Anwendung vorgenommen. Hier unterscheidet sich der Ansatz von Abbott/Peterson von der zugrundeliegenden Arbeit von Clark/Tennenhouse. Letztere schlagen vor, daß Datenmanipulationsfunktionen auf von der Anwendung festgelegten Einheiten, den oben beschriebenen Application Data Units, arbeiten, um eine für die Applikation sinnvolle Verarbeitung zu ermöglichen.

Word filter sollten aus Gründen der Leistungsfähigkeit nicht als Funktionen implementiert werden, da insbesondere die Parameterübergabe bei Funktionsaufrufen einen hohen Zeitaufwand erfordert. In [AP93] wird die Technik des Inlining vorgeschlagen, die auch von der Programmiersprache C bzw. C++ bekannt ist (siehe [Str92]). Hat man beispielsweise einen word filter zur Prüfsummenberechnung, wird der Quellcode bei jedem Aufruf an der entsprechenden Stelle eingefügt. In C kann dies auch durch die Verwendung von Makros erreicht werden.

Die Geschwindigkeitssteigerungen sind sehr stark von der verwendeten Hardware abhängig. Auf einer DecStation 5000/200 geben Abbott/Peterson einen Leistungszuwachs von 22-35% an, wobei der höhere Wert erreicht wird, wenn die Daten nicht gecacht werden, der niedrigere, wenn alle Daten im Cache gehalten werden. Die geringere Steigerung bei eingeschaltetem Cache ist darauf zurückzuführen, daß der Cache auch bei dem nicht optimierten Protokoll für einen Leistungszuwachs sorgt und sich daher die Optimierung nicht so stark auswirkt. Unter realen Bedingungen liegt der tatsächliche Geschwindigkeitszuwachs also zwischen diesen beiden Eckwerten.

Probleme können auftreten, wenn zuviele word filter integriert werden sollen, da in diesem Fall, abhängig vom Prozessortyp, nicht in ausreichender Zahl Register zur Verfügung stehen, um die benötigten lokalen Variablen aufzunehmen, die stattdessen auf dem Stack

abgelegt werden müssen.

In [AP93] haben Abbott und Peterson die Leistungsgewinne bei integrierter Ausführung von Datenmanipulationsfunktionen unter Verwendung der word filter gemessen. In Tabelle 4.1 sind die Ergebnisse aufgeführt.

	Leistungszuwachs in % auf		
	DS5000	HP720	Sparc I
CKSUM+BSWAP	22-35	20-32	25-35
BSWAP+PES	28-45	40-70	46-52
BSWAP+PES+CKSUM	37-66	47-87	55-68

Tabelle 9. Leistungsgewinn durch Integration

Betrachtet wurden folgende Datenmanipulationen:

- BSWAP: Vertauschung aufeinanderfolgender Bytes
- CKSUM: Prüfsummenberechnung über ein Feld von Integerwerten
- PES: Ein dem DES-Standard nachempfunder Verschlüsselungsalgorithmus

4.2 Geteilte Datenpakete (Segregated Messages)

Die bis jetzt beschriebenen Optimierungsmaßnahmen können auf beliebige Datenmanipulationsfunktionen angewandt werden. Im speziellen Fall der Anwendung auf ein Kommunikationsprotokoll ergibt sich jedoch das Problem der verschiedenen Sichtweisen der zu integrierenden Protokollschichten auf die Anwendungsdaten. Ein Protokollkopf, der von der Protokollinstanz einer Schicht den Anwendungsdaten vorangestellt wird, gehört aus Sicht der darunterliegenden Schicht zu den Anwendungsdaten. Daher ist eine integrierte Manipulation der Daten nicht ohne weiteres möglich, da die Schicht N beim Senden einer Nachricht erst alle Datenworte des Pakets bearbeitet haben muß, bevor sie den Datenheader komplett erstellen und das gesamte Paket an Schicht N-1 übergeben kann. Beim Empfang einer Nachricht kann umgekehrt nicht Schicht N auf denselben Anwendungsdaten wie Schicht N-1 arbeiten, da zu letzteren auch der Header von Schicht N gehört.

Abbott/Peterson schlagen das Konzept der „*geteilten Datenpakete*“ (segregated messages) als Lösung für dieses Problem vor. Hierbei werden auf allen Schichten die Protokollkontrollinformationen von den eigentlichen Anwendungsdaten, also den Nutzdaten der Anwendung, getrennt, indem deren Anfang durch einen Zeiger im jeweiligen Paketkopf angegeben wird (siehe Abbildung 65). Alle Protokollschichten haben folglich eine einheitliche Sicht auf die zu übertragenden Anwendungsdaten und können den Beginn dieser Daten im Datenpaket bestimmen.

Die Verarbeitung der Anwedungsdaten derart unterteilter Datenpakete kann integriert erfolgen. Der Zugriff auf die Headerdaten der einzelnen Schichten wird allerdings nicht

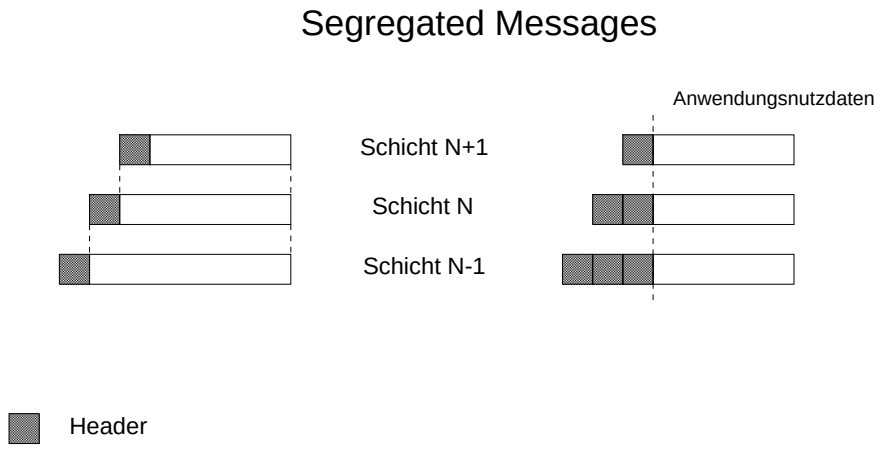

Abbildung 65. Geteilte Datenpakete

integriert, was jedoch keinen großen Einfluß auf den Leistungszuwachs, da die Anzahl der Zugriffe auf den Header im Vergleich zu den Zugriffen auf die Anwendungsdaten gering ausfällt. Allerdings müssen u. U. dieselben Operationen sowohl für die Anwendungsdaten als auch für die Headerdaten zweimal implementiert werden.

Bei bereits existierenden Protokolldefinitionen kann die Trennung der Verarbeitung von Anwendungs- und Headerdaten u. U. nicht durchgeführt werden, wenn das Protokoll eine unverzügliche Bearbeitung der gesamten Anwendungsdaten verlangt.

4.3 Der Drei-Stufen-Ansatz

In denen im vorigen Abschnitt ausgeführten Betrachtungen zur Optimierung von Protokollen wurden nur die Funktionen zur Datenmanipulation behandelt. Bei Anwendung der beschriebenen Optimierungsmaßnahmen sind jedoch auch die in Abschnitt 2.2 aufgeführten Funktionen zur Übertragungskontrolle zu berücksichtigen. Diese Funktionen, z. B. das Abschicken einer Bestätigung oder das Erhöhen der lokalen Sequenznummer, müssen in einer bestimmten Reihenfolge bezüglich der Datenmanipulationsfunktionen ausgeführt werden.

An folgendem Beispiel sollen die auftretenden Schwierigkeiten verdeutlicht werden. Betrachtet wird ein einfaches Protokoll, das folgende Operationen bietet:

– eine *send*-Operation, die eine Nachricht von Schicht N an Schicht N-1 übergibt

– eine *deliver*-Operation, die umgekehrt ein Datenpaket von Schicht N-1 zu Schicht N liefert.

Bei einem Übertragungsfehler werden die betroffenen Pakete verworfen und neu übertragen.

Implementiert man nun ohne weitere Vorkehrungen die obigen Optimierungen, kann folgende Situation eintreten:

Beim Empfang einer Nachricht beginnt die Protokollschicht N mit der Verarbeitung des eingehenden Pakets und reicht die bereits abgearbeiteten Teilstücke des Pakets an die nächste Schicht N+1 weiter, die ihrerseits mit der Verarbeitung beginnt. Auf dieser Schicht wird jetzt z. B. die Sequenzzahl der empfangenen Pakete erhöht. Die Protokollinstanz auf Schicht N stellt jedoch in der Zwischenzeit fest, daß am Ende des Datenpakets ein Fehler aufgetreten ist und wirft das Paket weg. Hier tritt nun ein inkonsistenter Zustand ein, da die Protokollinstanz auf Schicht N+1 bereits ihren Zustand geändert hat und mit der Verarbeitung des Pakets begonnen hat, dies aber nicht mehr rückgängig machen kann. D. h. es ist ein Rollback-Mechanismus notwendig, wie er auch in Datenbanksystemen Verwendung findet.

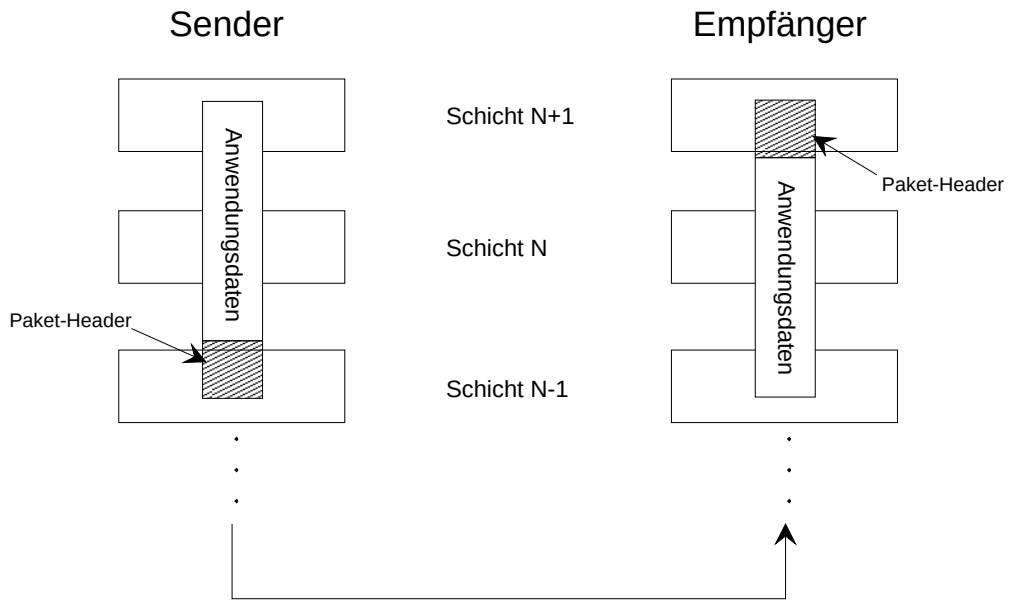

Abbildung 66. Gleichzeitige Verarbeitung eines Datenpakets in benachbarten Schichten

Die Operationen, die bei send- oder deliver-Diensten ausgeführt werden, teilen Abbott/Peterson in folgende Kategorien ein:

– *Data manipulation:* das Lesen und Schreiben von Daten

– *Header processing:* Bearbeitung der Protokollkontrollinformationen

– *External behavior:* außerhalb der Schicht sichtbare Operationen wie das Weitergeben von Nachrichten an benachbarte Schichten, das Senden von Empfangsbestätigungen und das Aktualisieren des Sequenzzählers

Diese Operationen unterliegen sowohl internen als auch externen Reihefolgebeschränkungen. Zu den internen Beschränkungen zählt z. B. die Berechnung der Prüfsumme: sie muß zuerst berechnet werden, bevor sie in den Header geschrieben werden kann.

Externe Reihenfolgebeschränkungen müssen zwischen zwei Protokollschichten beachtet werden. Beispielsweise kann, wenn auf einer Schicht eine Verschlüsselung erfolgt, eine höhere Schicht erst dann auf ihre Protokollkontrollinformationen zugreifen, nachdem die darunterliegende Schicht den Paketkopf entschlüsselt hat.

Interne Reihenfolgebeschränkungen können beim Design einer Protokollschicht berücksichtigt werden. Um externe Reihenfolgebeschränkungen zu erreichen, werden für die send- und deliver-Operationen auf den einzelnen Schichten drei Ausführungsstufen eingeführt. Diese sind

1. die Anfangsphase (*initial stage*)
2. die Datenverarbeitungsphase (*data manipulation stage*)
3. die Endphase (*final stage*)

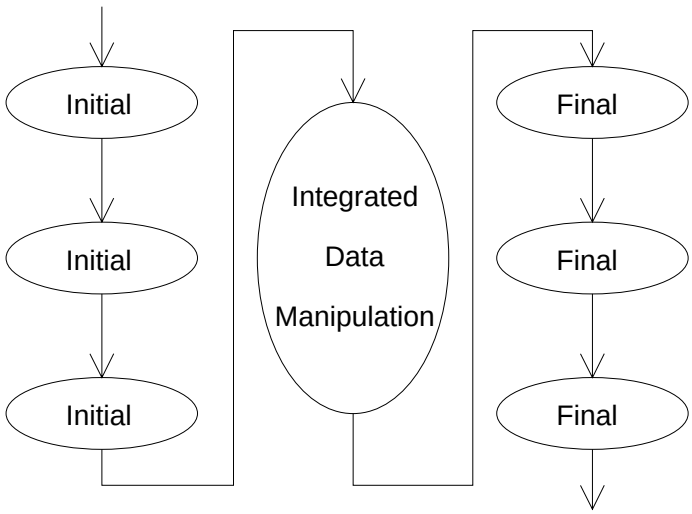

Abbildung 67. Abarbeitungsreihenfolge der einzelnen Stufen

Die Abarbeitung der Anfangsphasen der einzelnen Protokollschichten erfolgt seriell. In dieser Phase findet die Bearbeitung der Protokollkontrollinformationen beim Empfang von Datenpaketen statt. In der Datenverarbeitungsphase erfolgt die eigentliche integrierte Verarbeitung der Anwendungsdaten. In der letzten Phase werden die Protokollkontrollinformationen beim Senden von Datenpaketen erstellt und die Funktionen ausgeführt, die das externe Verhalten einer Schicht bestimmen.

Bei einer deliver-Operation an eine höhere Schicht führt diese Operationen, die ihr Verhalten nach außen bestimmen, erst in der *final stage* aus, die jedoch (s. Abbildung 67) sequentiell zu den Endphasen der übrigen Schichten abgearbeitet wird. Wird nun in einer unteren Schicht ein Paket wegen eines Fehlers verworfen, können in den Endphasen der darüberliegenden Schichten die vorherigen Aktionen rückgängig gemacht werden.

4.4 Automatische Protokollsynthese

Die in Kapitel 4 vorgestellten Techniken zur Optimierung von Kommunikationsprotokollen verstoßen durch die Integration der Protokollschichten gegen das Prinzip der Abgeschlossenheit und Unabhängigkeit von Schichten, das gewöhnlich beim Design von Protokollen gefordert wird (z. B. beim ISO/OSI-Referenzmodell). Dies macht die Implementierung, Wartung und Erweiterung des Kommunikationsprotokolls schwieriger.

Abbott/Peterson schlagen eine automatisierte Protokollsynthese vor, um aus einer modularen Protokollspezifikation eine optimierte Protokollimplementierung zu erzeugen. Dies ist vergleichbar mit dem „inlining“ von C/C++ Funktionen bzw. der Benutzung von Makros, die auf einer höheren semantischen Ebene (dem Quelltext) wie gewöhnliche Funktionen behandelt werden, tatsächlich jedoch bei jedem Aufruf in den ausführbaren Code kopiert werden.

Die Code-Teile einer integrierten Implementierung gehören entweder zu einer Protokollschicht, oder sie stellen Infrastrukturoperationen dar, zu denen Abbott/Peterson z. B. das Einlesen einer Nachricht zählen. Weiterhin haben die Komponenten der in Abschnitt 4.3 vorgestellten Protokollstufen jeder Protokollschicht gemeinsame Schnittstellen, beispielsweise haben word filter aller Protokollschichten die gleiche Schnittstelle.

Ein Prototyp des Syntheseprogramms akzeptiert Protokollspezifikationen in C Quelltextfragmenten und erzeugt eine integrierte Implementierung eines Protokollturms in C.

4.5 Hindernisse bei der Integration von Protokollen

Die Technik des Integrated Layer Processing kann nicht auf alle Protokolldefinitionen angewandt werden, da z. B. die gewünschte Funktionalität im Widerspruch zum ILP steht oder die Abbildung eines Protokollturm auf verschiedene Adreßräume erfolgt.

Wenn Protokollinstanzen benachbarter Schichten in verschiedenen Adreßräumen liegen, benötigt der Adreßraumwechsel soviel Zeit, daß eine durch Integration erreichte Optimierung nicht ins Gewicht fällt.

Weiterhin ist bedingt durch die Integration nur ein sequentieller Zugriff auf den Inhalt der Datenpakete möglich. Protokolle, die einen wahlfreien Zugriff verlangen, sind also für eine Optimierung durch ILP nicht geeignet.

Es existieren Protokolle, bei denen der Weg eines Pakets durch den Protokollturm sich zur Laufzeit ändern kann. Beispielsweise wird ein Paket, das nur weitergeleitet werden muß, nicht bis zur Spitze des Protokollturms durchgereicht, sondern es wird bereits auf einer der mittleren Schichten verarbeitet und wieder den Protokollturm hinabgeschickt. Auf diese Protokolle können die vorgesschlagenen Optimierungsmaßnahmen nicht angewandt werden, da sie eine statische Abfolge der Protokollschichten erfordern.

5 Fazit

Die Technik des Integrated Layer Processing läßt sich nicht auf alle Arten von Protokollen anwenden. Insbesondere bei der Optimierung von bereits existierenden Protokollen treten u. U. Schwierigkeiten auf, da diese teilweise die Voraussetzungen nicht erfüllen, die für eine Anwendung des ILP notwendig sind. Weiterhin kann man feststellen, daß die Geschwindigkeitssteigerung stark von der eingesetzten Hardware, insbesondere von der Größe und Auslastung des 2nd-Level-Caches, abhängig ist.

Bei der Entwicklung von neuen Protokollen koennen diese jedoch von vorneherein so ausgelegt werden, daß sie den Anforderungen von ILP genügen und ihre Optimierung keine Probleme aufwirft.

Die Anwendung des Application Level Framing, die auch unabhänigig vom ILP erfolgen kann, stellt eine gute Möglichkeit dar, eine häufige Segmentierung und Reassemblierung von Datenpaketen zu vermeiden und trägt so zur Erhöhung des Protokolldurchsatzes bei.

Ansatzpunkte für Hardware/Software Codesign in Kommunikationssystemen

Dimitrios Nikolaidis

Kurzfassung

Im folgenden Beitrag wird erläutert, was Codesign ist und welche Gründe es nötig machen, daß Codesign betrieben wird. Es werden Techniken und Methoden des Codesign sowie typische Codesign Probleme vorgestellt. Ferner werden die Protokollspezifikations- und -validationssprache Promela sowie ihre Hardware- und Softwarecompiler beschrieben. Der Beitrag endet mit der Behandlung eines Designbeispiels.

1 Einleitung: Was ist Codesign?

Hardware–Software (HW/SW) Codesign unterscheidet sich von der konventionellen Entwicklung von Hardware und Software dadurch,

daß die Entwicklung der HW und der SW stark voneinander abhängen.

Dies bedeutet, daß Entscheidungen, die das Hardwaredesign angehen, einen großen Einfluß auf das Design von Software haben und umgekehrt.

