INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwurf und Implementierung von Managementszenarien zu bestehenden Kommunikationsanwendungen

Bearbeiter : Alexander Keller

Aufgabensteller : Prof. Dr. Heinz–Gerd Hegering

Betreuer : Dr. Sebastian Abeck

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwurf und Implementierung von Managementszenarien zu bestehenden Kommunikationsanwendungen

Bearbeiter : Alexander Keller
Aufgabensteller : Prof. Dr. Heinz–Gerd Hegering
Betreuer : Dr. Sebastian Abeck

Abgabetermin : 15. Mai 1993

Ehrenwörtliche Erklärung

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Mai 1993

....
(Unterschrift des Kandidaten)

Meiner Mutter gewidmet, der ich meine Ausbildung verdanke.

Zusammenfassung

Die Arbeit beschreibt zunächst das Thema der Diplomarbeit sowie deren Umfeld und ordnet sie in die derzeit vom Münchner Netzmanagement Team behandelten Projekte ein.
Daran anschließend erfolgt eine Analyse der Problematik eines Managements verteilter Anwendungen im allgemeinen, die anhand eines konkreten Fallbeispiels, der verteilten Kommunikationsanwendung NWP, vertieft wird. Die Ausgangssituation und die Eigenschaften des NetzWerk–Programms werden geschildert; ebenso erfolgt eine Auflistung einiger für das Verständnis der nachfolgenden Kapitel wichtiger Definitionen. Eine Beschreibung allgemeiner Anforderungen an das Management des NetzWerk–Programms, bezogen auf die *Specific Management Functional Areas*, schließt sich an.
Im darauffolgenden Kapitel werden drei Ansätze für das Management verteilter Anwendungen vorgestellt, diskutiert und auf ihre Verwendbarkeit für die Aufstellung eines Managementkonzepts bewertet.
Kapitel 4 „Konzeption des NWP–Managements“ nimmt eine detaillierte Betrachtung der verteilten Kommunikationsanwendung vor, die für die zu entwickelnden Managementszenarien von großer Bedeutung ist. Hierbei stehen insbesondere folgende Aspekte im Mittelpunkt: Die derzeit vorhandenen Quellen für Managementinformation werden beschrieben; gleichzeitig wird die Funktionsweise von NWP im Detail untersucht. Diese Schritte sind für eine korrekte Modellierung des NetzWerk–Programms unerläßlich.
Parallel zur Beschreibung der technischen Gegebenheiten ist die Fragestellung, welche Anforderungen an ein NWP–Management gestellt werden, Gegenstand der Untersuchung. Die Beantwortung dieser Frage stellt die Basis der in Kapitel 5 entwickelten Managementszenarien dar. Die für NWP geforderte Managementfunktionalität wird in einzelne Module aufgeteilt, wobei jedes für sich eine für den Betrieb von NWP typische Situation behandelt. Der primäre Anwendungsbereich der Szenarien liegt in der Automatisierung von Vorgängen zur Fehlerverfolgung und –diagnose, die zum gegenwärtigen Zeitpunkt noch, mangels geeigneter Werkzeuge, in zeitraubender Handarbeit vom Bedienpersonal erledigt werden müssen.
Das sechste Kapitel beschäftigt sich mit der technischen Realisierung der im vorangegangenen Kapitel entworfenen Managementszenarien. Hierbei spielt sowohl die Modellierung des NetzWerk–Programms als auch die Integration der Szenarien in eine kommerzielle Managementplattform eine Rolle. Letzteres erfährt besondere Beachtung, da das Verfahren zum Einbringen von neuem Managementwissen in ein Netzmanagementsystem keine triviale Aufgabe ist.
Kapitel 7 schließlich faßt den Inhalt der vorliegenden Arbeit zusammen und diskutiert während der Implementierung aufgetretene Schwierigkeiten. Abschließend erfolgt ein Ausblick, der sich mit zukünftigen Entwicklungen im Rahmen der von dieser Arbeit behandelten Thematik befaßt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Übersicht

Dieses Kapitel der Diplomarbeit gibt in Abschnitt 1.1 das Thema der Arbeit im Wortlaut der Anmeldung wieder. Anschließend wird in Abschnitt 1.2 das Umfeld der Arbeit im Kontext der OSI–Netzmanagementarchitektur erläutert. Abschnitt 1.3 beschreibt die Einbettung der Arbeit in laufende Projekte des Münchner Netzmanagement (MNM–) Teams.

1.1 Thema der Diplomarbeit

„Entwurf und Realisierung einer logischen Kommunikationssicht für das Anwendungsmanagement“.

In Kommunikationssystemen tritt an verschiedenen Stellen managementrelevante Information auf, die dem Netzmanagement zur Verfügung gestellt werden muß.
Dieses umfaßt die „Gesamtheit der Vorkehrungen und Aktivitäten zur Sicherstellung des effektiven und effizienten Einsatzes eines Kommunikationssystems“ [HEGE 89].

Eine korrekte Auswahl sowie eine geeignete Strukturierung der bereitzustellenden Information bedingt eine standardisierte Netzmanagementarchitektur. Diese ist im Rahmen des ISO Management Frameworks [ISO 7498-4] durch die ISO erarbeitet worden und sieht vor, jedem Kommunikationssystem eine *Management Information Base* (MIB) zuzuordnen, in der die managementrelevante Information in Form von *Managed Objects* bereitgestellt wird.
Eine weitergehende Spezifikation der MIB–Inhalte wird durch Objektkataloge erreicht, die von verschiedenen Seiten (ISO, OSI Network Management Forum, Hersteller) aufgestellt wurden. Diese Objektkataloge genügen dem von ISO entwickelten Informationsmodell (*Structure of Management Information*) [ISO 10165] zur Beschreibung der *Managed Objects* in einer MIB, jedoch überwiegt hierbei eine Hardwarekomponenten–bezogene Sichtweise.

Im Rahmen dieser Arbeit soll unter anderem untersucht werden, inwieweit die *Generic Managed Object Classes* der ISO [ISO 10165-5] zur Modellierung verteilter Kommunikationsanwendungen verwendet werden können.
Darauf aufbauend werden einige für das Management dieser Kommunikationsanwendungen charakteristische Szenarien entwickelt und auf einer integrierten Netzmanagementplattform prototypisch implementiert.

1.2 Umfeld der Diplomarbeit

Das ISO–OSI–Referenzmodell [ISO 7498] erlaubt eine Strukturierung von Kommunikationssystemen nach Funktionalitätskriterien. Jede Schicht erfüllt eine spezifische Aufgabe und ist durch wohldefinierte Schnittstellen gegenüber den jeweils benachbarten Schichten abgegrenzt. Die Teilaufgaben, die jeder Schicht zugeordnet sind, sind für alle an der Kommunikation teilnehmenden Systeme gleich.
Jede Schicht stellt der darüberliegenden Schicht ihre Dienste zur Verfügung, jede höhere Schicht ergänzt die Dienste der darunterliegenden.

Innerhalb dieser einfachen Systematik von horizontalen, größtenteils virtuellen und vertikalen realen Kommunikationsbeziehungen läßt sich ein für den einwandfreien Betrieb eines Kommunikationssystems notwendiges effizientes und somit komplexes Netzmanagement nicht realisieren.
Daher wurden von ISO vier miteinander in Zusammenhang stehende Modelle entworfen, um einen geeigneten Beschreibungsrahmen für das Netzmanagement aufzustellen (Abbildung 1.1).

Im einzelnen sind dies:

(1) **Das Informationsmodell** (*Structure of Management Information*) [ISO 10165]:
Es legt die Konzepte zur Darstellung von Managementinformation fest; so zum Beispiel die Abstraktion realer Netzbestandteile wie Protokollimplementierungen oder Hardwarekomponenten als logische Beschreibungen. Diese werden als *Managed Objects* bezeichnet und stellen die für das Netzmanagement relevante Sicht auf die Netzressourcen dar.
Eine einheitliche Beschreibungsstruktur wird durch Verwendung von derzeit neun genormten *Templates* in ASN.1–Syntax sichergestellt.

Managed Objects mit gemeinsamen Eigenschaften werden zu Objektkatalogen zusammengefaßt und in der MIB abgelegt. Diese ist ein konzeptioneller Datenbehälter, der die Gesamtheit der Managed Objects eines Kommunikationssystems beinhaltet.

(2) **Das Organisationsmodell**:
Im Organisationsmodell werden die Begriffe „Manager“ und „Agent“ definiert sowie deren Beziehungen zueinander.

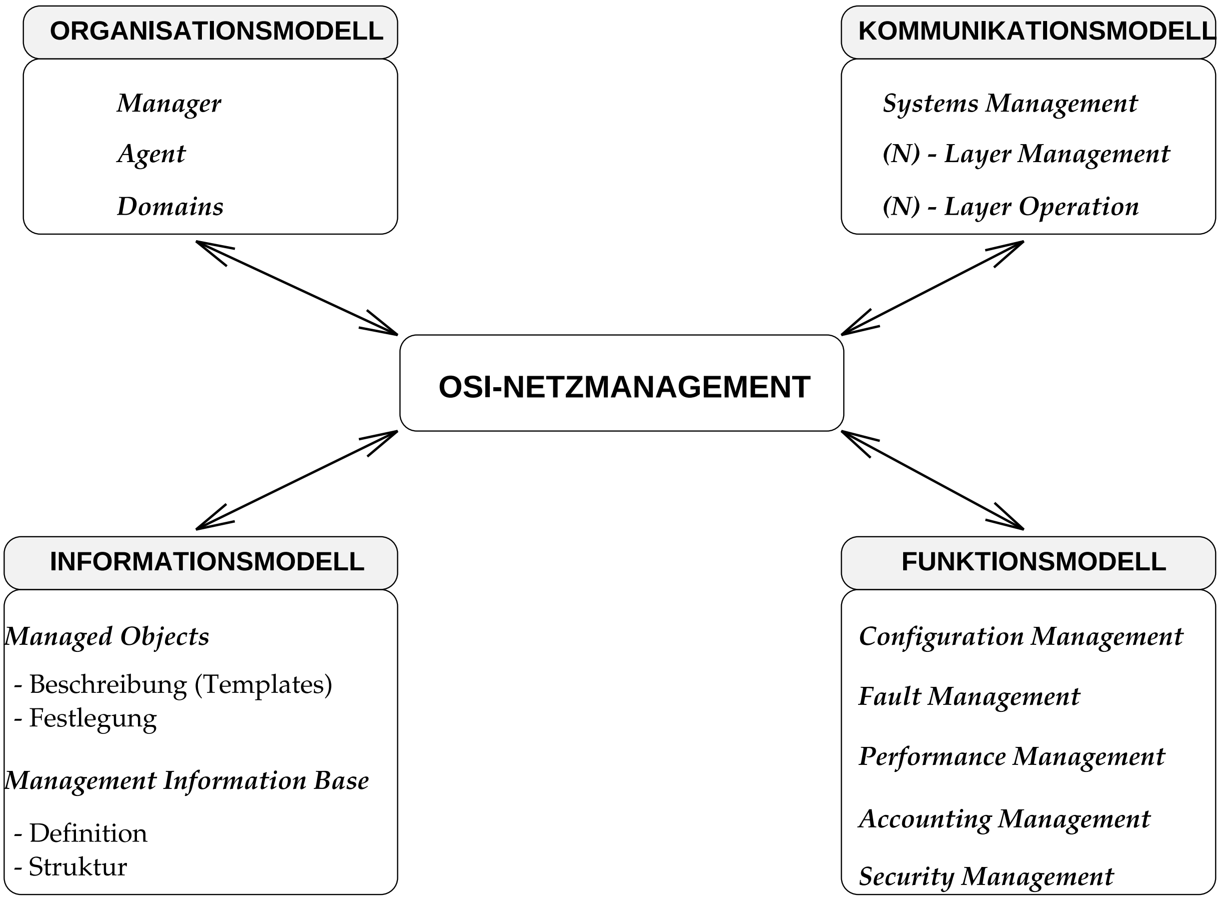

Abbildung 1.1: Die Teilmodelle des OSI–Netzmanagements

Der *Agent* stellt die Schnittstelle dar, mit der eine Managementanwendung, der *Manager*, mit den Managementobjekten in Verbindung treten kann.
Der Begriff der *Managementdomäne* im Sinne einer Gruppierung von Managed Objects nach organisatorischen Kriterien wird ebenfalls im Organisationsmodell eingeführt.

(3) **Das Kommunikationsmodell**:
Es behandelt die Aspekte der Kommunikation zwischen Manager und Agent sowie den Zugriff auf die MIB.
Während das Kommunikationsprotokoll zwischen Manager und Agent, das *Common Management Information Protocol* (CMIP) [ISO 9596-1] genormt ist, unterliegt die Kommunikation zwischen dem Agent und den Managementobjekten keinerlei Vorschriften, da es nicht möglich ist, ein Protokoll zu spezifizieren, welches die Gesamt-

heit von Managed Objects (sowohl Hardwarekomponenten als auch Softwaremodule) abdecken kann (Abbildung 1.2). Insbesondere sind diese Kommunikationsbeziehungen systemspezifischen Eigenschaften unterworfen.

Das Kommunikationsmodell nimmt eine Untergliederung in drei Management–Kategorien vor, die alle Zugriff auf die MIB haben:

- Schichtübergreifendes Management (*Systems Management*)
- Schichtenmanagement (*(N)–Layer Management*)
- Protokollmanagement (*(N)–Layer Operation*)

(4) **Das Funktionsmodell**:
Ausgehend von fünf allgemeinen Funktionsbereichen des Netzmanagements

- Configuration Management
- Fault Management
- Performance Management
- Accounting Management
- Security Management

wurden von ISO eine Vielzahl bereichsspezifischer Management–Hilfsfunktionen [ISO 10164] spezifiziert, die aufgrund ihrer Allgemeinheit flexibel verwandt werden können.

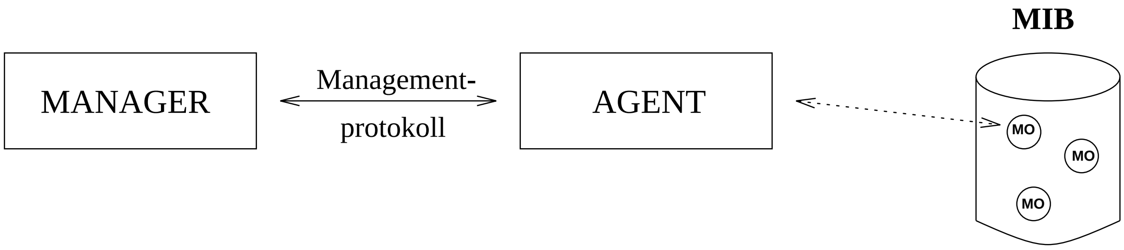

Abbildung 1.2: Kommunikationsbeziehungen: Manager, Agent und Managed Objects

1.3 Einordnung der Diplomarbeit in Projekte des MNM–Teams

Die innerhalb des MNM–Teams entwickelten Konzepte sollen mit Hilfe der integrierten Managementplattform SPECTRUM von Cabletron in der Praxis erprobt werden [MNM UEB].

Hierbei werden insbesondere zwei Richtungen verfolgt:

(1) Modellierung von Netzkomponenten:
Dieses Teilprojekt behandelt die Fragestellung, inwieweit Komponentenwissen mittels der *Inductive Modeling Technology* (IMT) [1]geeignet darzustellen und unter SPECTRUM nutzbar zu machen ist [ABLE 93].
Derzeit existieren Modelle, die innerhalb des MNM–Teams entwickelt wurden, für Sternkoppler (sogenannte *Hubs*) [ASWI 93], [MNM HUB]. Eine Autotopology–Funktion für einen Lannet–Hub [WIEN 93] befindet sich zur Zeit in der Entwicklungsphase.

(2) OSI–Migration:
Aufgrund der Offenheit und der vielseitigen Erweiterungsmöglichkeiten des Netzmanagementsystems SPECTRUM erscheint es machbar, Objektkataloge der ISO und des OSI Network Management Forums [FORUM 90] in die NM–Plattform zu integrieren.
Dies ermöglicht, von einer rein Hardwarekomponenten-bezogenen zu einer OSI–konformen Sichtweise überzugehen, welche die Modellierung *aller* sieben Schichten des ISO–OSI Referenzmodells für Kommunikationssysteme vorsieht [ALSE 92].
Ein Teilaspekt dieser Thematik wird von der vorliegenden Arbeit behandelt.

Der zweite Themenkomplex verdeutlicht die Absicht, einen Übergang von SNMP–basierten [RFC 1157] Netzmanagementlösungen hin zur OSI–Netzmanagementarchitektur zu vollziehen, obwohl diese zum gegenwärtigen Zeitpunkt noch von keiner marktfähigen Netzmanagementplattform explizit unterstützt wird.

Ein erster Schritt hierzu besteht darin, einzelne Schichten von Kommunikationssystemen zu modellieren und im Sinne des *(N)–Layer Managements* zu behandeln. Hierzu werden jeder Schicht sogenannte *Layer Management Entities* (LME) zugeordnet, die auf die der jeweiligen Schicht entsprechenden *Managed Objects* zugreifen und somit den Status der zu managenden Schicht verändern können.
Abbildung 1.3 stellt diesen Zusammenhang graphisch dar und zeigt die derzeit existierenden Schichtenmodelle auf. Es ist klar zu erkennen, daß bisher die Modellierung von Hardwarekomponenten (OSI–Schichten 1 bis 3) im Vordergrund stand, während erst in

[1] Auf die IMT wird in Abschnitt 3.3 detailliert eingegangen

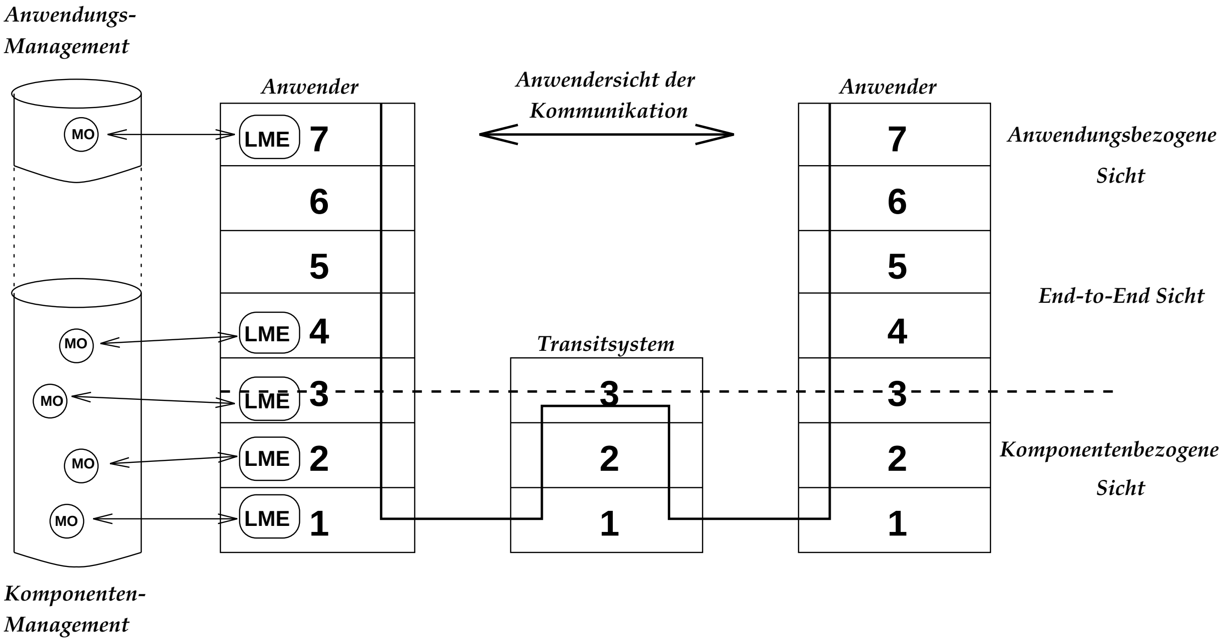

Abbildung 1.3: Das Schichtenmanagement in Verbindung mit dem OSI–Referenzmodell

jüngster Zeit Modelle für das Anwendungssystem (OSI–Schichten 4 bis 7) entwickelt wurden.

Kapitel 2

Anforderungsanalyse

Im folgenden Kapitel werden die Anforderungen an ein Management verteilter Kommunikationsanwendungen im allgemeinen (Abschnitt 2.1) sowie anhand eines konkreten Beispiels (Abschnitt 2.2) herausgearbeitet. Die Unterteilungen von Abschnitt 2.2 beinhalten eine kurze Beschreibung des „Netzwerk–Programms“ der BMW AG sowie die Anforderungen an das Management dieser verteilten Kommunikationsanwendung, welche im weiteren Verlauf der Arbeit Gegenstand der Untersuchungen sein wird.

2.1 Gründe für ein Management verteilter Anwendungen

Große Rechnernetze zeichnen sich, bedingt durch ihr Wachstum über mehrere Jahre, durch einen sehr hohen Grad an Heterogenität bezüglich nahezu aller für ein Netzmanagement relevanten Bereiche aus:

– **Hardware:**
  Vom PC über Workstations bis hin zu Mainframes ist bei vielen großen Unternehmen ein Querschnitt durch die Rechnertechnologie der letzten 20 Jahre vorhanden.

– **Betriebssysteme:**
  Bedingt durch die zum Teil von Herstellerseite forcierten Unterschiede zwischen Rechnerhardware verschiedener Anbieter ist eine Vielzahl proprietärer Betriebssysteme heute im Einsatz.

– **Übertragungssysteme:**
  Die Fixierung von Herstellern auf bestimmte Netztopologien und Übertragungstechnologien sowie zahlreiche Neuentwicklungen innerhalb der letzten Jahrzehnte haben auch auf diesem Gebiet zu einer hohen Diversifikation geführt.

– **Übertragungsprotokolle:**
Da erst in letzter Zeit die Bestrebungen der Standardisierungsorganisationen um Normung der Übertragungsprotokolle Früchte getragen haben, ist es nicht verwunderlich, daß ebenfalls im Bereich der Kommunikationsprotokolle eine hohe Zahl an größtenteils zueinander inkompatiblen de-facto-Standards vorliegt [BLAC 92].

Da es sich heutzutage weder Hersteller noch Anwender leisten können, getätigte Investitionen in Technologie, Geräte, Schulung und Personal nicht weiter zu nutzen, auf der anderen Seite jedoch die Geschäftstätigkeit der Anwender durch betriebssichere Systeme sichergestellt werden muß, ergibt sich die Notwendigkeit, für eine effektive und effiziente Überwachung der stark heterogenen Kommunikationssysteme ein integriertes Netzmanagement einzusetzen.
Dies schließt nicht nur ein Management von Hardwarekomponenten ein, sondern auch, und gerade, ein Management der höheren, anwendungsorientierten Kommunikationsschichten, also von Software-Modulen.

Gründe hierfür liegen im wesentlichen darin, daß große Anwender zur Überbrückung der Unterschiede zwischen den systemspezifischen und daher zueinander inkompatiblen Kommunikationsdiensten (wie z.B. *ftp, DECnet Copy*) eigene anwendungsbezogene Kommunikationsdienste entwickelt haben, auf denen die eigentlichen Anwendungen (z.B. CAD-Programme, Datenbanksysteme) aufsetzen und deren Aufgabe darin besteht, die Inkompatibilitäten der darunterliegenden Systemkomponenten für den Dienstnutzer transparent zu machen.
Hiermit verbunden ist

(1) eine Erhöhung der bereits beachtlichen Anzahl an Schichten und damit der Komplexität des Kommunikationssystems, da zusätzlich zu bestehenden Systembestandteilen wie der Netzinfrastruktur, den Kommunikationsprotokollen und systemnahen Kommunikationsdiensten ein weiterer anwendungsnaher Kommunikationsdienst bereitgestellt werden muß.

(2) die Notwendigkeit, neue Managementobjekte zu schaffen[1], bedingt durch die Einführung einer individuellen und daher nicht standardisierten Kommunikationsanwendung.

Moderne verteilte Kommunikationsanwendungen zeichnen sich unter anderem dadurch aus, daß sie keine monolithische Struktur besitzen, sondern aus einer Vielzahl verteilter, interdependenter und hierarchisch angeordneter Prozesse bestehen.
Daraus folgt, daß bereits der Ausfall eines Teilprozesses die ordnungsgemäße Funktion der gesamten Anwendung beeinträchtigen oder sogar zum Erliegen bringen kann.
Dies wiederum impliziert, daß sich ein Management der verteilten Anwendung nicht darauf beschränken kann, diese als Ganzes in der MIB abzubilden, sondern sämtliche Teilprozesse geeignet modelliert und analog zu den anderen Ressourcen eines Kommunikationssystems,

[1] Dieser Aspekt wird in Abschnitt 6.2 vertieft

wie z.B. Hardwarekomponenten in der Management-Informationsbasis (MIB) als *Managed Objects* abgelegt werden müssen, um ein den tatsächlichen Gegebenheiten angemessenes Management der verteilten Anwendung sicherzustellen (Abbildung 2.1).

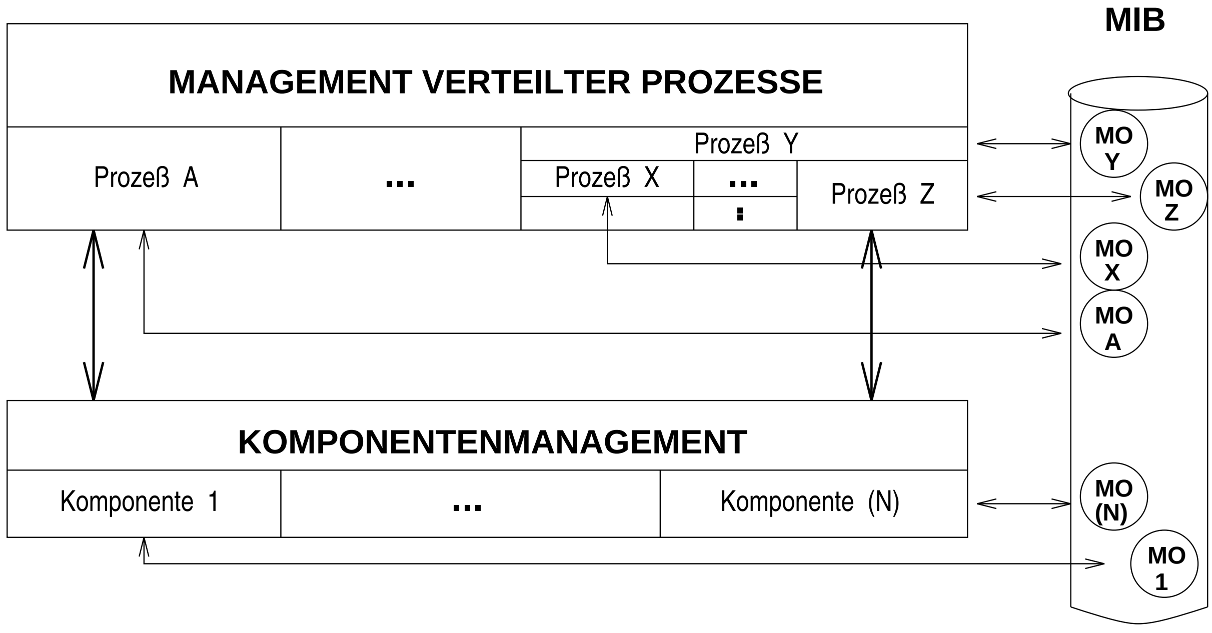

Abbildung 2.1: Komponenten- und prozeßbasierte Modellierung

Ein weiterer Aspekt, der Beachtung verdient, ist die Tatsache, daß ein Management verteilter Kommunikationsanwendungen keinesfalls isoliert möglich ist, sondern ein enger Zusammenhang mit Komponenten- und Systemmanagement besteht, da die korrekte Funktion eines anwendungsorientierten Kommunikationsdienstes das Vorhandensein der Dienste darunterliegender Schichten bedingt [NWP FK].
Diese Tatsache erschwert im Fehlerfalle die Fehlersuche und -diagnose, da ein bei der Nutzung des Dienstes aufgetretener Fehler nicht zwangsläufig ihm zuzuordnen ist. Es ist nämlich durchaus möglich, daß der Fehler in einer darunterliegenden Kommunikationsebene aufgetreten ist und, bedingt durch die Schichtung des Kommunikationssystems, erst an der Dienstschnittstelle bemerkt wird.

Ideal wäre es, in einer integrierten Netzmanagement-Plattform den gesamten Kommunikationsstack abzubilden, um Schicht für Schicht Aussagen darüber zu gewinnen, welche Teile des Systems einer Störung unterworfen sind. Der Operateur an der Netzmanagementplattform erhielte somit bereits eine signifikante Eingrenzung der Fehlermöglichkeiten, da das Netzmanagementsystem ihm bereits das betroffene Kommunikationssystem liefert sowie die darin enthaltene Schicht, in der der Fehler aufgetreten ist.

Die Qualität eines Netzmanagements hängt direkt von der Modellierungsgüte der in einem Kommunikationssystem enthaltenen Hard– und Softwarekomponenten ab.
Ziel des Modellierungsprozesses ist die Abbildung von Elementen der realen Welt in logische Objektbeschreibungen, die dem Netzmanagement zur Verfügung gestellt werden. Es liegt somit auf der Hand, daß der MIB, die das Ergebnis der Modellierung ist, eine zentrale Bedeutung im Netzmanagement zukommt.

Für die Realisierung der MIB sind insbesondere folgende Fragestellungen von entscheidender Bedeutung:

(1) **Notwendigkeit:**
Welche Informationen sind für das Management wichtig und sollen in der MIB enthalten sein ?