„Co“ bedeutet in erster Linie *zusammen, gemeinsam.* Hardware und

Software werden von einem Team parallel entwickelt, das den Designprozeß

führt, koordiniert und überwacht. Verschiedene Teams von Experten

entwickeln Systemkomponenten und lösen dann die Probleme, die dabei entstanden sind. *„Parallel“* ist in diesem Fall als *gleichzeitig* zu verstehen, weil Hardware und Software *zur gleichen Zeit* und abhängig voneinander entwickelt werden.

Codesign ist eigentlich nichts Neues. Allerdings entwickeln sich die verschiedenen Methoden zum HW/SW–Codesign erst jetzt ([Wol93]). Das Interesse

am Codesign und an gemeinsamer Entwicklung von Hardware und Software ist aus

folgenden Gründen in den letzten Jahren erneut gewachsen ([Sub93]):

- die wachsende Komplexität von Anwendungen, die eingebettete Systeme[40] einsetzen.
- die Notwendigkeit, die Kosten des Designs und des Tests solcher Systeme aus marktwirtschaftlichen Gründen zu senken und möglichst

  gering zu halten.
- die Fortschritte in wichtigen technologischen Bereichen wie Systemspezifikation, Simulationsumgebungen, formalen Methoden zum Design und zur Verifikation, Synthese und CAD (computer-aided-design).

[40] mit eingebetteten Systemen sind hier *reactive systems* gemeint, also Systeme, die zu schnellen Reaktionen fähig sein müssen, d.h. *Echtzeitsysteme.*

2 Techniken und Methoden

Traditionell basiert der Entwurf von heterogenen Systemen (also nicht–reinen HW– oder SW–Systemen) auf mehreren, aufeinanderfolgenden Entwicklungsschritten, nach Abschluß derer die Spezifikationen erfüllt sein sollen und ein Prototyp des Systems erstellt wird. (Heute wird auch Simulation eingesetzt, um ein System zu Testen.)

Im heutigen Einsatz basiert Codesign auf Methoden und Techniken,

die früher bereits erfolgreich eingesetzt wurden. Fortschritte werden in den Bereichen der Kommunikation zwischen den verschiedenen Tools, der tool–to–tool

Schnittstellen sowie der Partitionierung von HW und SW gemacht.

Man sieht also, daß die große Anzahl der Bereiche einen relativ breiten

Spielraum läßt, wie Codesign schließlich betrieben werden kann. Demzufolge gibt es keine universell einsetzbare Technik für den Entwurf von

HW/SW–Systemen. Verschiedene Ansätze sind vorhanden, aber es gibt natürlich auch einige Aspekte (s. Abb. 68 und [KL93]), die einen festen Rahmen bilden.

Im Verlauf dieses Abschnitts sollen dieser feste Rahmen sowie zwei Codesign–Ansätze

etwas näher betrachtet werden (s. auch [BR95]).

2.1 Rahmenstruktur beim Codesign

Der feste Rahmen einer Codesigntechnik besteht aus den folgenden Schritten :

- Analyse der Anforderungen und Bedingungen
- Systemspezifikation
- (Co–)Synthese von Hardware und Software
- (Co–)Simulation und Verifikation

Analyse der Anforderungen und Bedingungen In diesem Schritt werden die wichtigsten Charakteristika des Systems – natürlich

basierend auf den Anforderungen des Kunden und des (künftigen) Benutzers –

definiert. Die Bedingungen und Anforderungen des Kunden sind aber meistens

ungenügend bzw. nicht komplett, inkonsistent und weisen große Lücken auf.

Also ist das Ziel dieser Analyse herauszufinden, wo Information fehlt und wo Inkonsistenzen vorkommen. Designaspekte, die von der Analyse erfaßt werden, sind z.B. Marktwirtschaftlichkeit, Verbrauch (z.B.

von Strom), Echtzeitverhalten, Produktgröße, Kosten der Entwicklung und der

Produktion, Verläßlichkeit, Reparaturmöglichkeit, benutzte Stoffe usw.

Man sieht also, daß diese Analyse ein sehr wichtiger Schritt für die

Erstellung des Endproduktes darstellt, und daß sie sehr detailliert sein

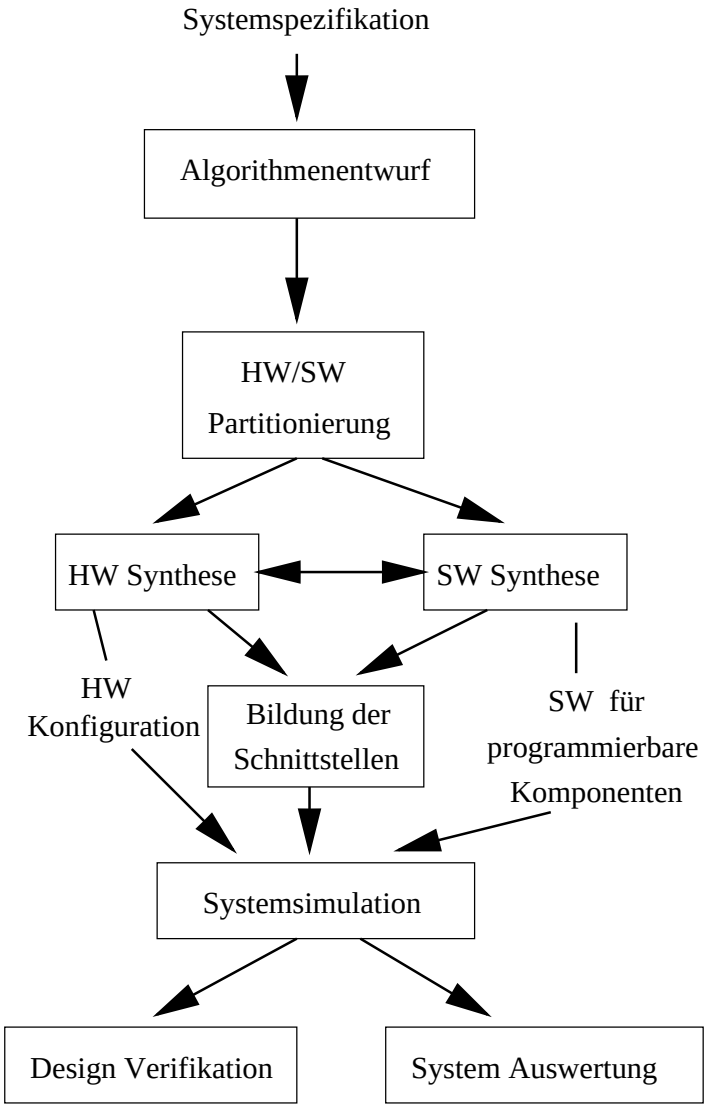

Abbildung 68. Die Rahmenstruktur beim Codesign.

muß, damit später bei (oder nach) der Erstellung des Endproduktes keine Fehler oder Schwierigkeiten entstehen, die auf fehlende Information zurückzuführen

sind.

Systemspezifikation Die Systemspezifikation ist das Resultat der Analyse.

Sie ist formal ausgedrückt und geeignet für elektronische Verarbeitung (so daß ein Designer Modellierungsalgorithmen daraus entwerfen kann).

Synthese von HW/SW Die Synthese ist das größte Problem und der wichtigste Schritt zur Entstehung eines Systems. Sie besteht aus zwei Schritten: der Partitionierung von HW und SW sowie dem Zusammenbinden der Teile.

Bei der Partitionierung von Hardware und Software muß sich der Designer entscheiden, welche Komponenten des Systems in Hardware realisiert werden

und welche in Software implementiert werden. Es ergibt sich die Frage, wie soll ein

Designer entscheiden, wie HW/SW zu partitionieren sind.

Vollautomatische Systempartitionierung ist leider noch nicht möglich. Anhand von automatisch herleitbaren Schätzungen kann der

Designer seine Entscheidungen treffen. Diese Technik ist oft befriedigend,

aber nicht für industrielle Anwendungen ausreichend.

Meistens werden deswegen deterministische und statistische Techniken verwendet

oder Methoden, die aus anderen „verwandten“ Bereichen wie VLSI–Design bekannt sind. Die deterministische Schätzung der Partitionierung erfordert

ein vollspezifiziertes Modell, bei dem alle Datenabhängigkeiten[41] behoben und alle Kosten der verschiedenen Komponenten bekannt sind.

Diese Methode führt zu sehr guten Partitionierungen, versagt aber, falls

Daten nicht bekannt oder Komponenten nicht verfügbar sind. Dann ist eine

statistische Schätzung nötig, die auf Analyse ähnlicher Systeme und

verschiedener Designparameter aufbaut ([Buc93]).

Ein anderes, auftauchendes Problem ist, *wann* diese Partitionierung erfolgen soll. Sie kann nämlich auf verschiedenen Abstraktionsebenen oder verschiedenen Stufen des Designs stattfinden. In der Industrie wird frühe Partitionierung bevorzugt, weil somit die ganze Entwicklung vorgeplant werden

kann und Designentscheidungen erleichtert werden. Dies erfordert aber, daß nur wenige (oder gar keine) Designänderungen (z.B. auf Grund von anders

gestellten Anforderungen des Kunden) erlaubt sind. Späte Partitionierung erlaubt dagegen solche (sogar häufige) Änderungen und erleichtert die Lösung der Probleme, die dabei auftreten.

Ein weiteres Problem sind die entstehenden Kosten und das Realzeitverhalten,

welches das System haben soll. Falls die Kosten minimiert werden sollen, wird eine Softwareimplementierung bevorzugt. Sollen Echtzeitabläufe garantiert

werden, so ist eine Hardwarerealisierung notwendig; die dabei entstehenden, hohen Kosten sind zweitrangig und müssen in Kauf genommen werden.

Nachdem Hardware und Software partitioniert worden sind, müssen sie auch

zusammengebunden werden, damit ein einheitliches System entsteht. Dies erfolgt über die Schnittstellen. Es müssen also HW/SW–Schnittstellen

definiert werden, die es ermöglichen, daß der HW– mit dem SW–Bereich

kommunizieren und zusammenarbeiten kann.

Simulation und Verifikation Anstatt einen Prototyp zu bauen, werden heutzutage Simulationen durchgeführt. In der Regel wird dabei die Hardware simuliert. Dafür sind natürlich

[41] Es

gibt drei Arten von Datenabhängigkeiten: i) echte Datenabhängigkeiten,
ii) Gegenabhängigkeiten und iii) Ausgabeabhängigkeiten. Mehr dazu in [GUS95], S. 55 ff.

HW–Simulationsprogramme notwendig und auch vorhanden.

Simulation ist ein Mittel, das System zu überprüfen und eventuell vorhandene

Fehler zu beseitigen. Mit Simulation kann betrachtet werden, wie sich das System intern und extern (also im Input/Output–Bereich) verhält ([DM93]). Die Simulation ist also ein Mittel, um die Funktionalität eines HW/SW–Systems

zu testen.

Um eine aussagekräftige Simulationen durchzuführen, ist es wichtig, die Hardwaresimulationsumgebung mit der tatsächlich benutzten Software zu verbinden ([TAS93]).

Es ist natürlich selbstverständlich, daß die Simulation die Richtigkeit (engl. correctness) des Systems nicht garantieren kann, weil nur eine

begrenzte Anzahl von I/O–Mustern betrachtet und analysiert werden können. Ist das entworfene System groß, so ist es nahezu unmöglich, alle I/O–Muster

zu erfassen und zu simulieren; die Simulation ist in diesem Falle ungenügend.

Trotzdem wird die Simulation eingesetzt, weil der

Bau eines möglicherweise fehlerhaften Prototypes mit sehr hohen Kosten verbunden ist und weil es fast unmöglich (und auch unwirtschaftlich) ist, nach jeder Behebung eines (durch Simulation und Verifikation gefundenen)

Fehlers im HW–Bereich einen neuen Prototyp zu bauen. Zu beachten ist, daß bei der

Simulation deutlich zwischen der Eingabe, der betrachteten Ausgabe des

Modells und dem Modell selbst unterschieden werden muß. Falls diese Unterscheidung gemacht wird, kann ein Modell erstellt werden, das je

nach experimenteller Analyse verschiedene Spezifikationen erfüllt.

2.2 Der konventionelle Ansatz

Zusätzlich zum betrachteten festen Rahmen kommen natürlich auch andere Schritte hinzu

(s. Abb. 69), wie die Hardware–Synthese und Konfiguration. Dabei wird die Hardware

entworfen und konfiguriert, die den von der Software erzeugten Code ausführt. Der entsprechende Schritt im SW–Bereich ist die SW–Erzeugung

und Parametrisierung; ein Schritt, in dem die Software für die entworfene

und konfigurierte Hardware geschrieben wird.

Was hier widersprüchlich erscheint, ist die Tatsache, daß in der HW–Synthese und Konfiguration die Hardware für die schon entworfene Software entworfen wird und umgekehrt. Dies ist aber die zentrale Idee im HW/SW Codesign. Hardware und Software werden gleichzeitig und abhängig voneinander entwickelt. Zu beachten ist, daß die Softwareerzeugung

stark von der Hardwarearchitektur abhängt. Dies führt dazu, daß diese Architektur früh genug gewählt und festgelegt werden muß. Ein weiterer wichtiger Schritt ist die Bildung von Schnittstellen, die

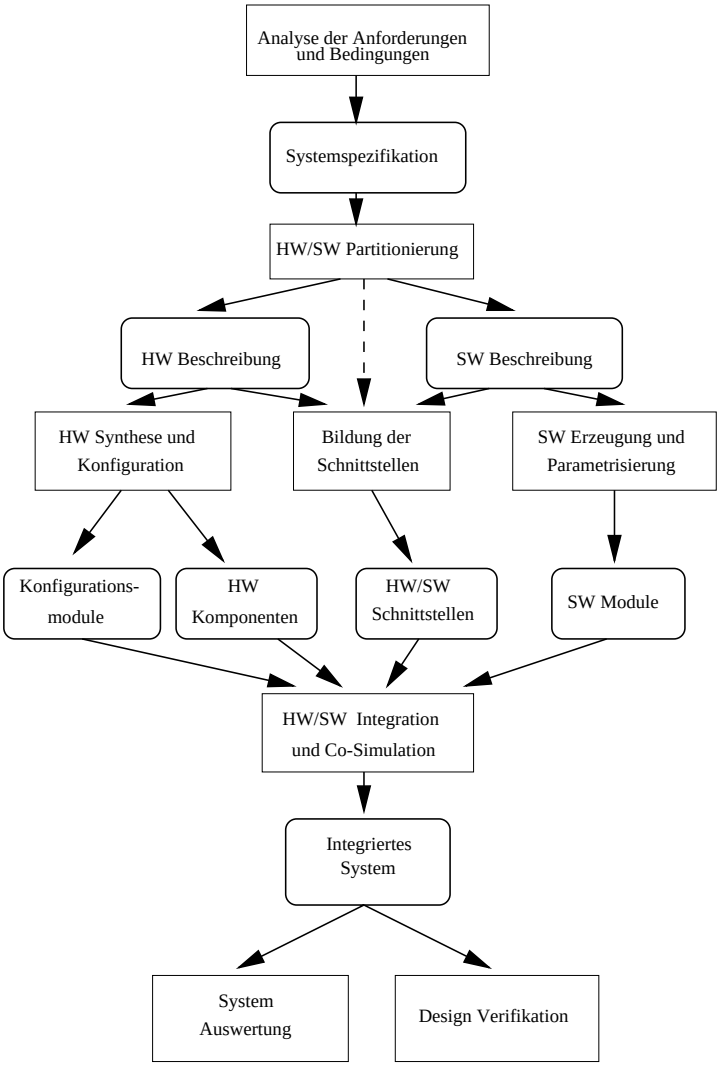

Abbildung 69. Der konventionelle Ansatz beim HW/SW Codesign.

zur HW/SW–Synchronisation dienen. Signalaustausch (im HW–Bereich), Semaphore[42] (im SW–Bereich) oder Interrupts werden dabei verwendet. Danach werden die Komponenten integriert und simuliert. Dabei entsteht das *integrierte*

System. Schließlich wird eine Design–Verifikation ausgeführt, die sichern

soll, daß das Ergebnis die Spezifikationen erfüllt.

2.3 Modellbasierter Ansatz

Die Abb. 70 zeigt einen Ansatz, der einer aus der Softwaretechnik bekannten

Methode stark ähnelt. Es werden Modelle (nacheinander) entworfen, die validiert und verfeinert (engl. refined) werden. Man sagt, ein Modell

sei *valid*, falls sein Input/Output–Verhalten konsistent zur Spezifikation des Systems ist. Im Verfeinerungsschritt (engl. refinement

[42] Die Realisierung des wechselseitigen Ausschlusses erfolgt in der Praxis oft durch *Semaphore*. Mehr dazu in [Eng89], S. 406

step) wird das System auf eine weniger abstrakte Ebene verfeinert. Diese schrittweise Verfeinerung ersetzt die Partitionierung und die Integration. Dies macht das Problem der Partitionierung nicht leichter. Die Handhabung der Designaufgabe ist jedoch leichter geworden, weil die validierten Objekte durch wohldefinierte Schnittstellen synchronisiert werden. Nachdem das verifizierte und simulierte System entstanden ist, erfolgt die Zuordnung der Technologie (engl. technology assignment).

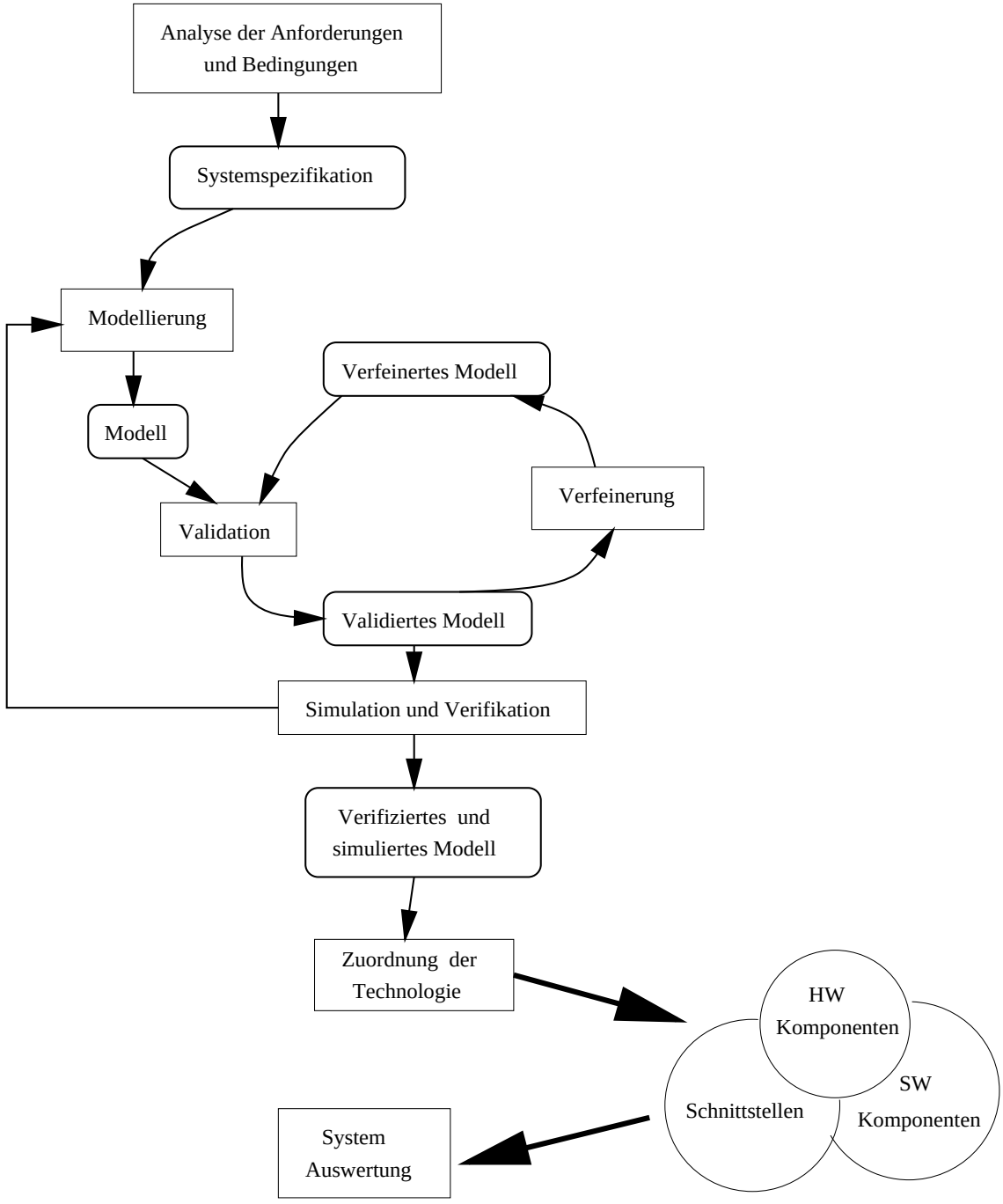

Abbildung 70. Der modellbasierte Ansatz beim HW/SW Codesign.