(2) **Vollständigkeit:**
Reichen die managementrelevanten Daten realer Elemente aus oder ist es notwendig, diese Daten um zusätzliche Informationen zu erweitern ?

(3) **Strukturierung:**
Kann die Struktur der zu speichernden Informationen in die MIB mit übernommen werden ?

(4) **Flexibilität:**
Inwieweit erlaubt die gewählte Modellierungstechnik, Änderungen und Erweiterungen der logischen Objektbeschreibungen vorzunehmen ?

Um der Komplexität moderner Kommunikationssysteme zu genügen, reicht es nicht aus, lediglich logische Objektbeschreibungen der Netzelemente aufzustellen, sondern es muß ebenfalls Managementwissen bezüglich der Elemente verfügbar gemacht werden.
Die Notwendigkeit hierfür liegt darin, daß Netzmanagement sich nicht nur auf rein passive Überwachungstätigkeiten beschränkt, sondern aktive Eingriffe in den Netzbetrieb erforderlich sind, um die Betriebssicherheit des Rechnernetzes zu gewährleisten. Insbesondere von Seiten großer Netzbetreiber wird erwartet, daß einfache Eingriffe selbständig von einem Netzmanagementsystem durchgeführt werden können, um die Netzoperateure von Routinetätigkeiten weitgehend zu entlasten.
In diesem Zusammenhang stellt sich die Frage, ab wann die durch Netzmanagement verursachte Netzlast eine kritische Größe bezüglich des Gesamtdurchsatzes des Netzes erreicht. Die Verwendung „dummer“ *Agents* führt in der Regel zu hohen Netzlasten, da vom Managerprozeß in gewissen Zeitabständen ein *Polling* der Agenten durchgeführt werden muß. „Intelligente“ *Agents*, die selbständig MIB–Variablen überwachen und die Überschreitung eines vordefinierten Schwellwertes an den Managerprozeß melden, sind gerade im Bereich großer Kommunikationssysteme das Mittel der Wahl.

Die skizzierten Themenkomplexe werfen folgende Fragestellungen auf:

(1) Welches Managementwissen über Netzelemente muß in die Modellierung eingebracht werden ?

(2) Welche Eingriffe in den Netzbetrieb sollen

(a) manuell durch den Netzoperateur

(b) automatisch durch die Managementstation

ausgeführt werden ?

(3) Was sind die Anforderungen an den *Agent* und wie können diese erbracht werden ?

(4) Welche Managementfunktionalität kann von dem Managerprozeß auf den *Agent* verlagert werden ?

(5) Ist es erforderlich, einen eigenen Managementkanal im Sinne eines *outband signaling* einzurichten, um einer Zunahme der Netzlast durch Managementvorgänge vorzubeugen ?

(6) Kann in großen Netzen ein zentrales Netzmanagement überhaupt durchgeführt werden oder ist ein *verteiltes* Netzmanagement sinnvoller ?

Die angeschnittenen Fragen stellen lediglich einen Querschnitt durch die Bereiche dar, die zwangsläufig bei der Realisierung eines Netzmanagements tangiert werden. Um die Zukunftssicherheit des Managementkonzeptes und die damit verbundenen Investitionen zu gewährleisten, ist es zwingend notwendig, die oben genannten Aspekte in die Planung des Netzmanagements mit einzubeziehen.

2.2 Fallbeispiel: Der anwendungsnahe Kommunikationsdienst NWP

2.2.1 Ausgangssituation

Bei der Bayerische Motorenwerke AG sind eine Vielzahl unterschiedlicher Rechnertypen (Personal Computer, Workstations, Mainframes...) im Einsatz, die ihrerseits unter verschiedenen Betriebssystemen (VM/CMS, MVS/TSO, VMS, UNIX in diversen Varianten, PRIMOS, MS-DOS) laufen.
Vernetzt sind diese Rechner mit unterschiedlichen Übertragungsnetzen (Ethernet, Token Ring) und Übertragungssystemen (BDT, NJE, RSCS, DECnet copy) sowie den dazugehörigen Übertragungsprotokollen (SDLC, BSC, TCP/IP, DDCMP).
Es ist unschwer zu erkennen, daß dieser Rechnerverbund einen unter allen Gesichtspunkten hohen Grad an Heterogenität aufweist.

Demgegenüber steht die Forderung, die an den jeweiligen Systemen arbeitenden Benutzer von den systemspezifischen Kommunikationsfunktionen abzuschirmen, um die Bedienfehlerwahrscheinlichkeit zu minimieren.

2.2.2 Eigenschaften von NWP

Diesen Aspekten trägt das **Netzwerk-Programm** (NWP) [NWP ALL] Rechnung:
Es handelt sich hierbei um ein rechner– betriebs– und übertragungssystemunabhängiges Programmsystem, welches dem Benutzer eine einheitliche Schnittstelle auf allen im Verbund zusammengeschlossenen Rechnern bietet, um Dateien innerhalb des Verbunds einfach und sicher zu senden bzw. anzufordern.
Ein Benutzer[2] kann mit Hilfe von NWP beispielsweise Dateien von einer UNIX-Workstation in einem Ethernet-Verbund zu einem unter VM/CMS laufenden Großrechner übertragen, ohne die systemspezifischen Filetransfer-Programme und deren Befehlssätze kennen zu müssen, die auf den Quell–, Transit– und Zielrechnern verfügbar sind.
Auch Meldungen und Fehlernachrichten der unterschiedlichen Betriebssysteme werden in ein NWP–einheitliches Format umgesetzt, um dem Benutzer deren Interpretation zu vereinfachen. Gleiches gilt auch für das Format von Dateinamen, das durch NWP automatisch den jeweiligen systemimmanenten Konventionen angepaßt wird [NWP BEN].

Ein NWP–Nutzer benötigt zur Durchführung beliebiger Datenübertragungen, unabhängig, von welchem System aus er NWP nutzt, lediglich einen einheitlichen Befehlssatz sowie Angaben über den Zielrechner, Dateien und eventuelle Ausgabegeräte.

NWP ermöglicht dem Benutzer unter anderem

- das Senden einer Datei von einem Rechner zu einem anderen (*gewöhnliche Filetransferfunktion*)
- das Senden einer Datei als Job (*Ausführen des Dateiinhaltes*)
- das Senden einer Datei zu einem Peripheriegerät (z.B. Plotter)
- das Anfordern einer Datei von einem beliebigen Rechner innerhalb des Verbunds

Als anwendungsnaher Kommunikationsdienst setzt NWP auf den systemspezifischen Kommunikationsdiensten sowie den darunterliegenden Schichten auf. Abbildung 2.2 zeigt eine schematische Übersicht des NWP-Verbunds sowie die Schichtung der Kommunikationsfunktionalität am Beispiel einer UNIX-Workstation.

[2] „Benutzer“ sind in diesem Zusammenhang nicht nur natürliche Personen, sondern auch Anwendungssoftware wie z.B. CAD-Programme.

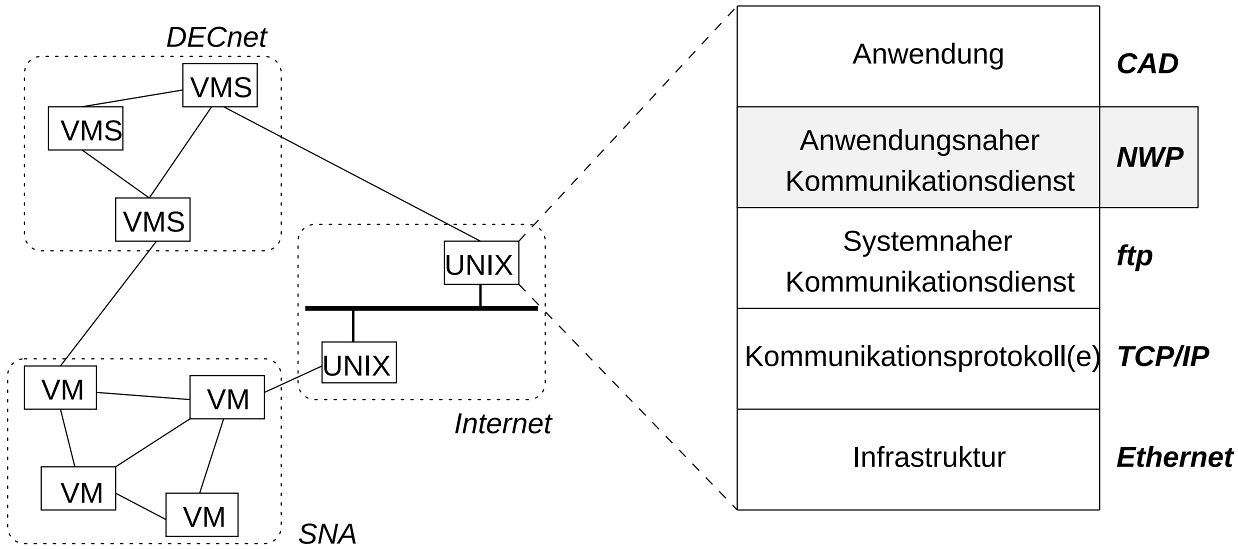

Abbildung 2.2: Einbettung von NWP am Beispiel eines UNIX–Systems

2.2.3 Definitionen im NWP–Kontext

Ein Rechner, welcher in seinem Kommunikationsstack über NWP verfügt, wird im folgenden als **NWP-Knoten** bezeichnet.
NWP–Knoten können Transportknoten (lediglich der NWP-interne Datentransport wird bereitgestellt) oder Nutzerknoten (zusätzlich zur Datentransportfunktion existiert eine NWP-Schnittstelle für den Benutzer) sein.
Die Gesamtheit aller NWP-Knoten heißt **NWP-Verbund**.

Als **Transaktion** wird eine Sequenz von Arbeitsschritten bezeichnet, welche von NWP ausgeführt wird, sobald der NWP–Dienstnutzer eine Datenübertragung angestoßen hat. NWP erzeugt nach Ausführung des Benutzerauftrags in jedem Falle eine Quittung, die Aufschluß darüber gibt, inwieweit bei der Übertragung Fehler aufgetreten sind.
Um dem Benutzer zu ermöglichen, den Status der von ihm initiierten Transaktion abzufragen oder diese Transaktion abzubrechen, generiert NWP außerdem für jede Transaktion einen netzweit eindeutigen **Transaktionscode**. Insbesondere bei Auftreten von Übertragungsfehlern ermöglicht der Transaktionscode genauere Untersuchungen bezüglich Art und Ursache des Fehlers [NWP NMa].

2.2.4 Anforderungen an ein NWP–Management

Zum gegenwärtigen Zeitpunkt umfaßt der NWP–Verbund

- 150 NWP–Knoten
- 6 verschiedene Betriebssysteme und
- 8 unterschiedliche Übertragungssysteme.

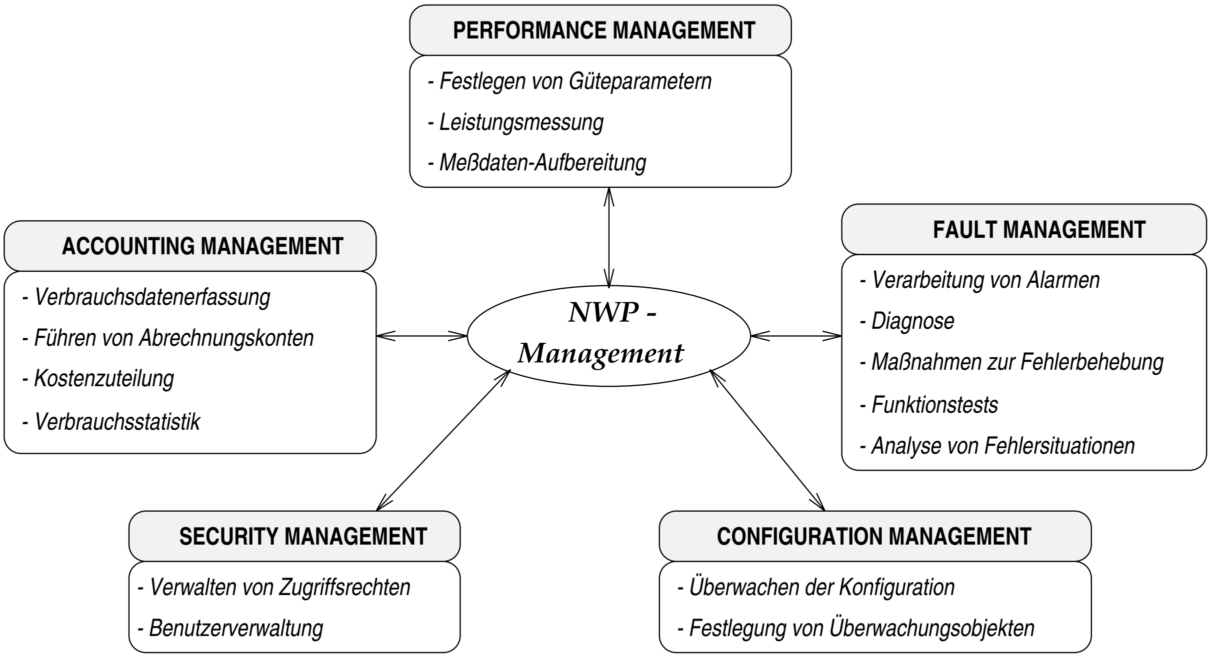

Abbildung 2.3: Allgemeine Anforderungen an das NWP–Management

Größe und Komplexität des NWP–Verbunds stellen hohe Anforderungen an das Management des anwendungsnahen Kommunikationsdienstes NWP.
Eine besondere Problematik resultiert aus der Tatsache, daß für das NWP–Management konkrete Netzelemente wie zum Beispiel Hardwarekomponenten nur eine untergeordnete Rolle spielen und primär abstrakte Größen wie Software–Module, Prozesse oder Warteschlangen vom NWP–Management erfaßt werden müssen.

Der Modellierungsprozeß wird hierdurch wesentlich aufwendiger, als er es für eine rein Hardwarekomponenten–bezogene Modellierung wäre.

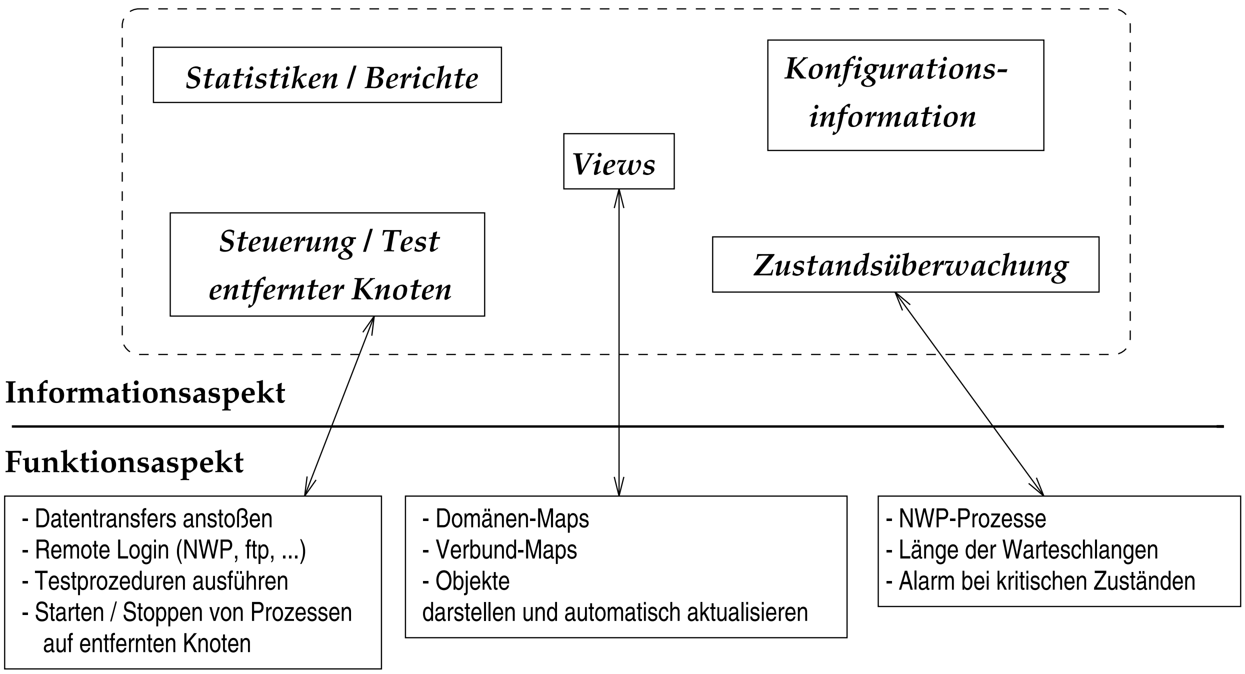

Abbildung 2.4: Anforderungen an das Management eines NWP–Knotens

Ein Beispiel soll diesen Sachverhalt erläutern:
Während die Enthaltenseinsrelation den Aufbau eines Sternkopplers reflektiert und somit zu dessen Modellierung unmittelbar verwendet werden kann (*Port* `ist_enthalten_in` *Board*, *Board* `ist_enthalten_in` *Hub*), ist es durchaus problematisch, in ähnlicher Weise Beziehungen zwischen verteilten Prozessen auszudrücken, da ein Teilprozeß zwar aus funktionaler Sicht einen Bereich abdeckt, der zur Erfüllung des Gesamtauftrages notwendig ist, dabei aber keineswegs physisch im übergeordneten Prozeß enthalten ist.
Anhand dieses Beispiels ist ersichtlich, daß Relationen, welche bei der Modellierung von Hardwarekomponenten hilfreich sind, ihren Zweck im Bereich verteilter Anwendungen unter Umständen nicht erfüllen und stattdessen neue Relationen wie zum Beispiel `hat_Teilprozesse` oder `erfordert` eingeführt werden müssen.

Dem gegenüber gelten die in Abschnitt 2.1 aufgeführten allgemeinen Fragestellungen auch für das NWP–Management.
Hierbei hat besonders der Aspekt der Akquisition und Verarbeitung von *Managementwissen* herausragende Bedeutung [ABEC 92], ermöglicht er doch erst die Umsetzung der vom Netzbetreiber gestellten Anforderungen an das NWP–Management: Nach den Kriterien des Funktionsmodells der OSI–Netzmanagementarchitektur gegliedert, sind in Abbildung 2.3 einige typische, das NWP–Management als Ganzes betreffende Anforderungen auf-

geführt.

Die am „grünen Tisch“ entstandenen *Specific Management Functional Areas* (SMFA) leisten auch in diesem Fall einen fundamentalen Beitrag zur Strukturierung von realen Managementaufgaben.
Neben allgemeinen Anforderungen, die den gesamten NWP–Verbund betreffen, existieren konkrete Ansprüche an das NWP–Knoten-spezifische Management. Abbildung 2.4 zeigt den funktionellen Aufbau eines NWP–Knotens.

Es ist klar zu sehen, daß der Informationsaspekt, der auf den logischen Objektbeschreibungen aufbaut, sowie der Funktionsaspekt, der das prozedurale Managementwissen beinhaltet, untrennbar miteinander verbunden sind. Es ist daher erforderlich, beide Aspekte bei der Modellierung der verteilten Kommunikationsanwendung NWP in angemessener Weise zu berücksichtigen, um der Komplexität von NWP gerecht zu werden.

Kapitel 3

Ansätze für ein Management verteilter Kommunikationsanwendungen

In diesem Kapitel werden drei voneinander unabhängige Ansätze vorgestellt, die für das Management im Rahmen dieser Arbeit relevant sind: Zwei Ansätze, nämlich ODP (Abschnitt 3.1) und die GMOC's (Abschnitt 3.2) stammen von der ISO; der Dritte (Abschnitt 3.3) bildet die konzeptionelle Grundlage für das Netzmanagementsystem, mit dem die Implementierung der in dieser Arbeit entwickelten Managementszenarien durchgeführt wurde. Eine kurze Bewertung (Abschnitt 3.4) schließt dieses Kapitel ab.

3.1 Open Distributed Processing (ODP)

Die Bemühungen, ein einheitliches Architekturmodell für das Management beliebiger Kommunikationsinstanzen und Netzressourcen zu schaffen, bilden die Grundlage der Realisierung verteilter Anwendungen. Beispielhaft seien an dieser Stelle das *Message Handling System* [CCITT X.400] oder das verteilte *Directory* [CCITT X.500] genannt; dem Bereich verteilter Anwendungen sind auch Systeme zur Gruppenarbeit, *Computer Supported Cooperative Work* (CSCW), zuzuordnen, die insbesondere in jüngster Zeit eine starke Verbreitung erfahren haben.

Im Rahmen des *Open Distributed Processing* [ISO N4888-2] erarbeitet ISO in Verbindung mit anderen Standardisierungsorganisationen ein allgemeines Referenzmodell zur Entwicklung verteilter Anwendungen in offenen, heterogenen Systemwelten.
In [GEIH 91] werden die Ziele von ODP wie folgt charakterisiert:

- Definition eines Referenzmodells für ODP im Sinne eines allgemeinen Rahmenwerkes zur Standardisierung verteilter Anwendungen in heterogenen Umgebungen.

- Entwicklung von Normen innerhalb dieses Rahmenwerkes, die verteilte Verarbeitung in offenen Systemen ermöglichen.

Während das OSI–Referenzmodell ausschließlich den Aspekt der Kommunikation zwischen heterogenen Systemen behandelt, umfaßt ODP darüber hinaus alle Belange, die für die Funktion verteilter Anwendungen notwendig sind [HEAB 93].
Dies beinhaltet beispielsweise Fragen der Modellierung von Softwarekomponenten, die Festlegung der Schnittstellen von Programmodulen sowie die Entwicklung von Sicherheits– und Managementkonzepten.

Ein derart umfassender Ansatz erfordert die Einbeziehung einer Vielzahl von Teildisziplinen der Informatik, angefangen bei den Methoden zur formalen Spezifikation über den Entwurf von Betriebssystemen und Programmiersprachen bis hin zum Software Engineering.
Die zahlreichen positiven Erfahrungen, die sowohl auf diesen Gebieten, als auch beim Netzmanagement–Informationsmodell der ISO mit dem objektorientierten Ansatz gemacht worden sind, ließen ihn zum Modellierungsparadigma auch für ODP werden.

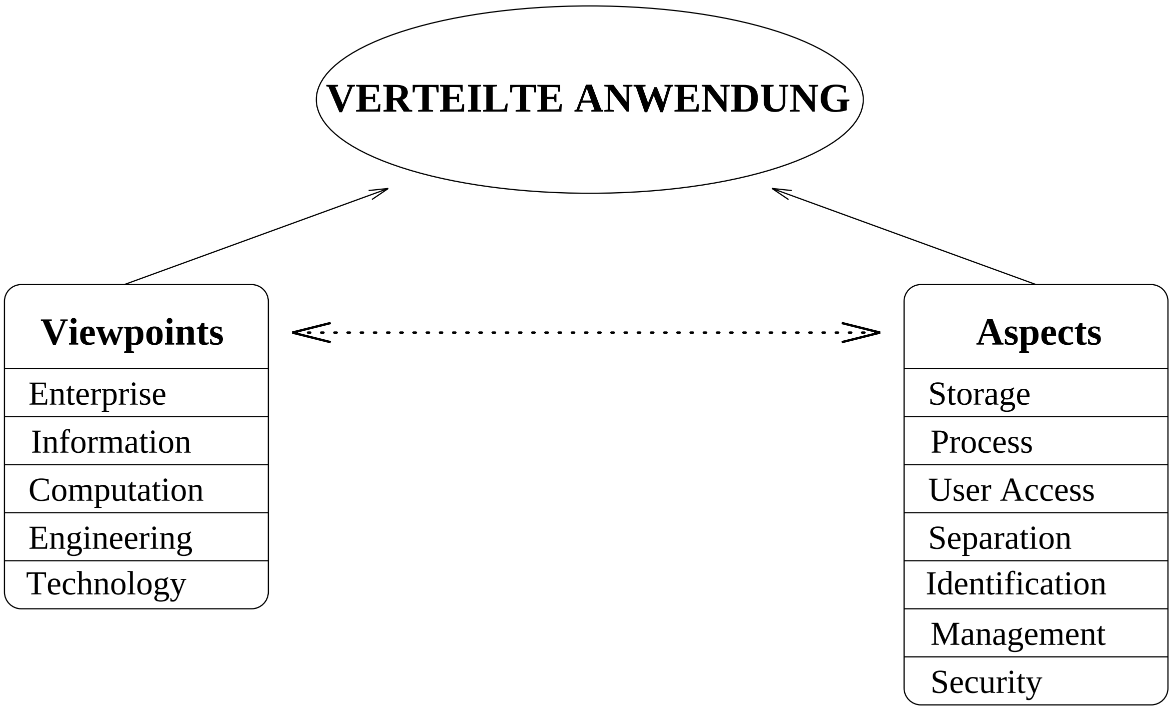

Abbildung 3.1: Viewpoints und Aspects einer verteilten Anwendung

Das Prinzip des „divide et impera“, die Aufgliederung größerer Probleme in zahlreiche kleinere und übersichtliche Teile, welches bereits bei der Entwicklung des OSI–

Referenzmodells mit Erfolg eingesetzt wurde, führt zur Aufteilung der ODP–Thematik in *Viewpoints* und *Aspects* (Abbildung 3.1).

In erster Linie sind die fünf *Viewpoints* zu nennen, welche jeweils spezifische Belange des Gesamtkomplexes betonen und von anderen Fragestellungen abstrahieren:

- **Enterprise**
  Dieser Viewpoint fügt die verteilte Anwendung in die strategischen Ziele und organisatorischen Strukturen der Unternehmung ein, die die verteilte Anwendung nutzt.

- **Information**
  Der Information Viewpoint behandelt die Aspekte der Strukturierung und des Flusses von Information zwischen den Komponenten der verteilten Anwendung, wie zum Beispiel Informationsquellen und –senken, informationsverarbeitende und informationsspeichernde Einheiten. Auch Fragen bezüglich der Gültigkeitsdauer von Information und der Datendefinition sind hier vertreten.

- **Computation**
  Unter dem Computation Viewpoint werden diejenigen Fragestellungen zusammengefaßt, welche die Struktur und die abstrakten Schnittstellen einer verteilten Anwendung betreffen.
  Darunter fallen Themenbereiche wie die Spezifikation der Softwarekomponenten und deren Interaktionen oder Untersuchungen bezüglich der Möglichkeiten einer Parallelisierung bestimmter Module.

- **Engineering**
  Die im vorangehenden Punkt erfolgte Spezifikation wird auf reale Systeme abgebildet. Hierzu zählen auch Vorgaben bezüglich der von der Anwendung zu erbringenden Leistung, des Speicherbedarfs und der benötigten Kommunikationssysteme.

- **Technology**
  Der Technology Viewpoint beinhaltet Beschreibungen der konkreten Hard– und Softwarekomponenten, welche die der verteilten Anwendung zugrunde liegende Infrastruktur bilden.

Viewpoints sind eine strukturbetonte Sicht auf eine verteilte Anwendung, wobei die Viewpoints *Enterprise* und *Technology* eine „top-down“ beziehungsweise „bottom-up“–Betrachtungsweise der Vorgaben darstellen: Während der Enterprise Viewpoint die Anforderungen an die verteilte Anwendung im Unternehmenskontext beschreibt, listet der Technology Viewpoint die bereits existierende Hard– und Softwarebasis auf. Beides sind Gegebenheiten, die den Rahmen abstecken, in dem sich die Entwicklung der verteilten Anwendung bewegen muß.
Demgegenüber stellen die Viewpoints *Information*, *Computation* und *Engineering* Empfehlungen für den eigentlichen Entwicklungsprozeß im Sinne des Software Engineerings dar (Abbildung 3.2).

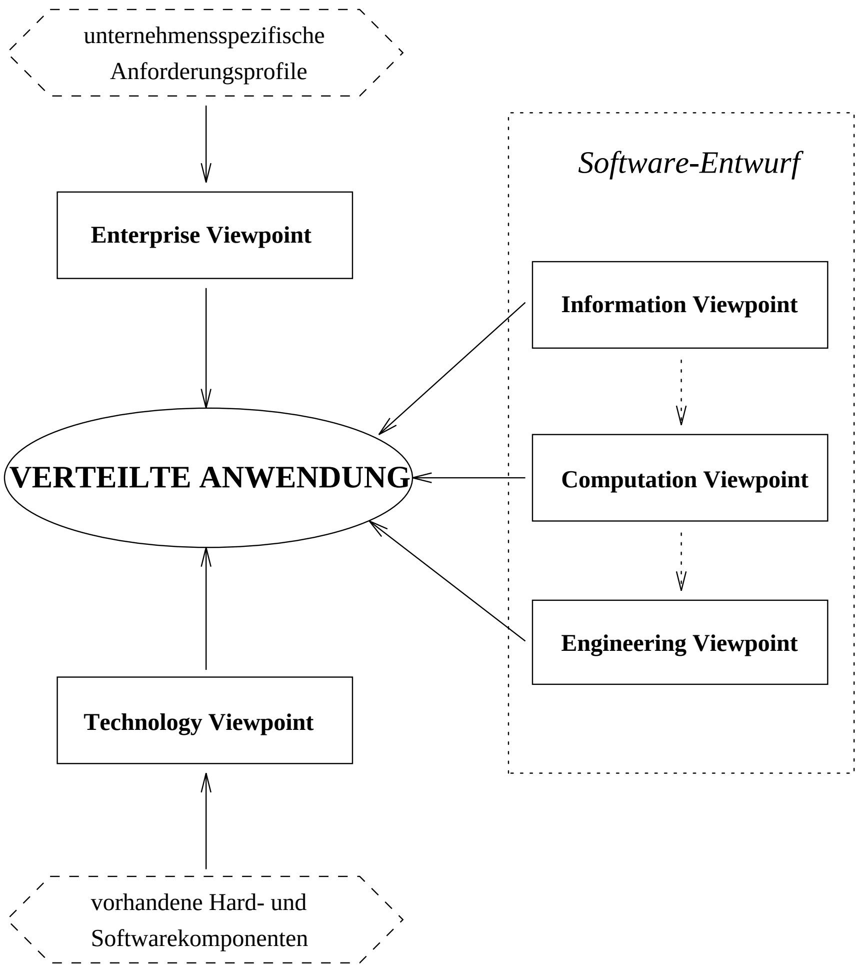

Abbildung 3.2: Die Viewpoints des Open Distributed Processing

Funktionale Charakteristika einer verteilten Anwendung hingegen werden durch sieben ODP–*Aspects* repräsentiert:

- Storage
- Process
- User Access
- Separation
- Identification
- Management

– Security

Viewpoints und *Aspects* sind nach unterschiedlichen Kriterien gruppierte, gleichwertige und interdependente Sichtweisen auf eine verteilte Anwendung.
Es ist sowohl möglich, eine Unterteilung von *Viewpoints* nach *Aspects* vorzunehmen, als auch umgekehrt. Welche Art der Unterteilung im Einzelfall letztlich gewählt wird, bleibt dem Entwickler überlassen.

Das ODP–Referenzmodell stellt, im Gegensatz zum OSI–Referenzmodell, einen „top-down“–Ansatz für verteilte Anwendungen dar und reflektiert die Bemühungen, die das Gebiet des Software–Engineering in den vergangenen Jahren erfahren hat. Das OSI–Referenzmodell stellt nur einen Teilbereich von ODP dar; den der Kommunikation.

3.2 Die *Generic Managed Object Classes* der ISO

Das in Abschnitt 1.2 angesprochene Informationsmodell [ISO 10165] stellt eines der Fundamente dar, auf denen die ISO–Netzmanagementarchitektur beruht.
Das Informationsmodell ist zum gegenwärtigen Zeitpunkt in sechs Teile gegliedert, die folgende Titel haben:

(1) Management Information Model

(2) Definitions of Support Objects

(3) Definitions of Management Attributes

(4) Guidelines for the Definition of Managed Objects

(5) Generic Management Information

(6) Requirements and guidelines for implementation conformance statement proformas associated with management information

Insbesondere der fünfte Teil ist für die vorliegende Arbeit von Bedeutung, spezifiziert er doch unter anderem eine nicht unbeträchtliche Zahl generischer Managementobjekte sowie eine Auswahl möglicher Attribute. Die Beschreibung erfolgt in ASN.1–Templatestrukturen.
In Anlehnung an die Terminologie des OSI–Referenzmodells sind die Managementobjekte so allgemein gehalten, daß sie nicht nur bei der Modellierung des Transportsystems hilfreich sind, sondern auch eine Basis zu einer für das Management adäquaten Darstellung des Anwendungssystems bilden. Abbildung 3.3 ordnet die *Generic Managed Object Classes* in die Schichtenstruktur eines OSI–konformen Kommunikationssystems ein.

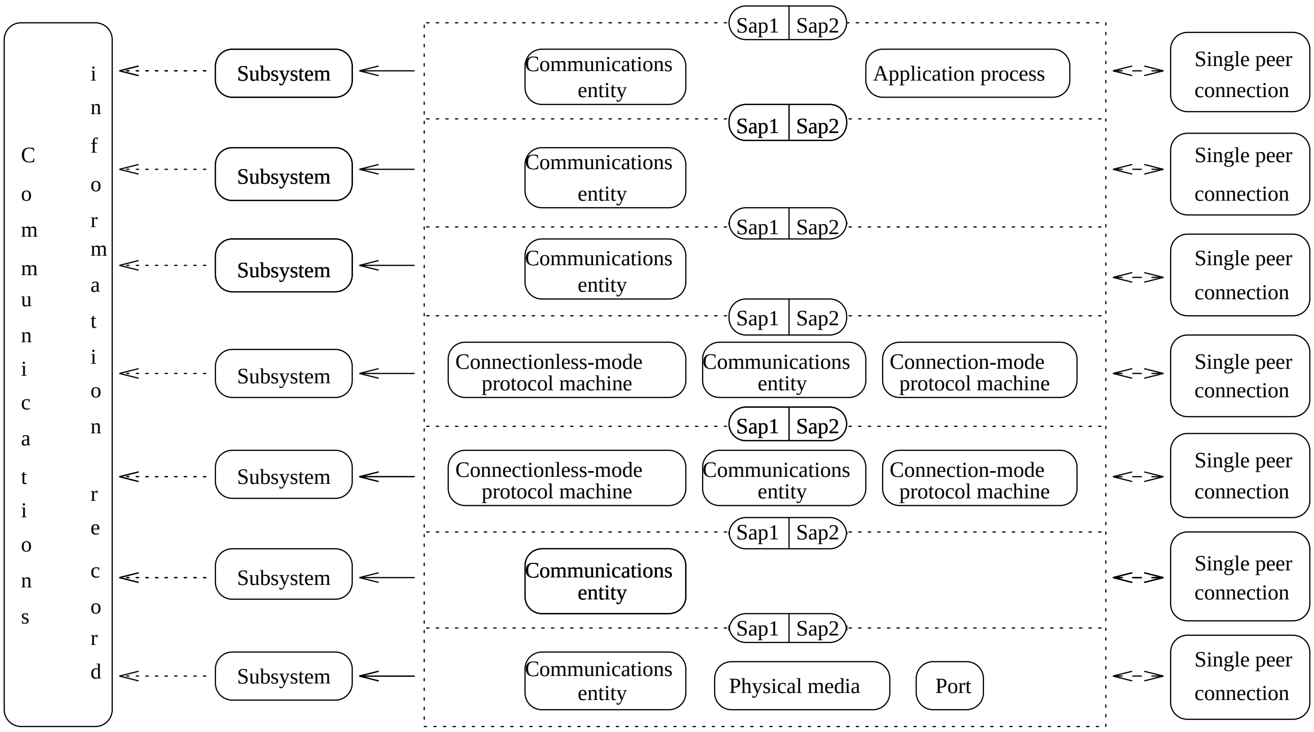

Abbildung 3.3: Die ISO–*Generic Managed Object Classes* im OSI–Schichtenmodell

Es ist klar zu erkennen, daß bei der Definition der *Generic Managed Object Classes* die bereits vom OSI–Referenzmodell bekannten Begriffe (wie SAP [1], Entity,...) direkt übernommen wurden.
Naheliegende und für das Netzmanagement notwendige Ergänzungen stellen die folgenden Begriffe dar:

- **Communications information record**
  Log–Dateien sind insbesondere bei der Fehlersuche und -diagnose ein unentbehrliches Hilfsmittel, da in ihnen die Systemaktivitäten festgehalten sind.
  Ihrer Bedeutung entsprechend, wurde für die Gesamtheit der Log–Informationen eines Systems eine eigene Objektklasse eingeführt.

- **Subsystem**
  Unter dieser Objektklasse werden alle Managementobjekte eines Systems zusam-

[1] Der Unterschied zwischen den Managementobjekten Sap1 und Sap2 besteht darin, daß die Adresse des Sap1 in Abhängigkeit von der des *Service Access Points* der darunterliegenden Schicht gebildet wird, während die Sap2–Adresse diesbezüglich keine Einschränkungen erfährt.

mengefaßt, die zu einer Schicht gehören.

Auffallend ist, daß Begriffe, die einen konkreten Bezug zu einer bestimmten Schicht haben (*Application process, Port, Physical media*) im „Draft International Standard 10165-5“ ausdrücklich mit dem Hinweis versehen sind, in der endgültigen Norm nicht mehr vorhanden zu sein, sofern aus ihrer Verwendung kein unmittelbarer Nutzen entsteht.

Anhand dieser Beispiele und bei Zugrundelegen des objektorientierten Modellierungsparadigmas ist ersichtlich, welche Aufgabe den *Generic Managed Objects* zugedacht ist:
Die *Generic Managed Object Classes* stehen in der Vererbungshierarchie für Managementobjekte an oberster Stelle, da mit Hilfe objektorientierter Methodiken (Vererbung, Allomorphismen) deren Eigenschaften ohne weiteres neuen und anwendungsspezifisch definierten Managementobjekten zugänglich gemacht werden können.

3.3 Die Inductive Modeling Technology (IMT)

Die *Inductive Modeling Technology* ist aus dem *Model Based Reasoning* hervorgegangen, das im Bereich wissensbasierter Systeme verwendet wird. Im Netzmanagement findet sie als Modellierungstechnik Anwendung, da der ihr inhärente objektorientierte Ansatz bei der Modellierung komplexer Kommunikationssysteme gewichtige Vorteile bietet:

– IMT ermöglicht, komplexe Probleme, die im Zusammenhang mit der Modellierung von Netzelementen auftreten, in atomare Einheiten zu gliedern, die anschließend wieder zu einem Ganzen zusammengefügt werden.

– Große Kommunikationsnetze sind einem ständigen Wandel unterworfen; entsprechend müssen Netzmanagementsysteme flexibel in bezug auf Erweiterungen und Modifikationen sein.
Der in IMT verwendete objektorientierte Ansatz unterstützt dies, da er, im Gegensatz zu prozeduralen Konzepten, eine Trennung zwischen Daten und Funktionen ermöglicht.

– Zwei Gegebenheiten prägen den Einsatz eines Netzmanagementsystems:

  (1) Die bereits existierende Hard– und Softwarebasis

  (2) Die Anforderungsprofile des Netzbetreibers an das Netzmanagement

  Es ist daher unerläßlich, zusätzlich zur bloßen Abbildung realer Netzkomponenten *Managementwissen* in den Modellierungsprozeß einfließen zu lassen.
  Durch die der Inductive Modeling Technology inhärente Trennung zwischen *Informationsaspekt* (Datenanteil) und *Funktionsaspekt* (Wissensanteil) ist es möglich, neue Anforderungsprofile in bereits existierende Netzmodelle einzubinden, ohne das gesamte Netzmodell neu konzipieren zu müssen [MNM IMT].

3.3.1 Grundlegende Begriffsdefinitionen

Von fundamentaler Bedeutung für die Inductive Modeling Technology [SPEC CON] ist der Begriff des **Modelltyps** (engl. Model Type). Er entspricht einer *Klasse* des objektorientierten Ansatzes.
Ein Modelltyp wird durch folgende Anteile charakterisiert:

- **Attribute**:
  Sie stellen das deklarative Wissen d.h. die Beschaffenheit eines Modelltyps dar. Ein Attribut beschreibt einen kleinen, abgegrenzten Teil eines Modelltyps.

- **Inference Handler**:
  Sie beinhalten das prozedurale Wissen d.h. das Verhalten eines Modelltyps. Inference Handler legen fest, wie ein Modelltyp auf Änderungen in seiner Umgebung und insbesondere seiner Attribute reagiert.
  Oft ist es empfehlenswert, zur Lösung einer komplexen Aufgabe mehrere Inference Handler einzusetzen. Die Gesamtheit dieser kooperierenden aufgabenspezifischen Inference Handler wird dann **Intelligence Circuit** genannt.

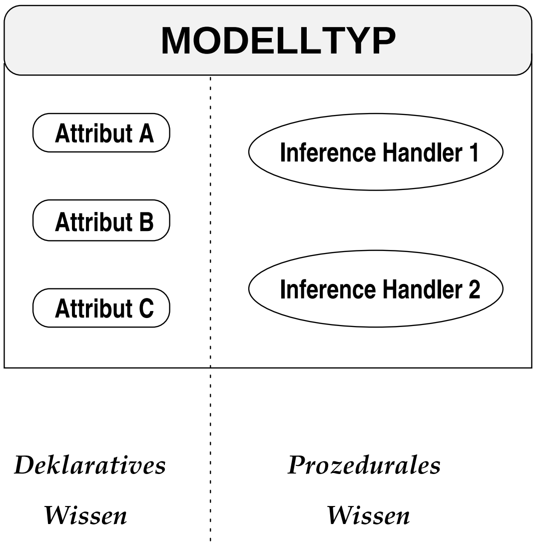

Abbildung 3.4: Die Anteile eines Modelltyps

Abbildung 3.4 stellt die Struktur eines Modelltyps graphisch dar.
Die Verwendung des objektorientierten Paradigmas durch die Inductive Modeling Technology gestattet es, sowohl deklaratives als auch prozedurales Wissen mehrerer Modelltypen an daraus abgeleitete Modelltypen zu vererben (Abbildung 3.5). Daraus folgt, daß Programmcode in abgeleiteten Modelltypen wiederverwendet werden kann.

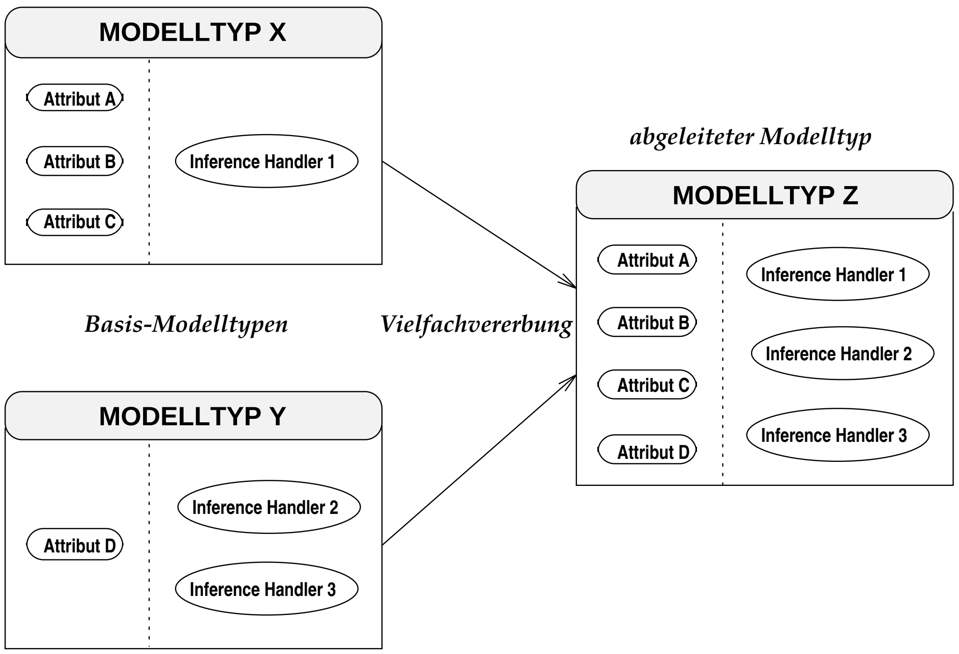

Abbildung 3.5: Möglichkeiten der Vererbung bei Modelltypen

Eine Instantiierung eines Modelltyps wird als **Modell** (engl. Model) bezeichnet und entspricht in der objektorientierten Terminologie der *Instanz einer Klasse*. Ein Modelltyp repräsentiert demnach eine Schablone für Modelle, die deren Struktur und Verhalten a priori festlegt.

Die Dynamik eines Kommunikationsnetzes bewirkt Änderungen der Attribute eines Modells. Die dem entsprechenden Modelltyp zugeordneten Inference Handler überwachen die Modellattribute und führen bei Veränderungen der Attribute bestimmte Aktionen auf den Modellen durch.
Beziehungen zwischen beliebigen Modelltypen werden durch **Relationen** (engl. Relations) ausgedrückt. In der Mathematik definiert man eine Relation als Teilmenge des kartesischen Produkts zweier Mengen. IMT verwendet diesen Begriff in abgewandelter Form: Eine Relation ist hier das kartesische Produkt der Menge aller im Netzmodell existierenden Modelltypen, wobei jedoch auch Spezialfälle wie 1:n und m:n Beziehungen vorkommen können.

Es liegt auf der Hand, daß mit dieser Semantik allein eine korrekte Netzmodellierung undurchführbar ist; eine Relation „`beinhaltet`“ würde somit nicht nur die Kombination „Kommunikationssystem `beinhaltet` Rechner“ zulassen, sondern auch die Aussage „Rechner `beinhaltet` Kommunikationssystem“.

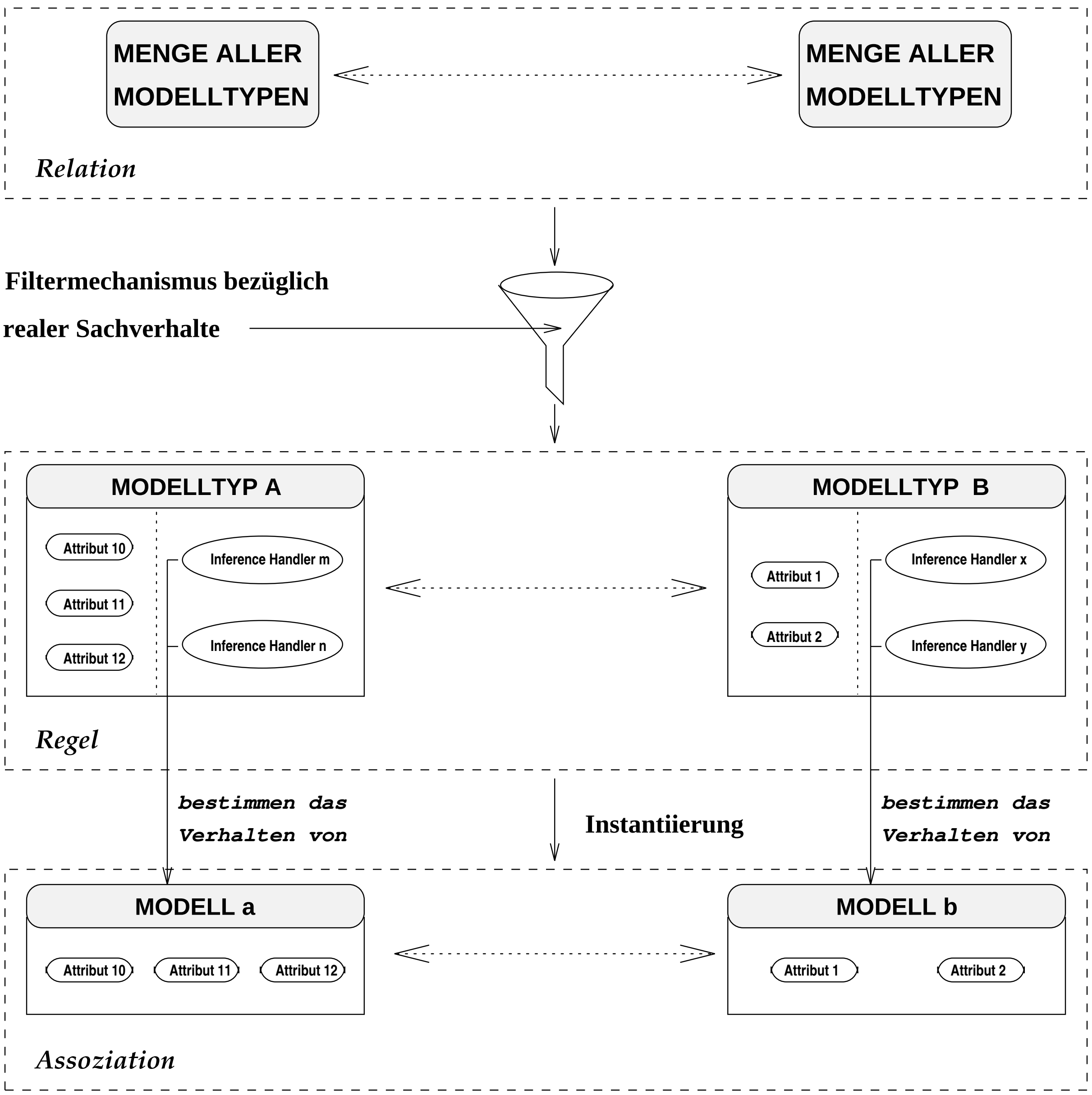

Abbildung 3.6: Übersicht: Wichtige Begriffe von IMT

Es ist daher notwendig, eine Restriktion auf den Relationen bezüglich der Richtigkeit durchzuführen. Hier leisten **Regeln** (engl. Rule) wertvolle Dienste.

Allgemein entspricht ein Satz mit der Syntax <Modelltyp> <Relationsname> <Modelltyp> einer Regel.

Tatsächlich werden beim Übergang von Relationen zu Regeln die ursprünglich vorhandenen Tupel nach Realitätsgesichtspunkten gefiltert d.h. es bleiben nur diejenigen Relationen erhalten, die auch in der Realität vorkommen. Diese Mengenpaare, in Verbindung mit den Namen der Relationen, sind dann die eigentlichen Regeln und müssen vom Entwickler explizit angegeben werden.
Regeln werden benötigt, um das Netzmanagementsystem in die Lage zu versetzen, selbständig Entscheidungen zu treffen und Schlüsse zu ziehen. Falsche Regeln führen unweigerlich zu falschen Schlußfolgerungen und damit zu Fehlern im Netzmodell des Managementsystems [SPEC KBG].

Für die Funktion eines Netzmodells reichen jedoch auch Regeln allein nicht aus, weil bis zu diesem Punkt weder die Regeln, noch die Modelltypen instantiiert wurden. Mit der Instantiierung der Modelltypen zu Modellen werden durch Inference Handler aus den Regeln **Assoziationen** (engl. Associations) aufgebaut.
Die Gesamtheit der Assoziationen stellt dann das funktionsfähige und semantisch korrekte Netzmodell dar: Der Versuch, im Netzmodell eine IEEE 802.5–Komponente in ein IEEE 802.3–Kommunikationssystem einzubringen, wird vom Netzmanagementsystem zurückgewiesen, da keine entsprechende Regel (sehr wohl jedoch die entsprechende Relation) bekannt ist.
Abbildung 3.6 gibt einen Überblick über die in diesem Teilabschnitt aufgeführten Begriffe.

3.3.2 Technische Realisierung von IMT

Die Inductive Modeling Technology bildet das konzeptionelle Gerüst der Netzmanagementplattform SPECTRUM von Cabletron Systems und ist vollständig in diese eingebettet [MNM ARB].
SPECTRUM setzt sich zusammen aus folgenden Bestandteilen:

- **Virtual Network Machine (VNM)**:
  Sie ist der zentrale Teil von SPECTRUM, der neben dem Netzmodell auch das gesamte Managementwissen in Form von Inference Handlern enthält.
  Innerhalb der Virtual Network Machine laufen zahlreiche *Threads* ab, die zum überwiegenden Teil Kontrollfunktionen (Pollen von Attributen, Entgegennehmen von Ereignismeldungen) wahrnehmen und Aktionen durch das Triggern von Inference Handlern anstoßen können. Die Ausführung eines Inference Handlers erfolgt dann innerhalb eines Threads.

- **Objektorientierte Datenbank**:
  In der Datenbank werden alle für das Managementsystem relevanten Informationen abgelegt: Modelltypen, Modelle, Attribute, Relationen, Regeln, Assoziationen...
  Der Zugriff auf die Datenbank erfolgt für den Benutzer transparent ausschließlich durch die VNM, die bei ihrer Initialisierung den gesamten Datenbankinhalt von der Platte in den Hauptspeicher lädt.

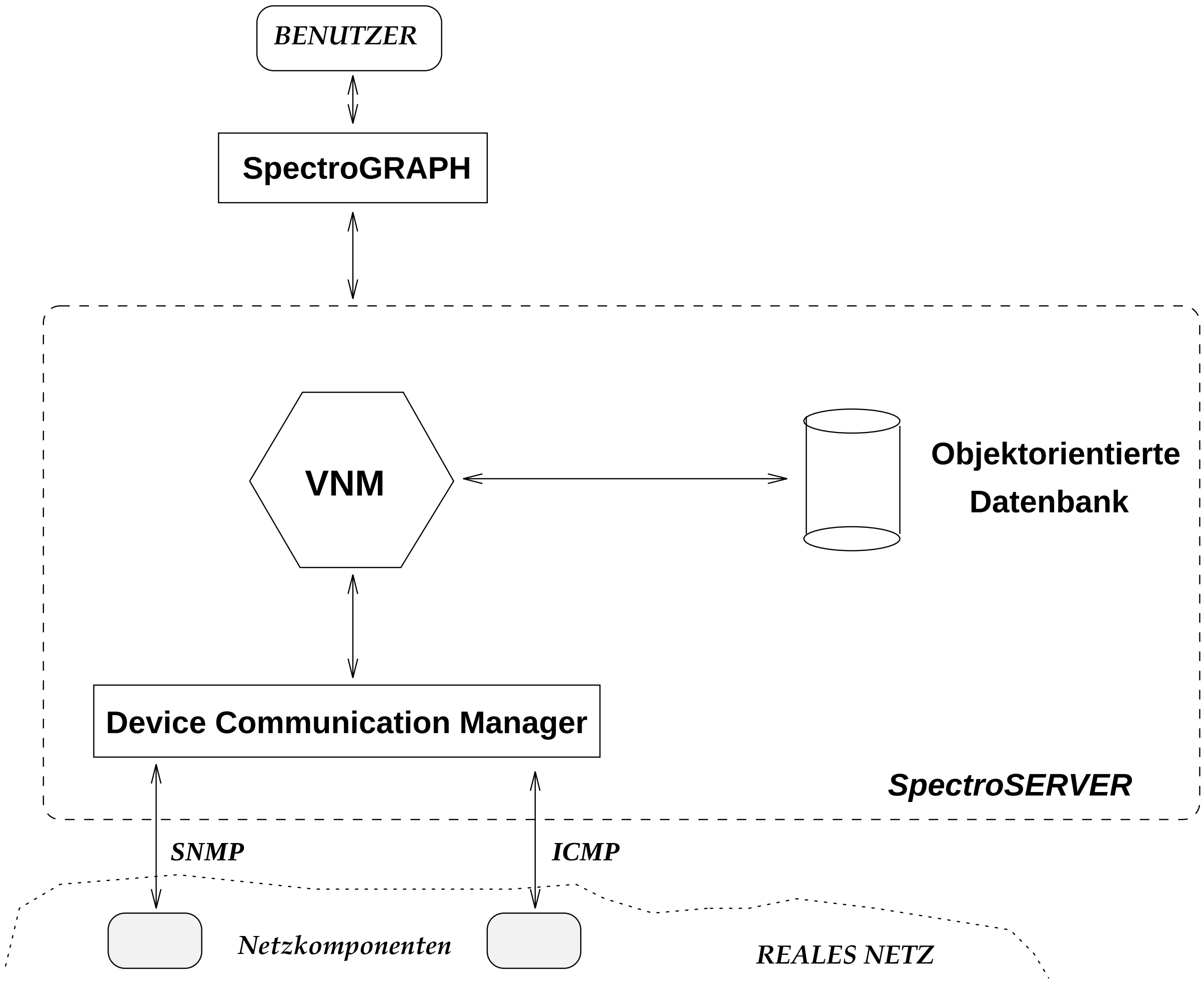

Abbildung 3.7: Das Netzmanagementsystem SPECTRUM

– **Device Communication Manager (DCM)**:
Da SPECTRUM verschiedene Managementprotokolle unterstützt, entspricht der DCM der VNM–Schnittstelle zu den verwendeten Managementprotokollen, die ihrerseits die Schnittstelle der Managementplattform zu den Agenten realer Netzkomponenten sind.
Aufgabe des DCM ist es beispielsweise, von den Agenten kommende SNMP–*Traps* in Meldungen an die VNM umzusetzen.

– **SpectroGRAPH**:
Diese OSF/Motif-Anwendung bildet die graphische Benutzerschnittstelle zu SpectroSERVER. Sie ermöglicht verschiedene Sichtweisen (geographische, topologische,

organisatorische) auf ein Rechnernetz und dessen Elemente.
Informationen werden durch *Icons* visualisiert; der aktuelle Status einer Netzkomponente wird durch die Farbe des zugehörigen Icons angezeigt.

Die Gesamtheit von VNM, DCM und Datenbank wird als **SpectroSERVER** bezeichnet (Abbildung 3.7).
SpectroGRAPH steht mit SpectroSERVER in einem *Client/Server*-Verhältnis; es ist daher möglich, mehrere SpectroGRAPHen an einem SpectroSERVER zu betreiben.
Ein Benutzer kann mit SpectroGRAPH unter anderem

– neue Modelle erzeugen

– Assoziationen zwischen Modellen herstellen (Das Verbinden von zwei Komponentenicons generiert in der VNM eine CONNECTS_TO Beziehung zwischen den Modellen)

– die Benutzer von SPECTRUM verwalten

also insgesamt Modifikationen an *Instanzen* vornehmen.
Um neues deklaratives Wissen (Modelltypen, Attribute, Relationen oder Regeln) in SPECTRUM einzubringen, sind Eingriffe in den SpectroSERVER nötig, die mit dem *Model Type Editor* (MTE) [SPEC MTE], einem OSF/Motif–konformen graphischen Werkzeug zum Erzeugen und Löschen von Modelltypen, Relationen und Regeln, vorgenommen werden.
Das Hinzufügen zusätzlichen prozeduralen Wissens (Inference Handler) hingegen bedeutet eine Erweiterung der VNM und kann weder mit SpectroGRAPH noch mit dem Model Type Editor, sondern nur durch Neucompilierung der VNM mit den Erweiterungen in Form von C++–Quelltext erfolgen. Das Verfahren hierzu ist in Abschnitt 6.3 beschrieben.