2.4 Vor- und Nachteile der beiden Ansätze

Generell kann keiner der beiden Ansätze als besser betrachtet werden. Der modellbasierte Ansatz ist eigentlich fortschrittlicher als der
konventionelle. Der schrittweise Entwurf des Modells macht es möglich, Änderungen der Spezifikation besser verarbeiten zu können. Der große Unterschied
besteht im Zeitpunkt der Partitionierung der HW/SW: im konventionellen Ansatz wird die Partitionierung früh gemacht, im modellbasierten fast am Ende. Die späte Partitionierung kann zu besseren Lösungen, geringeren
Kosten und schnellerem Entwurf führen. Dies bedeutet aber nicht, daß der
konventionelle Ansatz schlecht ist. Er hat den großen Vorteil, daß
die Entwurfsschritte sehr früh deutlich werden und somit (vielleicht)
stabilere Systeme entworfen werden können. Der konventionelle Ansatz wird in der Industrie sehr häufig eingesetzt, während der modellbasierte
Ansatz für Produkte, die sehr oft modifiziert und geändert werden
(z.B. aufgrund von geänderten Anforderungen, die vom Markt stark abhängen),
verwendet wird.

3 Codesign Probleme

Obwohl es heute viele CAD–tools gibt, müssen noch immer viele Codesign Probleme von Hand gelöst werden. In diesem Abschnitt sollen drei typische
Codesign Probleme betrachtet werden, ohne dabei auf große Einzelheiten einzugehen.
Eine ausführliche Behandlung dieser Probleme ist in [DM94] zu finden.

3.1 Befehlssatz–Prozessoren

Befehlssatz–Prozessoren (instruction–set–processors) sind Prozessoren, die Sätze von Instruktionen
ausführen. Diese Prozessoren sind das „Herz“ von Informationsverarbeitungssystemen
und müssen zuverlässig und von hoher Lebenserwartung sein, weil sie in
der Industrie eingesetzt werden und somit industrielle Entschlüsse beeinflussen. Beim Design von solchen Prozessoren gibt es drei Fragen, die HW/SW–Betrachtungen erfordern: Befehlssatz–Auswahl, Cacheentwurf und
Pipelinekontrolle.

Befehlssatz–Auswahl Die Entwerfer müssen Art, Anzahl und Format der Instruktionen
festlegen, die auszuführen sind, wenn ein Befehlssatz ausgewählt

wird. Die Kompatibilität dieser Prozessoren mit anderen Prozessoren beeinflußt diese Entscheidung stark. Auf der anderen Seite ist dieses Problem
weniger zwingend und abhängig von der Festlegung der Architektur von
ASIP's (Application Specific Instruction Processors). Folglich bringt die Befehlssatz–Auswahl die Auswertung von Kosten und Nutzen mit
sich. Die Entwerfer setzen dabei oft Auswertungsprogramme (benchmark programs) ein,
um den Nutzen zu „messen“, während die benutzte Hardware die Kosten
bestimmt.

Cacheentwurf Die wichtigsten Probleme hier sind die Größe des Cachespeichers und die
Garantie der Kohärenz. Die Designer müssen Abläufe mit verschiedenen Größen des Cachespeichers simulieren, um die angemessene schließlich bestimmen zu können.

Pipelinekontrolle Die Kontrolle einer Pipeline sollen verhindern, daß es
zu Pipeline–Hazards kommt (mehr dazu in [GUS95], S. 57 ff). Dazu werden, falls nötig, HW– oder SW–Techniken eingesetzt. Ein HW–Mechanismus
würde die Pipeline „spülen“, also entleeren. Eine SW–Lösung hingegen würde
die Reihenfolge der Instruktionen geeignet ändern und (falls nötig) no–operations
(NOP's) einfügen. Es ist klar, daß die getroffene Entscheidung, über die einzusetzende Technik die Leistung des Prozessors beeinflußt.

3.2 Eingebettete Systeme

Diese Systeme sind meistens einer Anwendung gewidmet und können in Größe und Komplexität stark variieren. Ein sehr beschränktes, eingebettetes System
wäre z.B. ein Waschmaschinenkontroller, ein komplexes beispielsweise ein System
zur Führung und zur Kontrolle eines Flugzeugs. Wie bereits erläutert, sind unter eingebetteten Systemen *reactive systems* zu verstehen. Dies sind Systeme, die auf das Umfeld reagieren müssen, meistens sogar in vorbestimmten zeitlichen Intervallen. Deswegen sind sie häufig als Echtzeitsysteme zu betrachten. Beispiele wären ein Verbrennungskontroller
in einer Maschine, ein Roboterkontroller, ein Mobiltelefon usw. Die Größe, das Gewicht und der Preis (und natürlich viele anderen Faktoren) machen es nötig,
daß ein eingebettetes System häufig neu entworfen werden muß. Dabei ist Codesign unentbehrilch.

3.3 Signalprozessoren

Signalprozessoren spielen eine sehr wichtige Rolle im Telekommunikationsbereich,
da dort Hochleistungsprozessoren mit günstigem Preis unentbehrlich sind.

Signalprozessoren sind meist einer (oder ganz wenigen) Aufgabe(n) gewidmet
und führen demzufolge entsprechende Programme aus. Der Designer muß den
Befehlssatz so auswählen, daß er die schnelle Ausführung von bestimmten Programmen unterstützt. Die Auswahl des Befehlssatzes beeinflußt aber, wie bereits erläutert, den Entwurf von Hardware und Software, was den Entwurf eines Signalprozessors
zu einem Codesignproblem macht.

4 Promela

Um Kommunikationsprotokolle zu beschreiben, benutzt man Promela, eine
parallele Programmiersprache, die von Gerard Holzmann und an den AT&T Bell Laboratories für die Spezifikation und Validation von Protokollen entwickelt wurde.
Eine ausführlichere Beschreibung von Promela ist in [WOB93] zu finden.
Ein Promela–Compiler besteht aus drei Modulen: einem Eingabemodul (front end) und zwei Ausgabemodulen (back ends). Das eine Ausgabemodul erzeugt C++ Code (geeignet für Kompilation auf eingebetteten Kontrollern), das andere erzeugt gate–level
Entwürfe, die als Eingabe von Hardware–Entwurfssystemen oder FPGA–tools (Field Programable Gate Arrays) geeignet sind.
Das eine Ausgabemodul übersetzt ein Promela–Programm in C++. Zusammen mit zusätzlichem C++ Code (nötig um externe Hardwareschnittstellen zu unterstützen)
kann dieser Code in eine ausführbare Version der Promela–Spezifikation kompiliert werden. Die ausführbare Datei heißt dann *Promela Executable*
(Pex). Das andere Ausgabemodul erzeugt eine Stromkreisbeschreibung in der Hardwarebeschreibungssprache VHDL (VHDL: VHSIC Hardware Description Language).
Die einzigen, nicht–automatisierten Schritte des Prozesses
sind der Entwurf des Promela–Codes und das Zusammenbinden der Hardware– und Softwareteile des Systems.

4.1 Beschreibung von Promela

Ein Promela–Programm kann aus mehreren Prozessen bestehen. Promela–Prozesse
werden parallel ausgeführt und können miteinander synchronisiert sein oder sie kommunizieren, indem Nachrichtenkanäle (engl. message channels) oder globale Variablen benutzt werden. Prozesse werden durch Prozeßtypdeklarationen
(die dann *proctype* heißen) definiert, die den Namen, die Parameter und den Körper (engl. body) des Prozesses spezifizieren. Der folgende Code deklariert
einen Prozeßtyp namens *Alpha* mit zwei Parametern und einer lokalen Variable.

```
proctype Alpha (int x, y)
   {int z; z = x+y; printf("%d", z)}
```

Somit wird der Typ des Prozesses deklariert; der Prozeß wird nicht ausgeführt.
Dies geschieht durch seinen Aufruf im initialen Prozeß *init*. Beispiel:

```
init {run Alpha (3, 5);
   run Alpha(2, 9)}
```

Der Prozeß *init* hat an sich keine Parameter. Offensichtlich weist Promela
große Ähnlichkeiten mit C auf: Die Prozesse sind Funktionen und der
init–Prozeß entspricht dem Hauptprogramm (main) in C.
Im vorigen Beispiel hat der Prozeß *init* zwei Kopien des *Alpha*–Prozesses gestartet. Alle drei Prozesse laufen parallel. Ein
Prozeß terminiert nicht, bis alle seinen Unterprozesse terminiert haben.
Befehle (engl. statements) in einem Prozeß können ausführbar oder nicht–ausführbar sein. Kann ein Befehl nicht ausgeführt werden, so wird
der Prozeß blockiert, bis der Befehl ausführbar wird. Kanäle können
als `typed` oder `untyped` deklariert werden. Im folgenden Beispiel wird `a` als `untyped` deklariert und `b` als ein Kanal, der Nachrichten des Typs `byte` transportiert. Maximal 10 solche Nachrichten können von b gespeichert
werden.

```
chan a;
chan b=[10] of {byte};
```

Ein Sendebefehl schickt dem Kanal b im folgenden Beispiel eine Nachricht
mit dem Wert 23:

```
b!23;
```

Ein anderer Prozeß könnte diesen Wert auf folgende Weise empfangen:

```
b?x;
```

Der Prozeß erhält einen Wert vom Kanal und setzt ihn in die Variable x.
Ein Sendebefehl ist ausführbar, falls im Kanalpuffer genügend Platz vorhanden ist. Ein Empfangebefehl ist ausführbar, falls der Kanalpuffer
nicht leer ist.
Der if–Befehl erlaubt eine bedingte Auswahl von verschiedenen Alternativen.
Jede Alternative besteht aus einem Wächter, dem Befehle folgen können. Im Falle, daß ein oder mehrere Wächter ausführbar sind, wird einer

von ihnen nichtdeterministisch ausgewählt und ausgeführt. Falls keiner ausführbar ist, blockiert der Prozeß, bis ein Wächter ausgeführt werden

kann.

Der do–Befehl hat eine ähnliche Syntax, wird aber ausgeführt, bis ein

break–Befehl kommt. Ein Beispiel ist in der Abb. 71 zu sehen.

```
i=1;
do
:: (i<=10) -> printf("%d\n", i);
              i=i+1
:: (i> 10) -> break
od
```

Abbildung 71. Ein Beispiel zum do–Befehl.

4.2 Der Software Compiler

Das eine Ausgabemodul erzeugt zwei Dateien: die Pex.h– und die Pex.cpp–Datei.

Die Pex.h–Datei enthält Deklarationen für jede C++–Klasse, die produziert wird. Eine Klasse wird für jede proctype–,

jede Kanal– und typedef–Deklaration generiert. Die Pex.cpp–Datei

enthält die Deklarationen für alle globalen Daten im Promela Programm. Hinzu kommt der Code für die process–bodies und die main–Funktion. Die main–Funktion erzeugt einen initialen Prozeß und schickt ihn dem Prozeßplaner.

Darauf wird der Prozeßplaner aufgerufen, damit er die aktiven Prozesse laufen läßt.

Weil mehrere aktive Prozesse gleichzeitig auf eine Variable zugreifen können, ist es wichtig, einen Mechanismus zu haben, der dieses Problem lösen kann. Der C++ Code benutzt den lazy copy mechanism, um deklarierte Variablen zu implementieren (so daß auch große Datenstrukturen, die sich

selten ändern, effizienter zu behandeln sind). Er funktioniert folgendermaßen: Jede Variable hat einen Referenzzähler und ein Prozeß darf die Variable überschreiben, falls der Referenzzähler

1 ist. Ist dies nicht der Fall, so muß eine neue Kopie der Variable mit Referenzzähler 1 erstellt werden, damit die Variable überschrieben

werden kann. Der Referenzzähler der ursprünglichen Kopie wird um 1 verringert. Ist der Referenzzähler 0, so wird der für die Variable allokierte

Speicher freigesetzt.

Prozeßplanung Der Prozeßplaner kontrolliert die Ausführung von Prozessen in einem Pex Programm. Die main–Funktion schickt dem Prozeßplaner den initialen Prozeß und

ruft den Prozeßplaner auf, damit er die Prozesse laufen läßt. Nachdem alle Prozesse terminiert haben, wird die run–Funktion beendet und
das Pex Programm terminiert.
Der Prozeßplaner benutzt eine zyklische Schleife von aktiven Prozessen
und ruft die body–Funktion des Prozesses am Kopf der Schlange auf. Zwischen
den Prozeßaufrufen prüft er, ob ein Interrupt–Handler
Ergebnisse hat, die vom Prozeßplaner zu behandeln sind. Falls ein Prozeß terminiert, wird er aus der Schlange entfernt und der von ihm belegte Speicher freigesetzt. Wird ein Prozeß blockiert, so wird er vom Prozeßplaner aus der Schlange entfernt. Er kann der Schlange später wieder hinzugefügt werden, indem die wake–up Funktion des Prozeßplaners
aufgerufen wird.

Interrupt handling Man kann Hardware Interrupts mit Hilfe der Runtime–Bibliothek einbinden. Die
Echtzeituhr, die seriellen Ports und die Tastatur können Interrupts erzeugen, auf die Pex Programme antworten. Für die Kommunikation zwischen
den Interrupt–Handler und dem Rest des Pex Programms wird die globale Variable
intrpt_flags benutzt. Ein Interrupt–Handler darf aus Stabilitätsgründen
keine Aktionen auf eigene Initiative ausführen (wie z.B. wake–up eines
Prozesses); das übernimmt der Prozeßplaner, dem die entsprechende
Information vom Interrupt–Handler mitgeteilt wird.

Schnittstellen zur Hardware Die Promela Kommunikationskanäle kommunizieren mit der externen Hardware. Promela Prozesse, die Nachrichten bzw. Meldungen über die externen Kanäle senden
und empfangen, kommunizieren mit einem Hardwareelement. Die
Kanäle müssen extern durch Benutzung des Schnittstellenbefehls *interface_chan* deklariert werden.

4.3 Der Hardware Compiler

Promela Befehle und Ausdrücke werden auf primitive, elementare Komponenten wie Flip–Flops und logische Gatter abgebildet. Jeder Befehl hat eine Eingabe S (für Start), die eine Befehlsausführung veranlaßt, und drei Ausgaben T, F und B. Ein Puls auf T bedeutet, daß der Befehl erfolgreich
ausgeführt wurde (T für Terminated). Falls auf F ein Puls liegt, heißt es,
daß der Befehl (aus welchen Gründen auch immer) nicht ausgeführt wurde
(F steht für Failed). Ein Puls auf B zeigt an, daß ein `break`
vorliegt: die do–Schleife muß sofort verlassen werden. In der Abb. 72 sind die Realisierungen der einfachen Befehle `skip` und `break`

zu sehen.

Abbildung 72. Die Befehle a) *skip* und b) *break*.

Außerdem werden Addition und Subtraktion (aber nicht immer Multiplikation und Division), logische Funktionen sowie bitweises `UND, ODER` und `NICHT` von Datenwörtern unterstützt.

Variablen werden als Register implementiert. Jede Variable hat einen Lese–Port (read port) und einen Schreib–Port (write port). Variablen werden auf 0 initialisiert, aber Ausdrücke wie z.B. `byte x=5` sind auch erlaubt.

Die Sendeschnittstelle eines Kanals besteht aus den *Request* und *Acknowledge* Leitungen. Die Emfangschnittstelle hat eine *Request*, eine *Acknowledge* und eine *Probe* Leitung. Hinzu kommt natürlich eine Leitung, durch die Daten geschickt bzw. empfangen werden.

Falls Synchronisation initiiert werden soll, setzt der Sender oder der Empfänger `Req` auf high. Geht `Ack` auf high, so erfolgte die Synchronisation. Falls `Ack` auf low bleibt, hat es nicht funktioniert: Es muß erneut versucht werden.

Zu beachten ist, daß die Hardwarekanäle keinen Speicher besitzen, während die Softwarekanäle in der Lage sind, Nachrichten bzw. Meldungen zu speichern.

5 Beispiel: Alternating-Bit-Protokoll

Ein einfaches Kommunikationssystem besteht aus einem Software–Prozeß, der Daten über einen zuverlässigen Kanal sendet, und einem Hardware–Prozeß, der den zuverlässigen Kanal erzeugt, indem er das Alternating-Bit-Protokoll auf zwei unzuverlässige Kanäle anwendet. Die Schnittstelle zwischen Hardware und Software muß entworfen werden (s. Abb. 73).

Das Promela–Programm der Software Komponente liest Zeichen von der Standardeingabe ein und schickt die Bytes über die HW/SW–Schnittstelle dem Hardware–Kanal (*hw_in*). Die HW/SW–Schnittstelle kann als

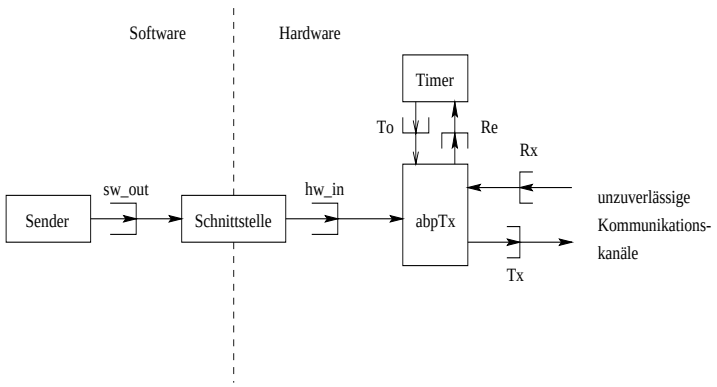

Abbildung 73. Das Kommunikationssystem.

ein einziger Prozeß modelliert werden. Dieser Prozeß empfängt Bytes vom Software–Kanal *sw_in* und sendet sie dem Hardware–Kanal *hw_in*. Das Promela–Programm der Hardware Komponente erhält Bytes vom Hardware–Kanal und schickt diese dem externen Kanal (*Rx, Tx*), wobei das Alternating–Bit–Protokoll benutzt wird. Jede ausgehende Nachricht besteht aus den Datenbytes und einem zusätzlichen Bit. Jede Nachricht muß bestätigt werden, sonst wird sie nach einem Timeout erneut gesendet.

Für die Handhabung des Timeouts ist ein Prozeß *timer* notwendig, der durch einen Zähler realisiert werden kann. Dabei wird in jeder Taktperiode eine Variable namens `counter` um 1 erhöht. Erreicht `counter` den Wert 255, signalisiert der Prozeß *timer*, daß der Timeout vorbei ist. Der Zähler kann auf 0 gesetzt werden (etwa wenn die Bestätigung gekommen ist), indem der Transmitprozeß (Prozeß *abpTx*, in Abb. 76 zu sehen) diesen Wunsch über den Reset–Kanal *Re* signalisiert. In Abb. 74 ist der Prozeß *timer* zu sehen.

```
{ bit x = 0;
  do
  :: Re?x -> counter = 0
  :: (counter == 255) -> if :: To!x :: Re?x fi;
                            counter = 0
  :: counter + counter + 1
  od
}
```

Abbildung 74. Der Prozeß *timer*.

Zu beachten ist folgendes: Promela unterstützt keine Synchronisation, wenn dabei keine Daten übertragen werden. Daher muß der Transmitprozeß eine Null–Nachricht dem Prozeß *timer* senden. Es müssen also '0'-Bits über die Kanäle `Re` und `To` gesendet werden.
Die HW/SW–Schnittstelle ist als ein Promela–Prozeß modelliert, der Bytes vom Kanal *sw_out* empfängt und diese dem Kanal *hw_in* weitersendet.

Der Prozeß ist in Abb. 75 zu sehen.

```
proctype  glue()
{  byte x;
   do
   :: sw_out?x -> hw_in!x
   od
}
```

Abbildung 75. Der Prozeß *glue*.

```
hardware chan hw_in = [0] of {byte},
              To = [0] of  {bool},       /* Timeout */
              Re = [0] of  {bool},       /* Setze Timer auf 0 */
              Rx = [0] of  {bool},       /* Rx Kanal */
              Tx = [0] of  {byte, bool}; /* Tx Kanal */


hardware proctype abpTx()
{
  bool r=0, s=0;
  byte mesg=0;

  do
  :: hw_in?mesg ->
           do
           Tx!mesg.s;
              Re!0;
              if
              :: Rx?r -> if
                         :: (r==s) -> break
                         :: (r!=s)
                         fi
              :: To!0
              fi
           od;
           s= !s
  od
}
```

Abbildung 76. Der Transmitprozeß.

6 Fazit

Obwohl Codesign nichts Neues ist, wird es erst jetzt als Entwurfsstrategie von der Industrie akzeptiert. Der Grund dafür ist, daß es keine einheitliche, standardisierte Codesign–Technik gibt. Die zwei betrachteten Ansätze lassen einen sehr breiten Spielraum. Der modellbasierte Ansatz ist eleganter und bietet dem Designer eine größere Flexibilität an, als der konventionelle Ansatz. Für die Unterstützung von Codesign beim Entwurf

von Kommunikationssystemen gibt es die Spezifikationssprache Promela. Der Beitrag wird mit einer kurzen Beschreibung von Promela und der Behandlung eines Beispiels abgeschlossen.

Traffic Shaping in ATM-Netzen

Matthias Jacob

Kurzfassung

Traffic Shaping in ATM-Netzen entspricht der Modellierung des ausgesendeten Datenstromes für die ausgehandelten Verkehrsparameter. Diese Funktionalität ist in den Endgeräten implementiert. In diesem Beitrag wird zunächst auf die Grundverfahren und deren Optimierungen eingegangen, die sich u.a. mit denen der Nutzungskontrolle decken. Anschließend erfolgt ein kurzer Abriß über die Verkehrsüberwachung. Ein weiterer wichtiger Punkt ist die Staukontrolle im Netz selber : Tritt ein Stau innerhalb des Netzes auf, müssen bestimmte Zellen verworfen werden. Am Ende wird ein Resumee anhand eines Praxistests gezogen, bei dem die 3 Grundverfahren verglichen werden.

1 Einführung

Um Datenflüsse in Gigabit-Netzwerken zu kontrollieren, werden bestimmte Mechanismen eingesetzt. Es wird dabei i.a. zwischen präventiver und reaktiver Verkehrs- bzw. Staukontrolle unterschieden. Zu den präventiven Maßnahmen zählen sowohl die Zugangs- als auch die Nutzungskontrolle. Die Zugangskontrolle erfolgt beim Verbindungsaufbau: Es wird geprüft, ob die erforderliche Bandbreite überhaupt verfügbar ist. Um die ausgehandelte Datenrate anschließend kontinuierlich zu überprüfen, wird netzwerkseitig die Nutzungskontrolle eingesetzt.
Bei den reaktiven Maßnahmen wird zwischen der Prioritätskontrolle und der reaktiven Staukontrolle unterschieden. Die Prioritätskontrolle bezieht sich auf das Prioritätsbit in den Zellen und verwirft bei Überlastsituationen Pakete niedrigerer Priorität. Unter reaktiver Staukontrolle werden Mechanismen verstanden, die Auswirkungen von Stausituationen beseitigen sollen. Hierunter fällt z.B. die Vorwärts-Stauerkennung.
Das eigentliche Traffic-Shaping besteht aus den zusätzlich zur Nutzungskontrolle eingesetzten Möglichkeiten, wie beispielsweise dem Verzögern von Paketen, um die Burstiness eines Datenstromes zu kontrollieren.
Allgemeine Traffic Shaping Algorithmen sollten die folgenden Eigenschaften besitzen : [Eck92]

- **Einfachheit** - Da die Algorithmen aufgrund der hohen Verarbeitungsgeschwindigkeit oft in Hardware implementiert werden, ist die Einfachheit eine sehr wichtige Eigenschaft.

- **Flexibilität** - Z.B. besitzt B-ISDN als eine Anwendung von ATM sehr viele Dienste und Anwendungen. Daher muß sich ein Traffic Shaping Algorithmus sehr schnell an neue Anforderungen anpassen können.

- **Robustheit** - Selbst wenn einige Kontrollmechanismen ausfallen sollten, muß der Datenverkehr immer noch geregelt werden. D.h. das System muß weitestgehend unabhängig von anderen Systemkomponenten bzw. dem ankommenden Datenstrom sein.
- **Kontrollierbarkeit** - Sowohl der Datenverkehr als auch Stauungen müssen effektiv kontrolliert werden, so daß eine optimale Ressourcenauslastung ohne Leistungseinbußen möglich ist.

1.1 Eigenschaften und Aufgabenbereiche von Traffic Shaping

Traffic regeln Ein Sender sollte seinen Datenausstoß regeln, denn alleine die Kapazitätszusicherung reicht nicht aus, um das Netzwerk vor Überlastungen zu schützen. Wird z.B. ein 100 MB-Paket in 1 Sekunde ausgesendet, kann leichter ein Pufferüberlauf auftreten als bei 1 KB-Paketen, die jeweils in Abständen von $10\mu s$ gesendet werden. Die Datenrate beträgt jeweils 800 MBit/s. Eine Zerlegung des Paketes innerhalb des Netzes widerspricht insofern dem Konzept von ATM, das ausschließlich eine Transportfunktion unterstützt, und wäre innerhalb eines akzeptablen Leistungrahmens nur schwierig zu realisieren.

Traffic überwachen Um sicherzustellen, daß das ausgehandelte Muster auch tatsächlich gesendet wird, muß das Netzwerk die eingehenden Daten ständig überprüfen, damit keine Unregelmäßigkeiten im Netz auftreten.

2 Verfahren zur Nutzungskontrolle

2.1 Grundverfahren

Leaky Bucket Beim einfachen *Leaky-Bucket Algorithmus* [Par94] (siehe Abb. 77) werden die Zellen in einer FIFO-Schlange[43] abgelegt, die dadurch charakterisiert ist, daß sie

1. die Kapazität β besitzt und
2. die Zellen mit der Rate ρ emittiert.

Ist die Kapazität überschritten, werden alle ankommenden Zellen verworfen.

Dieser Algorithmus erzielt die folgenden Effekte :

1. Die Kapazität β beschränkt die *Zellverzögerung* in dem Verkehrsmuster.
2. Die *maximale Burst-Größe* wird durch β eingegrenzt.

[43] Leaky-Bucket deswegen, weil die Schlange einen undichten Eimer simulieren soll.

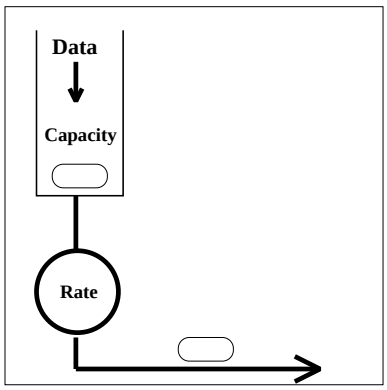

Abbildung 77. Leaky-Bucket Algorithmus

3. Es kann dem Netzwerk zugesichert werden, daß kein Datenverkehr, der eine höhere Rate als ρ besitzt, emittiert wird.

Dieses Muster kann sehr einfach überwacht werden - es wird genau dann verletzt, wenn zwei aufeinanderfolgende Zellen in weniger als $\frac{1}{\rho}$ übermittelt werden. Es kann sowohl für Zell- als auch für Datagrammdienste eingesetzt werden.

Jumping Window Der *Jumping Window Algorithmus* [Rat93] begrenzt durch den Parameter N_{JW} die Anzahl der Zellen, die ein Netzwerk von einer Quelle in dem Intervall T_{JW} durchläßt. Wird die Anzahl der angekommenen Zellen gleich N_{JW}, werden keine Zellen mehr angenommen. Nach Ablauf des Intervalls wird der Zähler wieder zurückgesetzt. Eine Variation wird in [Zit95] erläutert : Der *triggered Jumping Windows-Algorithmus.* Um aufeinanderfolgende Fenster zusammenzuhalten, beginnt ein Fenster erst mit der Ankunft der ersten Zelle.

Moving Window Im Gegensatz zum Jumping Windows Algorithmus (Abschnitt 2.1) wird das Intervall T_{MW} genau dann gestartet, wenn eine Zelle angekommen ist. Der globale Zähler wird dabei um 1 erhöht. Ist das Intervall abgelaufen, wird der Zähler um 1 erniedrigt. Die obere Grenze für die Anzahl der Zellen ist N_{MW}. Wird diese überschritten, tritt die Nutzungskontrolle in Erscheinung und verwirft oder markiert die zusätzlichen Zellen.

2.2 Prioritätsschemata (allgemein)

Die bisherigen, isochronen Schemata waren dahingehend begrenzt, daß sie nur feste Datenraten unterstützten. Wird eine Verbindung mit variabler Bitrate etabliert, muß diese ihre maximale Bitrate belegen. Dies hat zur Folge, daß sehr viel Netzkapazität verlorengeht.

Um dieses Problem zu beheben, wird in jedem Paket ein Prioritätsbit (CLP = Cell Loss Priority) vorgesehen. Es existieren nun zwei Realisierungsmöglichkeiten :

1. Die sendende Anwendung markiert die Zellen, die anwendungsspezifisch weniger wichtige Informationen beinhalten. Tritt nun eine Überlastsituation auf, wird ein Teil der markierten Zellen verworfen.

2. Das Netzwerk kontrolliert den Fluß, indem es die Zellen, welche die ausgehandelte Bitrate übersteigen, markiert bzw. aus dem Verkehr zieht. Die Markierung erfolgt dabei am Netzzugang (anhand des CLP-Bit). Das Verwerfen einer markierten Zelle wird anschließend an einem überlasteten Netzknoten durchgeführt.

Bei der zweiten Methode kann es passieren, daß wichtige, anwendungsspezifische Informationen verlorengehen. Allerdings ist es als Kontrollschema sehr benutzerfreundlich, da während einer Situation, in der das Netzwerk nicht ausgelastet ist, über der ausgehandelte Kapazität übertragen werden kann.
Zwei Einschränkungen besitzt das Prioritätsschemaverfahren :

1. Die garantierte Kapazität entspricht etwa 50% der Bandbreite, da für die niedrig priorisierten Pakete ein gewisser Puffer vorhanden sein muß. Sonst würde keine Chance bestehen, diese überhaupt zu übertragen.

2. Da die meisten Switches FIFO-Puffer zur Speicherung der Pakete benutzen, muß eine Überlastsituation immer vorhergesehen werden, weil Pakete in der Mitte eines FIFO-Puffers nicht selektiert werden können. Über die Effektivität solcher Vorhersagealgorithmen ist bisher noch nicht viel bekannt.

2.3 Bursty-Traffic Verfahren

Die Bursty-Traffic Verfahren sind optimierte Leaky-Bucket Verfahren, die bei burstartigem Verkehr eingesetzt werden. Die Behandlung bei Verletzungen der ausgehandelten Parameter wird der Anwendung überlassen.

Token Bucket Der *Token-Bucket Algorithmus* [Par94] basiert auf dem Leaky-Bucket Verfahren. Es werden Token, die mit der Rate ρ eintreffen, in dem FIFO-Puffer gespeichert. Dieser besitzt die Kapazität β. Grundsätzlich sind 3 Fälle zu unterscheiden, wenn ein Paket der Länge $b < \beta$ gesendet werden soll (maximale Paketgröße $= \beta$) :

Der FIFO-Puffer ist voll : Das Paket wird gesendet, und b Token werden aus dem FIFO-Puffer entfernt.

Der FIFO-Puffer ist leer : Das Paket muß b Token abwarten, bis es gesendet werden darf.

Der FIFO-Puffer ist teilweise voll : B Token sind in dem FIFO-Puffer.

1. $b \leq B$: Das Paket wird sofort gesendet, und es werden b Token entfernt.

2. $b > B$: Das Paket muß $b - B$ Token warten.

Im Unterschied zu Leaky Bucket läßt Token Bucket *bursty traffic* zu, begrenzt ihn aber so, daß nicht mehr als $\beta + (\tau/\rho)$ Token das System in einem Intervall τ durchlaufen. Asymptotisch wird eine Übertragungsrate von ρ erreicht.
Token Bucket selbst verwirft keine Pakete. Das Management des Datenstromes wird der Anwendung überlassen. Sie kann die Puffergrenzen bestimmen, anhand derer die Pakete verworfen bzw. markiert werden. Die Implementierung erfolgt relativ einfach über einen Zähler, um die Token zu verwalten, und einen Timer, der die Ankunft der Token bestimmt. Schwieriger wird die Realisierung eines Überwachungsmechanismus, da in einem bestimmten Intervall τ die Bitrate ρ durchaus übertroffen werden kann. Lediglich ihr asymptotisches Verhalten konvergiert gegen ρ.

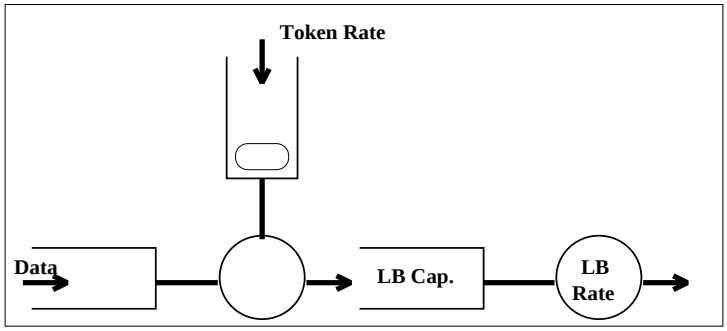

Abbildung 78. Token Bucket with Leaky Bucket Rate Control

Token Bucket with Leaky Bucket Rate Control Beim einfachen Token-Bucket Verfahren kann es zu Überlastproblemen kommen, wenn der FIFO-Puffer vollständig geleert wird, d.h ein Burst gesendet wird. Um diese Unregelmäßigkeiten zu unterdrücken, wurde das Schema *Token Bucket with Leaky Bucket Rate Control* [Par94] entwickelt. Hinter den Token Bucket wird noch ein Leaky Bucket mit derselben Kapazität β und der Rate C gesetzt. Daraus ergibt sich eine maximale Übertragungsrate von C und eine Obergrenze der durchschnittlichen Übertragungsrate von ρ.
Die Implementierung dieses Verfahrens erfordert zwei Timer und zwei Zähler. Die Überwachung von C ist relativ einfach, schwierig wird nur die Ratenkontrolle von ρ. Analog wird in [Zit95] der *doppelte Leaky-Bucket* definiert. Die eine Schlange, die eine sehr kleine Kapazität, jedoch eine sehr große Rate besitzt, begrenzt die maximale Burstgröße. Die andere dient zur Regulierung der mittleren Rate.

Der virtuelle Leaky-Bucket Mechanismus Der virtuelle Leaky-Bucket Algorithmus ist im Prinzip eine Umsetzung der in 2.2 behandelten Prioritätsverfahren. Es existieren 2 Puffer. Ist der erste überfüllt, werden die Pakete nicht verworfen, sondern als "niedrigpriorisiert„ markiert. Dadurch wird eine bessere Resourcenauslastung erreicht.

3 Verkehrsüberwachung

An den Endpunkten des Netzes und zwischen den Netzknoten wird jeweils eine Überwachungsfunktion dazu benutzt, den ausgehandelten Datenstrom zu kontrollieren. Diese kann sich sowohl auf eine einfache virtuelle Verbindung (VC) als auch auf einen ganzen virtuellen Pfad (VP) beziehen. Grundlegende Überwachungsfunktionen wurden bereits in 2.1 vorgestellt. Im folgenden sollen diese weiter erläutert werden.

3.1 Vergleich der Überwachungsalgorithmen

Beim schlechtest möglichen Übertragungsmuster werden alle Zellen zu Beginn akkumuliert gesendet (maximaler Burst). Es folgt eine Ruheperiode, bis der Timer abgelaufen ist. Bei den verschiedenen Verfahren sieht das wie folgt aus :
(Δ = Zellenzwischenankunftszeit, N_i = Puffergröße von Verfahren i, T_i = Periode von Verfahren i)

- **Moving Window :**
  Burstlänge : N_{MW}
  Ruheperiode : $T_{MW} - (N_{MW} - 1)\Delta$
- **Jumping Window :**
  Burstlänge : N_{JW}
  Ruheperiode : $2T_{JW} - (2N_{JW} - 1)\Delta$
- **Leaky Bucket :**
  Burstlänge : $N_{LB,max} > \frac{N_{LB}-1}{1-\frac{\Delta}{D}}, \Delta < D$
  wobei ($D = \frac{1}{BitRate}$) und $N_{LB,max}$ die kleinste Ganzzahl ist, die die Bedingung erfüllt
  Ruheperiode : $N_{LB,max}(D - \Delta) + \Delta$

Setzt man nun die Fensterparameter wie folgt

$N_{MW} = N_{LB,max} = 2N_{JW}$, sowie
$T_{MW} = T_{LB} = 2T_{JW}$

sieht man, daß alle 3 Mechanismen denselben, burstartigen Verkehr verarbeiten können.

4 Staukontrolle

Bei VBR-Anwendungen[44] werden die Ressourcen im Netz in der Regel nach der durchschnittlichen Bitrate belegt. Beabsichigt wird damit eine möglichst gute Auslastung des Netzes. Probleme bereiten aber die Lastspitzen. Um diese zu bewältigen, können an den Netzknoten die folgenden Aktionen ausgelöst werden ([Zit95]):

- Verwerfen zusätzlicher Zellen

[44] VBR = Variable Bit Rate

– Kennzeichnen zusätzlicher Zellen

– Verzögern zusätzlicher Zellen

– Verwenden adaptiver Mechanismen

Das Verwerfen zusätzlicher Zellen sollte nur im absoluten Notfall eingesetzt werden. Ist eine bestimmte Lastgrenze überschritten, kann es aber durchaus sinnvoll sein, die Zellen, die darüber hinausgehen, zu markieren. Damit fällt bei einem Überlauf die Entscheidung einfacher, welche Zellen zu verwerfen sind. Ob ein Verzögerungsmechanismus eingesetzt werden kann, hängt stark von der Art der Anwendung ab. Eine Maßnahme, die auch oft eingesetzt wird, ist die Vorwärts-Stauerkennung. Durch Setzen des EFCI-Bits (explicit forward congestion indicator) an einem Netzwerkelement mit angestauten Zellen, erfährt der Empfänger, daß auf dem VC/VP möglicherweise ein Stau aufgetreten ist. Dieser kann dem Sender daraufhin über AAL-Funktionen eine Meldung senden, daß z.B. Shaping- oder Fehlerkontrollmaßnahmen ergriffen werden sollen.