3.3.3 Anwendungsbeispiel

Das hohe Abstraktionsniveau und die Vielzahl der Definitionen und Begriffe verdecken insbesondere für denjenigen, der sich zum ersten Mal mit IMT befaßt, die in ihrer Eleganz und Flexibilität bestechenden Möglichkeiten, die dieses Modellierungs– und Implementierungskonzept bietet [DEV 92].
Im folgenden soll anhand eines einfachen Beispiels aus dem täglichen Leben demonstriert werden, daß auch komplexe Abläufe, geeignete Strukturierung vorausgesetzt, mit IMT beherrschbar sind.
Man nehme an, daß ein Basis-Modelltyp `Fahrzeug` existiere, der unter anderem folgende Attribute habe:

– Lenkrad

– Motor

- Anzahl der Räder
- Scheinwerfer
- Farbe

Will man die Fahrfähigkeit eines Fahrzeugs untersuchen, so wird man einen Intelligence Circuit `Prüfe_technischen_Zustand` kreieren, der seinerseits aus mehreren Inference Handlern besteht:

- `Prüfe_ob_Lenkrad_vorhanden`
- `Prüfe_Motor_auf_Funktion`
- `Prüfe_Anzahl_der_Räder`
- `Prüfe_Scheinwerfer_auf_Funktion`

Das Attribut „Farbe" ist für den technischen Zustand eines Fahrzeugs irrelevant; es wird daher auch nicht durch Inference Handler überwacht.
Von `Fahrzeug` können nun andere Modelltypen abgeleitet werden wie `PKW` oder `LKW`, die sowohl die Attribute als auch die Inference Handler von `Fahrzeug` erben.
Der Modelltyp `PKW` kann außerdem gegenüber `Fahrzeug` um

- ein neues Attribut „Kofferraumdeckel" sowie um
- einen neuen Inference Handler `Prüfe_ob_Kofferraumdeckel_geschlossen`

erweitert werden.
Eine Instanz des von `PKW` abgeleiteten Modelltyps `BMW 325i`, also ein Modell, wäre dann zum Beispiel `M-XY 1234` (Das Fahrzeug mit dem amtlichen Kennzeichen M-XY 1234).
Um Besitzverhältnisse auszudrücken, wird eine Relation *ist Eigentum von* eingeführt.
Eine Regel hieße dann etwa: `PKW` *ist Eigentum von* `Mensch` und die entsprechende Assoziation lautet folgendermaßen: `M-XY 1234` *ist Eigentum von* `Herrn Meier` [2].

Will Herr Meier nun wissen, ob er mit seinem BMW 325i eine Fahrt unternehmen kann, zeigt ihm der Bordcomputer an, ob

(1) der BMW 325i mit der Nummer M–XY 1234 in einwandfreiem technischen Zustand ist (dies geschieht mit dem von `Fahrzeug` ererbten Intelligence Circuit).

(2) der Kofferraumdeckel geschlossen ist (mit Hilfe des PKW–spezifischen Inference Handlers `Prüfe_ob_Kofferraumdeckel_geschlossen`)

[2]Hierbei wird unterstellt, daß `Herr Meier` ein Modell des Modelltyps `Mensch` ist, der wiederum vom Modelltyp `Säugetier` abgeleitet wurde...

Falls Herr Meier seinen BMW 325i verkauft, meldet ein Inference Handler `Prüfe_Eigentum`, der die Relation *ist Eigentum von* überwacht, den Besitzerwechsel der zuständigen Kraftfahrzeug-Zulassungsstelle.

Dieses kurze Beispiel zeigt, daß IMT eine Technik ist, die dem Entwickler ein mächtiges und durchgängiges Werkzeug in die Hand gibt, um Beziehungen bzw. Sequenzen von Aktionen verschiedenster Art zu modellieren bzw. zu steuern.
In der Domäne komplexer Kommunikationssysteme sind solche Eigenschaften „conditiones sine qua non“ für ein erfolgreiches Netzmanagement.

3.4 Bewertung – State of the Art

Im vorliegenden Kapitel wurden drei Ansätze diskutiert, die für das Netzmanagement zum aktuellen Zeitpunkt große Bedeutung haben.
Es ist wichtig, sich vor Augen zu führen, daß diese Ansätze nicht etwa in Konkurrenz zueinander stehen, sondern sich gegenseitig ergänzen können, da jeder einen anderen Grad der Abstraktion vorsieht.

Open Distributed Processing ist insbesondere für die Analyse und die Konzeption eines Managementsystems wichtig: Eine genaue Erfassung der Gegebenheiten und eine präzise Definition der Ziele wird explizit durch die Viewpoints *Technology* und *Enterprise* ermöglicht. Die weiteren Viewpoints zielen darauf ab, dem Entwickler Handlungsempfehlungen für den Software-Entwurf zu geben.

Das Hauptziel der Generic Managed Object Classes liegt darin, eine problemadäquate Strukturierung für Managementobjekte zu erreichen, indem allgemeine Objektklassen spezifiziert werden. Überlegungen, in welcher Beziehung Managed Objects zueinander stehen und wie sie am günstigsten angeordnet sein müssen, um deren Eigenschaften auf effiziente Weise weitervereben zu können, sind an dieser Stelle des Entwicklungsprozesses wesentlich.

Auf diese Vorüberlegungen ist die Inductive Modeling Technology angewiesen, da sie für die Bildung konkreter Managementobjekte und deren Implementierung maßgeblich ist.

Von der Managementaufgabe bis zur Erstellung eines Managementsystems werden die in Abbildung 3.8 dargestellten Teilschritte durchlaufen; ODP, die GMOC's und IMT liefern die dazu benötigten Verfahren.

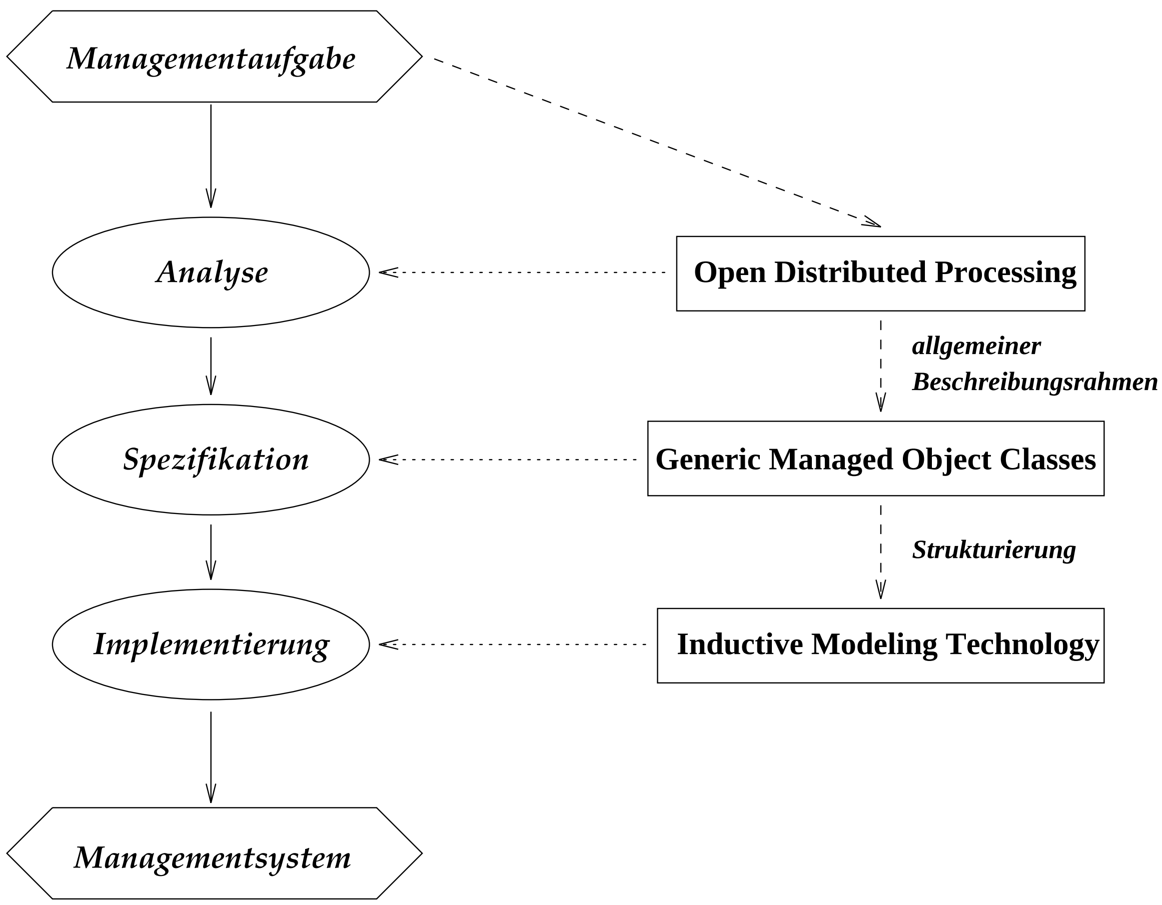

Abbildung 3.8: Aufstellung einer Managementlösung mit ODP, GMOC und IMT

Kapitel 4

Konzeption des NWP–Managements

In diesem Kapitel erfolgt eine Analyse des Managements von NWP gemäß ausgewählter ODP Viewpoints. Abschnitt 4.1 beleuchtet den Technology Viewpoint d.h. die Ausgangssituation, während Abschnitt 4.2 den Enterprise Viewpoint untersucht, also die Anforderungen, die von Seiten des Netzbetreibers an ein NWP–Management gestellt werden. Abschnitt 4.3 schließlich behandelt die Thematik der Akquisition von NWP–Managementwissen; es wird aufgezeigt, in welcher Form managementrelevante Information bezüglich NWP vorliegt und wie diese einem Managementsystem zugänglich gemacht werden kann.

4.1 Betrachtung von NWP unter Funktionalitätsgesichtspunkten

Für das Aufstellen eines tragfähigen Managementkonzeptes ist es zwingend notwendig, sich einen detaillierten Einblick in die technischen und funktionalen Zusammenhänge des zu managenden Systems –hier: der verteilten Kommunikationsanwendung NWP– zu verschaffen.
Wie bereits in Abschnitt 2.2 ausgeführt wurde, ist in der Regel eine Vielzahl von NWP–Knoten mit unterschiedlichen Kommunikationsprotokollen und -infrastrukturen an einer Dateiübertragung beteiligt.

Im folgenden wird ein umfassender Einblick in die technischen Abläufe einer Dateiübertragung mit dem NetzWerk–Programm gegeben und das *NWP Control Program* als zentraler Bestandteil eines NWP–Knotens beschrieben.

4.1.1 Technische Realisierung von Filetransfers

Soll eine Datei übertragen werden, erhält das Netzwerk–Programm über die Benutzerschnittstelle, das NWP–Menü, folgende Angaben von Seiten des Benutzers:

– den Namen des lokalen NWP–Knotens

– die Benutzerkennung am lokalen Knoten

– Angaben zur Datei (Name, Typ) auf dem lokalen NWP–Knoten und den gewünschten Übertragungsmodus (synchron, asynchron)

– den Namen des NWP–Zielknotens

– die Benutzerkennung am Zielknoten

– das Benutzerpasswort für den Zielknoten

– Angaben zur Datei bezüglich des Zielknotens (Platte, Verzeichnis, Name, Typ)

Nachdem der Benutzer dem NetzWerk-Programm den Auftrag zum Senden einer Datei erteilt hat, erzeugt NWP einen netzweit eindeutigen Transaktionscode, der sich aus dem Namen des Ursprungsknotens sowie einer vierstelligen Dezimalzahl, die noch nicht für bisher an diesem Knoten initiierte Transaktionen verwandt worden ist, zusammensetzt [1]. Zu beachten ist, daß der Knoten*name*, also eine logische Adresse der OSI–Schicht 7 und nicht etwa eine IP–Adresse verwandt wird, da Schicht–3–Adressen aufgrund der Heterogenität des Verbunds nicht eingesetzt werden können: Einige NWP–Knoten verfügen über keine IP–Adresse.
Eine wichtige Konsequenz daraus ist, daß NWP logisches Routing auf der OSI–Schicht 7 praktiziert, um die Dateien durch das Netz zu transferieren.
Die Wege, über die die zu übertragenden Daten laufen, sind von vornherein festgelegt d.h. das NetzWerk-Programm verwendet statisches Routing. Jeder NWP–Knoten besitzt eine Wegedatei im ASCII–Format mit N Einträgen (wobei N die Gesamtzahl der Knoten des NWP–Verbunds ist), die unter anderem die in Tabelle 4.1 gezeigten Angaben umfaßt.

Die Wegenummer gibt an, welche Knotensequenz eine Datei nimmt; es können bis zu 9 verschiedene Routen von einem Ursprungs– zu einem Zielknoten konfiguriert werden.
Die Angabe „Format“ enthält Information, ob der Dateiname auf dem Zielknoten einem fixen (F) Format unterliegt (wie es der Fall bei VM–Rechnern ist), oder ob dieses variabel (V) gewählt werden kann (zum Beispiel bei UNIX–Systemen).

[1] Die vierstellige Dezimalzahl ist für die Eindeutigkeit des Transaktionscodes verantwortlich. Mit jeder neu angestoßenen Transaktion wird ein auf dem lokalen NWP–Knoten befindlicher Zähler modulo 10000 inkrementiert.

Tabelle 4.1: Die Wegedatei eines NWP–Knotens

WEGE-NUMMER	URSPRUNGS-KNOTEN	ZIEL-KNOTEN	NÄCHSTER KNOTEN	FORMAT	WEGE-INFO
1	zrzvm3	catek	malibu	V	H
3	titan	bmwsys	rrz1	F	I
...	...	...	...	...	...

Die Wegeinformation beinhaltet Angaben, in welcher der folgenden Netzarten der Dateitransfer erfolgt:

- **H**omogenes Netz d.h. es muß keine Konvertierung des Dateiinhaltes erfolgen
- **I**nhomogenes Netz d.h. es müssen Anpassungen an unterschiedliche Zeichensätze (ASCII, EBCDIC...) und Satzlängen vorgenommen werden
- **S**treng homogenes Netz d.h. alle Knoten auf dem Weg haben dasselbe Betriebssystem
- **L**okales Netz d.h. Ursprungs– und Zielknoten sind identisch und die Menge der Zwischenknoten ist leer; dies mag auf den ersten Blick merkwürdig erscheinen, jedoch kann dies durchaus bei CAD–Systemen vorkommen, die auf die NWP–Programmierschnittstelle [NWP PRO] aufgesetzt sind.

Jedem NWP–Knoten sind also alle anderen Knoten des NWP–Verbunds bekannt, allerdings nicht die vollständige Route zu ihnen, da in der Wegedatei lediglich der benachbarte und der Zielknoten verzeichnet sind.

Nachdem der Transaktionscode erzeugt wurde, wird ihm ein Präfix, das aus zwei Buchstaben besteht, vorangestellt. Die Konkatenation aus Präfix und Transaktionscode ergibt dann einen netzweit eindeutigen Dateinamen, der auch Aufschluß über die Art der in ihm enthaltenen Daten gibt (Abbildung 4.1).

Insgesamt existieren fünf verschiedene Präfixe für Dateinamen, die für netzweite Dateiübertragungen relevant sind:

- **CS**:
  Dieses Präfix haben Sende–Kontrolldateien, die ausschließlich beim Start einer Transaktion angelegt werden.
  Sie kommen nur auf dem NWP–Knoten vor, auf dem eine Transaktion initiiert wurde.

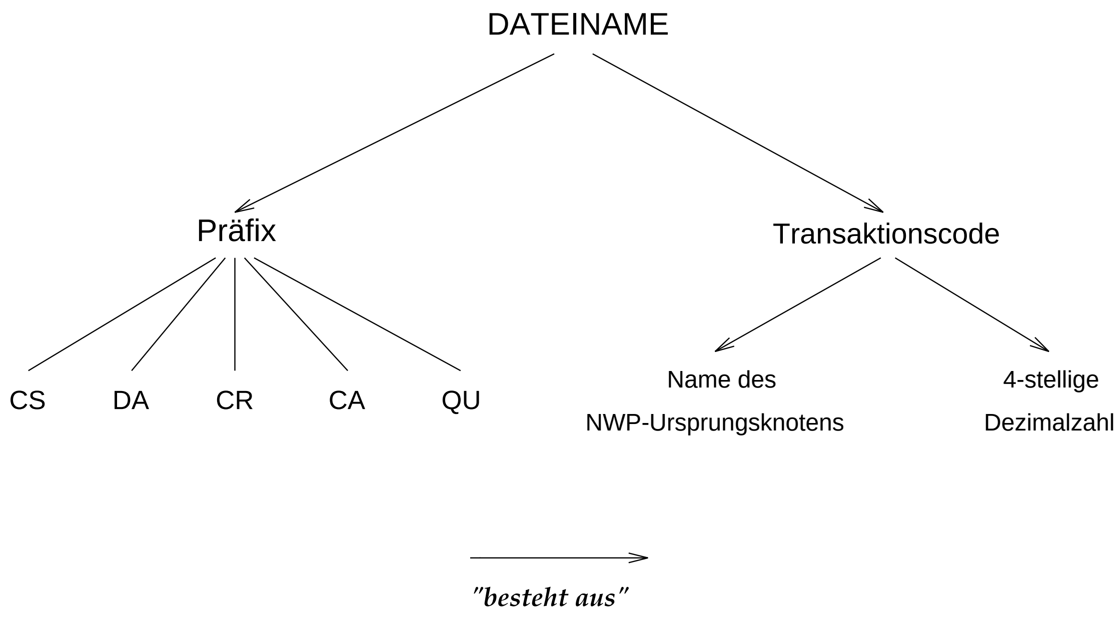

Abbildung 4.1: Generierung eines netzweit eindeutigen Dateinamens

- **DA**:
  Hiermit sind Benutzerdateien gekennzeichnet, die Nutzdaten enthalten.
  Sie werden für eine Transaktion auf all den NWP–Knoten angelegt, die Teil der Route vom Ursprungs– zum Zielknoten sind.

- **CR**:
  CR–Dateien sind sogenannte *Remote Node Kontrolldateien* und bilden das Gegenstück zu den CS–Dateien: Sie beinhalten Kontrollinformation bezüglich der Transaktionen, die *nicht* am aktuellen Knoten initiiert wurden und kommen daher auf allen Knoten einer Route vor mit Ausnahme des Ursprungsknotens. Jeder Zwischenknoten ergänzt diese Dateien um seine Log–Einträge.

- **CA**:
  Dateien mit dem Präfix CA sind Administrations– und Kontrolldateien und enthalten Aufträge der Systemverwaltung (zum Beispiel Update der Wegedatei, Herunterfahren eines NWP–Knotens).

- **QU**:
  Das Präfix QU ist den Quittungsdateien vorbehalten. Sie werden vom Zielknoten nach Eintreffen der CR– und DA–Dateien und der Ausführung der geforderten Aktionen durch den Zielknoten erzeugt und zum Ursprungsknoten über dieselbe Route, allerdings in entgegengesetzter Richtung, transportiert, die die Benutzerdaten

genommen haben. QU–Dateien werden wie DA–Dateien behandelt d.h. die Zwischenknoten nehmen an ihnen keine Änderungen vor.

Die Benutzerschnittstelle, das NWP–Menü, übergibt anschließend die zur Übertragung anstehende Datei dem Empfangsserver [2]des Knotens, der sie umbenennt (d.h. ihr neuer Dateiname lautet dann DA*Knotenname*xxxx) und eine CS–Datei (CS*Knotenname*xxxx) mit Kontrollinformation erzeugt.
Beide Dateien werden dann auf dem Ursprungsknoten im Eingabebereich des NWP-*Control Programs* abgelegt. Das *Control Program* ist der Hauptbestandteil von NWP und wird im folgenden Teilabschnitt detailliert behandelt. Für die Fortführung der Diskussion der Dateiübertragungsmechanismen im NWP–Verbund wird das Control Program vorerst als „black box“ angesehen.
Nach Verarbeitung der CS– und DA–Dateien durch das Control Program werden diese anschließend einem von mehreren sogenannten *Sendeservern* übergeben. Zum Begriff „Server“ ist im Zusammenhang mit NWP folgendes anzumerken:
Mit „Server“ ist nicht etwa das Gegenstück zu „Client“ in einer *Client/Server*–Beziehung gemeint; „Server“ ist hier ein Synonym für „Ablauf eines Programms“, was in unterschiedlichen Systemwelten jeweils mit verschiedenen Begriffen umschrieben wird: In der VM–Welt als „Task“ und auf UNIX–Systemen als „Prozeß“. Um in der heterogenen NWP–Systemwelt konsistente Begriffsdefinitionen zu schaffen, wurde für die genannten Bezeichnungen der Oberbegriff „Server“ eingeführt.

Ein Sendeserver hat die Aufgabe, die Dateien mit Hilfe eines systemspezifischen Kommunikationsdienstes (ftp, DECNET copy...) zum benachbarten NWP–Knoten zu übertragen. Bekanntlich kann man im NWP–Verbund nicht davon ausgehen, daß die Nachbarknoten eines NWP–Knotens alle denselben systemnahen Kommunikationsdienst verwenden. An einem NWP–Knoten müssen daher Sendeserver für alle an den Nachbarknoten verfügbaren Dateitransfer–Protokolle vorgehalten werden.

Nachdem der für die Übertragung ausgewählte Sendeserver die Nutzdaten– und Kontrolldateien dem Nachbarknoten übergeben –genauer: im Eingabebereich des Control Program des Nachbarknotens abgelegt– hat, wartet der NWP–Knoten auf die Quittung, die von diesem Nachbarknoten kommen muß.
Der Nachbarknoten (und alle weiteren NWP–Knoten bis hin zum Zielknoten) vollzieht dieselbe Prozedur (Control Program ausführen, Log–Information in Kontrolldatei schreiben, Sendeserver auswählen und diesem die Dateien übergeben, auf Quittung warten) wie der Ursprungsknoten, allerdings mit einer Ausnahme: Der Kontrolldateiname hat nun das Präfix CR, weil der aktuelle Knoten nicht der Ursprungsknoten ist.

Sind die Dateien am Zielknoten angekommen, werden die Nutzdaten (unter Beachtung der Angaben des Benutzers am Ursprungsknoten) abgespeichert und die Quittungsdatei für die gesamte Übertragung erzeugt.

[2]Der Empfangsserver eines NWP–Knotens ist dafür verantwortlich, Daten von der Benutzerschnittstelle oder einem anderen Knoten entgegenzunehmen und der Knoten–lokalen Software zur Verfügung zu stellen.

Nun ist der erste Teil der Übertragung abgeschlossen und der gesamte Vorgang kehrt sich um: Der Zielknoten ist nunmehr Startknoten; die Quittungsdatei durchläuft denselben Weg in umgekehrter Richtung, den die DA–Datei genommen hat; der ehemalige Ursprungsknoten ist jetzt Zielknoten.
Bezüglich der Dateinamen ist bei der Rückübertragung der Quittung folgendes von Bedeutung:

(1) Der Transaktionscode des Dateinamens bleibt *unverändert*, weil das Senden der Quittung zur selben Transaktion gehört wie der Sendevorgang für die Nutzdaten.

(2) Die Kontrolldatei für die Rückübertragung hat nicht das Präfix CS (obwohl der Knoten jetzt strenggenommen Ursprungsknoten ist), sondern behält das Präfix CR bei.

Die Quittungsdatei wird demnach auf der gesamten Route von einer CR–Kontrolldatei begleitet, die ihrerseits von jedem Knoten auf der Route um Log–Informationen ergänzt wird.

Sind die Quittungs– und Kontrolldateien am Ursprungsknoten eingetroffen, ist die Transaktion beendet. Abbildung 4.2 skizziert die Reihenfolge der Schritte einer Dateiübertragung mit NWP.

Bei Dateiübertragungen praktiziert das NetzWerk–Programm Teilstreckenvermittlung nach dem *store-and-forward* Prinzip.

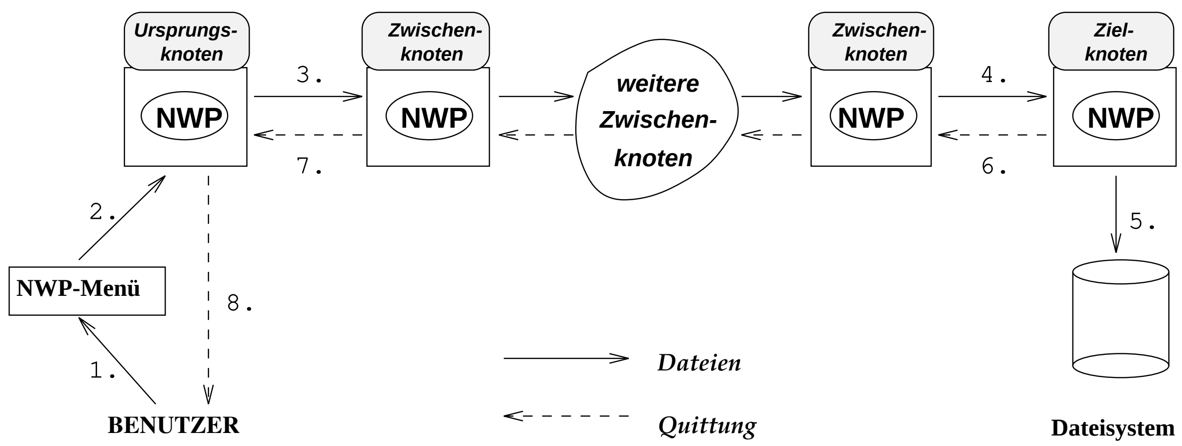

Abbildung 4.2: Dateiübertragung mit dem NetzWerk-Programm

4.1.2 Das NWP Control Program

Aus dem vorangegangenen Teilabschnitt ist bekannt, wie Dateiübertragungen innerhalb des NWP–Verbunds ablaufen. Im folgenden wird die Funktionsweise der zentralen Steuersoftware eines NWP–Knotens, das *NWP Control Program*, erläutert.

Ausgangssituation für das Control Program ist das Eintreffen von Dateien im sogenannten *Eingabebereich*. Der Eingabebereich ist eine Warteschlange, in die alle ankommenden Dateien eingereiht werden.
Auf jedem NWP–Knoten existiert ein Dateiverzeichnis, das das Zwischenspeichern aller eintreffenden Dateien übernimmt. Der Eingabebereich eines NWP–Knotens besteht beispielsweise bei einem UNIX–System aus folgendem Verzeichnis:

`/usr/spool/nw/tmp`

Daten können an einem NWP–Knoten auf zwei Arten eintreffen:

(1) Über das NWP–Menü initiiert ein Benutzer den Transfer einer Datei. In diesem Fall ist der NWP–Knoten Ursprungsknoten und der Transaktionscode beinhaltet seinen Namen. Die Dateinamen im Eingabebereich des Control Program haben die Präfixe DA und CS.

(2) Der betreffende NWP–Knoten ist Zwischenknoten auf der Route vom Ursprungs– zum Zielknoten. Die Daten werden ihm vom vorangehenden NWP–Knoten übergeben; der aktuelle Knoten ist für das Weitersenden der Daten zuständig.
Im Falle einer Benutzerdatenübertragung haben die Dateien im Eingabebereich die Präfixe DA und CR; bei der Rückübertragung der Quittungsdatei liegen Dateien mit den Präfixen QU und CR in dem Verzeichnis des Eingabebereiches.

Das *Control Program* ist zuständig für

(1) das Lesen der Dateien, die vom Empfangsserver (auf UNIX–Systemen: *TCPSvr*) im Eingabebereich abgelegt wurden. Wird eine Transaktion vom Control Program abgearbeitet, so werden die zu dieser Transaktion gehörenden Dateien umbenannt und in die Arbeitsverzeichnisse gebracht.
Die Dateinamen haben dann die Präfixe CW (Kontrolldatei) bzw. DW (Datei mit Nutzdaten) und stehen in den Dateiverzeichnissen `/usr/spool/nw/c` bzw. `/usr/spool/nw/d`. Das „W“ im Dateinamen zeigt an, daß die Dateien bearbeitet werden („in **W**ork“).
Die Schnittstelle zwischen den einzelnen Bereichen des NWP Control Program ist also auf Basis von Dateien realisiert.

(2) die Konfiguration des Übertragungsweges d.h. die Bestimmung des Knotens, der als nächstes die Daten erhält.
Für den Fall, daß der NWP–Knoten Zielknoten ist, wird versucht, die Datei in das

(am Ursprungsknoten vom dortigen Benutzer festgelegte) Verzeichnis unter dem vorgesehenen Namen abzulegen.

(3) das Schreiben von Log–Information in die zur Transaktion gehörende CR–Datei.

(4) die Übergabe der Dateien an den Sendeserver. Gelingt der Transfer der Dateien zum nächsten NWP–Knoten d.h. können die CW– bzw. DW–Dateien dem Empfangsserver des nächsten Knotens übergeben und in CR– und DA–Dateien umbenannt werden, so werden die DA– und CR–Dateien auf dem aktuellen Knoten gelöscht. Eine Kopie der DW–Datei wird solange im Arbeitsbereich aufbewahrt, bis die zu dieser Transaktion gehörende Quittung zurückübertragen wurde.
Sollte die Übertragung scheitern, nachdem der NWP–Knoten erfolgreich passiert wurde, ist die CW–Datei das einzige Mittel, das den Nachweis erbringen kann, daß der Fehler zwischen dem lokalen NWP–Knoten und dem Zielknoten aufgetreten sein muß. CW–Dateien sind ein wichtiges Hilfsmittel bei der Lokalisierung von Übertragungsfehlern und werden daher erst vier Tage nach Beendigung der Transaktion gelöscht.
Die Anzahl der CW–Dateien an einem NWP–Knoten ist demnach immer größer oder gleich der Anzahl der DW–Dateien, da Dateien mit dem Präfix CW den Status der Transaktion enthalten.

(5) das Warten auf die Quittung, die vom Zielknoten kommt und zum Ursprungsknoten übertragen werden muß.
Falls der NWP–Knoten Zielknoten ist, generiert er die Quittung, bestimmt den Nachfolgeknoten und übergibt die Dateien an den zuständigen Sendeserver.