5 Traffic-Shaping Algorithmen

5.1 Buffering

Verletzen Zellen eine ausgehandelte Rate, können diese gepuffert werden. Dadurch entsteht aber eine zusätzliche Verzögerung für diese Zelle (Jitter). Es werden im folgenden die Begriffe „kritische Rate“ und „Spitzenrate“ eingeführt, um den Zusammenhang zwischen der Verzögerung und der Pufferkapazität zu erläutern.
Liegt die zu kontrollierende über der kritischen Rate, so bedeutet dies, daß die Puffer-

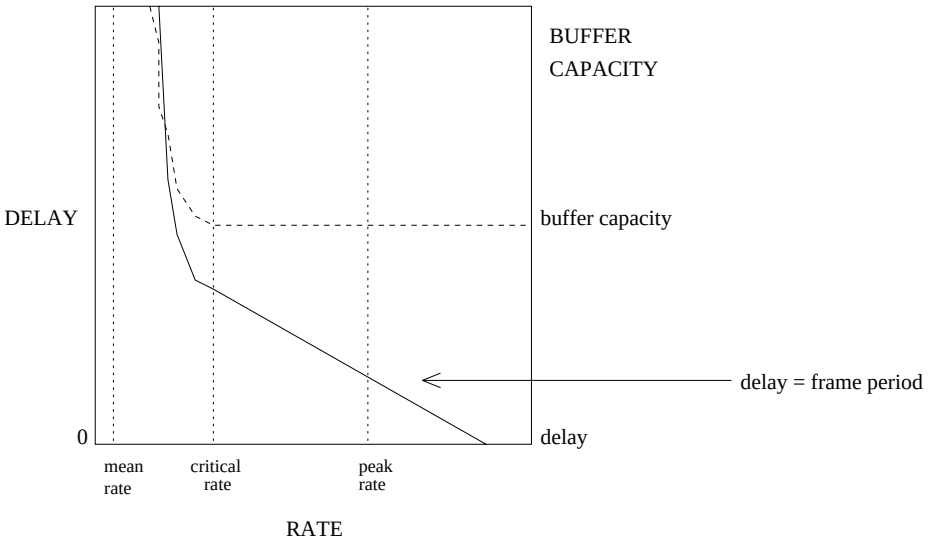

Abbildung 79. Verzögerung and Pufferkapazität als Funktion der Rate

größe konstant bleiben kann, ohne daß Zellen verlorengehen (jeder Rahmen wird also vollständig emittiert, bevor ein neuer eintritt). Wird die Spitzenrate erreicht, ist die

Verzögerung genau eine Rahmenperiode groß, d.h. auch der theoretische Fall, daß die Rahmen immer die volle Länge besitzen, wird ohne Pufferüberlauf bewältigt. Der Abstand zwischen der Spitzenrate und der kritische Rate ist je nach der Rahmengröße kleiner oder grösser. Bei dieser Betrachtung wird von einer vollständigen Übertragung ausgegangen, d.h. es darf kein Zellenverlust erfolgen. Um z.B. interaktive Videoanwendungen zu realisieren ist die Verzögerung jedoch unbrauchbar (sie sollte bei $n \cdot 100ms$ liegen). Hierbei ist es aber weniger wichtig, daß tatsächlich alle Rahmen bei dem Empfänger ankommen. Es ist also sinnvoller, die Verzögerung zu betrachten, wenn eine gewisse Verwurfswahrscheinlichkeit der Zellen angenommen wird (siehe Abbildung 82). Man stellt fest, daß die Kurve bei niedrigen Raten erheblich flacher wird, als bei den hohen - was eigentlich auch zu erwarten war. Für diese niedrigen Raten wird die Qualität der Übertragung ein sehr großes Problem darstellen.

5.2 Allgemeine Traffic Shaping Modi

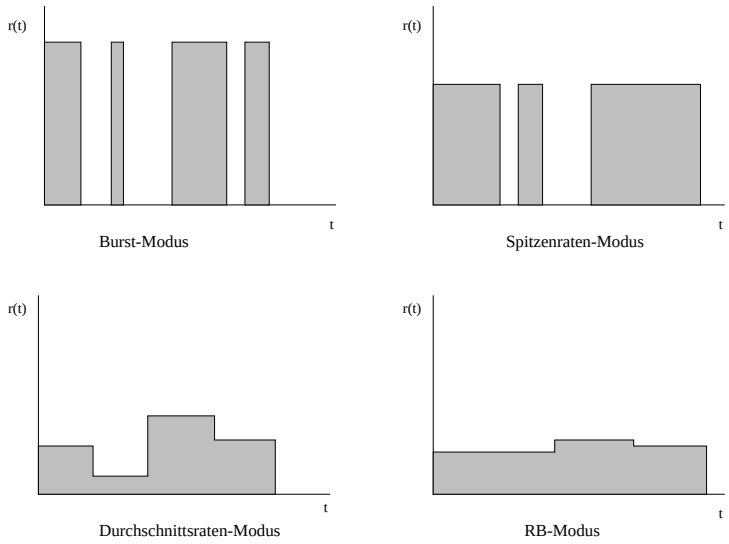

Abbildung 80. Traffic Shaping Modi

Um bestimmte Zellverzögerungen zur Reduzierung der Burstiness zu erreichen, wurden bestimmte Traffic Shaping Modi eingeführt. Dadurch wird der Datenstrom sehr gut charakterisiert. Ein Rahmen ist hier definiert als eine logische Einheit von Zellen, die z.B. ein Bild (M-JPEG) oder eine Bildmodifikation (MPEG) darstellen.

- **Burst Modus :** Die Rahmen werden im Ein/Aus-Muster auf der gesamten Bandbreite gesendet.
- **Spitzenraten-Modus :** Die Rahmen werden im Ein/Aus-Muster mit der Spitzenrate gesendet. Die Spitzenrate wird dabei als maximale Rahmengröße dividiert durch die Rahmenperiode der Datenquelle definiert.
- **durchschnittliche Rahmenperiode :** Die Rahmen werden während einer Rahmenperiode mit einer konstanten Rate gesendet. Die Rate ergibt sich als die Rahmengröße dividiert durch die Rahmenperiode.

– **RB-Modus :** Die Rahmen werden mit einer konstanten Rate ausgesendet. Diese wird anhand eines speziellen Algorithmus bestimmt, der eine Begrenzung des Delay-Jitters sowie der Puffergröße zusichert.

Für die letzten beiden Verfahren wird aus Leistungsgründen eine Hardwareimplementierung vorausgesetzt, die eine Änderung der Segmentierungsrate nach jeder Rahmenperiode ermöglichen kann.

5.3 Traffic Shaping Modi für das Kurzzeitverhalten

Für das Kurzzeitverhalten gibt es zwei prinzipiell unterscheidbare Modi :

– **EF-Mode :** Die Zellen eines Rahmens werden gleichmäßig über die gesamte Rahmendauer verteilt. Dies wird mittels eines Puffers im Sender erreicht. Auf diese Weise wird die maximale Bitrate verringert. Der Nachteil ist aber, daß eine zusätzliche Verzögerung eingefügt wird.

– **CF-Mode :** Alle Zellen werden im Burst-Mode gesendet, bis die maxmimale Kapazität erreicht ist. Danach tritt eine Ruheperiode ein. Mit diesem Modus wird eine sehr geringe Verzögerung auf Kosten eines sehr bursthaften und variablen Datenverkehrs erreicht.

In der Regel wird immer eine Mischform aus beiden Modi eingesetzt, da beide eigentlich nur eine Extremsituation darstellen.

5.4 Cell Spacing

Cell Delay Variation (CDV) Unter „Cell Delay Variation (CDV)“ versteht man die unterschiedliche Verzögerung zwischen den Zellen. Es tritt ein Streuungseffekt auf, wenn diese Verzögerung größer als T ist. Wenn sie jedoch kleiner ist, werden die Zellen akkumuliert (T ist hierbei die Sendeperiode einer Zelle). Allgemein läßt sich dieser Prozeß wie folgt beschreiben :[45]
Die Zelle n wird zum Zeitpunkt $n.T$ erzeugt und erfährt eine Verzögerung W_n :
$t_n = n.T + W_n$,
(W_n poissonverteilt, $n.T$ ist der Erzeugungszeitpunkt von Zelle $\#n$)

W_{max} und W_{min} ergeben sich aus folgenden Gleichungen :
$Pr\{W_n > W_{max}\} \sim 10^{-10}$
$Pr\{W_n < W_{min}\} \sim 10^{-10}$
mit $\delta = W_{max} - W_{min}$ folgt nun :
$Pr\{\mid W_n - W_0 \mid > \delta\} \sim 10^{-10}$
Daraus ergibt sich für die Ende-zu-Ende Grenzfunktion von CDV $\frac{\delta}{T}$, wodurch die CDV in einem Netz eindeutig charakterisiert wird. Anhand dieses Verhältnisses und der Sendeperiode T kann man auf δ schließen, das die Spannweite der möglichen Verzögerungen widerspiegelt.

[45] Pr = Wahrscheinlichkeit(Probability)

Virtual Scheduling Algorithm Der Virtual Scheduling Algorithm (VSA(T, τ)) [BGSC92] (siehe Abb. 81) wird dazu verwendet, festzulegen, ob eine Zelle verzögert werden soll, und zu welchem Zeitpunkt sie reemittiert wird. Hierzu wird der *TRT (theoretical reemission timer)* benutzt. Es treten die folgenden Fälle auf :

1. $TRT_n + T - t_n > \tau$ - Zelle #n wird verworfen, $TRT_n^+ = TRT_n$
2. $TRT_n + T - t_n \leq 0$ - Zelle #n wird sofort weitergesendet, $TRT_n^+ = t_n$
3. $0 < TRT_n + T - t_n \leq \tau$ - Zelle #n wird gepuffert, $TRT_n^+ = TRT_n + T$

Anmerkung : TRT_n^+ bezeichnet den TRT nach der Ausführung von VSA, TRT_n den TRT vor der Ausführung des VSA. τ ist ein Parameter, der dazu eingeführt wurde, um etwaige Verzögerungen auszugleichen. Anhand einer Heuristik läßt sich die Bedingung $\tau = \delta$ ableiten, um die Wahrscheinlichkeit eines Zellenverlustes kleiner als 10^{-10} zu halten. T entspricht der größten Zellemissionsperiode.
Je nach Prioritätsschema wird die Zelle bei Schritt 1 verworfen oder markiert.
Durch den VSA werden also die Cell Delay Variations ausgeglichen.
Der VSA unterscheidet in ATM-Netzen auch zwischen den verschiedenen Zelltypen. Es existieren getrennte TRT und τ sowohl für $OAM - F_5$ (end-to-end) als auch für user-to-user Zellen (CLP=0/CLP=1). FRM-Zellen (network-to-network) werden entweder immer aktzeptiert oder überhaupt nicht.
Konkret sieht die Realisierung eines solchen Spacer-Controllers folgendermaßen aus :

- **Context Memory :** Für jede Verbindung werden die folgenden Variablen gespeichert : T_0, TRT_0, τ_0, T_{0+1}, TRT_{0+1}, τ_{0+1}, T_{OAM}, TRT_{OAM}, τ_{OAM} sowie der boolesche Ausdruck "FRM nicht akzeptiert !". Außerdem werden dort die Zähler für die nicht angenommenen und aktzeptierten Zellen abgelegt.
- **Virtual reemission schedule management block :** Hier ist der VSA implementiert. Der TRT wird für jede Zelle gesetzt, und die Policing-Maßnahmen werden ausgelöst.
- **Reemission memory :** Dieser Puffer dient dazu, die Zellen vor dem Reemittieren zwischenzuspeichern. Er muß $\frac{\delta_0}{\Delta}$ Zellen speichern können, wobei δ_0 den längsten Verzögerungsabstand zwischen zwei Zellen beschreibt, und Δ der Parameter für die Zellemissionszeit am Netzwerk ist.

Der VSA sollte immer in demselben Element realisiert sein, wie der Leaky Bucket-Algorithmus bzw. der Kontrollalgorithmus.

5.5 Framing

Beim Framing werden Rahmen synchron von einer Quelle gesendet. Ankommende ATM-Zellen benutzen den naechsten erreichbaren und noch nicht vollständig gefüllten Rahmen. Die Quelle besitzt einen festen Sendezeitplan.

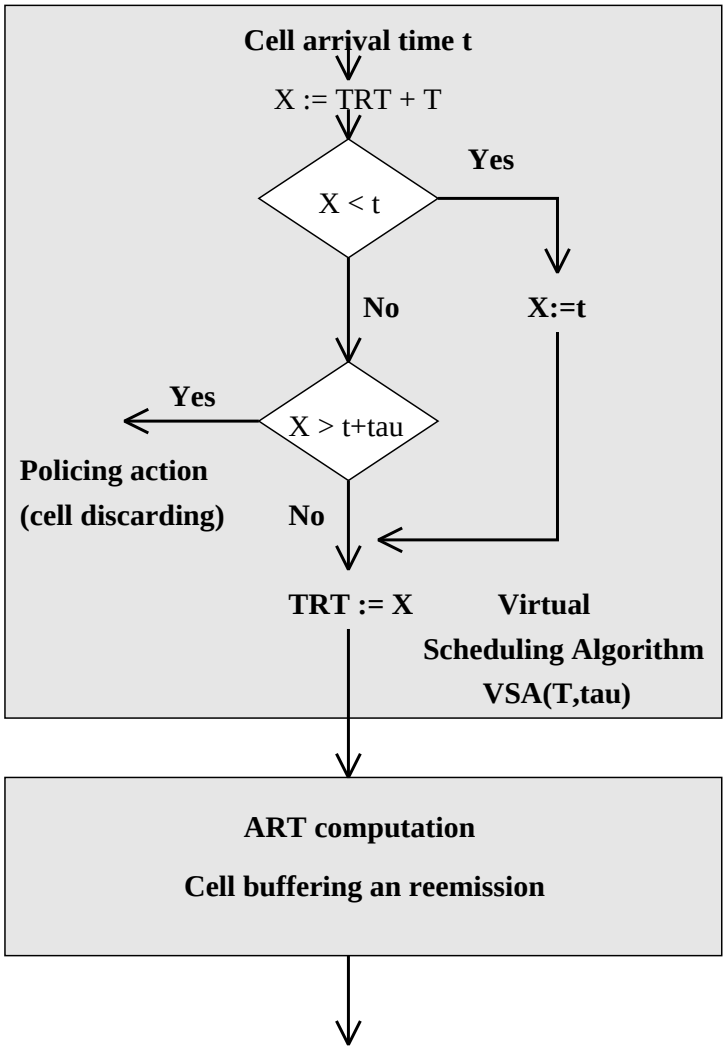

Abbildung 81. Virtual Scheduling Algorithm

Anmerkung : ART = actual reemission timer

6 Praxistests von Traffic-Shaping-Verfahren

6.1 Einführung

Anhand zweier Datenquellen soll die Häufigkeit ermittelt werden, mit der die vom Netzwerk zugewiesene Rate in Abhängigkeit der Puffergröße verletzt wird [Rat93]. Diese Häufigkeit ist somit ein Maß dafür, wie groß der Puffer gewählt werden sollte, um eine für die jeweilige Anwendung akzeptabele Verlustquote zu erhalten.

6.2 Beschreibung des Experiments

Datenquellenbeschreibung

Star Wars : Dient als Beispiel für einen Video-On-Demand-Service. Die Datenmenge umfaßt 171.000 Rahmen, die mit einer Rate von 24 Rahmen pro Sekunde gesendet werden.

EVA_CAM : Diese Videosequenz soll eine Videophone-Anwendung demonstrieren. Der Sender arbeitet dabei mit einer Rate von 25 Rahmen pro Sekunde.

Data set	Maxmimum	Mean	Minimum	Burstiness factor
Start Wars	18,15 Mbit/s	6,43 Mbit/s	1,99 Mbit/s	2,82
EVA_CAM	13,96 Mbit/s	2,63 Mbit/s	0,329 Mbit/s	5,31

Die Versuche wurden im *EF-Modus* durchgeführt, wobei die zu überwachende Bitrate zwischen der Durchschnitts- und Spitzenrate variiert wurde.

Ergebnisse Folgende Phänomene konnten beobachtet werden :

- Die Erhöhung der Pufferkapazität von 1 auf 2 verursacht eine erhebliche Verringerung der Häufigkeit eines Paketverlustes. Das kann damit begründet werden, daß die maximale Burstlänge bei einer Pufferkapazität von 1 immer 1 beträgt. Wird hingegen die Kapazität auf einen Wert größer als 1 gesetzt, ist die maximale Burstlänge immer proportional zu $(1 - \Delta/D)^{-1}$ (siehe Abschnitt 3). Dieser Wert kann aber bei hohen Bitraten sehr groß werden.
- Bei einem Pufferlimit größer als 2 sinkt die Verlusthäufigkeit nur bei Werten in der Nähe der Spitzenrate.
- Um die Durchschnittsrate zu überwachen, wäre ein sehr großer Puffer sowohl bei „Star Wars“ als auch bei „EVA_CAM“ notwendig. Dadurch entsteht aber eine lange Reaktionszeit des Algorithmus. Außerdem vergrößert sich die Verzögerung der Rahmen.

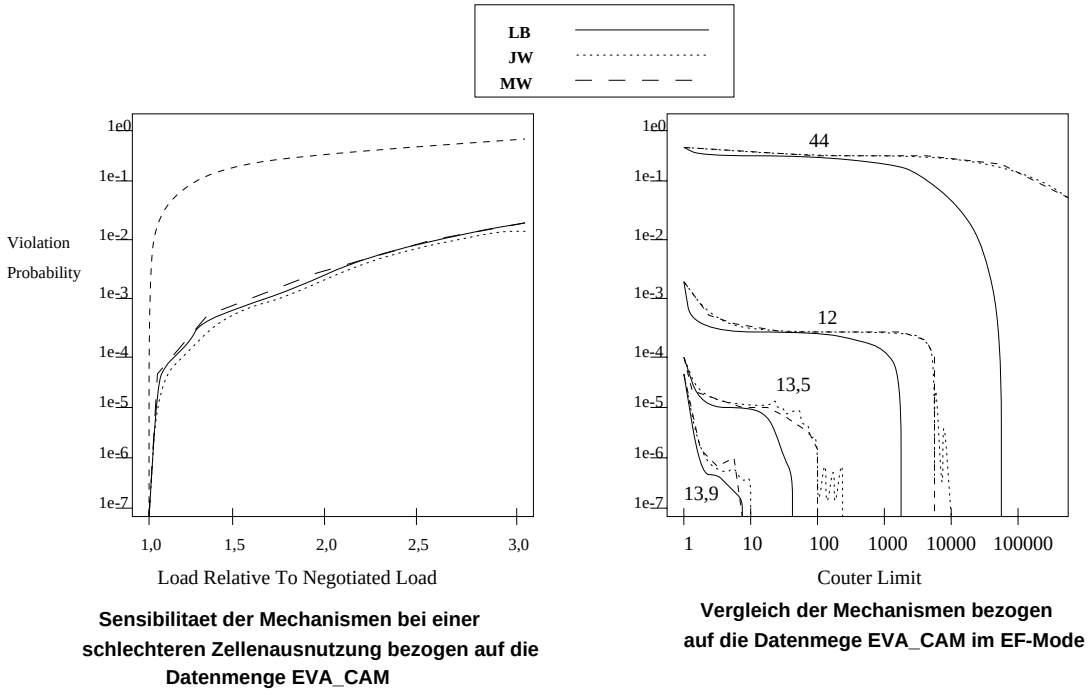

Abbildung 82. Vergleich der 3 Kontrollalgorithmen

Bei dem Vergleich aller 3 Verfahren (Leaky Bucket, Jumping Window, Moving Window) schneidet der Leaky Bucket Algorithmus am besten ab. Aufgrund der Fenstertechnik fallen die Kurven von Jumping Window und Moving Window nicht monoton ([Rat93]). Weiterhin wurde das Verhalten aller 3 Algorithmen bei einer Verringerung der Nutzinformationen pro Zelle getestet. Dadurch werden mehr Zellen emittiert - es entsteht eine Überlastsituation. Alle 3 Mechanismen sind relativ weit von der Ideallinie entfernt, wie Abbildung 82 zeigt.
Weiterhin muß zwischen dem Durchschnitt der Fehlerhäufigkeit und den lokalen Häufigkeiten unterschieden werden. Die lokale Fehlerhäufigkeit kann durchaus größer sein als der Durchschnitt. Dies wirft aber erhebliche Probleme bei den Nutzungskontrollalgorithmen auf. Sie können zwar die Daten bei einer dem Durchschnitt äquivalenten Verstoßhäufigkeit verarbeiten, sind aber mit den lokalen Verletzungen überfordert. Vergleicht man die beiden Modi CF und EF miteinander, fällt auf, daß der Fluß, der im CF-Modus gesendet wurde, im Gegensatz zum EF-Modus erst nach einer weiteren Vergrößerung des Puffers abfällt.

6.3 Zusammenfassung

In jedem Falle sollte die Spitzenrate der Übertragung in einem gewissen Rahmen gehalten werden, damit - wenn auch ein Qualitätsverlust dabei in Kauf zu nehmen ist - die Ressourcenbelegung sowie die Kontrollfunktion effizient durchgeführt werden können. Ein weiteres Problem sind die Langzeitmessungen des QoS[46]. Denn ein Verlust akkumulierter Zellen ist für die Anwendung problematischer als ein zufällig verteilter Verlust. Für interaktive Anwendungen kommt nur eine Ratenkontrolle nahe der Spitzenrate in Betracht, da bei niedrigeren Raten die Verzögerung zu groß wird. Selbst bei Raten nahe der Durchschnittsrate treten schon Probleme mit der Puffergrösse auf.

7 Ergebnisse

7.1 Zukunft von ATM bei Videoübertragung (als spezielles Anwendungsgebiet von Traffic Shaping)

Die meisten Probleme bei Videoübertragungen bereiten die burstartigen Übertragungen. Diese Art von Traffic ist für das Netzwerk am schwierigsten zu kontrollieren, da die durchschnittliche Rate oft um ein Vielfaches überschritten wird. Es sind sehr große Ressourcenbelegungen notwendig, und die Verzögerung wird für interaktive Anwendungen unakzeptabel. Bei weniger zeitkritischen Anwendungen, wie z.B. dem Star Wars-Film aus 6.2, spielt die Verzögerung eine geringere Rolle. Hingegen muß dort die Bildqualität sehr gut sein, was große Ressourcen in Anspruch nimmt.
Als Fazit kann gefolgert werden, daß die bisherigen Algorithmen noch nicht sehr befriedigend sind, was auch das Ergebnis aus 6.2 zeigt. Die Algorithmen sind insgesamt von der Ideallinie noch relativ weit entfernt.

[46] QoS = Quality of Service

Das GIGAswitch / FDDI-System

Kioumars Namiri

Kurzfassung

Das GIGAswitch / FDDI-System ist eine Hochleistungs-Netzwerkkomponente, die als Brücke mit integrierten Routingfunktionen charakterisiert werden kann. Es bietet bei vollem Ausbau Anschlüße für insgesamt 34 FDDI-Ringe, die durch das Gerät bei Bedarf mit Ihrer vollen Kapazität verbunden werden kann. Dies wird durch eine leistungsfähige Switch-Architektur erreicht, die maximal 3,6 GBits verarbeiten kann.

Das GIGAswitch-System besteht aus einer Backplane, in die bis zu 4 verschiedene Kartentypen gesteckt werden können: Der Non-blocking Crossbar, der Switch Control Processor (SCP), die Clockcard und bis zu 11 Linecards.

Der SCP ist die zentrale Anlaufstelle vieler Pakete im GIGAswitch. Er beinhaltet Implementierungen über der MAC-Schicht (IP, SNMP), lernt Adressen mit Hilfe der Linecards und verwaltet die GIGAswitch-Konfiguration. Die Clockcard generiert den Takt und stellt den Speicher für Managementparameter zur Verfügung. Das Crossbar-Modul beinhaltet die Crossbar-Bausteine.

1 Einführung in die FDDI-Technologie

FDDI (Fiber Distributed Data Interface) ist ein flexibles Hochgeschwindigkeitsnetzwerk, das vielseitig verwendbar ist. Im allgemeinen unterscheidet man 3 Gebiete, in denen FDDI schwerpunktmäßig eingesetzt werden kann: Backbone-Bereich, Front- und Backend-Vernetzung.

Als Backbone-Netzwerk zur Kopplung lokaler Netzwerke, wie Ethernet oder Token Ring, ist FDDI ideal geeignet. Durch die hohe Datenrate von 100 Mbit/s kann ein FDDI-Backbone netzübergreifend Datenverkehr aus vielen 'langsamen' LANs bewältigen, ohne daß es zur Kapazitätsengpässen kommt. Die maximale Netzausdehnung von 200 Kilometern erlaubt den Aufbau ausgedehnter Betriebs- oder Betriebsgelände-Verkabelung.

Die Anbindung einzelner LANs kann über Bridges, Switches und Router erfolgen. Sie können Netze auf unterschiedliche Weise miteinander verbinden. Bridges arbeiten hardwarenah und koppeln Netze auf der Schicht 2. Router operieren hingegen auf der Schicht 3. Zweite Einsatzvariante von FDDI ist die direkte Kopplung von Rechnern über FDDI, auch Front-End-Bereich genannt. Die hohe Datenrate ist besonders bei der Vernetzung von leistungsfähigen Workstations und File-Servern interessant, wenn dort auf Anwendungen mit hohen Ansprüchen an Netzbandbreite wie CAD, Bildverarbeitung (u.a. Röntgenbilder) oder Multimedia-Applikationen laufen.

Drittes Einsatzgebiet ist der Back-End-Bereich. Hier geht es um die Anbindung von Höchstleistungsrechnern (Mainframes, Parallel- oder Vektorrechner) an schnelle Peripheriegeräte wie Tape- und Diskcontroller oder Videoeinheiten.

FDDI-Netzwerkstrukturen basieren auf einem Ring, auf den alle Stationen mit Hilfe eines Tokenprotokolls zugreifen können. Möchte ein Station bei Empfang des Tokens keine Daten übertragen, so gibt sie alle das Senderecht sofort weiter. Hat sie jedoch selbst Daten zu übertragen, sendet sie ihre eigene Daten und gibt darauf den Token weiter.

Eine Besonderheit von FDDI ist das Doppelring-Konzept. Es gibt 2 gegenläufige Ringe, die beide eine Übertragungsrate von 100 Mbit/s bieten. Die Datenübertragung erfolgt nur auf einem, dem aktiven Ring; der zweite Ring dient als sogenannter Backup-Ring. Dieser hält einen alternativen Pfad für den Fall von Kabelunterbrechungen oder Stationsausfällen bereit. Ihrer Funktion entsprechend heißt der aktive Primär- und der andere Sekundär-Ring. Ein an ein FDDI-Netzwerk anschließbares Gerät bezeichnet man als Station. Neben FDDI-Endgeräten wie Interface-Karten in Rechnern, Bridges oder Routern zur Anbindung anderer Netze gibt es noch ein besondere Klasse von Stationen, welche die Bildung komplexer FDDI-Netzwerktopologien erlaubt: Konzentratoren. Je einer stellt für eine Anzahl von weiteren Geräten (auch weiteren Konzentratoren) einen Zugangspunkt zum FDDI-Ring her. Dabei läuft der aktive Ring durch den Konzentrator und alle angeschlossenen Stationen.

Ein weitere Klassifikationsmöglichkeit von FDDI-Geräten sind die Anschlußfunktionen, über die sie verfügen. Zur Class A zählen solche Stationen, die direkt an den Doppelring anschließbar sind. Sie heißen je nach Typ dual-attachment station (DAS) oder dual-attachment concentrator (DAC). Stationen der Class B benötigen für die Anbindung an den Doppelring einen Konzentrator. Sie sind lediglich mit dem Primärring verbunden und heißen entsprechend single-attachment station (SAS) bzw. single-attachment concentrator (SAC).

Die als Ports bezeichneten Stationsanschlüsse bestehen jeweils aus einem Sender und einem Empfänger. Man unterscheidet hier 4 Typen: A, B, M und S. Der A-Port bildet die Verbindung zum eingehenden aktiven Ring und zum ausgehenden Backup-Ring. Der B-Port stellt den umgekehrten Fall dar. Über M-Ports verfügen nur Konzentratoren, die eine Verbindung zu einer Station (SAS, DAS) oder einem weiteren Konzentrator herstellen. Den S-Port findet man nur in Geräten der Class B. Er kann entweder mit einem S- oder einem M-Port verbunden werden.

Statt eines großen Standardisierungsdokumentes existieren mehrere, die jeweils Teilbereiche definieren. Von den sieben Schichten des OSI-Referenzmodells definiert der Standard die Schichten 1 und 2a (MAC).

Im Fall von Netzen mit geteiltem Medium - bei LANs ist das die Regel - sieht das OSI-Referenzmodell die Aufteilung der Schicht 2 in zwei Unterschichten vor: Die untere kontrolliert den Medienzugang (MAC) und die obere ist für den Verbindungsaufbau und die Datensicherung zuständig (LLC). Innerhalb von FDDI beschreibt die MAC-Schicht (Media Access Control) das Medienzugangsverfahren.

2 Übersicht über das GIGAswitch-System

Das GIGAswitch-System stellt im Bereich der Hochleistungskommunikation eine leistungsfähige Netzwerkkomponente dar. Es ist in der Lage, FDDI-Ringe mit voller Bandbreite miteinander zu verbinden. Die Verarbeitungskapazität des GIGAswitch-Systems beträgt maximal 3,6 Gbits/s mit 6,25 Millionen Verbindungen in der Sekunde.

Das GIGAswitch-System besteht aus einer Backplane, in die verschiedene Karten gesteckt werden können [SKO+94]. Drei Karten müssen im GIGAswitch-System vorhanden sein: Der Non-blocking Crossbar, der Switch Control Processor (SCP) und die Clockcard. Zusätzlich können noch 11 Linecards installiert werden, die je nach Typ entweder zwei FDDI-Ports (DAS oder SAS) oder vier FDDI-Ports (nur SAS) zur Verfügung stellen. An jeden der DAS-Ports kann ein FDDI-Doppelring angeschlossen werden. An die SAS-Ports können Stationen oder Konzentratoren mit einer Punkt-zu Punkt-Verbindung angeschlossen werden. Bei dieser Art der Verbindung kann auf das Austauschen des Tokens verzichtet werden und beide Stationen können auf jeweils einer Leitung senden. Die dabei erreichbare Datenübertragungsleistung liegt bei 140-150 Mbit/s.

Das GIGAswitch-System ist eine transparente Brücke mit Routingfunktionen. Die Entscheidung über das Weiterleiten oder Verwerfen von Dateneinheiten wird anhand der Hardware-Adressen getroffen. Ist der empfangenden Linecard aus früheren Paketen die Hardware-Adresse des Pakets bekannt, sendet sie das Paket über den Crossbar direkt an die Linecard, über welche die entsprechende Station erreichbar ist.

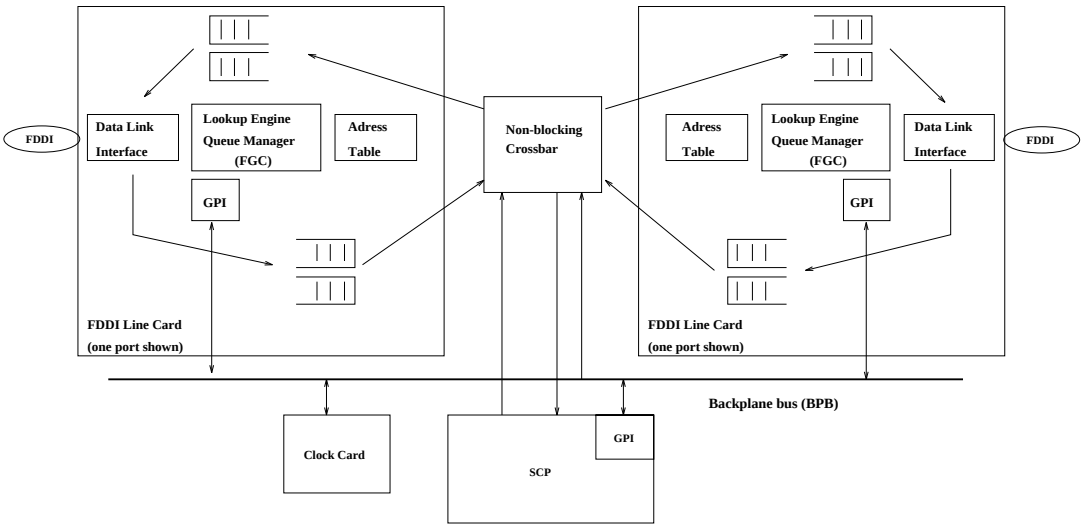

Abbildung 83. GIGAswitch System block diagram

3 Die Architektur des GIGAswitch-Systems

Die Architektur erlaubt ein schnelles, einfaches Weiterleiten der Pakete durch das Verkürzen der 48-Bit-Adressen der Pakete in eine kürzere Adresse, sobald ein Paket in den

Switch ankommt. Es wird ein Header angelegt, der die kürzere Adresse, die Ankunftszeit und den Ankunftsort des empfangenes Pakets beinhaltet.

Der Switch enthält weiterhin einen Algorithmus für die schnelle und effiziente Zuweisung der Pakete über den Crossbar (Kreuzschienenverteiler), der in dem GIGAport VLSI-Chip (GPL) implementiert ist.

Jeder Port hat eine Tabelle für Adressen, einen hardwareunterstützten Suchmechanismus und einen Warteschlangen-Manager.

Der Switch Control Processor (SCP) hat einige zentralen Aufgaben. Er beinhaltet Implementierungen von Protokollen über der MAC-Schicht (IP, SNMP), lernt Adressen mit Hilfe der Line-Cards, versendet Multicast-Pakete und solche deren Ziel ihm nicht bekannt sind, verwaltet die GIGAswitch-Konfiguration und stellt Netzwerkmanagementfunktionen über SNMP zur Verfügung. Die Clock-Card generiert den Takt und stellt den Speicher für Managmentparameter zur Verfügung. Das Crossbar Modul beinhaltet die Crossbar-Bausteine.

Die genauere Funktionsweise dieser Module wird in späteren Kapiteln erläutert.

4 Design und Implementierung

4.1 Struktur und Aufbau des Switches

Der Kern eines GIGAswitch-Systems ist der schnelle nicht blockierende, voll-duplex 36x36x6 Crossbar, der aus 3 36x36x2 custom VLSI Chips besteht. Die FDDI-Line-Karten sind durch den Crossbar miteinander verbunden. Die Daten zwischen den Modulen und dem Crossbar können gleichzeitig in beiden Richtungen übertragen werden. Durch die Verwendung des Crossbars kann ein Eingangsport mit mehreren Ausgangsports gleichtzeitig verbunden werden. Ein 6-Bit Datenpfad durch den Crossbar bietet eine Geschwindigkeit von 150 Mbit/s bei einem Takt von 25 MHz (5 Bits für die Kodierung eines 4-Bit-Datenworts plus einem zusätzlichen Paritätsbit). Jeder Crossbar-Chip hat ungefähr 87000 Gatter und ist in CMOS-Technik implementiert.

Die Ports des Crossbars im GIGAswitch-System haben physikalische und logische Adressen. Die physikalischen Portadressen sind eine Funktion des Steckplatzes in dem sich die Karte befindet. Die logischen Portadressen werden vom SCP zugewiesen, der eine Abbildung von der logischen auf die physikalische Adresse beim Initalisieren der Karte vornimmt. Manche der logischen Portadressen sind reserviert, z.B ist der logische Port mit der Adresse 0 ist immer für den SCP reserviert.

In der Regel benutzt der Crossbar für seine Zuweisungen an Ausgabeports logische Adressen. Der Zuweisungsmechanismus wird ẗake a ticket“genannt: Wenn eine Line Karte ein Paket an einen Ausgabeport senden möchte, bekommt sie von diesem Port ein Ticket. In diesem Ticket wird die Position in der Warteschlange des Ausgabeports angezeigt. Auf diese Weise kann die Line Karte den Crossbar auffordern, eine Verbindung zu diesem Ausgabeport herzustellen. Die Zuweisung erfolgt in Form eines verteilten Algorithmus, welcher durch GIGAPort-Interface-Chips (GPI) im SCP und in den Line-Cards realisiert ist. Die Übertragung des Tickets und der Informationen zur Verbindung durch den GPI

wird über einen Bus im Switch-Backplane abgewickelt. Durch Benutzung dieses Buses im Switch können Verbindungsunterbrechungen vermieden werden. So verdoppeln sich die Übertragungsraten, und dadurch ist der GIGAswitch in der Lage, 6,25 Millionen Verbindungen in der Sekunde zu schaffen.

Das GPI erlaubt eine Zusammenfassung mehrerer physikalischer Ports zu einer logischen Adresse, wodurch eine sogenannte ẗhunt group"geschaffen wird. Über die Größe und die Mitglieder einer hunt group gibt es keine Einschränkungen, die Mitglieder können sogar auf verschiedenen Line Cards im Switch verteilt sein. Wenn an eine hunt group gesendet wird, verteilt der take a ticket Zuweisungsmechanismus dynamisch den Verkehr auf die physikalischen Ports und es wird einer Verbindung zum nächsten freien Port geschaffen. Für dieser Zuweisungs- und Verteilungsmechanismus wird keine zusätzliche Zeit benötigt.

Adreẗssuche Eine optimal operierende Bridge muẗ in der Lage sein, auf jedem Port jedes Paket zu empfangen. Sie soll weiterhin sofort entscheiden können, ob sie das Paket weiterleitet oder verwirft. Im schlimmsten Fall beträgt die eingehende Paketrate bei FDDI 440.000 Pakete pro Sekunde pro Port. Da 3 Felder (Frame Control (FC), Zieladresse (DA) und Quelladresse (SA)) pro Paket untersucht werden müssen, führt die FDDI-Line-Karte bis zu 1,3 Millionen Suchoperationen pro Sekunde pro Port aus, worunter 880.000 Adressen mit 48 Bit sind. Die 48-Bit-Suche muß in einer Tabelle mit 16K Einträgen vorgenommen werden, damit große LAN's darin verwaltet werden können. Die Suchfunktion ist an jedem Port realisiert, um den benötigten Durchsatz zu erhalten.

CAMs (content adressable memory) stellen ca. 1K Einträge pro CAM-Chip zur Verfügung. Dies ist aber nicht als Implementierung einer 16K Tabelle für die Adreßsuche geeignet. Die älteren Bridge-Produkte benutzen eine in Hardware implementierte binäre Suche für die 48-Bit-Adressen. Diese binäre Suche benötigt im Durchschnitt 13 Leseoperationen für eine 16K Adreßmenge und schnelle, teuere RAM's, um die Verzögerung zu minimieren. Um die gewünschte Leistung kostengünstig zu erreichen, wurde auf dem FGC ein hochoptimierter Hash-Algorithmus implementiert, der die Quell- und Zieladresse sucht. Diese Suche benötigt maximal 4 Lese-Operationen im RAM, das auch bei der Packetpufferung eingesetzt werden kann.

Der Hash-Algorithmus ermittelt eine neue 48-Bit-Adresse und benutzt dabei die Technik ders Polynom-Divisions in GF(2). Die unteren 16 Bit der daraus resultierenden Adresse werden benutzt, um den Eintrag in der 64K Hashtabelle auszuwählen. Jeder Eintrag in der Hashtabelle besteht aus einem Zeiger und einer Zahl zwischen 1 und 7, die angibt, wieviele Elemente in diesem Eintrag schon kollidiert sind. Sind zwischen 1 und 7 Elemente auf diesem Eintrag kollidiert, so zeigt der Zeiger auf die Wurzel eines balancierten Binärbaumes der Höhe 1, 2 oder 3. Die oben genannten 4 Lesezugriffe setzen sich aus einem Zugriff auf die Hashtabelle und max. 3 Zugriffen auf diesen Baum zusammen.