Zusammenfassend kann festgestellt werden, daß das Control Program verantwortlich ist für die Überwachung und Steuerung aller Aktivitäten, die NWP betreffen.

Ein (stark vereinfachter) Algorithmus für das NWP Control Program ist nachstehend skizziert:

```
SOLANGE (Eingabebereich nicht leer)

   NIMM CW-Datei;
   LIES Zielknotenname AUS CW-Datei;

   FALLS Aktueller_Knoten = Zielknoten
      SCHREIBE DW-Datei AUF Platte IN Verzeichnis UNTER Dateiname ALS Typ

      FALLS fehlerfrei                     \\ kein Fehler beim Schreiben
         Quittung := generiere_positive_quittung
      SONST                                \\ Fehler beim Schreibvorgang
         Quittung := generiere_negative_quittung;
```

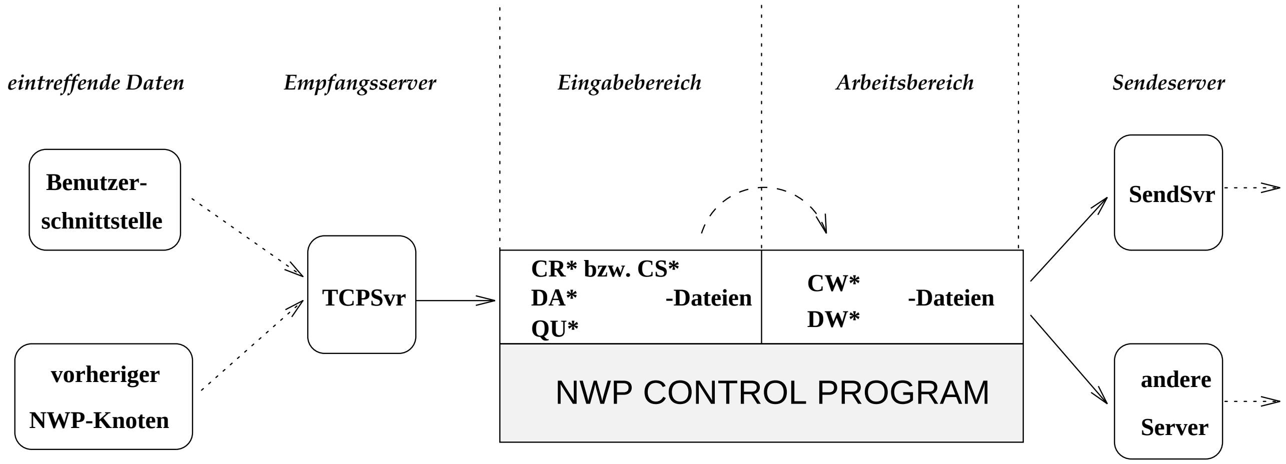

Abbildung 4.3: Abarbeitung von Aufträgen in einem NWP–Knoten

```
      EXTRAHIERE Zielknoten AUS Dateiname; \\ fuer Senden der Quittung
      ERMITTLE Naechster_Knoten AUS Wegedatei;
      UEBERGEBE Quittung AN Sendeserver (Naechster_Knoten)

   SONST                                   \\ Datei weiterleiten
      ERMITTLE Naechster_Knoten AUS Wegedatei;
      UEBERGEBE Datei AN Sendeserver (Naechster_Knoten);

ENDE
```

Abbildung 4.3 zeigt schematisch die Vorgehensweise des NWP Control Program.

Als Sendeserver kommt bei der Übertragung zu UNIX–Systemen der auf FTP basierende *SendSvr* in Frage. Bei Dateiübertragungen zu anderen Systemwelten werden an Stelle von SendSvr die entsprechenden systemspezifischen Sendeserver ausgewählt.

4.2 Betreiberspezifische Anforderungsprofile an ein NWP–Management

In Abschnitt 2.2 (und speziell in Teilabschnitt 2.2.4) wurden bereits einige allgemeine Anforderungen an das Management verteilter Kommunikationsanwendungen skizziert. Dieser Abschnitt konkretisiert die beschriebenen Anforderungen [3], die die Grundlage der in dieser Arbeit entwickelten Managementszenarien bilden, aus zwei unterschiedlichen Blickwinkeln:

(1) Aus der Sicht des *Netzwerkleitstands*, dessen Aufgabe neben der passiven Überwachung von Netzkomponenten des gesamten BMW–Kommunikationssystems darin besteht, den zahlreichen Benutzern von bei der BMW AG installierten Software als Anlaufstelle zur Verfügung zu stehen, falls ein Problem mit einem beliebigen Programm auftritt.
Der Netzwerkleitstand dokumentiert dieses Problem, führt erste, einfache Diagnosemaßnahmen durch und gibt diese Informationen an eine Abteilung weiter, die für die Software, bei deren Benutzung der Fehler aufgetreten ist, technische Unterstützung leistet.

(2) Die zweite Sichtweise ist die der Software–spezifischen Abteilungen, deren Mitarbeiter über detaillierte Kenntnisse des betroffenen Programms verfügen und in der Regel bereits an der Entwicklung der Software beteiligt waren.
Im Falle von NWP werden Informationen über Probleme, die beim Betrieb von NWP auftreten, vom Netzwerkleitstand (und zum Teil auch direkt von betroffenen Benutzern) an die Abteilung *NWP–Support* weitergeleitet.
Dort wird mit eingehenden Diagnosemaßnahmen der Fehler lokalisiert und anschließend behoben.

Es liegt auf der Hand, daß beide Bereiche unterschiedliche Anforderungen haben. Der Netzwerkleitstand betreut neben NWP noch eine Vielzahl anderer Programme und kann aus zeitlichen Gründen lediglich einfache Tests durchführen. Die tatsächliche Fehlerbehandlung liegt beim NWP–Support.

4.2.1 Anforderungen von Seiten des Netzwerkleitstands

Im Brennpunkt der Anforderungen des Netzwerkleitstands an ein NWP–Management stehen die Überwachung und die betriebliche Unterstützung des NetzWerk–Programms.

Hierzu ist es notwendig, alle NWP–Knoten mit ihren Schicht–7–Namen auf der Netzmanagementplattform graphisch darzustellen.

[3] Die Anforderungen an das NWP–Management unter Berücksichtigung *aller* betriebs– und übertragungssystemspezifischen Belange sind zu umfassend, um im Rahmen dieser Arbeit eingehend gewürdigt zu werden. Die folgenden Untersuchungen beziehen sich daher speziell auf UNIX–Systeme, die zahlenmäßig am stärksten innerhalb des NWP–Verbunds vertreten sind.

Aufgrund der beträchtlichen Anzahl von NWP–Knoten (gegenwärtig 150) ist es sehr zeitaufwendig, dies von Hand zu tun; ein automatisches Erkennen der Knoten durch das Managementsystem (Autotopologie–Funktion) wird gefordert, um ein hinreichendes Maß an Aktualität zu gewährleisten.

Bei der graphischen Anzeige des NWP–Verbunds sollte jeder NWP–Knoten durch ein *Icon* repräsentiert sein, das durch adäquate Einfärbung den Zustand des Knotens anzeigt.
Beispiele für Ereignisse, die Zustandsänderungen induzieren und von auf den Knoten befindlichen Agenten an die Plattform gemeldet werden sollten, sind:

– Link–Down des NWP–Knotens

– Link–Up des NWP–Knotens

– Fehler am NWP–Knoten

– Restart des auf dem NWP–Knoten befindlichen Agenten

Für den Netzwerkleitstand ist es insbesondere im Fehlerfall [NWP DOK] von großer Bedeutung, die Route verfolgen zu können, die eine Datei und ihre Quittung genommen haben. Es sollte möglich sein, daß das Managementsystem, nachdem Start– und Zielknoten eingegeben wurden, die gesamte Knotenfolge ausgibt. Zur Zeit muß von den Operateuren mangels geeigneter Unterstützung durch eine Netzmanagementplattform ein sehr zeitaufwendiges Verfahren durchlaufen werden, falls der Transfer einer Datei scheitert:

(1) Login am Ursprungsknoten

(2) Lesen der Logfiles des Knotens um zu überprüfen, ob die Dateien

    (a) korrekt empfangen und

    (b) erfolgreich zum nächsten NWP–Knoten gesendet wurden

(3) Überprüfen der Prozeßtabelle, um zu sehen, ob NWP (bzw. das Control Program, die Empfangs– und Sendeserver) läuft

(4) Anzeigen der Warteschlangen

(5) Logout

Diese Prozedur muß für alle NWP–Zwischenknoten, die auf der für die Dateiübertragung vorgesehenen Route liegen, solange wiederholt werden, bis der Knoten gefunden wird, an dem Schwierigkeiten aufgetreten sind.
Anschließend wird von einem Rechner des Leitstands versucht, eine TELNET–Session zu dem betroffenen Knoten aufzubauen oder einen FTP–Login durchzuführen, um die Funktionsfähigkeit der systemnahen Kommunikationsdienste, der Kommunikationsprotokolle sowie der Infrastruktur des NWP–Knotens zu überprüfen. Über das Ergebnis dieser

Bemühungen wird der NWP–Support informiert, der weitere Schritte einleitet.
Die Notwendigkeit einer Automatisierung dieses Suchvorgangs ist offensichtlich.

Ein weiteres Problem folgt aus dem statischen Routing: Fällt ein NWP–Knoten aus, so müssen die Wegedateien an allen Knoten, die den ausgefallenen Knoten enthalten, von Hand geändert werden; ein Vorgang, der ebenfalls automatisiert werden sollte.

Großer Wert wird auf eine klare und übersichtliche Präsentation wichtiger NWP–spezifischer Daten und Parameter gelegt. Diese sollten in themenbezogene Sichten aufgegliedert sein (Organisation, Topologie, Konfiguration, Standort), um problemadäquate Informationen schnell zu visualisieren.

4.2.2 Anforderungen aus der Sicht des NWP–Supports

Im Gegensatz zum Netzwerkleitstand benötigt der NWP–Support Mittel, die neben weitreichenden Monitoring–Funktionen auch Eingriffe in den Betrieb von NWP zulassen, um eine erfolgreiche Fehlersuche und –behebung zu gewährleisten.
Hierbei ist es besonders wichtig, systemspezifische Eigenschaften eingehend zu beachten, da sich bei den zahlreich vorhandenen UNIX–Workstations Fehlersituationen ergeben, die beispielsweise bei Großrechnern kaum vorkommen dürften:

- Treffen mehrere Dateien mit CAD–Daten gleichzeitig an einer Workstation ein, die als NWP–Knoten fungiert, ist es durchaus möglich, daß der verfügbare Platz in dem für NWP relevanten Dateisystem `/usr/spool/nw` nicht ausreicht.
- In der Regel sind UNIX–Workstations innerhalb des NWP–Verbunds nicht dedizierte NWP–Knoten d.h. eine Workstation wird beispielsweise von einem Ingenieur zum Erstellen von Konstruktionsplänen benutzt; die NWP–Prozesse laufen im Hintergrund. Es kommt vor, daß der Besitzer einer solchen Maschine mit Hilfe seiner *Root*–Berechtigung die Zugriffsrechte der für NWP wichtigen Verzeichnisse derart modifiziert, daß NWP keine Daten mehr darin ablegen kann.
- Im schlimmsten Fall fährt ein Besitzer einer Workstation diese nach Beendigung seiner Tätigkeit herunter, ohne zu beachten, daß die Maschine gleichzeitig ein NWP–Knoten ist.
  Die Folge ist ein Totalausfall des Knotens: Dateien können nicht mehr über diesen Knoten gesendet werden und Quittungsdateien für bereits übertragene Dateien werden aufgehalten.
- Die NWP–Prozesse sind auf UNIX–Maschinen in der Datei `crontab` eingetragen und werden in periodischen Abständen angehalten und neu gestartet. Ebenso wird die Prozedur `nwpcleanup`, die die Datenbereiche des NetzWerk–Programms bereinigt, aus der `crontab` aufgerufen.
  Bei der Konfiguration (oder durch versehentliches Löschen) kann es passieren, daß die NWP–spezifischen `crontab`–Einträge entfernt werden; die unmittelbare Folge ist, daß NWP an diesem Knoten nicht mehr benutzt werden kann.

– Obwohl seit geraumer Zeit das *Domain Name System* (DNS) zur Ermittlung der Rechneradressen im Einsatz ist, existieren noch Rechner, die die lokale Adreßtabelle `/etc/hosts` benötigen, in der unter Umständen nicht alle NWP–Knoten erfaßt sind. Bei Übertragungswegen, die über ein solches System führen, kann es zur Fehlermeldung „unknown host“ kommen.

Anhand der geschilderten Problemfälle lassen sich direkt die Anforderungen des NWP–Supports an das NWP–Management ableiten:

(1) Es ist notwendig, die Menge des freien Speicherplatzes in den für NWP relevanten Dateisystemen zu überwachen und Schwellwerte für den Füllgrad zu definieren, bei deren Überschreitung der Agent eine Meldung an die Managementstation schickt.

(2) Ein NWP–Knoten sollte derart in das Netzmodell der Managementstation abgebildet werden, daß bestimmte (UNIX–) Kommandos, ausgelöst an der Managementstation, auf dem NWP–Knoten ausgeführt werden. Hierzu zählen insbesondere:

   – Das Anhalten eines NWP–Teilprozesses
   – Das Neustarten eines NWP–Teilprozesses
   – Das Lesen von Log–Information
   – Das Löschen, Kopieren und Umbenennen von Dateien
   – Das Anzeigen und Editieren von Dateiinhalten (Wegedatei, crontab)
   – Das Anzeigen der Länge und der Inhalte einer Eingangswarteschlange
   – Das Anzeigen von Systemressourcen (Prozessorauslastung, Plattenplatz)

(3) Es sollte möglich sein, durch „Anklicken“ eines bestimmten Bereiches des NWP–Knotenicons, eine TELNET–Session von der Managementstation aus auf dem ausgewählten NWP–Knoten zu starten, um die unterhalb von NWP liegenden Schichten des Knotens und die Kommunikationsinfrastruktur des NWP–Verbunds schnell prüfen zu können.

(4) Tests bezüglich der Funktionsfähigkeit von NWP auf einzelnen Knoten bzw. frei definierbaren Teilsegmenten des NWP–Verbunds sollten von der Managementstation aus eingeleitet werden können.

Die obengenannten Beispiele zeigen, daß die Problemschwerpunkte stark von den eingesetzten Rechnersystemen abhängig sind. So sind die Managementanforderungen an einen Workstationcluster gänzlich andere als an einen Großrechnerverbund. Im Zuge der *Downsizing*– bzw. *Rightsizing*–Bestrebungen und der damit einhergehenden Verbreitung von Workstations erhalten die beschriebenen Probleme eine immer größere Brisanz, die nur durch adäquate Managementkonzepte abgeschwächt werden kann.

4.3 Akquisition von NWP–Managementinformation

Von großer Bedeutung für das NWP–Management ist die Fragestellung, welche Managementinformationen bereits vorliegen und inwieweit diese in ein Managementsystem integriert werden können. Reichen diese Informationen nicht aus, um NWP und seine Bestandteile vollständig zu beschreiben, müssen neue Objekte eingeführt werden.

Im folgenden werden Quellen dargestellt, die Managementinformation enthalten.

– **ISYKON**
  ISYKON ist ein auf dem relationalen Datenbanksystem ORACLE aufbauendes statisches Dokumentationssystem [ISY IK], das eine Beschreibung des NWP–Verbunds enthält und zentral gehalten wird.
  In ISYKON sind alle NWP–Knoten und deren Charakteristika erfaßt wie zum Beispiel

  - der Knotenname
  - das Betriebssystem des Knotens
  - die NWP–Versionsnummer
  - der Standort
  - die Area und das Subnetz, denen der Knoten angehört
  - eine Beschreibung der Knotenhardware
  - der Ansprechpartner für den Knoten
  - zusätzliche organisatorische Information
  - die am Knoten vorhandenen Empfangs– und Sendeserver
  - das Verzeichnis der Log–Dateien und der Pfad des NWP–Eingabebereiches
  - weitere knotenspezifische Konfigurationsinformation

  Diese Angaben können vom Benutzer über eine SQL–Kommandoschnittstelle abgefragt werden.
  Bezüglich der Aktualität der in ISYKON enthaltenen Daten ist zu beachten, daß die Informationen nicht dynamisch verändert werden. Beispielsweise wird die Erweiterung eines NWP–Knotens um einen neuen Sendeserver nicht automatisch in ISYKON vermerkt, sondern muß von Hand eingetragen werden. Daher sind in ISYKON keine Informationen enthalten, die einer hohen Dynamik unterliegen, wie zum Beispiel der Status eines NWP–Knotens.

– Netzwerk Informations Datei (**NIDAT**)
  Im Gegensatz zu ISYKON ist die NIDAT auf jedem NWP–Knoten verfügbar. Sie wird periodisch aus ISYKON erzeugt und besteht aus einer Teilmenge der ISYKON–Daten. Unter anderem enthält die NIDAT:

  - die Namen der NWP–Knoten
  - die Nummern der auf den jeweiligen Knoten vorhandenen NWP–Versionen
  - Angaben zu den Betriebssystemen der NWP–Knoten
  - Systembeschreibungen der Knoten
  - die logischen Netze, denen die Knoten zugeordnet sind.

  Die Netzwerk Informations Datei ist eine statische Netzbeschreibung in tabellarischer Form, die im ASCII–Format vorliegt. Automatisierte Schreib– und Lesezugriffe sind daher für ein Managementsystem leichter zu handhaben, als der Datenaustausch mit ISYKON.

– **Wegedatei**
  Ebenso wie die NIDAT ist die Wegedatei im ASCII–Format auf jedem NWP–Knoten vorhanden. Aufgrund der Tatsache, daß die Wegedatei bereits in Abschnitt 4.1 beschrieben wurde, wird an dieser Stelle auf eine erneute Darstellung verzichtet. Wegen des verhältnismäßig einfachen Datenzugriffs und der damit verbundenen Aktualisierungsmöglichkeiten durch das Managementsystem ist die Wegedatei für das NWP–Management von großem Interesse.

– **Log–Dateien**
  Jede Aktivität des NWP Control Programs wird in einer Knoten–lokalen Log–Datei festgehalten.
  Sie liegt im ASCII–Format vor und enthält unter anderem folgende Parameter:

  - Ursprungsknoten
  - Zielknoten
  - Zeitstempel für den Beginn der Transaktion
  - Zeitstempel für das Ende der Transaktion
  - Übertragungsmodus
  - Zustandsbyte, ob die Transaktion erfolgreich abgeschlossen wurde
  - Transaktionscode
  - Größe der Datei in Kilobyte

  Die Log–Dateien, die auf jedem NWP–Knoten mit dem Betriebssystem UNIX im Verzeichnis `/usr/spool/nw/log` stehen, reflektieren die Dynamik des NetzWerk–Programms und spielen für die in Kapitel 5 entwickelten Managementszenarien eine wichtige Rolle.

ISYKON, NIDAT sowie die Log– und Wegedateien sind die derzeit verfügbaren Quellen, aus denen Managementinformation abgeleitet werden kann. Allen, mit Ausnahme der Log–Dateien, ist gemeinsam, daß die in ihnen enthaltenen Informationen statischer Art sind.

Das NWP–Management ist jedoch primär auf dynamische Informationen angewiesen, um auf Änderungen der Systemparameter in akzeptabler Frist reagieren zu können bzw. selbständig Modifikationen an den Daten (wichtig im Zusammenhang mit dynamischem Routing) vornehmen zu können.
Ideal wäre es, ISYKON im Managementsystem abzubilden und einen bidirektionalen Datentransportpfad für den Informationsaustausch zwischen dem relationalen Datenbanksystem und der Management Information Base zu schaffen. Ist dies möglich, steht einer Erweiterung von ISYKON um dynamische Informationen prinzipiell nichts mehr im Wege.

Eine Untersuchung der Aspekte einer Datenintegration unter Gesichtspunkten des *Enterprise Managements* erfolgt zum gegenwärtigen Zeitpunkt im Rahmen einer Arbeit des Münchner Netzmanagement Teams [WIED 93].

In der vorliegenden Arbeit muß jedoch davon ausgegangen werden, daß ISYKON–Daten nicht in den Modellierungsprozeß einfließen können.
Die in den ASCII–Dateien abgelegten Informationen reichen nicht für eine umfassende Modellierung der verteilten Kommunikationsanwendung NWP aus, bieten jedoch eine Fülle an Zusatzinformationen, die, insbesondere im Fall der Wege– und Log–Dateien, für das NWP–Management eine wertvolle Unterstützung sind.
Um der Dynamik des NWP–Verbunds gerecht werden zu können, ist es erforderlich, zusätzliche Managementobjekte zu definieren.

Kapitel 5

Managementszenarien für das NetzWerk–Programm

In diesem Kapitel werden die Managementszenarien beschrieben, die aus den Anforderungen des Netzbetreibers abgeleitet wurden. Aus Gründen der Übersichtlichkeit umfaßt ein Abschnitt jeweils ein Szenario, das zuerst eingehend erläutert wird und anschließend, soweit möglich, als Basis für eine Implementierung, in Pseudocode realisiert ist.

5.1 Operationen auf Attributen, Erstellen von Views

Von großer Bedeutung für die folgenden Managementszenarien sind Operationen auf Attributen von Managementobjekten.
Hierzu zählen insbesondere:

– das Einlesen von Attributwerten eines Managementobjektes

– das Vergleichen des Istwertes eines Attributes mit einem vorgegebenen Sollwert

– das Anwenden arithmetischer Operationen auf Attributwerte

– das Schreiben eines Attributwertes

Diese Operationen betreffen hauptsächlich die in Abschnitt 4.3 aufgeführten Attribute.

Aufgrund der Tatsache, daß die SPECTRUM Inference Handler API Funktionen anbietet, die direkt die obengenannte Funktionalität besitzen, unterbleibt eine Pseudocode–Darstellung an dieser Stelle.

Um die zahlreichen Daten, die das NWP–Management erfordert, strukturieren zu können, ist es notwendig, Views einzuführen, die die Aspekte

– Konfiguration

– Organisation
– Topologie
– Anwendungsinformation
– Diagnose

umfassen.
Eine eingehende Beschreibung dieser Sichten erfolgt in Abschnitt 6.4.

Für die Akzeptanz der NWP–Managementlösung von Seiten der Benutzer ist eine klare und übersichtliche Darstellung der Managementdaten unerläßlich. Wichtig ist insbesondere eine Einbettung der Managementfunktionen in die adäquaten Sichten, da nur so gewährleistet ist, daß Managementinformation und Managementwissen auch für den Benutzer zusammengehörig erscheinen. So müssen zum Beispiel aus der Diagnose–Sicht, deren Aufgabe es ist, eingetroffene Alarme und Zusatzinformationen zur Fehlersuche bereitzuhalten, Funktionen, die weitreichende Diagnosemöglichkeiten bieten, aufrufbar sein.

SpectroGRAPH gestattet dem Benutzer, Attribute zur Laufzeit in bestehende Sichten zu integrieren. Weitergehende Veränderungen wie die Integration von neuen Sichten in SpectroGRAPH erfordern die View API.

5.2 Erstellen einer NWP–Map, Autotopologiefunktion

Um eine Übersicht über den aktuellen Status aller NWP–Knoten zu erhalten, ist es notwendig, jeden einzelnen Knoten als *Icon* in einer logischen Map darzustellen.
Aufgrund der großen Anzahl von NWP–Knoten innerhalb des NWP–Verbunds ist es ein mühsames Unterfangen, jeden einzelnen Knoten von Hand zu erfassen.

Da in der auf jedem NWP–Knoten verfügbaren Wegedatei alle anderen Knoten sowie deren jeweilige Nachbarknoten verzeichnet sind, läßt sich ein Algorithmus angeben, der eine Übersicht aller NWP–Knoten in einer topologischen Sicht generiert und logische Verbindungen zwischen ihnen erzeugt.

Die Gestaltung des *Icons* für den Modelltyp `NWP-Knoten` ist in Abschnitt 6.4 erläutert und in Abbildung 6.10 dargestellt.

Ein Algorithmus für die Autotopologiefunktion lautet wie folgt:

```
LIES Wegedatei eines beliebigen NWP-Knotens;
ERZEUGE 2-spaltige Knotenliste aus 2. und 4. Spalte der Wegedatei;
Modelltyp := NWP_Node;
```

```
SOLANGE (Knotenliste nicht leer)

   Aktueller_Knoten := 1.Spalteneintrag aus Knotenliste;
   Naechster_Knoten := 2.Spalteneintrag aus Knotenliste;

   TRAGE Aktueller_Knoten MIT Modelltyp IN VNM-Tabelle EIN;
   ERZEUGE CONNECTS-TO Relation (Aktueller_Knoten, Naechster_Knoten);

   STREICHE Aktueller_Knoten aus Knotenliste;
   STREICHE Naechster_Knoten aus Knotenliste;
;
ENDE
```

Der Algorithmus nutzt dabei die Tatsache aus, daß SpectroGRAPH zur Erzeugung von *Icons* und Verbindungen auf VNM–Informationen angewiesen ist und eine VNM–interne Tabelle (im Algorithmus als VNM-Tabelle bezeichnet) auswertet, die existierende Modelle zu Modelltypen enthält. Ebenso bewirkt das Einzeichnen einer Verbindung zwischen zwei Modellen die Erzeugung einer CONNECTS–TO Relation.
Aus diesem Grunde muß lediglich die VNM–Tabelle um die Namen der Modelle, die den Modelltyp `NWP_Node` besitzen, ergänzt werden sowie die CONNECTS–TO Relation um die entsprechenden Tupel.

SpectroGRAPH erzeugt mit diesen Angaben automatisch eine NWP–Map, die sämtliche Knoten als auch die zwischen ihnen bestehenden Verbindungen enthält.

5.3 Transaktionskontrolle

Da für Dateiübertragungen mit dem NetzWerk–Programm keine obere Schranke für die Übertragungszeit angegeben werden kann, ist es nützlich, den Fortschritt einer aktiven Transaktion prüfen zu können.

Dies geschieht, indem der Benutzer nach Selektion des Menüpunktes „Transaktionskontrolle“ den Transaktionscode sowie den Zielknoten angibt. Der Algorithmus ermittelt daraus den Ursprungsknoten (der lexikalisch im Transaktionscode enthalten ist) und prüft, ob ein NWP–Knoten auf der Route zum Zielknoten eine Datei mit dem Präfix DA zum gegebenen Transaktionscode in ihrem Eingangsbereich enthält.
Die zum Transaktionscode gehörende DA–Datei kommt auf der gesamten Route nur an einem einzigen NWP–Knoten vor, da sie gelöscht wird, sobald die im Arbeitsbereich des Knotens befindliche DW–Datei an den Sendeserver übergeben wurde und dieser die DW–Datei auf dem nächsten NWP–Knoten in eine DA–Datei umbenennen konnte [1].
Derjenige NWP–Knoten, in dessen Eingabebereich eine DA–Datei liegt, muß folglich der für diese Transaktion aktuelle Knoten sein.

[1] vergleiche hierzu Teilabschnitt 4.1.2

Analog zu den DA–Dateien werden auch die Quittungsdateien (Präfix QU) gehandhabt. Hier läßt sich anhand des Vorhandenseins der QU–Datei im Eingabebereich der NWP–Knoten ermitteln, wie weit die Transaktion fortgeschritten ist.

Es muß daher zwischen drei Fällen unterschieden werden:

(1) Im Eingabebereich des Knotens ist die zum Transaktionscode passende DA–Datei abgelegt:
Die Transaktion befindet sich in der Übertragungsphase und der NWP–Knoten ist der gesuchte Knoten.

(2) Im Eingabebereich des Knotens ist die zum Transaktionscode passende QU–Datei abgelegt:
Zur Zeit läuft die Rückübertragung der Quittung, die an diesem Knoten angekommen ist.

(3) Im Eingabebereich des Knotens ist weder eine DA– noch eine QU–Datei enthalten:
Dieser Knoten liefert keine für die Aufgabenstellung verwertbaren Informationen. Aus seiner Wegedatei wird daher der nächste Knoten ermittelt.

Zu beachten ist, daß die Quittungsdatei immer *dieselbe* Route wie die Datendatei nimmt; die Reihenfolge der Knoten kehrt sich allerdings um.

Der folgende Algorithmus realisiert dieses Szenario:

```
EINGABE(Transaktionscode);
EINGABE(Zielknoten);

ERMITTLE Ursprungsknoten AUS Transaktionscode;
Aktueller_Knoten := Ursprungsknoten;
gefunden := FALSE;
Eingabebereich := /usr/spool/nw/tmp;

SOLANGE (nicht gefunden)

   LOGIN (Aktueller_Knoten);
   Logoutknoten := Aktueller_Knoten;

   FALLS DATransaktionscode-Datei IN Eingabebereich
         \\ 1. Fall: DA-Datei gefunden
         AUSGABE "Daten an" Aktueller_Knoten "eingetroffen";
         gefunden := TRUE;

   SONST FALLS QUTransaktionscode-Datei IN Eingabebereich
               \\ 2. Fall: QU-Datei gefunden
```

```
            AUSGABE "Quittung an" Aktueller_Knoten "eingetroffen";
            gefunden := TRUE;

      SONST \\ 3. Fall: Suche fortsetzen mit naechstem Knoten
            ERMITTLE Naechster_Knoten ZU Aktueller_Knoten AUS \\
                                                    Wegedatei;
            Aktueller_Knoten := Naechster_Knoten;

   LOGOUT (Logoutknoten);
 ;
ENDE
```

Diese Funktion kann auch dazu benutzt werden, im Falle eines ausgefallenen Knotens herauszufinden, welchen Namen dieser Knoten hat. Vorausgesetzt wird in diesem Fall, daß eine Störung der NWP–Software (Control Program, Empfangs– und Sendeserver) vorliegt; bei Fehlern in der Infrastruktur eines Knotens ist dieser Algorithmus natürlich machtlos.
Sollte das Control Program ausfallen, legt der Empfangsserver alle erhaltenen Dateien zwar weiterhin im Eingangsbereich ab; sie können aber nicht mehr umbenannt werden und bleiben als DA– oder QU–Dateien im Eingangsbereich liegen. Da der obengenannte Algorithmus nach bestimmten DA– bzw. QU–Dateien sucht, kann der defekte NWP–Knoten schnell gefunden werden.