Wenn mehr als 7 Adressen in der Hashtabelle miteinander kollidieren sollten, werden die übergelaufenen Adressen in einem Assoziativspeicher (GCAM) in der FDDI-Line-Cards abgespeichert. Dieser Fall kommt aber sehr selten vor. Nach mehr als zwölf Zuweisungen von Adressen in den GCAM wird entschieden, daß es sich um eine ungeeignete Wahl der Hashfunktion handelt. Anschließend sucht der SCP eine bessere 48-Bit Hash-Funktion,

die er auf die FGLs neu verteilt. Die FGLs bilden entsprechend der neuen Funktion eine neue Hash-tabelle.

Paketpufferung Die FDDI-Line-Card stellt Paketpuffer am Ein- und Ausgang eines jeden FDDI-Ports zur Verfügung. Ausgangspufferung wird benutzt, wenn der Ausgang vom FDDI-Link kurzzeitig nicht verfügbar ist. Eingangspuffer für die Pakete wird während der Feststellung des Ausgabeports benötigt. Ein- und Ausgabepuffer sind in mehrere FIFO-Schlangen für verschiedene Verkehrsarten unterteilt. Switche mit einer FIFO-Warteschlange am Eingangsport können mit dem Problem des 'head-of-line-blocking' konfrontiert werden. Head-of-line-blocking liegt dann vor, wenn ein Paket am Anfang einer Warteschlange für eine beschäftigten Port vorgesehen ist und gleichzeitig weiter hinten in der Schlange ein Paket für einen nicht beschäftigten Port vorhanden ist. Der wichtigste Faktor beim head-of-line-blocking ist die Verteilung des Paketverkehrs innerhalb des Switches. Head-of-line-blocking tritt jedoch bei normalem Verkehr sehr selten auf, und die hunt groups sind ein geeignetes Mittel zur Reduzierung von head-of-line-blocking, falls es doch vorkommen sollte.

Die Multicast Pakete bedürfen einer besonderen Behandlung im GIGAswitch-System. Während eine sehr hohe Packetdurchsatzrate einen Netzwerkengpaß verursachen kann, haben die Erfahrungen mit großen LAN's gezeigt, daß bei niedriger Durchsatzrate der gleiche Engpaß durch Multicast-Pakete zustande kommen kann. Eine hohe Anzahl von Multicast-Paketen in einem LAN führt schnell zu Überlastung. Daher können diese Pakete im GIGAswitch-System durch Filter vom Systemmanager beschränkt werden. Die Abarbeitung der Multicast-Pakete ist im Gegensatz zu den Unicast-Paketen zentralisiert und findet in Software auf dem SCP statt.

4.2 Strategien zur Sicherheit bei Überlastung des Systems

Das Netzwerk sollte stabil bleiben, auch wenn der GIGAswitch sehr stark überlastet ist. Dies ist der Fall, wenn die Rate der ankommenden Pakete ein Maximum erreichen. Die Technik, die eine regelmäßige, stabile Abwicklung des Durchsatzes garantiert, benutzt Preallokation von Speichern für Pakete und Warteschlangenmethoden. Es werden keine Lösungen gesucht, die nur ein robustes System garantieren. Sie sollten darüber hinaus für einen hohen Durchsatz bei Überlast sorgen.

Der SCP ist die zentrale Anlaufstelle vieler Pakete im GIGAswitch. Dazu zählen unter anderem Pakete, die geflutet werden müssen, 802.1d Kontrollpakete, Inter Card Command-Pakete (ICC) und Simple Network Management Protokoll-Pakete (SNMP). Die 802.1d Kontrollpakete sind Dateneinheiten des 802.1d Spanning Tree Algorithmus, der eine stabile Netzwerktopologie garantiert. Die ICC-Pakete sorgen für die interne Kommunikation zwischen den verschiedenen Karten. Die SNMP-Pakete dienen der Kontrolle des GIGAswitch Systems.

Die Crossbar-Access-Control-Hardware (XAC) auf dem SCP ist für die Vermeidung des Verlust wichtiger Pakete bei Überlast zuständig. Durch einen Parsingvorgang versucht die XAC unter den ankommenden Paketen die wichtigen zu erkennen. Die Empfangsgarantie von jedem Pakettyp kann der XAC durch das Bereitstellen von Pufferspeichern für

jedem Pakettyp erreichen. Durch Zusammenarbeit von Hard- und Software wird immer eine Überblick über den augenblicklich verbrauchten Puffern je Pakettyp gegeben. Die ankommenden Pakete, denen kein Pufferplatz zugeordnet werden kann, werden durch das XAC verworfen. Die zu flutende Pakete, die ohne eine Ratenlimitierung dem SCP erreichen, werden ebenso im Falle einer Überlastung des SCP verworfen. Manche Pufferquoten, z.B die der ICC-Paketen, können so angepaßt werden, daß sie grundsätzlich zu keiner Verwerfung führen.

Wenn der SCP die Abarbeitung der Pakete in FIFO-Reihenfolge vornimmt, kann der Empfang von jedem Pakettyp gesichert werden. Bei dieser Methode bleibt jedoch die zeitliche Reihenfolge bei der Bearbeitung wichtiger Pakete nicht berücksichtigt. Deshalb werden im ersten Schritt die ankommenden Pakete abhängig von ihrer Abarbeitungspriorität in Warteschlangen gesetzt. Durch Spaltung einer Warteschlange in mehrere wird eine Bearbeitungshierarchie für die wichtigeren Pakete geschaffen.

Der SCP benutzt den Eingangsport zur Klassifizierung ankommender Pakete. Somit kann eine unakzeptable Rate ankommender Pakete früher erkannt werden und entsprechend die Puffervergabe verweigert werden. Eine Überlastung des Hintergrundverkehrs im System durch den Einfluß solcher Pakete kann so reduziert werden. Eine einfache Unterscheidung könnte im System zu unangenehmen Vorkommnissen führen, wie z.B das Belegen aller Puffer durch Multicastpakete oder Pakete mit unbekannter Zieladresse.

Ein weiterer Faktor, der zur Instabilität des Systems führen kann, ist das Auftreten von Aktivitäten, die unterbrechungsgesteuert sind. Solche Aktivitäten werden vom System vorrangig behandelt. Dabei ist es aber möglich, daß so viele Unterbrechungen auftreten, daß die CPU keine Zeit mehr für andere Aufgaben hat. Es gibt mehrere Möglichkeiten auf dem SCP, um die Anzahl dieser Unterbrechungen zu limitieren. Eine Methode sieht die Kombination einer Unterbrechung mit einer Pulsmaske vor. Diese Pulsmaske wird durch die Software vergeben. Nachdem das Auftreten einer Unterbrechung festgestellt wurde, wird die Pulsmaske ausgelöst. Die Unterbrechung wird damit in einer gewissen Zeitspanne verhindert.

Es gibt zwei verschiedene Interrupttypen. Bei dem ersten Typ wird eine feste Anzahl von Aktivitäten sequentiell nacheinander ausgeführt. Darum ist es in dem Falle ausreichend, die Anzahl der Unterbrechungen zu limitieren. Der zweite Typ kommt dann vor, wenn die Pakete schneller ankommen, als der Softwaretreiber zu ihrer Bearbeitung Zeit benötigt. Daher sollte hier ein Mechanismus gefunden werden, um die Bearbeitungszeit dieser Unterbrechungen zu begrenzen. Dazu fragt der Softwaretreiber für die Paketverarbeitung zyklisch die Mikrosekundenuhr des Systems ab, um die Bearbeitungszeit zu messen. Ist die festgelegte Grenze für die Bearbeitungszeit erreicht, wird die Bearbeitung durch den Softwaretreiber beendet.

5 Module des Systems und ihre Funktionen

5.1 Die Clock Card

Die Clock Card generiert im System den Takt und stellt eine Anzahl von zentralen Funktionen zur Verfügung. Diese Funktionen sind dort plaziert und nicht auf der Backplane,

da diese passiv und damit zuverlässiger sein soll.

Die Clock Card dient zur Speicherung der 48-bit IEEE 802 Adressen, die dem Switch selbst zugewiesen sind. Ferner enthält sie die Zuweisungslogik für den Backplane-Bus, die für die Initialisierung und einen Teil der Kommunikation zwischen den Modulen zuständig ist.

Die Managementparameter sind in stabilen Speichern auf der Clock Card abgelegt. Der Vorteil gegenüber der Plazierung im SCP liegt darin, daß so die Koordination von Updates zwischen den SCP-Moduln vereinfacht wird. Die Clock Card wählt auch das SCP-Modul aus, das für die Kontrolle des GIGAswitch Systems zuständig ist. Die Clock Card generiert und verteilt die Systemzeit an die einzelnen Module durch einen gemeinsamen Speicher auf den GPI Chips der jeweiligen Module. Das Ein- und Ausbauen von Modulen wird auch durch die Clock Card entdeckt, da sie laufend alle Slots der Backplane über den Module Identification Bus, welcher die Slots verbindet, abfragt.

Die Plazierung dieser Funktionen in die Clock Card erhöht nicht wesentlich die Komplexität dieses Modules, sorgt aber für eine Erhöhung der Zuverlässigkeit und Reduzierung der Komplexität in anderen Modulen.

5.2 Der Switch Control Processor

Der SCP besteht aus einem MIPS R3000 Prozessor mit 64 KByte Befehls- und Datencache, Schreibpuffer, 16 MByte DRAM, 2 Mbyte Flash-Speicher und einem Crossbaranschluß [WO95]. Der große DRAM Speicher stellt Puffer für die Pakete zur Verfügung und enthält eine Adreßdatenbank für den Switch. Die Crossbarverbindungen werden durch das XAC kontrolliert. Das XAC enthält einige robuste Funktionen, die im Falle der Überlastung des GIGAswitches eingesetzt werden. Sie sorgen für eine reibungslose Abarbeitung der Pakete bei hohem Durchsatz. Das XAC ist in 2 programmierbaren Gate-Arrays mit RAM implementiert. Der Grund für die Einsatz des MIPS-Prozessor im GIGAswitch war die gute Entwicklungs-und Simulationsumgebung für diesen Prozessor.

Es wurde eine Simulation des SCP benutzt, um ein optimales Hard- und Softwaredesign zu erhalten. Ergebnisse aus der Simulation der SCP-Software wurden als Eingabedaten bei der Cachsimulation benutzt. Diese Cachsimulation berücksichtigte auch den Zugriffsaufwand für die verschiedenen Teile der Speicherhierarchie hinter dem Cache. Man stellte fest, daß ungefähr die Hälfte der vom Compiler erzeugten Codes Lade- und Speicherbefehle waren, so daß die Größe der Caches und die Kapazität der Schreibpuffer von großer Bedeutung sind. Die Ergebnisse der Cachesimulation wurden dann zur Optimierung von Software-Komponenten benutzt. Einige Teile der Software wurden darauf aus dem typischen Pfad der Paketabarbeitung genommen. Ebenso sind manche elementaren Operationen neu in Assembler codiert worden, was zu kompakterem Code und weniger Speicherzugriffen führte.

5.3 Die FDDI Line Card

Eine FGL-Karte enthält ein FDDI-Port Subsystem pro Port (2 für FGL-2, 4 für FGL-4) und ein Prozessor Subsystem. Die FDDI-Subsysteme sind vollkommen unabhängig

voneinander Der Prozeß des Sendens eines Paketes von einem FDDI-LAN zu einem anderen ist immer identisch, egal ob die FDDI-Ports auf derselben FGL-Karte liegen oder nicht. Das Prozessor-Subsystem arbeitet mit einem 68302er Mikroprozessor, mit 1 Mbyte DRAM, 512 Kbyte ROM und 256 Kbyte Flash-Speicher. Der Flash-Speicher wird zur Initialisierung der Konfiguration und des Setups, zur Fehlererkennung sowie Firmware Funktionen benutzt. Eine FGL-Karte enthält spezielle Tochterkarten, nämlich ModPMD-Karten, um verschiedene physikalische Medienanbindungen zu implementieren. Die verschiedene PMDs unterstützen Single- bzw. Multimode Glasfasern, die jeweils für Entfernungen bis zu 2 bzw. 40 Kilometern eingesetzt werden können. Es sind auch PMDs für den Anschluß an Unshielded Twisted Pair (UTP) Kupferkabel der Kategorie 5 verfügbar, die eine Entfernung von maximal 100 Metern ermöglichen.

Sowohl FGL-2 als auch FGL-4 können 4 PMDs beinhalten. Die FGL-2 bietet Anschluß an zwei FDDI-Netze. Dabei kann jeder Anschluß entweder SAS (1 PMD) oder DAS (2 PMDs) sein. Die FGL-4 unterstützt nur SAS-Anschlüsse.

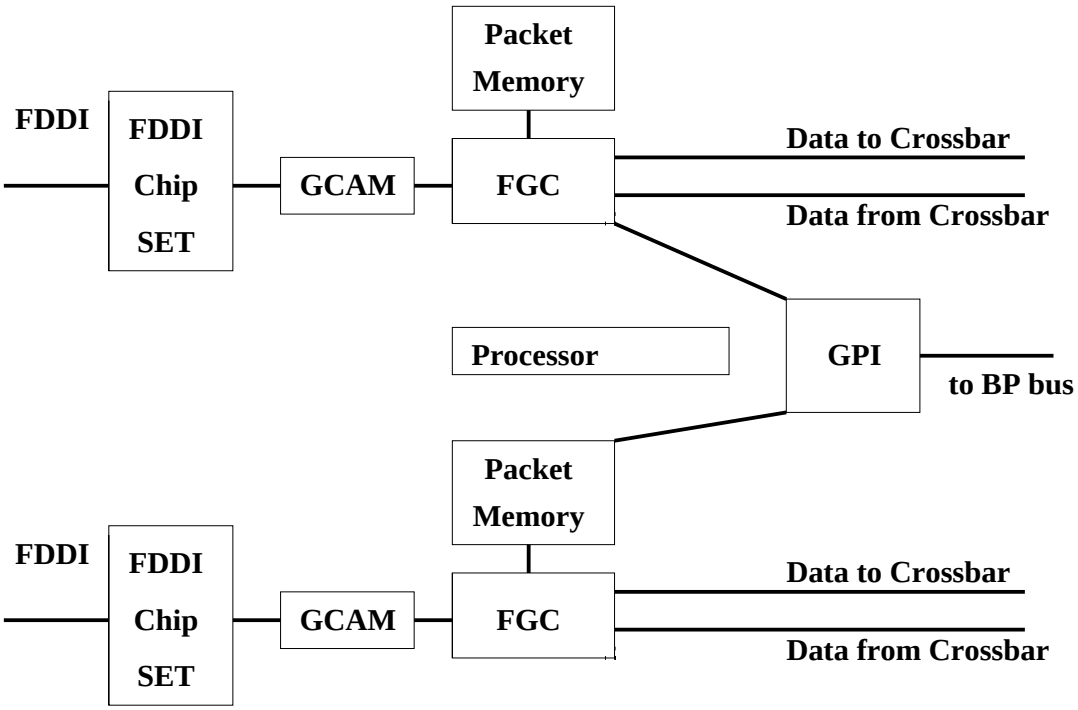

Abbildung 84. FGL2 Block Diagram

Den Kern der FGL-Karte bietet die Paketweiterleitungskomponente, die aus zwei DEC-Chips (FGC und GCAM) besteht.

FDDI to GigaPort Controller (FGC) Der FGC-Baustein ist ein großer Gate-Array-Chip mit ca. 250.000 Transistoren. Er stellt Funktionen zur Verwaltung der Warteschlangen und Verarbeitungsfunktionen für die Pakete auf der FGL zur Verfügung. Er kontrolliert ebenfalls den korrekten Paketfluß zum Crossbar und zu den FDDI-Data-Link-Subsystemen. Er verteilt die Pakete nach ihrem Typ in Warteschlangen, deren Größe durch die Firmware zum Startzeitpunkt festgelegt wird. Der FGC bestimmt die zu bear-

beitenden Pakete und entscheidet ferner, an welchen Port sie weitergeleitet werden. Er stellt zudem fest, ob ein Paket neue, ihm unbekannte Quelladressen enthält. Außerdem besitzt er noch Paket- und Bytezähler.

GIGAnet Content Addressable Memory (GCAM Chip) Der GCAM-Chip hat 2 Datenschnittstellen. Eine 16-Bit-Schnittstelle, die durch den Prozessor benutzt wird, und eine 8-Bit-Schnittstelle, die einen Vergleich mit Paketfeldern im Fluge ermöglicht. Es können 1-Byte Frame-control-Felder, 6 Byte Quell- und Zieladressen, 1 Byte DSAP-Felder und 5 Byte SNAP Protokollidentifikatoren verglichen werden. Dabei kann von FGC alle 80 ns ein neuer Vergleich gestartet werden.

6 FDDI Bridge Implementierung

Die FDDI Line Card und der SCP haben jeweils zwei 100 Mbit/s Simplexverbindungen zum Crossbar. Der Paketverkehr kann simultan in beide Richtungen fließen. Die Quelladresse (SA), die Zieladresse (DA) und der Protokolltyp (PID) von jedem Paket wird im GIGAswitch durch die FGC-Hardware auf der Line Card durchsucht. Das Ergebnis dieser Suche ergibt die Entscheidung darüber, ob ein Paket gefiltert, verworfen oder weitergeleitet wird. Falls ein Paket weitergeleitet wird, wird ein kleiner Header angelegt und das Paket in die Warteschlange zum Crossbar eingereiht.

Die Puffer für die Kontrollpakete der Bridge werden aus einem separaten Pool vergeben. So können keine Konflikte mit den eigentlichen Datenpakete im System entstehen. Die meisten Pakete fließen durch den Crossbar zu einer anderen Line Carte, welche die Pakete auf einen bereits bestimmten FDDI-Ring weitergibt. Falls der Ausgangsport frei sein sollte, beginnt das GIGAswitch System mit der Beförderung des Pakets, sobald ein Fortsetzungssignal vom FGC vorliegt.

Für den Durchsatz mancher Pakete ist der SCP verantwortlich. Zu dieser Gruppe zählen die Multicast-Pakete und Pakete mit unbekannter DA.

Eine Bridge kann genauere Informationen über die Lokation von 48-Bit MAC-Adressen im Netzwerk gewinnen, wenn sie alle Pakete auf jedem angeschlossenem LAN untersucht und dabei feststellt, auf welchem Bridge-Port welche Quelladressen vorkommen. Durch Kooperation zwischen dem SCP und den Line Cards wird eine Haupttabelle auf dem SCP erzeugt, in der die Abbildung von 48-Bit-Adressen auf Ports gespeichert wird. Zusätzlich wird durch Kooperation zwischen dem SCP und den Line Cards versucht, eine schwache Konsistenz zwischen der Haupttabelle auf dem SCP und den Tabellen auf den Line Cards zu sichern.

Falls eine Line Card ein Paket erhält, dessen Quelladresse nicht in der lokalen Adreßtabelle eingetragen ist, schickt sie eine Kopie dieses Pakets an den SCP. Der SCP speichert die Eintragung zwischen dem Empfangsport und der 48-Bit Adresse in seiner Hauptadreßtabelle und benachrichtigt alle Line Cards über die neue Adresse. Zusätzlich fragt der SCP die Line.Cards in regelmäßigen Intervallen ab, ob neue Adresse vorliegen, da es möglich ist, daß die Kopie des Pakets am Eingang zum SCP verloren geht. Die FGC-

Hardware auf der Line Card kennzeichnet neu entdeckte Adressen dadurch, daß sie ein Source-Seen-Bit in ihrer Adreßtabelle setzt.