5.4 Überwachung der Eingabewarteschlange

Eine weitere wichtige Größe, die es zu überwachen gilt, ist die Eingabewarteschlange eines jeden NWP–Knotens, in der vom Empfangsserver entgegengenommene Daten abgelegt werden. Die Warteschlange eines NWP–Knotens besteht aus dem Verzeichnis `/usr/spool/nw/tmp` und enthält Dateien mit bis zu fünf verschiedenen Präfixen.

Für das NWP–Management ist es wichtig, die Namen der Dateien sowie deren Größe feststellen zu können, um bereits frühzeitig eine drohende Überlastsituation zu erkennen und diesen Knoten für weitere eintreffende Daten sperren zu können.

Ein Algorithmus, der auf die Eingabe des Knotennamens durch den Benutzer alle Dateien im Eingabebereich dieses Knotens sowie deren Größe ermittelt, kann sich direkt auf den UNIX–Befehl `ls` abstützen, um diese Parameter zu ermitteln. Vier Schritte müssen nach der Eingabe durch den Benutzer erfolgen:

(1) Ein *remote login* auf dem NWP–Knoten

(2) Die Ausführung des `ls`–Kommandos

(3) Die Darstellung der erhaltenen Daten in übersichtlicher Form auf dem Bildschirm der Managementplattform

(4) Logout

5.5 Anzeigen einer NWP–Route

Um einen bei einer Übertragung aufgetretenen Fehler lokalisieren zu können, ist es wichtig, einen Überblick über die auf der für den Dateitransfer vorgesehenen Route liegenden Knoten zu erhalten.

Nach Wahl des Menüpunkts „NWP–Route anzeigen" wird der Benutzer aufgefordert, den Ursprungs– und den Zielknoten einzugeben. Daraus errechnet der Algorithmus alle NWP–Knoten, die auf der Route liegen und gibt dies am Bildschirm in Form einer Liste an.

Der Operateur an der Managementplattform kann mit Hilfe der erhaltenen Knotenliste anhand der NWP–Map, die den aktuellen Zustand aller NWP–Knoten enthält, prüfen, ob die Route geändert werden muß, weil ein auf ihr liegender NWP–Knoten ausgefallen ist.

Der Algorithmus zum Anzeigen einer Route lautet:

```
EINGABE (Ursprungsknoten);
EINGABE (Zielknoten);
Aktueller_Knoten := Ursprungsknoten;

SOLANGE (Aktueller_Knoten != Zielknoten)

   LOGIN (Aktueller_Knoten);
   Logoutknoten := Aktueller_Knoten;
   ERGAENZE Knotenliste UM Aktueller_Knoten;
   ERMITTLE Naechster_Knoten ZU Aktueller_Knoten AUS Wegedatei;
   Aktueller_Knoten := Naechster_Knoten;
   LOGOUT (Logoutknoten);
 ;
\\Zielknoten erreicht

ERGAENZE Knotenliste UM Aktueller_Knoten;
AUSGABE(Knotenliste);

ENDE
```

5.6 Befehlsausführung auf entfernten NWP–Knoten

Um dem NWP–Support die Fehlerbeseitigung zu ermöglichen, ist es notwendig, die Ausführung bestimmter Befehle auf entfernten NWP–Knoten zuzulassen.

Beispiele für solche Befehle sind:

– Neustart des NWP Control Programs: `/bin/nwpstart`.

– Beenden des NWP Control Programs: `/bin/nwpstop`.

– Bereinigen der NWP–Datenbereiche: `/bin/nwpcleanup`.

– Eröffnung einer TELNET–Sitzung zu einem NWP–Knoten, um UNIX–Kommandos ausführen zu können.
  Hierbei sind insbesondere folgende Befehle von Interesse:

  - Ändern von Zugriffsrechten für Dateien und Verzeichnisse: `chmod`
  - Modifizieren des Inhabers einer Datei oder eines Verzeichnisses und seiner Gruppe: `chown` bzw. `chgrp`
  - Löschen von Dateien oder Verzeichnissen: `rm` bzw. `rmdir`
  - Wechsel des aktuellen Verzeichnisses `cd` und Anzeigen des Verzeichnisinhalts: `ls`

Die Befehle `nwpstart`, `nwpstop` und `nwpcleanup` stehen bereits auf jedem Knoten zur Verfügung. Dasselbe gilt für die UNIX–Kommandos.
Sämtliche Befehle können daher in einer TELNET–Sitzung auf dem betroffenen NWP–Knoten abgesetzt werden und müssen daher nicht neu implementiert werden.

Das Eröffnen einer TELNET–Sitzung sollte durch Auswahl eines Bereiches des NWP–Knotenicons mit der Maus möglich sein. Diese Funktionalität wird bereits durch das Managementmodul *MaestroVision* erbracht (vgl. hierzu Abschnitt 6.2).

Überlegungen, welches Aussehen ein *Icon* haben sollte, das einen NWP–Knoten repräsentiert, sind in Abschnitt 6.4 „Graphische Aufbereitung der Managementdaten“ beschrieben.

Obwohl es technisch mit Hilfe der Inference Handler API möglich ist, etwa den Neustart oder das Beenden des Control Programs automatisch zu veranlassen, stehen solchen Tätigkeiten Bedenken des Bedienpersonals gegenüber: Derartig folgenschwere Eingriffe in den Betrieb des NetzWerk–Programms sollten besser von qualifiziertem Fachpersonal ausgeführt werden, als von einem Programm, das auf der Managementplattform läuft.

Im Zusammenhang mit der Ausführung von Systemkommandos auf entfernten Knoten ist auch die für das NWP–Management überaus wichtige Fähigkeit zu sehen, Inhalte der Log–Dateien, der NIDAT und der Wegedatei nicht nur lesen, sondern auch editieren zu können.

Sollte ein NWP–Knoten ausfallen, so kann das Bedienpersonal mit Hilfe der Editierfunktion die Wegedateien der umliegenden Knoten umkonfigurieren.
Auch in diesem Falle ließe sich die Operateurtätigkeit automatisieren, vorausgesetzt, die Wegedatei wird um alternative Routen ergänzt.

5.7 Überwachung der Plattenbelegung

In Teilabschnitt 4.2.2 wurde die Notwendigkeit begründet, eine Überwachung der auf einem NWP–Knoten verfügbaren Plattenkapazität durchzuführen.

Unter Performance–Gesichtspunkten erscheint es jedoch zweifelhaft, die Managementplattform mit dieser Funktionalität auszustatten, da die periodische Ausführung einer Prozedur auf jedem NWP–Knoten insgesamt eine nicht zu unterschätzende Netzlast verursacht.

Sinnvoller wäre es stattdessen, diese Funktionalität auf dem Agent zu erbringen, indem ein neues Attribut `Free_Diskspace` des Modelltyps `NWP_Node` in die NWP–MIB integriert und dieses mit einem Schwellwert (zum Beispiel Free_Diskspace: 10%) versehen wird.

Aufgabe des Agents ist es dann, die prozentuale Belegung des NWP–Eingabebereiches `/usr/spool/nw/tmp` mit Hilfe des UNIX–Kommandos `df` zu ermitteln und mit dem Schwellwert zu vergleichen. Sind weniger als beispielsweise 10% des gesamten Speicherplatzes verfügbar, sollte der Agent selbständig einen Alarm an die Managementplattform senden, der dort durch Verfärbung des Knotenicons und einer entsprechenden Meldung dem Operateur mitgeteilt wird.
Dieser kann anschließend Schritte zur Behebung des Problems einleiten.

Kapitel 6

Implementierung der Managementszenarien

Am Anfang dieses Kapitels, in Abschnitt 6.1, stehen Überlegungen, wie eine konkrete Managementlösung für die in den vorangegangenen Kapiteln erläuterte Managementproblematik erarbeitet werden kann.
Ein Managementkonzept umfaßt immer zwei Gesichtspunkte: Den der Agent–bezogenen Ziele d.h. die Fragestellung, welche Managementfunktionalität von den einzelnen NWP–Knoten erbracht werden sollte (beschrieben in Abschnitt 6.2) und den Gesichtspunkt des *Managers*, also der zentralen Managementstation (Abschnitt 6.3).
Anschließend wird in Abschnitt 6.4 aufgezeigt, wie die Managementdaten übersichtlich graphisch aufbereitet werden können.

6.1 Integration von NWP–Managementfunktionen in die Netzmanagementplattform SPECTRUM

Ausgehend von den in Abschnitt 4.2 erläuterten Managementanforderungen und der in Abschnitt 4.3 beschriebenen Managementinformation wird im folgenden aufgezeigt, wie diese für eine konkrete Implementierung in die Netzmanagementplattform SPECTRUM integriert werden können. Abbildung 6.1 skizziert die Vorgehensweise:

(1) Die Managementinformation, die als Wegedatei, NIDAT, Log–Dateien sowie zusätzlicher, neu eingebrachter Größen vorliegt, kann auf zwei verschiedene Arten zur Erzeugung eines NWP–Modells in SPECTRUM integriert werden:

   (a) Die erste Möglichkeit besteht darin, anhand der im *Request for Comments* „Structure and Identification of Management Information for TCP/IP–based Internets“ [RFC 1155] spezifizierten Konventionen Managementobjekte und deren Attribute vermöge einer Teilmenge der „Abstract Syntax Notation One“

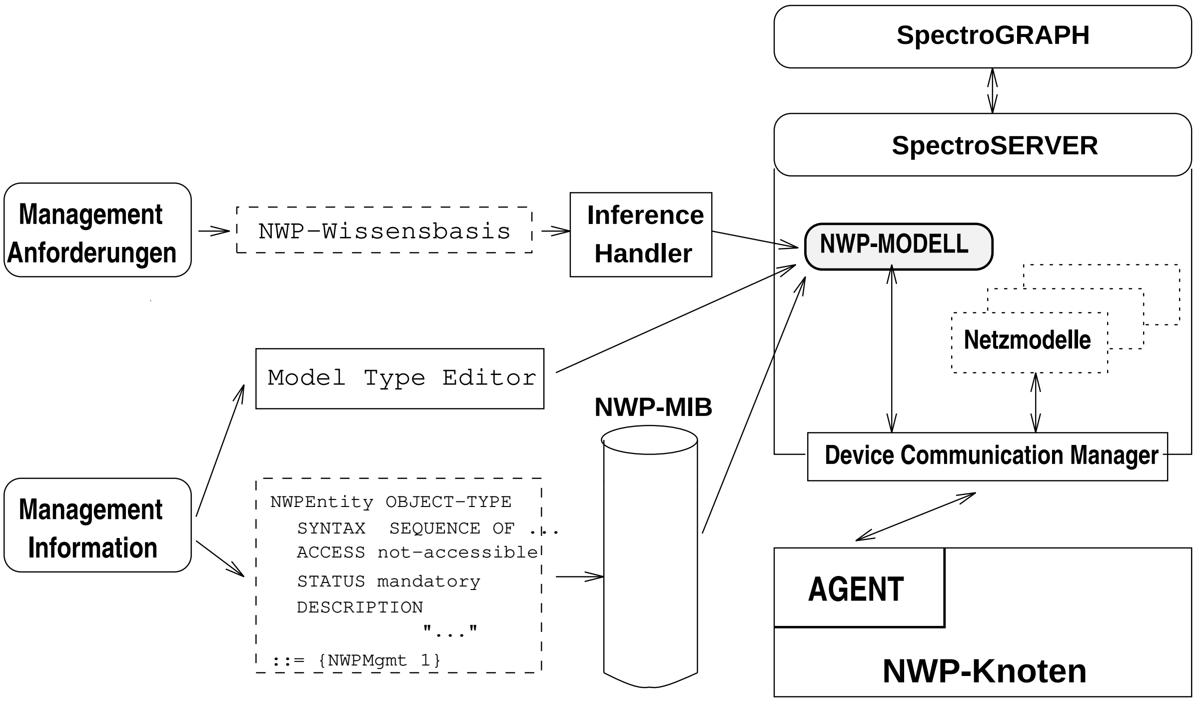

Abbildung 6.1: Vorgehensweise unter Architekturgesichtspunkten

[GORA 90] zu formalisieren.
Hierbei ist es notwendig, sich zu überlegen, an welcher Stelle die Managementobjekte in die Hierarchie der MIB–II [RFC 1213] eingefügt werden sollen.
Das Ergebnis dieses Vorgangs ist eine MIB–II-konforme NWP–spezifische *Management Information Base*, die in Abbildung 6.1 als „NWP–MIB"dargestellt ist.

Beachtenswert ist, daß SPECTRUM noch nicht darauf zugreifen kann, da die NWP–MIB noch nicht in das Netzmanagementsystem eingebracht wurde.
Dies wiederum geschieht durch einen MIB–Compiler, der in der Lage ist, die ASN.1–Makros in Modelltypen umzusetzten. Hiermit werden dann die logischen Objektbeschreibungen des NWP–Modells erzeugt.

(b) Alternativ zum oben genannten Verfahren können neue deklarative Informationen auch direkt mit dem *Model Type Editor* in SPECTRUM integriert werden. Der Benutzer kann damit zusätzlich zu Modelltypen und deren Attributen auch Relationen und Regeln definieren und das semantische Datenmodell der Netzmanagementplattform direkt erweitern.

(2) Die Managementanforderungen werden gesammelt und strukturiert. Die NWP–Wissensbasis umfaßt damit sämtliches für das NWP–Management notwendige pro-

zedurale Wissen und ist der Ausgangspunkt für die Implementierung der in Kapitel 5 beschriebenen Managementszenarien.

Um das in der NWP–Wissensbasis gesammelte Wissen in das NWP–Modell der Netzmanagementplattform SPECTRUM integrieren zu können, muß es geeignet formalisiert werden. Das Ergebnis dieses Prozesses sind C++-Methoden, sogenannte *Inference Handler*, die auf den logischen Objektbeschreibungen [1] des NWP–Modells aufsetzen und die geforderte Managementfunktionalität erbringen.

Das Verfahren zur Generierung neuen prozeduralen Wissens ist in Abschnitt 6.3 ausführlich beschrieben.

6.2 *MaestroVision*: Ein SPECTRUM–konformer Agent für das Systems Management

Dieser Abschnitt zeigt anhand eines bestehenden kommerziellen Agenten auf, welche Funktionalität ein Agent haben muß, um mit dem Netzmanagementsystem SPECTRUM kooperieren und die verteilte Kommunikationsanwendung NWP managen zu können.

6.2.1 MaestroVision–Kurzbeschreibung

Das Managementmodul MaestroVision von Calypso Software Systems, Inc. erweitert das Netzmanagementsystem SPECTRUM um systemmanagementrelevante Belange und ist zum gegenwärtigen Zeitpunkt für verschiedene UNIX–Derivate erhältlich.

Der Teil von MaestroVision, der den Agent darstellt, erlaubt, systemspezifische Parameter zu überwachen.
Im einzelnen sind die folgenden Bereiche eines UNIX–Systems in parametrisierter Form auswertbar:

– **Filesystems**:
MaestroVision folgt den Konventionen und teilt Filesysteme in remote d.h. über das *Network File System* (NFS) gemountete und lokale Dateisysteme auf und überwacht diese.

– **Memory**:
Es können statistische Daten bezüglich der Belegung sowohl des Haupt– als auch des Hintergrundspeichers (*Swap*–Bereich) ermittelt werden.

[1] In der SPECTRUM–Terminologie *Modelltypen* genannt

– **Processor**:
  Die aktuelle Auslastung eines oder mehrerer Prozessoren wird dem Netzmanagement zur Verfügung gestellt.

– **Application**:
  Die derzeit aktiven Prozesse können ermittelt werden; zusätzlich wird festgestellt, ob eine Anwendung den ihr zugewiesenen Speicherbereich überschreitet.
  Insbesondere die erste Funktion spielt für die vorliegende Arbeit eine bedeutende Rolle.

Ein Charakteristikum von MaestroVision ist, daß die oben genannten Systembestandteile selbständig vom *MaestroVision Server*, dem Agent–Prozeß, überwacht werden und eine Zustandsänderung der Komponentenattribute bzw. ein Überschreiten der vordefinierten Schwellwerte automatisch dem Managerprozeß gemeldet werden, so daß dieser kein *Polling* der Agenten vornehmen muß.

Die Vorteile eines solchen „intelligenten“ Agents liegen auf der Hand:

– Die Aktualität der Daten ist jederzeit gewährleistet, da keine Pollingzyklen mehr auftreten.

– Die Netzlast, hervorgerufen durch das Management, sinkt spürbar, da nur dann Daten zwischen Manager und Agent ausgetauscht werden, wenn eine signifikante Zustandsänderung einer Systemkomponente auftritt.

Falls erforderlich, kann ein Polling der Attribute erfolgen; die Pollingintervalle sind vom Benutzer frei konfigurierbar. Die obengenannten Vorteile erfahren dann allerdings Einbußen.

Um den Zugriff auf die Managementdaten von SPECTRUM aus zu ermöglichen, ist es notwendig, SpectroSERVER um MaestroVision–spezifische Managementfunktionalität zu erweitern; eine Aufgabe, die der *MaestroVision Client* erfüllt.

Der MaestroVision Client, der auf der Managementstation installiert ist, stellt sicher, daß die von den MaestroVision Servern auf den jeweiligen Systemen überwachten Systemressourcen und die damit verbundenen Meldungen verarbeitet, aufbereitet und dargestellt werden. Er ermöglicht eine vollständig in SpectroGRAPH eingebettete Repräsentation der gemanagten Objekte d.h. die Ressourcen–spezifischen neuen *Icons* sind in die vorhandenen Sichten integriert.
Dies impliziert die Einführung neuer Modelltypen, die den oben genannten Systemressourcen entsprechen. Abbildung 6.2 enthält eine Übersicht über diese Modelltypen.

Die Integration der MaestroVision–spezifischen Modelltypen in die SPECTRUM Modelltypenhierarchie gibt jedoch Anlaß zu Kritik, da hierbei die Konventionen von SPECTRUM verletzt wurden:
In der Regel werden vom Modelltyp `Location` ausschließlich diejenigen Modelltypen abgeleitet, die sich auf räumliche Gesichtspunkte beziehen, also `Country`, `Site`, `Section` oder `Floor`.

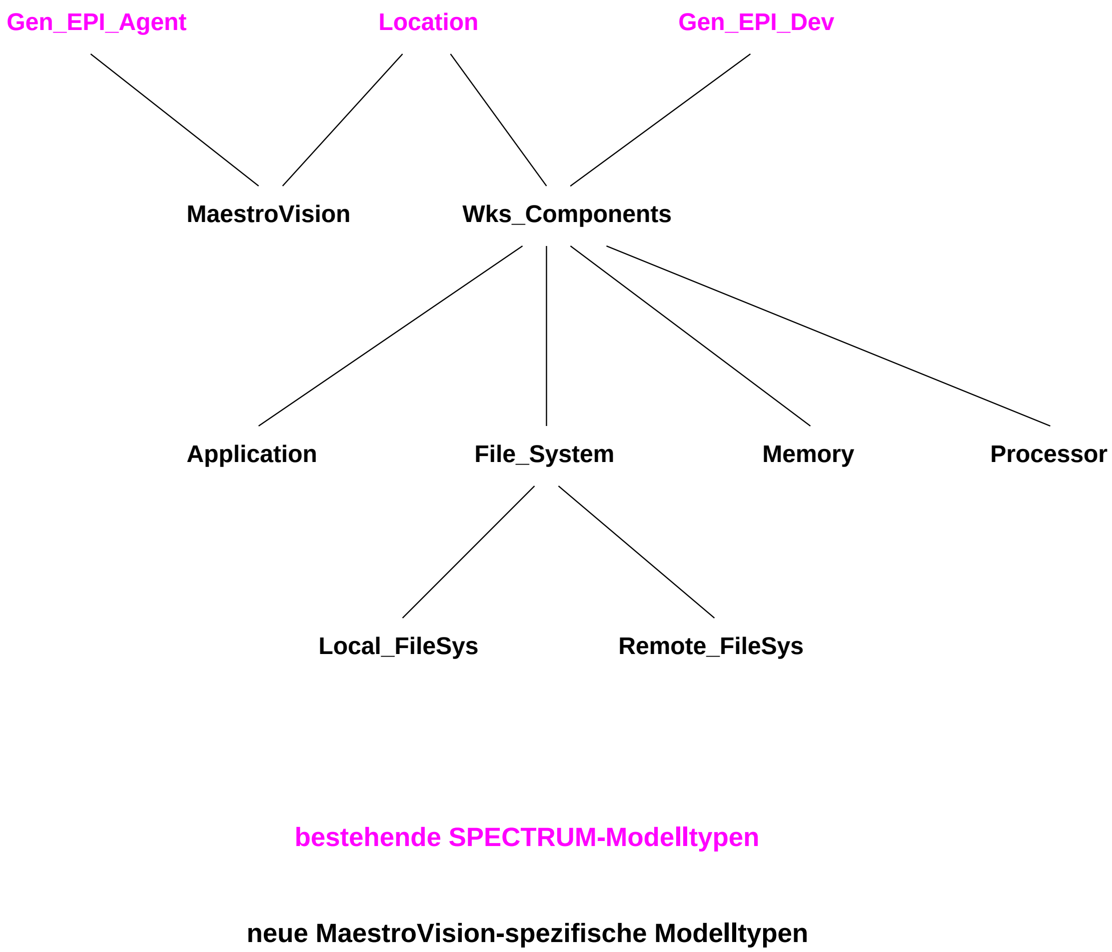

Abbildung 6.2: *MaestroVision*–spezifische Modelltypen

Schwer nachvollziehbar erscheint, weshalb MaestroVision den Modelltyp „Komponenten von Workstations“ (`Wks_Components`) und damit weitere Modelltypen wie `Application`, `File_System`, `Memory` und `Processor` davon ableitet.
Die Modelltypenhierarchie stellt einen strukturierten Beschreibungsrahmen dar, in dem Modelltypen unter dem Gesichtspunkt ihrer logischen Zugehörigkeit von einem gemeinsamen allgemeinen Modelltyp abgeleitet und somit unter einem Oberbegriff zusammengefaßt werden.
Es liegt auf der Hand, daß `Wks_Components` nahezu keine Gemeinsamkeiten mit `Region` oder `Site` hat und daher dessen Eingruppierung in `Location` fragwürdig erscheint.
Es steht außer Frage, daß bereits in der rudimentären Form der Hierarchie Modelltypen existieren, die dem Wesen der neu eingeführten Modelltypen wesentlich besser entsprechen, als es der Fall bei `Location` ist. Als Beispiele seien hier `Component`, `Device` oder

`Software_Application` genannt. Man wäre daher besser beraten gewesen, diese Modelltypen als Basis für neue zu verwenden, allerdings wären die neuen Modelltypen dann über die gesamte Modelltypenhierarchie verstreut.

Die freie Erweiterbarkeit der SPECTRUM Modelltypenhierarchie birgt daher die Gefahr, daß bei derart freizügiger Integration neuer Modelltypen die Systematik über kurz oder lang verloren geht.

Zur Kommunikation zwischen Client und Server kommt ein auf TCP/IP basierendes, eigenständiges Protokoll zum Einsatz, das über das *Extended Protocol Interface* an den *Device Communication Manager* und damit an SpectroSERVER angeschlossen ist. Abbildung 6.3 zeigt die technische Realisierung von MaestroVision und das Zusammenspiel zwischen dessen Bestandteilen *MaestroVision Client* und *MaestroVision Server*.

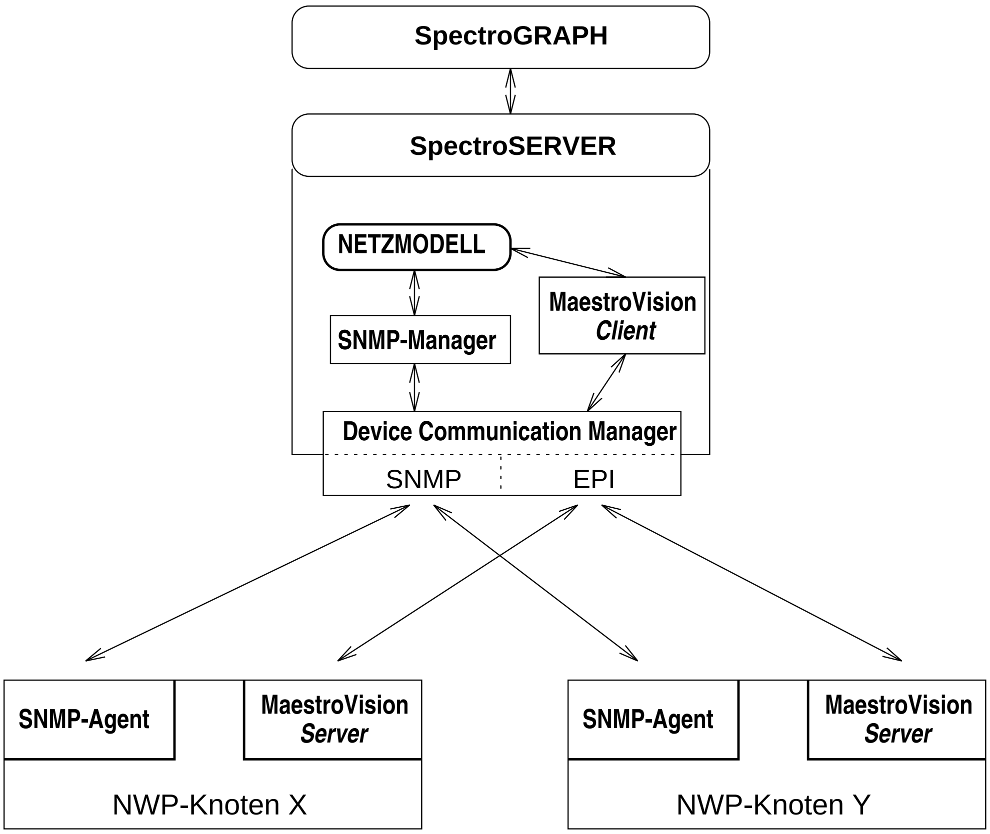

Abbildung 6.3: Integration von MaestroVision in SPECTRUM

6.2.2 Integration NWP–spezifischer Modelltypen in SPECTRUM

Aufgrund der Tatsache, daß SPECTRUM keine für die Modellierung des NetzWerk–Programms adäquaten Modelltypen bietet, werden die von MaestroVision zur Verfügung gestellten Modelltypen als Basis–Modelltypen angenommen.

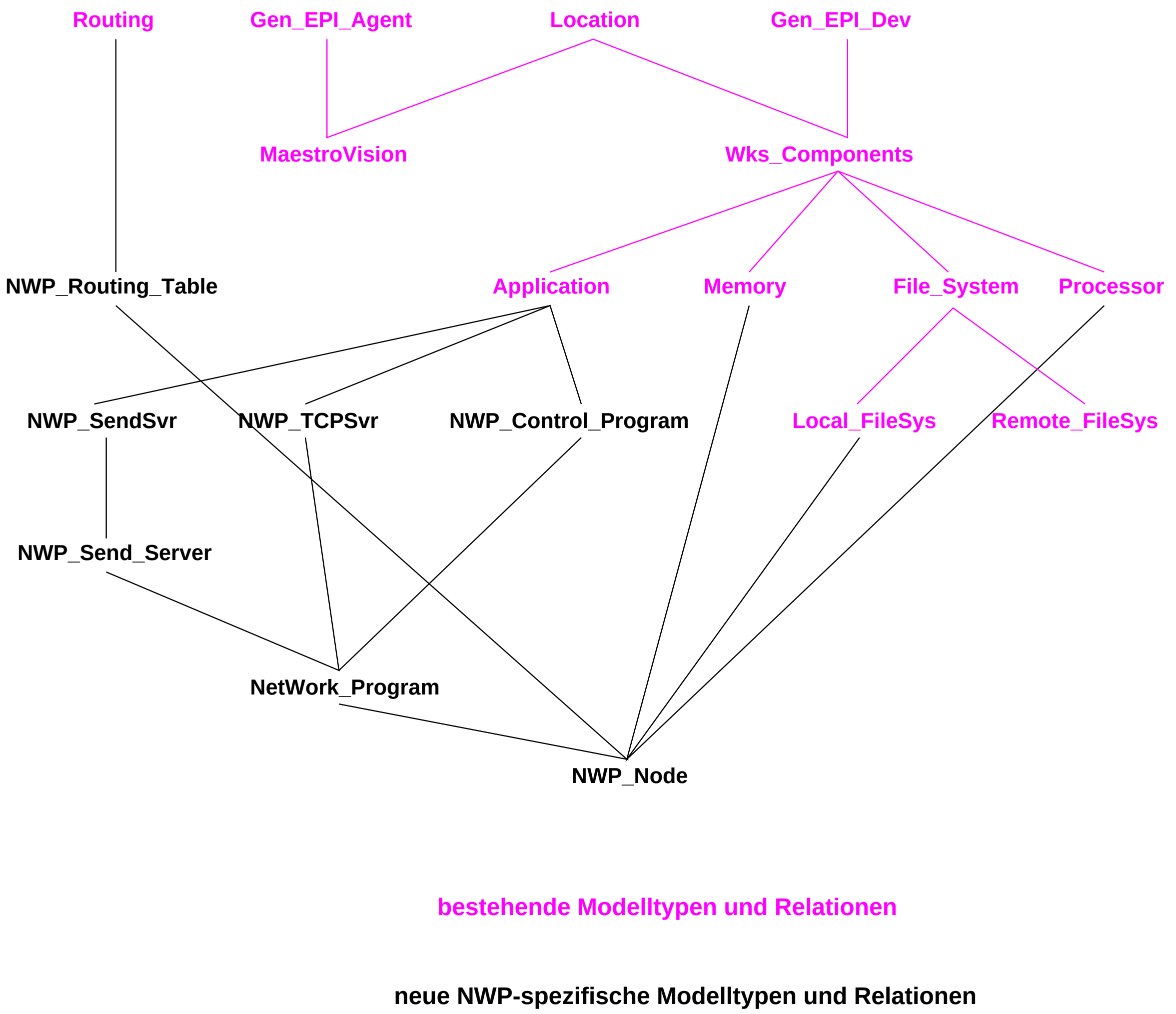

Abbildung 6.4: Erweiterung von SPECTRUM um NWP–spezifische Modelltypen

Die wichtigsten Elemente eines NWP–Knotens, die für eine Modellierung in Frage kommen, sind:

– das NWP Control Program

- der Empfangsserver TCPSvr
- der Sendeserver SendSvr
- das lokale Dateisystem `/usr/spool/nw`
- der Hauptspeicher des Knotens
- der Prozessor des Knotens
- die Wegedatei
- die NIDAT
- die Log–Dateien

Für diese Bestandteile werden, sofern sie nicht bereits durch die MaestroVision–Modelltypen abgedeckt sind, neue Modelltypen eingeführt und von bestehenden Modelltypen abgeleitet. Abbildung 6.4 zeigt auf, wie eine derartige Modellierung [2]aussehen kann.

Die aufgezeigte Modellierung ist konform zur SPECTRUM–„Philosophie", da zuerst die einzelnen Bestandteile integriert und anschließend zu einem Ganzen, dem `NWP_Node`, zusammengefaßt werden.

6.3 Programmierung mit den SPECTRUM Level–II Toolkits

Teilabschnitt 3.3.2 deutete bereits an, daß für bestimmte Anforderungen bestehende Tools wie der Model Type Editor nicht ausreichen. Beispiele für solche Anwendungen sind das Einbringen neuen prozeduralen Wissens, die Erstellung neuer Sichten oder die Erweiterung von SPECTRUM um die Fähigkeit, neue Managementprotokolle zu unterstützen. Zu diesem Zweck besitzt SPECTRUM Programmierschnittstellen, die als **Level–II Toolkits** bezeichnet werden und in Abbildung 6.5 dargestellt sind.

6.3.1 Übersicht über die Level–II Toolkits

- **SpectroSERVER API**:
  Sie bietet dem Entwickler die Möglichkeit, neue Anwendungen zu schreiben, die auf SpectroSERVER und damit auf die Daten von SPECTRUM zugreifen. Aufgrund der *Client/Server*–Beziehung, in der die Anwendungen zu SpectroSERVER stehen, können (theoretisch) beliebig viele voneinander unabhängige Anwendungen gleichzeitig Kommunikationsbeziehungen zu SpectroSERVER unterhalten.

[2]Um die Übersichtlichkeit nicht zu beeinträchtigen, sind die NIDAT sowie die Log– und Wegedateien nicht in der Abbildung eingezeichnet, obwohl auch sie modelliert wurden.

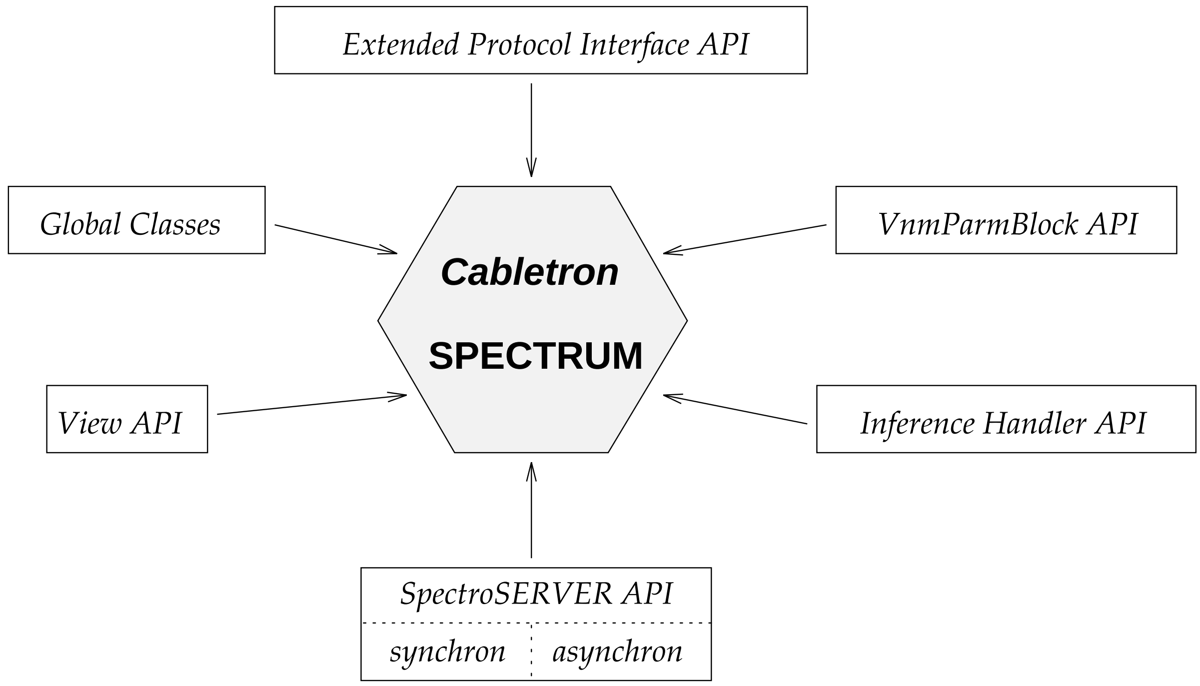

Abbildung 6.5: Die SPECTRUM–Programmierschnittstellen

Man unterscheidet zwischen der *asynchronen* [SPEC SSA] und der *synchronen* [SPEC SSS] SpectroSERVER API, die in Abhängigkeit von der zu erstellenden Anwendung vom Entwickler ausgewählt werden:
Über die asynchrone SpectroSERVER API wird eine Verbindung aufgebaut, bei der die Anwendung SpectroSERVER Aufträge übermittelt, ohne daß ihr Programmlauf durch Warten auf das Resultat unterbrochen wird. Ist der Auftrag von SpectroSERVER abgearbeitet worden, erhält das Anwendungsprogramm eine Nachricht, daß die angeforderten Daten bereitstehen.
Interaktive Anwendungen und graphische Benutzerschnittstellen machen von der asynchronen SpectroSERVER API Gebrauch. Das derzeit komplexeste Beispiel einer Anwendung, die die asynchrone SpectroSERVER API nutzt, ist SpectroGRAPH, die graphische Benutzerschnittstelle von SPECTRUM.
Anwendungen, die Daten (Attribute, Relationen...) oder Berichte und Statistiken von SpectroSERVER abfragen, benutzen in der Regel die *synchrone* SpectroSERVER API.
Änderungen und Erweiterungen der *Virtual Network Machine* können mit der SpectroSERVER API nicht vorgenommen werden.

- **View API**:
  Die View API wird benötigt, um neue Sichten und neue Funktionalität in SpectroGRAPH einzuführen. Eigene *Client*–Anwendungen können mit Hilfe dieses Toolkits SpectroGRAPH beispielsweise um *cut and paste*–Möglichkeiten erweitern.

- **Extended Protocol Interface API**:
  Soll ein neues Managementprotokoll in SPECTRUM integriert werden, ist die Extended Protocol Interface API [SPEC EPI] erforderlich.
  Mit ihr kann der *Device Communication Manager* erweitert werden, um SPECTRUM in die Lage zu versetzen, Komponenten und Systeme zu managen, deren Agenten weder SNMP noch ICMP unterstützen.

- **Global Classes**:
  Die Global Classes [SPEC GLO] sind keine Programmierschnittstelle im eigentlichen Sinne, da sie nicht unmittelbar die Bestandteile von SPECTRUM ergänzen.
  Sie stellen dem Entwickler jedoch allgemeine Methoden und Datenstrukturen (Buffers, Trees, Hash-Tables...) zur Verfügung und müssen daher bei Verwendung der APIs mit eingebunden werden.

- **VnmParmBlock API**:
  Zur Kommunikation mit seinen *Clients* verwendet SpectroSERVER ein Protokoll, das auf TCP/IP basiert. Die Protokolldateneinheiten, die damit ausgetauscht werden, sind sogenannte *VnmParmBlocks*.
  Die VnmParmBlock API [SPEC VPA] stellt dem Entwickler diese für SPECTRUM essentielle Datenstruktur und eine Vielzahl von Methoden, die auf den VnmParmBlocks arbeiten, zur Verfügung und muß in Anwendungen, die Level–II Toolkits verwenden, eingebracht werden.

- **Inference Handler API**:
  Die Inference Handler API [SPEC IHA] schließlich erlaubt Erweiterungen der *Virtual Network Machine*. Neues prozedurales Wissen kann mit ihrer Hilfe in SPECTRUM integriert werden; so kann zum Beispiel veranlaßt werden, daß bei der Änderung von Attributen, die durch Inference Handler überwacht werden, vom Entwickler spezifizierte Aktionen ausgelöst werden.
  Die Inference Handler API erlaubt die mit Abstand weitestgehenden Eingriffe in SPECTRUM (mit all ihren Konsequenzen) und erfordert vom Entwickler ein detailliertes Verständnis der inneren Zusammenhänge von SPECTRUM, da fehlerhafte Inference Handler in der Regel das Gesamtsystem zu Fall bringen oder zumindest erheblich beeinträchtigen können.

Alle Level–II Toolkits werden in der Sprache C++ [STRO 91] programmiert und erlauben dem Entwickler, die Netzmanagementplattform SPECTRUM in jeder Hinsicht zu erweitern.
In Abhängigkeit der Aufgabenstellung und der erforderlichen Integrationstiefe der Anwendung vollzieht sich die Auswahl der Toolkits; unter Umständen werden alle Application

Programming Interfaces gebraucht. Abbildung 6.6 zeigt die Abhängigkeitsbeziehungen zwischen den SPECTRUM Level–II Toolkits.

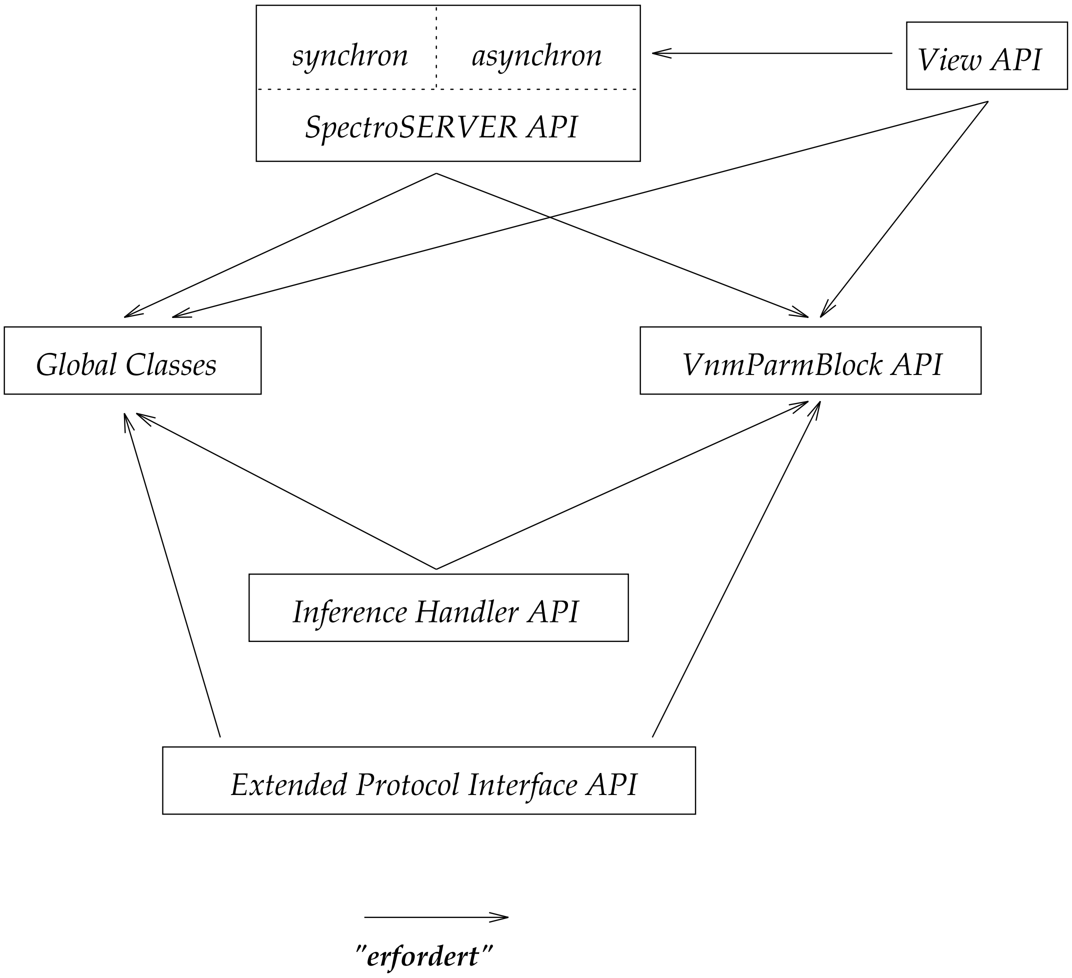

Abbildung 6.6: Abhängigkeitsbeziehungen zwischen den Level–II Toolkits

6.3.2 Inference Handler Application Programming Interface

Im folgenden wird anhand eines Beispiels das Verfahren erläutert, das die Erweiterung der *Virtual Network Machine* um neues prozedurales Wissen zuläßt.

Wie in Abschnitt 3.3 beschrieben, sind Inference Handler Teile von Modelltypen. Ebenso wie das deklarative Wissen (Attribute, Relationen,...) eines Modelltyps kann auch das prozedurale Wissen an davon abgeleitete Modelltypen vererbt werden. Abbildung 6.7 stellt

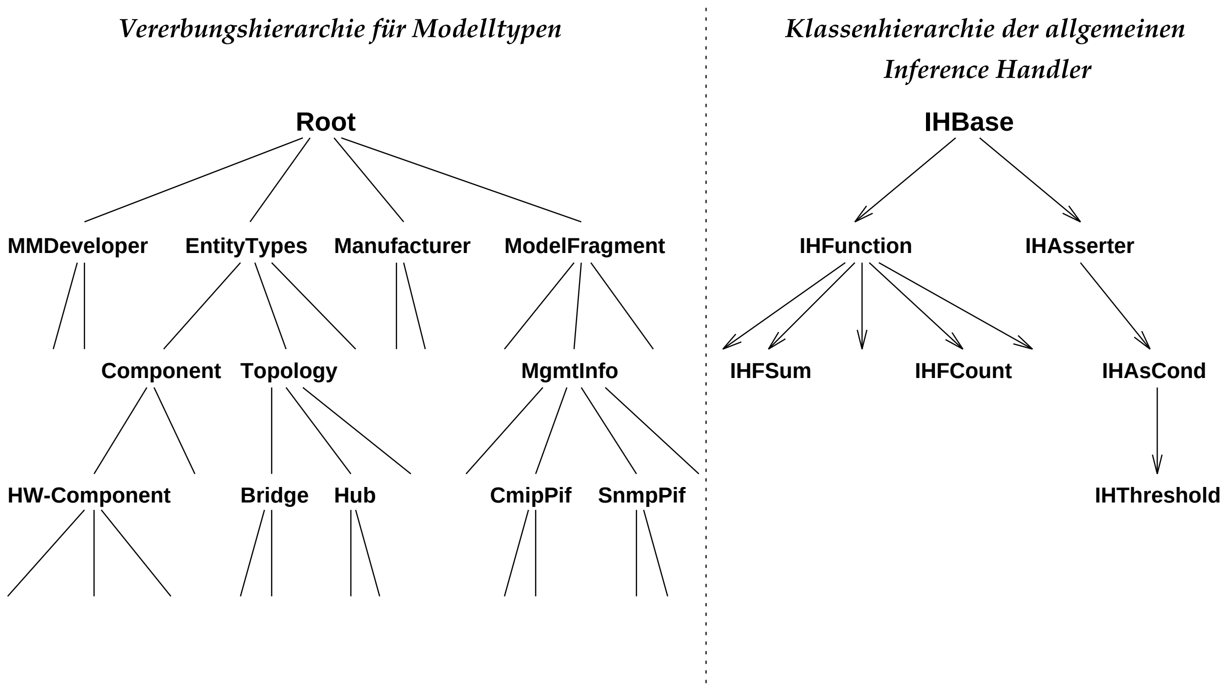

Abbildung 6.7: Vererbung von Modelltypen und Inference Handlern

die Vererbungshierarchie von Modelltypen der Klassenhierarchie der allgemeinen (*general purpose*) Inference Handler gegenüber.
Die allgemeinen Inference Handler stellen dem Entwickler eine Vielzahl von Methoden zur Verfügung, die aufgrund ihrer Vielseitigkeit auf allen Attributen von Modelltypen arbeiten. In erster Linie werden Methoden angeboten, die arithmetische Operationen ermöglichen:

– Summen–, Differenzen– und Quotientenbildung (CsIHFSum, CsIHFSub, CsIHFDiv)

– Berechnung von Durchschnittswerten (CsIHFAvg)

– Zählfunktionen (CsIHFCount)

– Ermitteln von Maxima (CsIHFMax)

Zusätzlich gibt es Methoden zur Schwellwertüberwachung (CsIHThreshold).

Es ist wichtig, sich vor Augen zu führen, daß die allgemeinen Inference Handler nur ein sehr kleiner Teil der insgesamt vorhandenen Inference Handler sind. Sie stellen Hilfsfunktionen dar, die an keine spezifischen Modelltypen gebunden sind und erscheinen in einem

eigenständigen Ast des Vererbungsbaumes.
CsIHBase [3] ist die Basisklasse für *alle*, d.h. sowohl für allgemeine als auch für spezielle, an Modelltypen gebundene, Inference Handler.

Auf den ersten Blick mag es merkwürdig erscheinen, daß die Darstellung der Inference Handler Klassenhierarchie lediglich eine geringe Anzahl umfaßt und Modelltyp–bezogene Inference Handler (d.h. Inference Handler, die für einen konkreten Modelltyp Funktionalität erbringen) nicht erfaßt sind.
Der Grund hierfür ist, daß Cabletron Modelltyp–spezifische Inference Handler grundsätzlich nicht publiziert, da dringend davon abgeraten wird, Vielfachvererbung im Zusammenhang mit Inference Handlern zu verwenden. Daher wird *Multiple Inheritance* auch nicht unterstützt und eine Veröffentlichung der Inference Handler unterbleibt, so daß neue Inference Handler lediglich von `CsIHBase` abgeleitet werden können. Die Erfahrung hat gezeigt, daß Vielfachvererbung, obwohl programmiertechnisch ohne weiteres möglich, äußerst komplexe Seiteneffekte auf Attribute (die unter Umständen von anderen Inference Handlern überwacht werden) zur Folge hat und daher eine häufige, kaum entdeckbare Ursache für schwerwiegende Fehler ist.

Die Folge ist, daß eine der wichtigsten Eigenschaften des objektorientierten Paradigmas, die *Multiple Inheritance*, eingebüßt wird und jeder neu erzeugte Inference Handler auf der gleichen Stufe wie `CsIHFunction` und `CsIHAsserter` in der Klassenhierarchie steht.

In Abbildung 6.7 sind nur die allgemeinen Inference Handler dargestellt, da die Hierarchie aller Inference Handler aus den oben erwähnten Gründen flach und damit sehr breit wäre, da eine Ableitungstiefe, wie sie bei der Vererbungshierarchie für Modelltypen vorkommt, hier nicht existieren darf.

Aufgrund der großen Anzahl an Modelltypen konnte in Abbildung 6.7 lediglich der oberste Teil der Hierarchie berücksichtigt werden.

Die vier von *Root* direkt abgeleiteten Modelltypen stellen einen allgemeinen Beschreibungsrahmen auf, indem die folgenden Fragenkomplexe durch jeweils eine Klasse repräsentiert werden:

– **MMDeveloper**: Wer ist der Entwickler des Modelltyps ?

– **EntityTypes**: Von welcher Art ist der Modelltyp ?

– **ModelFragment**: Was ist darin enthalten ?

– **Manufacturer**: Wie lautet der Hersteller der Komponente, die durch den Modelltyp dargestellt wird ?

Beim Durchlaufen der baumartigen Hierarchie von *Root* abwärts werden die Modelltypen immer konkreter, bis auf der untersten Stufe Modelltypen wie „SNMP–Workstation“ oder „Coax–Segment“ stehen. Hersteller– und produktspezifische Modelltypen können davon ohne großen Aufwand abgeleitet werden.

[3] Cs (mit kleinem „s“) bedeutet „Class“

Von großem Interesse für die vorliegende Arbeit ist die Fragestellung, wie man neue Inference Handler in bestehende Modelltypen einbringen kann.

Das anschließende Beispiel geht von folgendem Szenario aus:
Für einen CABLETRON–Sternkoppler mit der Typenbezeichnung IRM2 soll das Attribut „DeviceCRC“, ein Zähler für MAC–Frames, bei denen ein Prüfsummenfehler festgestellt wurde, überwacht werden und bei einer Änderung dieses Attributs der vorherige sowie der neue Wert am Bildschirm ausgegeben werden.

Zur Erläuterung des Verfahrens sind jeweils nach der Beschreibung der Teilschritte die entsprechenden C++–Codefragmente aufgeführt. Der vollständige Abdruck der für den neuen Inference Handler erforderlichen Quelltextdateien befindet sich in Anhang B.

Zuerst muß man sich Klarheit darüber verschaffen, wie man auf einen Modelltyp und seine Attribute zugreifen kann:
Jeder Modelltyp und jedes seiner Attribute sind eindeutig durch einen *Model Type Handle* bzw. einen *Attribute–Identifier*, beides Hexadezimalziffern, identifizierbar. Im vorliegenden Fall hat der Modelltyp CS_IRM2_HUB (CS {großes „S“} steht hier für „Cabletron Systems“ und *nicht* etwa für „Class“) den Model Type Handle 10059; der Attribute–Identifier für „DeviceCRC“ lautet 10e33.

Diese Werte müssen in einem *Header–File* dem zu entwickelnden Inference Handler bekanntgemacht werden, was durch die nachfolgenden Zeilen geschieht:

```
class CsIRM2Hub
{
        public:
                enum
                {
                  DeviceCRC                   =       0x10e33
                };
};

// Model Type Handle
#define IRM2_TEST_MTYPE         0x10059
```

Der Entwurf des eigentlichen Inference Handlers folgt immer dem nachstehend angegebenen Ablauf:

(1) Jeder Inference Handler muß von CsIHBase abgeleitet werden, damit er mittels Vererbung mit der benötigten Grundmenge von Methoden ausgestattet wird.
Im folgenden Beispiel lautet der neue Inference Handler `CsIHTest` und wird in der Kopfzeile seines Constructors von `CsIHBase` abgeleitet:

```
 CsIHTest::CsIHTest( CsMTypeHandle& in_m_t_handle )
: CsIHBase ( in_m_t_handle )
```

(2) Zuerst muß geprüft werden, ob das zu überwachende Attribut auch tatsächlich im ausgewählten Model Type vorkommt. Im Fehlerfalle wird eine Meldung ausgegeben.

```
CsVnmMTypeHandle *vmth = new CsVnmMTypeHandle( in_m_t_handle );

// Verify that the model type contains the attribute DeviceCRC.

if ( ! vmth->has_attr ( CsIRM2Hub::DeviceCRC ))
{
    eout( "No such attribute: DeviceCRC" << endl );
      set_error( CsError::FAILURE );
}
```

(3) Ist das Attribut Teil des Modelltypen, läßt sich der Inference Handler durch `reg_attr_change` registrieren, um von Änderungen des Attributes zu erfahren. Scheitert die Registrierung, wird eine Fehlermeldung ausgegeben.

```
    // Register to receive changes in DeviceCRC

    if (!reg_attr_change( CsIRM2Hub::DeviceCRC ))
{
   eout( "Couldn't register for changes in DeviceCRC." << endl);
   eout( "m_t_handle = " << hex << m_t_handle << dec << endl );
   set_error( CsError::FAILURE );
   return;
}
else
{
   tout( "Registered for CsIRM2Hub::DeviceCRC change" << endl);
}
```

Der Registrierungsmechanismus zeichnet die Inference Handler API gegenüber allen anderen SPECTRUM–Programmierschnittstellen aus:
Mit seiner Hilfe kann man ohne großen Programmieraufwand sicherstellen, daß während der gesamten Laufzeit des Inference Handlers (bzw. solange, bis man im Programmtext explizit die Registrierung aufhebt) Änderungen des überwachten Datenobjekts automatisch von der VNM [4] an den Inference Handler gemeldet werden. Anders ausgedrückt, legt die Registrierungsmethode, die vom Inference Handler aufgerufen wird, fest, welche Änderungen innerhalb des Netzmodells der VNM die Ausführung des Inference Handlers veranlassen [SPEC IHC].

[4] präzise ausgedrückt, wird diese Tätigkeit vom *Notification Manager*, einem *Thread* der VNM, erledigt.

(4) Eng damit verbunden ist der *Trigger*–Mechanismus:
Hiermit kann vom Entwickler veranlaßt werden, daß eine vordefinierte Funktion bei einer bestimmten Konstellation (im hier behandelten Fall: bei der Änderung des Attributes DeviceCRC) getriggert wird.

Trigger–Funktionen müssen *außerhalb* des Constructors als eigenständige Methoden implementiert sein. Obwohl alle Trigger–Methoden bereits in `CsIHBase` als *virtual* definiert sind und mittels Vererbung auch dem Inference Handler zur Verfügung stehen, müssen sie vom Entwickler unbedingt neu definiert werden, da ihr Rumpf leer ist. Ihr Wesen entspricht daher dem rein virtueller (*pure virtual*) Methoden.

Die folgende Methode `trig_attr_change` wird vom *Notification Manager* der VNM angestoßen, falls das Attribut `DeviceCRC` sich ändert. Daraufhin werden der aktuelle und der vorherige Wert gelesen und anschließend ausgegeben. Dies geschieht durch direktes Abstützen auf die Funktionalität von `CsChangeNode`; im Change Node sind der aktuelle und der vorherige Wert eines Attributes zusammen mit der Attribute ID gespeichert.

```
void CsIHTest::trig_attr_change( const CsModelHandle& mh,
                  const CsChangeNode * change )
{
        tout("CsIHTest::trig_attr_change() mh=("<< hex << mh << dec <<
        ") activated." << endl );

        int errors = * (int *)(change->get_cur_value());
        int prev_errors = * (int *)(change->get_prev_value());

        // if DeviceCRC has not changed, do nothing

        if (errors != prev_errors)
                {
            switch( change->get_attr_id())
                        {
                        case CsIRM2Hub::DeviceCRC:
                                {
                                 iout("CsIHTest:trig_attr_change()
                     mh=(" << hex << mh << dec << ") Errors changed
                     from "<< prev_errors << " to " << errors << endl);
                                 break;
                                }
                        default:
                                {
                                eout( "Model with mh=(" << hex << mh
                                 << dec << ") is only registered for
```

```
                                DeviceCRC " << endl );
                               }
                       }
               }
        tout( "CsIHTest::trig_attr_change() ending."  << endl );
}
```

Damit ist die Implementierung des Kernstücks des Szenarios, der Inference Handler `CsIHTest`, abgeschlossen.
Abblauffähig ist er allerdings noch nicht, da `CsIHTest` noch nicht in den Modelltyp IRM2_TEST_MTYPE integriert ist.

(5) Um den Inference Handler instantiieren zu können, muß innerhalb des Model Intelligence Node, dessen Funktionsweise weiter unten erläutert wird, folgende Funktion definiert werden:

```
static void CsIRM2HubMI( CsMTypeHandle& m_t_handle )
{
    // Attach CsIHTest IH to IRM2Hub Model Type
    // and if unsuccessful, unattach CsIHTest IH.

    CsIHTest * test_ih = new CsIHTest( m_t_handle ) ;
    if ( test_ih->get_error() == CsError::FAILURE )
    {
        CsDelete( test_ih );
    }
}
```

Zu beachten ist, daß ein Inference Handler gelöscht werden muß, sobald ein Fehler bei seiner Integration in den Modelltyp auftritt. Das darf jedoch *ausschließlich* in der Funktion des MI–Nodes geschehen; ein Löschen des Inference Handlers an einer anderen Stelle (zum Beispiel in einer Trigger–Methode des Inference Handlers) führt zu einem Systemabsturz der VNM (*coredump*) und impliziert den Verlust der Datenbank.
Entsprechende Vorsichtsmaßnahmen (Sicherung der Datenbank, Anlegen einer Kopie des SpectroSERVER) sind daher unbedingt einzuhalten.