Wenn Stationen in einem LAN ihre Position wechseln, werden ihre Adressen in der Adreßtabelle gelöscht, falls innerhalb einer vom Management spezifizierten Zeitspanne die alten Adressen nicht mehr auftauchen. Dieser Prozeß wird mit 'Aging' bezeichnet. Die Line Card, die mit dem entsprechenden LAN verbunden ist und die jeweilige Adresse enthält, ist für das Aging zuständig. Die Firmware auf der FGL überprüft in regelmäßigen Abständen die Adreßtabelle nach dem Source-Seen-Bit und nach Adressen, die veraltet sind. Die veralteten Adressen werden markiert und aus Aufwandsgründen erst dann gelöscht, wenn die Tabelle voll ist.

Das GIGAswitch-System sollte schnell reagieren, falls die Quelladresse eines Systems von einem Switch-Port zu einem anderen wechselt. Diese Situation kann dann vorkommen, falls sich die LAN-Topologie ändert. Die Tatsache, daß eine Quelladresse jetzt an einem neuen Port auftaucht, kann für die Optimierung des Aging-Prozeßes ausgenutzt werden. Die Firmware auf der FGL überprüft die Tabelle auf Adressen, die neu sind aber nicht zu diesem Port gehören. Diese Adressen werden in einer Liste an den SCP übermittelt. Der SCP kann dann schnell die nötigen Anbindungen veranlassen.

7 Netzwerkmanagement vom GIGAswitch

Der GIGAswitch wird mit dem Simple Network Management Protokoll (SNMP) verwaltet. SNMP benutzt “get “, “get-next “und “set “Nachrichten für die Abfrage und Veränderung der zu verwaltende Objekte im GIGAswitch.

Ein einzelnes SNMP Set kann mehrere Objekte gleichzeitig verändern, wobei entweder alle oder keines der angesprochenen Objekte aus der Sicht der Management Station sich verändern sollten. Schlägt die Änderung eines Objektes fehl, werden auch die anderen Objekte wieder auf ihre ursprünglichen Werte zurückgesetzt, bevor eine Antwort an die Management Station geschickt wird.

Der SNMP führt eine übergreifende Kontrollfunktion aus und nimmt Steuerungsaufgaben wahr.

8 Resümee

Der GIGAswitch ist eine Bridge mit Routingfunktionen. Der GIGAswitch bietet bei vollem Ausbau Anschlüsse für insgesamt 32 FDDI-Ringe, die durch das Gerät bei Bedarf mit ihrer vollen Kapazität verbunden werden können. Dazu besitzt der GIGAswitch eine leistungsfähige Switcharchitektur, die eine maximale Datenrate von 3.6 GBit/s bewältigen kann. Einige Möglichkeiten und Techniken wurden aufgezeigt, die dafür sorgen, Robustheit und hohen Durchsatz im GIGAswitch zu erreichen. Der GIGAswitch gehört bezüglich Durchsatz zu den leistungsfähigsten Netzwerkkomponenten. Das Maximum der Switchingsrate von 6,25 Millionen Verbindungen pro Sekunde an variablen Paketen beinhaltet zudem ¨hunt groups“ohne Leistungsverlust.

Der GIGAswitch ist in einem weiten Anwendungsbereich (Vernetzung von Workstations oder als LAN- bzw. MAN-Backbones) erfolgreich eingesetzt worden. Ein oder mehrere GIGAswitch Systeme können auch zur Konstruktion eines WAN-Backbones eingesetzt werden. Das WAN ist dann mit Routern umgeben, um die privaten LANs vom WAN-Backbone zu isolieren. Das GIGAswitch System kann so konfiguriert werden, daß es seine Bandbreite, sowohl tausenden von konventionellen LAN-Benutzern als auch immer häufiger vorkommenden Anwendungen wie Videokonferenzen, Multimedia und verteilte Anwendungen zur Verfügung stellen kann.

Teil III

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[ABBY89] H. Amara, T. Balraj, T. Barzilai und Y. Yemini. *PSi: A Silicon Compiler for fast Protocol Processing.* Elsevier Science Publishers. 1989.

[Abd93] H. M. Abdu. *Managing Distributed Applications.* The University of Western Ontario, Canada. 1993.

[ADA82] ADA. Reference Manual for the Ada Programming Language. *D.O.D., Ada, Joint Program Office*, Juli 1982.

[AP93] Mark B. Abbott und Larry L. Peterson. Increasing Network Throughput by Integrating Protocol Layers. *IEEE/ACM Transactions on Networking* **1**(5), Oct 1993, Seite 600–610.

[Bap91] Subodh Bapat. OSI Management Information Base Implementaton. *Integrated Network Management*, 1991, Seite 172–187.

[Bes91] Karl Beschoner. Die Einbettung des OSI-Netzmanagements in das TRANSDATA-Netzmanagement CNM. In *Kommunikation in verteilten Systemen*, Seite 393 – 405. Springer Verlag, 1991.

[BGL91] D. Bacon, A. Goldberg und A. Lowry. *Hermes–A Language For Distributed Computing.* Prentice Hall. 1991.

[BGSC92] P.E. Boyer, F.M. Guillemi, M.J. Serve und J.P. Coudreus. Spacing Cells Protects and Enhances Utilization of ATM Network Links. *IEEE Network* Band 6, 1992, Seite 38–49.

[BHG87] P. Berenstein, V. Hadzilacos und N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison Wesley. 1987.

[BHK94a] Anatol Badach, Erwin Hoffmann und Olaf Knauer. *High Speed Internetworking.* Addison-Wesley, Bonn. 1994.

[BHK94b] Anatol Badach, Erwin Hoffmann und Olaf Knauer. *High Speed Internetworking.* Addison-Wesley. 1994.

[BHS93] Donald F. Box, Duke P. Hong und Tatsuya Suda. Architecture and Design of Connectionless Data Service for a Public ATM Network. *Department of Information and Computer Science, University of California, Irvine, CA 92717-3425, Infocom*, 1993.

[Bie93] Ernst W. Biersack. Performance Evaluation of Forward Error Correction in an ATM Environment. *IEEE Journal on Selected Areas in Communication* **11**(4), May 1993, Seite 631–640.

[Bir90] K. P. Birman. *Isis - A Distributed Programming Enviroment.* Cornell University, Inthaca. 1990.

[BMD95] J. Berghoff, C. Mönch und O. Drobnik. An Approach to Decentralized Management of Distributed Applications. In *Anwendungsunterstützung für heterogene Rechnernetze.* Technische Universität Bergakademie, 1995.

[BN84] A. Birell und P. Nelson. Implementing Remote Procedure Calls. *ACM TOCS* (2-1), Februar 1984.

[BR95] K. Buchenrieder und J.W. Rozenbit. Codesign: An Overview. In *Codesign Computer-Aided Software/Hardware Engineering.* IEEE Press, 1995.

[Buc93] K. Buchenrieder. Codesign and Concurrend Engineering. *IEEE Computer*, Jan 1993, Seite 85 – 86.

[BY92] T. S. Balraj und Y. Yemini. PROMPT - a destination oriented protocol for high-speed networks. *Proceedings of the IFIP WG 6.1/WG 6.4 Second International Workshop on Protocols, Stockholm*, May 13-15 1992.

[Cal87] Bertschas Callager. *Data Networks.* Prentice Hall. 1987.

[CD94] Ken Cox und John DeHart. CMAP Specification 3.0. Technical Report WUCS-94-21, Department of Computer Science, Washington University, Campus Box 1045, One Brookings Drive, St. Louis, MO 63130-4899, July 1994.

[CEG+95a] Georg Carle, Hiroshi Esaki, Aloke Guha, Keiji Tsunoda und Kumiko Kanai. Draft Proposal for Specification of FEC-SSCS for AAL Type 5. ATM Forum Technical Committee, 10. - 14. April 1995.

[CEG+95b] Georg Carle, Hiroshi Esaki, Aloke Guha, Keiji Tsunoda und Kumiko Kanai. Necessity of an FEC Scheme for ATM Networks. ATM Forum Technical Committee, 10. - 14. April 1995.

[Che89] G. Chesson. *XTP/Protocol Engine Design.* Proceedings of the IFIP WG 6.1/6.4 Workshop, Rüschlikon. 1989.

[Com94a] ATM Committee. *atm-forum-94-0168R1.ps: ATM-Forum/94-0168R1 : Credit-Bases FCVC Proposal for ATM.* ATM Committee. 1994.

[Com94b] ATM Forum Technical Committee. *ATM-Forum/94-0439:Action Items for Credit-Based FCVC Proposal.* Internet-ATM-Forum. 1994.

[Cow93] J. Cowan. *OSIMIS GDMO compiler user manual.* University College London, UK. 1993.

[CT90] David D. Clark und David L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *SIGCOMM*, Oct 1990, Seite 200–208.

[DDK+90] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister und H. Rudin. A Survey of Light-Weight Transport Protocols for High-Speed Networks. *IEEE Trans. on Communication* **38**(11), November 1990, Seite 2025–2039.

[Dit91] Andreas Dittrich. Composite Managed Objects. In *Kommunikation in verteilten Systemen*, Seite 378 – 392. Springer Verlag, 1991.

[DM91] Lars Dittermann und Klaus Moth. Flow Enforcement Algorithms for ATM Networks. *IEEE Journal on Selected Areas in Communications* **9**(3), Apr 1991, Seite 343.

[DM93] G. De Michelli. Extending CAD Tools and Techniques. *IEEE Computer*, Jan 1993, Seite 85 – 87.

[DM94] G. De Micheli. Computer-Aided Hardware/Software Codesign. *IEEE Micro*, Aug 1994, Seite 10 – 16.

[Eck92] A.E. Eckberg. B-ISDN/ATM Traffic and Congestion Control. *IEEE Network* Band 6, 1992, Seite 28–37.

[Ehr92] R. Ehrlich. Konstruktion eines GDMO-Übersetzers in C++-Klassen. Diplomarbeit, Universität Karlsruhe, Institut für Telematik, 1992.

[Eng89] H. Engesser. *Duden: Informatik. Ein Sachlexikon für Studium und Praxis.* Dudenverlag. 1989.

[FHHS93] A. Fischer, H. Herpers, D. Holden und S. Sievert. The DOMAINS management language. *Integrated Network Management III*, 1993.

[FZ93] O. Festor und G. Zorntlein. Formal description of managed object behaviour, A rule base approach. *Integrated Network Management III*, 1993.

[Gar91] Klaus Garbe. *Management von Rechnernetzen.* Teubner. 1991.

[Gra94] M. Graf. Traffic Shaping of VBR Video in ATM Endsystems. *4th Open Workshop on High Speed Networks, Brest*, 1994.

[GUS95] W. Görke, S. Ungerer und D. Schmid. *Rechnerstrukturen: Skript zur gleichnahmigen Vorlesung.* Universität Karlsruhe. 1995.

[HA93] Heinz-Gerd Hegering und Sebastian Abeck. *Ingegriertes Netz- und Systemmanagement.* Addison-Wesley. 1993.

[HAB91] Heinz-Gerd Hegering, Sebastian Abeck und Thomas Boehnke. Converting MIB Descriptions into MIB Implementations. In *Second IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Santa Barbara, California*, 1991.

[HABH94] Hegering, Abeck, Boehnke und Heiler. Erweiterung von OSI-Management-Objektbeschreibungen als Basis für eine effektive Implementierung der Managementinformationsbasis. *Informatik Forschung und Entwicklung*, 1994, Seite 93 – 106.

[HKB93] J. H. Hong, M. J. Katchabaw und M. A. Bauer. *Distributed Applications Management Using the OSI MF.* University of Western Ontario. 1993.

[HKM93] E. L. Hahne, C. R. Kalmanek und S. P. Morgan. Dynamic window flow control on a highspeed wide-area data network. *Computer networks and ISDN Systems* Band 26, 1993, Seite 29–41.

[ISO] ISO. Liason statement to ITU-TS SG7 on FDT use of specifying managed object behaviour. ISO/IEC JTC 1/SC 21 N 7981.

[ISO89] ISO. "Information Processing Systems–Open System Interconnection–Management Information Service Part 2: Common Management Service Element". Dezember 1989.

[Kaf92] Franz-Joachim Kaffels. *Netzwerk-Management:Probleme, Standards, Strategien.* DATACOM. 1992.

[KBC94a] H. T. Kung, T. Blackwell und Alan Chapman. *ATM-Forum/94-0085:Use of Flow Control for Effecitive Statistical Multiplexing and Notes on Implementation.* Internet-ATM-Forum. 1994.

[KBC94b] H. T. Kung, T. Blackwell und Alan Chapman. *ATM-Forum/94-0282: Adaptive Credit Allocation for Flow-Controlled VCs.* Internet-ATM-Forum. 1994.

[KBC94c] H. T. Kung, Trevor Blackwell und Alan Chapman. Credit-Based Flow Control for ATM Networks. *Proc.ACM SIGCOMM'94 Symposium on Communications Architectures,Protocols ans Applications*, 1994, Seite 101–114.

[KC94a] H. T. Kung und Kolin Chang. Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks. *Proc. INFOCOM'95*, 1994, Seite 239–252.

[KC94b] H. T. Kung und Koling Chang. *ATM-Forum/94-0080R1:Performance of Credit-Based Flow Control on Challenge Configurations.* Internet-ATM-Forum. 1994.

[KC94c] H. T. Kung und Alan Chapman. *The FCVC (Flow-Controlled Virtual Channels) Proposal for ATM Networks.* Internet-ATM-Forum. 1994.

[KKS87] A. S. Krishnakumar, B. Krishnamurthy und K. Sabnani. *Translation of formal protocol specifications to VLSI design.* Protocol Specification, Testing and Verification VI, North Holland. 1987.

[KL93] A. Kalavade und E. Lee. A Hardware/Software Codesign Methodology for DSP Applications. *IEEE Design & Test of Computers*, Sep 1993, Seite 16 – 28.

[KM95] H. T. Kung und R. Morris. Credit-Based Flow Control for ATM Networks. *IEEE Network* **9**(2), March/April 1995, Seite 40–48.

[Lac95] Hans Lackner. ATM - Völlig losgelöst. *Datacom* Band 1, 1995, Seite 60–66.

[LBE93] Bernd Lamparter, Otto Böhrer und Wolfgang Effelsberg. Vorausschauende Fehlerkorrektur für multimediale Datenströme. In Wolfgang Effelsberg (Hrsg.), *Verteilte Multimediasysteme.* KG. Saur Verlag, 18. - 19. Feb 1993, Seite 61–75.

[Lip90] S.B. Lippman. *C++ Primer.* Addison Wesley. 1990.

[MCD93] K. Marzullo, R. Cooper und Wood M. D. *Tools for Distributed Application Management.* Cornell University. 1993.

[MS91] J.D. Moffett und M.S. Sloman. *The Representation of policies as System Object, in Proc. Conf. Organiz. Comput. Syst.* Atalanta. 1991.

[MS94] David E. McDysan und Darren L. Spohn. *ATM — Theory and Application.* McGraw-Hill, New York. 1994.

[OK91] Hiroshi Ohta und Tokuhiro Kitami. A Cell Loss Recovery Method Using FEC in ATM Networks. *IEEE Journal on Selected Areas in Communication* **9**(9), Dec 1991, Seite 1471–1483.

[OON88] H. Ohnishi, T. Okada und K. Noguchi. Flow Control Schemes and Delay / Loss Tradeoff in ATM Networks. *IEEE Journal on Selected Areas in Communications* **9**(6), Dec 1988, Seite 1609.

[Par94] C. Partridge. *Gigabit Networking.* Addison-Wesley. 1994.

[Rat93] E.P Rathgeb. Policing of Realistic VBR Video Traffic in an ATM Network. *International Journal of Digital and Analog Communication Systems* Band 6, 1993, Seite 213–226.

[Ros91] M.T. Rose. *The Simple Book, An Introduction to Management of TCP/IP-based Internets.* Prentice Hall. 1991.

[Ros95] Ortwin Rose. *Lastkontrolle in Hochgeschwindigkeitsnetzen.* Verlag Shaker. Dissertation an der Universität Karlsruhe, Fakultät für Informatik, 1995.

[Sat95] Shirish S. Sathaye. *ATM Forum Traffic Managment Specification.* ATM Committee. 1995.

[Sch87] M. Schwartz. *Telecommunications Networks, Protocols, Modeling and Analysis.* Addison Wesley. 1987.

[Sch95] G. Schäfer. Einsatz des OSI-Management-Framework im Bereich Anwendungsmanagement. In *Forschungs- und Arbeitsgebiete des Insititut für Telematik.* G. Krüger G. Schäfer(Hrsg) Interner Bericht 22/95 Universität Karlsruhe, 1995.

[Sei94a] J. Seitz. *Integration heterogener Netzwerkmanagementarchitekturen.* VDI. 1994.

[Sei94b] Jochen Seitz. *Integration heterogener Netzwerkmanagementarchitekturen.* VDI Verlag. 1994.

[Sei94c] Jochen Seitz. *Netzwerkmanagement.* Thomson Publishing. 1994.

[Sie94] Gerd Siegmund. *ATM — die Technik des Breitband-ISDN.* R. v. Decker, Heidelberg. Ausgabe 2, 1994.

[SKO+94] Robert J. Souza, P.G. Krishnakumar, Cüneyt M. Özveren, Robert J. Simcoe, Barry A. Spinney, Robert E. Thomas und Robert J. Walsh. The GIGAswitch System: A High-Performance Packet Switching Platform. Digital Technical Journal 6, Digital Equipment Corporation, 1994.

[SN89] K. Sabnani und A. Netravali. A high Speed Transport Protocol for Datagramm / Virtual Circuit Networks. *CCR* **19**(4), September 1989, Seite 246–157.

[Sou94] Nader Soukouti. Automatic Translation of OSI Managed Object Classes to C++ Classes. *IEEE Journal on selected areas in communications* **12**(6), Aug 1994, Seite 1011–1019.

[Sta94] William Stallings. *ISDN and Broadband-ISDN.* Macmillan Publishing, New York. Ausgabe 2, 1994.

[Str92] Bjarne Stroustrup. *Die C++-Programmiersprache.* Addison-Wesley. Ausgabe 2, 1992.

[Sub93] P.A. Subrahmanyam. Hardware/Software Codesign, Cautious Optimism for the Future. *IEEE Computer*, Jan 1993, Seite 84.

[Tan89] A. Tanenbaum. *Computer Networks, Second Edition.* Prentice Hall. 1989.

[TAS93] D. Thomas, J. Adams und H. Schmit. A Model Methodology for Hardware/Software Codesign. *IEEE Design & Test of Computers*, Sep 1993, Seite 6 – 15.

[Teb93] Guillene Teboul. *Verfahren der Verkehrsüberwachung in ATM-Netzwerken.* Diplomarbeit Universität Karlsruhe, Institut für Telematik. 1993.

[VS94] Brett J. Vickers und Tatsuya Suda. Connectionless Service for Public ATM Networks. *Department of Information and Computer Science, University of California, Irvine, CA 92717-3425, ebenfalls in: IEEE Communications Magazine*, August 1994.

[WMC93] M.D. Wood, K. Marzullo und R. Cooper. *Tools for Distributed Application Management.* Cornell Universität. 1993.

[WO95] Robert J. Walsh und Cüneyt M. Özveren. The GIGAswitch Control Processor. *IEEE Network* **9**(1), Januar/Februar 1995, Seite 36–43.

[WOB93] A.S. Wenban, J.W. O'Leary und G.M. Brown. Codesign of Communication Protocols. *IEEE Computer*, Dec 1993, Seite 46 – 52.

[Wol93] W. Wolf. Hardware/Software Codesign. *IEEE Design & Test of Computers*, Sep 1993, Seite 5.

[ZBDP94] M. Zimmermann, J. Berghoff, P. Dömel und B. Patzke. Integration of Managed Objects into Distributed Applications. In *Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management.* Toulouse, France, 1994.

[Zit95] M. Zitterbart. *Hochleistungskommunikation I.* Oldenbourg. 1995.