(6) Schließlich ist ein global definierter *Model Intelligence Node* (MI–Node) zu erzeugen, um den neuen Inference Handler in den existierenden Modelltyp zu integrieren. Im Beispiel lautet der neue Inference Handler CsIHTest, der Modelltyp IRM2_TEST_MTYPE.
Ein MI–Node dafür lautet:

```
static CsMINode mi_node ( IRM2_TEST_MTYPE, CsIRM2HubMI ) ;
```

Um Namenskonflikte zu vermeiden, sollte der MI–Node die Speicherklasse „static“ haben.

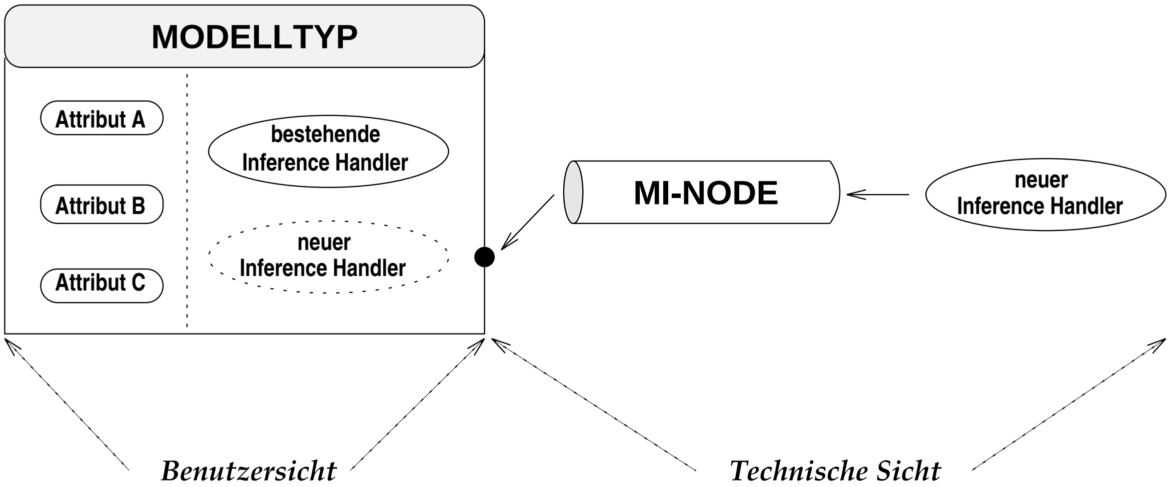

Abbildung 6.8: Erweiterung eines Modelltypen um neues prozedurales Wissen

Abbildung 6.8 zeigt die technische Sichtweise d.h. die Rolle des MI–Nodes als Bindeglied zwischen dem neuen Inference Handler und dem Modelltyp sowie die Benutzersicht. Für den Benutzer erscheint das Einbringen des Inference Handlers transparent, er kann nicht feststellen, ob der Inference Handler bereits Bestandteil des Modelltyps war oder erst im nachhinein integriert wurde.

Ähnlich verhält es sich mit Modelltypen, die von dem erweiterten Modelltyp abgeleitet werden: Der neue Inference Handler erscheint nun als deren integraler Bestandteil.

(7) Zuletzt wird der Inference Handler compiliert und zusammen mit den Quelldateien für die VNM gelinkt, um eine neue *Virtual Network Machine* zu generieren.
Das *Makefile* [ORAM 91] ist zusammen mit anderen Konfigurationsdateien in Anhang A abgedruckt.

Dem Verständnis der in dieser Arbeit verwendeten C++–Ausgabebefehle dient die folgende Anmerkung:
Gerade in der Testphase eines Inference Handlers ist es sehr wichtig, über jeden Schritt informiert zu werden, der von dem Inference Handler vollzogen wird. Im obenstehenden (und insbesondere in dem in den Anhängen abgedruckten) Programmcode kommen daher Begriffe wie „iout", „tout" oder „eout" vor; dies sind einfache Makros, die von Cabletron speziell zum Testen von Inference Handlern entwickelt wurden. Sie ersetzen den in C++ gebräuchlichen *iostream*–Befehl `cout` und können bedarfsspezifisch verwandt werden:

– Sollen Fehlermeldungen ausgegeben werden, kommt `eout` (Error-Output) in Frage

– Wird die Ausgabe von Trace–Informationen gewünscht, sollte `tout` im Quelltext vorkommen

– Warnungen werden mit `wout` auf die Standardausgabe umgelenkt

– Ist beabsichtigt, Informationen über die interne Ausführung einer Methode auszugeben, steht `iout` zur Verfügung

– Detaillierte Debug–Information bezüglich der Ausführung einer Funktion ist mit `dout` erhältlich

Zur Laufzeit des fertig compilierten und gebundenen Programms werden, entsprechend der auf der Kommandozeile angegebenen Optionen, die jeweiligen Ausgabeoperationen ausgeführt bzw. unterdrückt. Der Entwickler kann also eine Selektion der Ausgaben vornehmen, die mit `cout` nicht möglich wäre.
Zu beachten ist, daß dieser Ausgabemechanismus nur für selbst geschriebene Funktionen gilt und im Inference Handler mit `define MY_DEBUG 1` das Debug–Flag gesetzt werden muß; Hoffnungen, durch Angabe entsprechender Optionen nachträglich Informationen über das „Innenleben" der VNM zur Laufzeit zu erhalten, erfüllen sich nicht, da Cabletron diese Makros bei der Entwicklung der VNM nicht aktiviert hat.

Informationen zur Syntax der Makros, die leicht von der des `cout`–Befehls abweicht, sowie detaillierte Ausführungen zu den Debug–Makros findet man in Anhang B von [SPEC IHA] sowie in der Datei `Spectrum/GLOBL/include/CsDebug.h`.

Die folgende Abbildung 6.9 faßt die Schritte, die zur Instantiierung eines neuen Inference Handlers erforderlich sind, zusammen und skizziert die Interaktionen zwischen dem Inference Handler, dem Modell und dem *Notification Manager* innerhalb der *Virtual Network Machine*.

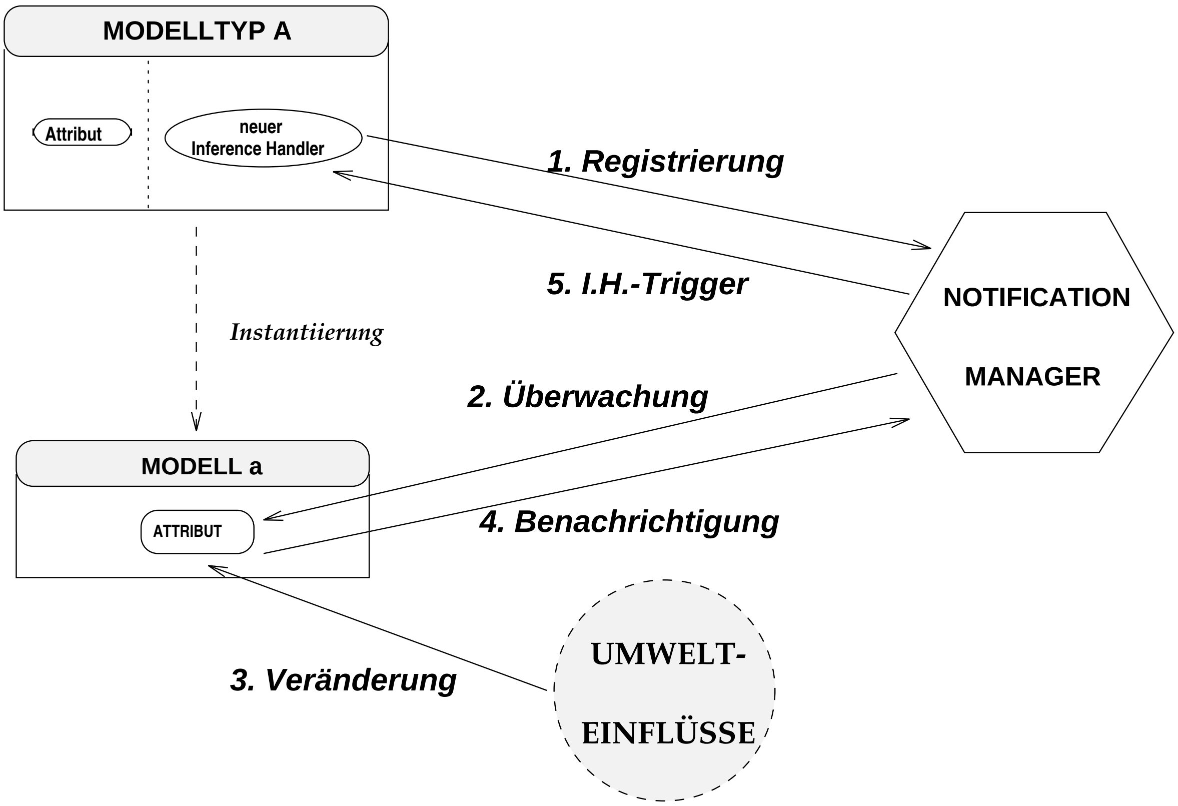

Abbildung 6.9: Verhalten eines Inference Handlers zur Laufzeit

6.4 Graphische Aufbereitung der Managementdaten

Wie in Kapitel 5 bereits angedeutet wurde, steht und fällt die Akzeptanz eines Managementsystems mit der klaren und anschaulichen Darstellung der zahlreichen Daten auf der Managementstation.

Die Netzmanagementplattform SPECTRUM bietet dem Entwickler auch in dieser Hinsicht gute Möglichkeiten, Erweiterungen vorzunehmen:
Bereits ohne Programmierkenntnisse kann ein Benutzer Sichten auf ein zu managendes System erstellen, die beispielsweise die Gesichtspunkte Topologie, Organisation und räumliche Lage umfassen.
Für jede Hard– und Softwarekomponente eines Systems können *Icons* definiert werden, die, in den einzelnen Sichten plaziert, Aufschluß über den momentanen Betriebszustand der Komponente geben. Beliebige Attribute eines Modelltyps lassen sich problemlos in eine Sicht integrieren und mit anderen Attributen kombinieren.

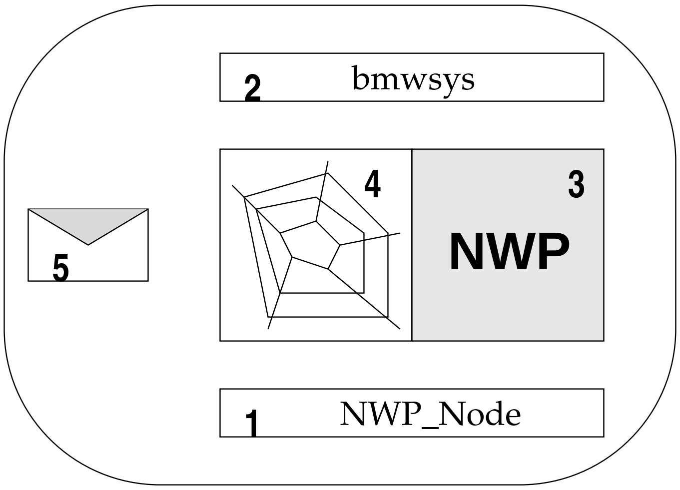

Abbildung 6.10: Graphische Darstellung des Managementobjekts „NWP–Knoten“

Grundvoraussetzung für den Zugriff auf die für das NWP–Management erforderlichen Sichten

– Konfiguration

– Anwendung

– Organisation

– Diagnose

ist die Definition eines *Icons* für das Managementobjekt `NWP_Node` in der *Topology View*. Einen Realisierungsvorschlag dafür zeigt Abbildung 6.10.

Die numerierten Felder des *Icons* haben folgende Bedeutung:

(1) Hier steht der Name des Modelltyps (im Beispiel: `NWP_Node`). Durch doppeltes „Anklicken“ gelangt man in die *Application View*.

(2) In diesem Feld ist der Schicht–7–Name des NWP–Knotens eingetragen. Ein „Anklicken“ bewirkt das Öffnen der *Organisation View*.

(3) Dieser Bereich ist Diagnoseinformationen vorbehalten. Die Farbe des Hintergrundes zeigt den gegenwärtigen Zustand des Knotens an (zum Beispiel bedeutet grün: „in Ordnung“ und rot: „schwerer Fehler aufgetreten“).
Entsprechend wird bei der Selektion diese Bereiches mit dem Mauszeiger die *Diagnostic View* geöffnet.

(4) Das (Spinnen–) Netz, BMW–internes Symbol für das NetzWerk–Programm, ermöglicht dem Benutzer die Eröffnung einer TELNET–Sitzung auf dem betreffenden NWP–Knoten.

(5) Mit einem Mausklick gelangt man in die *Configuration View* des Knotens.

Nachstehend werden die Inhalte der einzelnen Views erläutert.

Die *Application View* enthält dynamische Informationen bezüglich NWP sowie Funktionen, die Eingriffe in den NWP–Betrieb erlauben:

– Durch Auswahl des Parameters `laufende Transaktionen` erscheint eine Auflistung der Transaktionscodes aller derzeit noch nicht abgeschlossenen Transaktionen, die über den Knoten laufen.

– Analog dazu können vor weniger als vier Tagen beendete Transaktionen abgefragt werden. Die zeitliche Beschränkung rührt daher, daß eine CW–Datei vier Tage nach Abschluß der Transaktion gelöscht wird (vgl. hierzu Teilabschnitt 4.1.2).

– Die Option `Prüfe Transaktion` öffnet eine Dialogbox, in die der Benutzer den Code sowie den Ursprungsknoten einer Transaktion eingeben kann und als Ergebnis die Knotensequenz tabellarisch angezeigt bekommt, die die Daten während der Übertragung genommen haben.
Die Prozedur, die diese Funktionalität erbringt, prüft, ob an den Knoten, die aus der Wegedatei ermittelt wurden, CW–Dateien mit dem vorgegebenen Transaktionscode existieren d.h. es ist möglich, den Übertragungsvorgang einer Datei zu überwachen.

– Log– und Systemdateien können durch Auswahl der Felder `Lies crontab`, `Lies NIDAT`, `Lies Wegedatei` und `Lies Log-Information` angezeigt werden.

– Als Ergänzung des vorherigen Punktes kann man Editiermöglichkeiten für crontab, NIDAT und Wegedatei vorsehen.

– Die Anzeige der zum gegenwärtigen Zeitpunkt auf dem NWP–Knoten ablaufenden Prozesse ist durch Wahl des Unterpunktes `Prozeßtabelle` möglich.

Die *Organisation View* vereinigt alle Informationen, die das organisatorische Umfeld eines NWP–Knotens bilden. Hierzu zählen Angaben bezüglich des Standortes, der Abteilung und der Benutzergruppen (sogenannte *logische Netze*) eines NWP–Knotens.

In der *Diagnostic View* werden sämtliche Parameter zusammengefaßt, die Einfluß auf den Status eines NWP–Knotens haben.
Insbesondere können hier die *Event*– und *Alarmlogs* abgerufen werden, die eine wertvolle Hilfe zur Fehleranalyse sind.

Die Konfiguration eines NWP–Knotens, und zwar sowohl die der Hard– als auch der Softwarekomponenten wird in der *Configuration View* dargestellt.

Sie beinhaltet beispielsweise die Versionsnummer des NWP Control Program, die Namen der Sende– und Empfangsserver oder Angaben zur Knotenhardware.
Man kann sich leicht vorstellen, daß zusammen mit den Daten der Kommunikationsinfrastruktur des Knotens eine sehr große Menge an Konfigurationsinformation anfällt.
Eine Lösung zur Strukturierung dieser Daten gemäß dem ISO–OSI–Referenzmodell bestünde darin, dem Pfeilsymbol eine neue Semantik zu geben:
Mit jedem „Anklicken“ dieses Symbols gelangt man in eine gemäß OSI „niedrigere“ Schicht d.h. befindet man sich in einer Schicht–7–View, bewirkt eine Selektion des Symbols das Öffnen einer Schicht–6–View. Auf der OSI–Schicht 1 angekommen, würde durch erneute Auswahl des *Icons* eine neue Schicht–7–View geöffnet.

Die in diesem Abschnitt genannten Beispiele zur Erweiterung dieser Arbeit um graphische Darstellungsmöglichkeiten zeigen, daß sich die gesamte NWP–Managementfunktionalität nahtlos in die von SPECTRUM zur Verfügung gestellte graphische Umgebung einfügen läßt. Außer der *Application View* existieren bereits alle obengenannten Sichten (in mehr oder weniger ähnlicher Funktion) d.h. es müßte ohne Programmieraufwand möglich sein, dem NWP–Management das *look–and–feel* einer OSF/Motif–Benutzerschnittstelle in Vereinigung mit der Managementfunktionalität von SPECTRUM zu geben.
Die *Application View* schließlich, das Rückgrat des NWP–Managements, läßt sich mit Hilfe der View API in Verbindung mit der asynchronen SpectroSERVER API in SPECTRUM integrieren.

Kapitel 7

Schlußbemerkung und Ausblick

Diese abschließende Kapitel faßt die Ausgangssituation und die Aktivitäten, die zum Entwurf und zur Implementierung der Managementszenarien erforderlich waren, kurz zusammen und beschreibt, in welcher Hinsicht Möglichkeiten der Erweiterung der in der vorliegenden Arbeit behandelten Thematik bestehen.

Zu Beginn wurde analysiert, weshalb allgemein ein Management verteilter Anwendungen notwendig ist; im Brennpunkt der Untersuchungen stand fortan die verteilte Kommunikationsanwendung NWP.
Verschiedene Ansätze für das Management verteilter Anwendungen wurden anschließend beschrieben und nach Kriterien der praktischen Anwendbarkeit bei der Erstellung eines Managementkonzeptes bewertet.

Da das Aufstellen eines tragfähigen Managementkonzeptes Detailkenntnisse des zu managenden Systems erfordert, wurden in Kapitel 4 die technischen Eigenschaften des NetzWerk–Programms eingehend geschildert sowie die derzeit verfügbaren Möglichkeiten, NWP–Managementinformationen zu akquirieren.

Für die Entwicklung der Managementszenarien und die Akzeptanz der aufzustellenden Managementlösung war es von grundlegender Bedeutung, sich Klarheit darüber zu verschaffen, welche Probleme beim Betrieb von NWP auftreten und welche Anforderungen die davon hauptsächlich betroffenen Benutzergruppen an ein NWP–Management richten. In mehrstündigen Diskussionen und Gesprächen mit dem Personal des Netzwerkleitstands sowie der NWP–Gruppe konnten wertvolle Informationen gewonnen werden, die prägenden Einfluß auf den Entwurf der Managementszenarien hatten.

Schließlich stand die technische Realisierung der Szenarien d.h. deren prototypische Implementierung auf der kommerziellen Netzmanagementplattform SPECTRUM im Blickpunkt, die aufgrund des hohen Maßes an Komplexität den zeitlich bei weitem größten Anteil beanspruchte.
Als Ausgangsbasis für die Implementierung wurden das *Inference Handler Application Programming Interface*, da es als einziges Toolkit die für die vorliegende Arbeit dringend benötigte Funktionalität bot, und das Managementmodul *MaestroVision* gewählt, bei

deren Benutzung jedoch ungeahnte Schwierigkeiten auftraten:

– Von MaestroVision lag bis zuletzt lediglich eine Demonstrations–Version vor, der, bis auf eine Produktankündigung im Umfang weniger Seiten, keinerlei Dokumentation beilag. Obgleich unmittelbar nach dem Eintreffen der Demo–Version Bemühungen eingeleitet wurden, eine vollständige Lizenz (und damit adäquate Dokumentations– und Supportmöglichkeiten) zu erhalten, gelang dies binnen 14 Wochen nicht.

  Die Folge war, daß sämtliche Informationen bezüglich MaestroVision mühsam in langwierigen Terminalsitzungen herausgefunden werden mußten; letztendlich war jedoch der Modellierung von NWP in SPECTRUM Erfolg beschieden.

– Die Probleme mit der SPECTRUM Inference Handler API lagen in einem anderen Bereich:
  Aufgrund der Tatsache, daß es sich bei der Inference Handler API um ein neues Produkt handelte, für das bis zum Beginn dieser Arbeit noch keine Erfahrungen vorlagen, geriet die vorliegende Arbeit unter anderem zu einer Nagelprobe der Stabilität der Inference Handler API.

  Das Ergebnis war unbefriedigend, da Probleme beim Link–Vorgang des SpectroSERVERs auftraten und es trotz intensiven Nachrichtenaustauschs während 12 Wochen mit Cabletron England nicht gelang, den Fehler zu beheben.
  Als Fehlerursache wurden uns mutmaßliche Inkonsistenzen zwischen den Compilerversionen genannt, da der (von Cabletron) zum Erstellen der C++-Libraries verwendete Compiler eine andere Version aufwies als derjenige, der uns vorlag (und der laut Cabletron zur Entwicklung verwendet werden sollte).
  Die Marktreife einer neuen SPECTRUM–Version, die für Februar 1993 angekündigt war und bei der der erwähnte Fehler (angeblich) nicht mehr aufgetreten wäre, wurde inzwischen auf August 1993 verschoben (zu spät, um für die vorliegende Arbeit noch relevant zu sein).

  Die Support– und Versionenproblematik—eine unendliche Geschichte...
  Das Münchner Netzmanagement Team hat die Notwendigkeit eines Kooperationsabkommens mit Cabletron erkannt und entsprechende Schritte hierzu eingeleitet, um in Zukunft sicherstellen zu können, daß aufgetretene Fehler in einer akzeptablen Zeitspanne behoben werden.

Wegen der obengenannten Gründe konnten die implementierten Szenarien daher nicht in der Praxis erprobt werden. Nichtsdestoweniger lassen die erarbeiteten Lösungen eine Beurteilung der Möglichkeiten eines NWP–Managements zu.

Obwohl sich sie entwickelten Managementszenarien primär auf UNIX–Systeme beziehen, lassen sie sich größtenteils auch auf DEC– oder SNA–Systeme, die als NWP–Knoten fungieren, übertragen.
Anpassungen müssen insbesondere an Großrechner innerhalb der SNA–Welt erfolgen, da bereits deren Hardwarevoraussetzungen von denen herkömmlicher Workstations differieren.

In diesem Zusammenhang leistet das Managementmodul *BlueVision* gute Dienste, da es SPECTRUM um die *NetView/6000*–Funktionalität erweitert und somit erlaubt, SNA Hard– und Softwarekomponenten zu managen.
Für das NWP–Management essentielle Modelltypen wie „Application“ oder „Process“ sind dadurch verfügbar.

Gelingt es, ein Managementmodul für die DECnet–Welt mit ähnlich gelagerter Funktionalität zu entwickeln, steht einer umfassenden, integrierten Managementlösung für das NetzWerk–Programm nichts mehr im Wege, da die drei großen Systemwelten UNIX, SNA und DEC in eine einheitliche Managementplattform eingebettet sind.

Eine weitere wichtige Frage, die in Abschnitt 4.3 angesprochen wurde, bezieht sich auf die Anbindung von relationalen Datenbanksystemen an SPECTRUM.
Im konkreten Fall NWP bedeutet dies die Integration der (bis jetzt) statischen Netzdatenbank ISYKON. Gelingt es, die Integration unter Beachtung der Konsistenz– und Integritätsbedingungen zu vollziehen d.h. ist es möglich, auf die SPECTRUM–Daten mit Hilfe der Abfragesprache SQL und auf ISYKON–Daten mit dem Model Type Editor zuzugreifen, ist die Erweiterung von ISYKON um dynamische Daten nur noch eine Frage der Zeit.
Im Endausbau wäre ISYKON demnach eine dynamische, NWP–spezifische Wissensbasis, die neben laufend aktualisierten Hardwarebeschreibungen auch momentan ausgefallene Knoten und zum Abfragezeitpunkt gültige Routen beinhaltet. Dies impliziert dynamisches Routing.

Mit der Integration von immer zahlreicheren Komponenten, Systemen und Anwendungen steigt jedoch auch die Anzahl der anfallenden Daten. Hierzu sind zwei Anmerkungen zu machen:

(1) Die Netzlast, verursacht durch das Netzmanagement, erhöht sich durch die Verfügbarkeit immer größerer Datenmengen, selbst bei Verwendung intelligenter Agenten.
Es stellt sich zwangsläufig die Frage, ob es Sinn macht, daß alle Systeme von einer zentralen Managementstation verwaltet werden, oder ob es nicht besser ist, eine hierarchische Managementarchitektur einzusetzen; auf SPECTRUM angewandt, hieße das, die *Virtual Network Machine* verteilt zu betreiben d.h. an ausgezeichneten Punkten des Netzes bereits frühzeitig unterschwellige Alarme zu verarbeiten, bevor diese zu ernsthaften Problemen für den Betrieb des gesamten Netzes werden.
Tatsächlich sind bereits heute intelligente Sternkoppler, sogenannte *Multi Media Access Center*, erhältlich, die eine auf einem Einschub untergebrachte vollwertige Workstation aufnehmen können. Es ist durchaus vorstellbar, daß eine auf das jeweils zu überwachende Netzsegment zugeschnittene VNM auf dem Workstation–Board innerhalb eines Hubs Managementinformation vorverarbeitet.

(2) Die derzeit unter SPECTRUM verfügbaren Sichten reichen im Falle großer Datenvolumina mit Sicherheit nicht aus, um alle benötigten Attribute darzustellen:
Momentan kann man sich damit behelfen, einzelne besonders wichtige Attribute in

eine Sicht zu integrieren; dies geht jedoch nur, solange nur eine beschränkte Anzahl von Netzdaten verfügbar ist.
Im Zuge weitergehender Integrationsbemühungen gelangt man jedoch bald an eine kritische Grenze, wobei die Daten aufgrund ihrer Fülle nicht mehr in übersichtlicher Form darstellbar sind.

Hier könnte eine Einteilung der Views anhand der Systematik des ISO–OSI–Referenzmodells erfolgversprechend sein.
Eine Realisierung der *OSI–Views* für einen NWP–Knoten würde gestatten, schichtenspezifische Analysen des Kommunikationssystems zu ermöglichen.
Der konkrete Fall eines „abgestürzten" NWP–Prozesses, ohne daß ein Fehler in den darunterliegenden Schichten aufgetreten ist, ließe sich problemlos diagnostizieren, da die graphische Darstellung der Schicht, in der der Prozeß sich befindet, eine andere Farbe annimmt, während die Farbe der *Icon*–Bestandteile, die die darunterliegenden Schichten repräsentieren, unverändert bliebe.

Im wesentlichen lassen sich die Erkenntnisse für das Management der verteilten Kommunikationsanwendung NWP auch auf verteilte Anwendungen im allgemeinen Sinne übertragen: Im Zuge der Datenintegration sind Fragestellungen wie die Anbindung relationaler Datenbanksysteme an Anwendungen von großem Interesse; die Überwachung des Zustands einer verteilten Anwendung sowie das Lesen und Modifizieren von Konfigurationsinformation und Systemdateien hat auch in der „Nicht–NWP–Welt" seine Berechtigung. Die entwickelten Managementszenarien lassen sich prinzipiell, bis auf wenige Ausnahmen, nicht nur bei Filetransfersystemen, sondern auch bei diversen anderen verteilten Anwendungen einsetzen.

Insgesamt wurde eine Basis geschaffen, die unter Verwendung der Programmiersprache C++ und der Inductive Modeling Technology eine flexible Weiterentwicklung im Hinblick auf eine integrierte Managementlösung für das NetzWerk–Programm erlaubt. Die Erstellung einer graphischen Benutzeroberfläche, die die beschriebenen Sichten beinhaltet, die vollständige Integration vorhandener Datenbestände sowie die Ausdehnung des Managementkonzepts auf weitere Systemwelten sollten die Schwerpunkte bei der Weiterentwicklung der in der vorliegenden Arbeit entworfenen Konzepte bilden.

Anhang A

Konfigurationsdateien

Nachstehend sind die Dateien `make.defs` und `Makefile` abgedruckt, die für die Generierung eines um neues prozedurales Wissen erweiterten SpectroSERVERs maßgeblich sind. `make.defs` beinhaltet unter anderem Suffix–Regeln und Definitionen, auf die das `Makefile` angewiesen ist. Der Inference Handler, der hiermit in die VNM integriert wird, wurde in Abschnitt 6.3 beschrieben und ist im folgenden Kapitel abgedruckt.

A.1 make.defs

```
########################################################################
##
##    Entwurf und Implementierung von Managementszenarien zu verteilten
##    Kommunikationsanwendungen
##    Diplomarbeit
##    Technische Universitaet Muenchen
##
##    Workfile:             make.defs
##    Directory:            /proj/Spectrum3/IHAPI
##    Original Author:      Jeffrey J. Rodgers
##    Adapted by:           Alexander Keller
##    Date:                 11/3/93
##
########################################################################

CPP = /lib/cpp -P $(INCLUDES)

CC = cc
C++ = /mnt/sd1h/proj/C++/CC
C++STDLIB = -lC
```

```
NOSHAREDLIBS = -Bstatic

SYSCFLAGS = -DCS_DEBUG -DBSD

LD = ld
LDFLAGS =

AR = ar
ARFLAGS = rv

RANLIB = ranlib

MAKEDEPFLAGS = -I /proj/C++/SC1.0/include/CC

LN = ln -s

SYSLIBS =

RM = rm -f

##
##  Standard Targets:
##
##  local - compile everything in the current directory
##
##  link - link all executables in the current directory, then recurse down
##
##  relib - force .o's into libraries, then recurse.
##
##  depend - generate dependencies for current directory, then recurse down
##
##  all - does ``init'', ``depend'', ``down'', and ``link''
##
##  init - initialize scripts, and other once-only stuff.
##
##  down - recurses down, then does ``local''
##
##  up - does ``local'', and recurses up
##
##  default - does ``down'' and ``link'' (default when you just type "make")
##
##
##  The targets init, link, and depend have *_here equivalents that exist
```

```
##  in the Makefile of each directory.  The local target is also defined
##  in Makefile.  The rest are defined in make.defs.
##
##
##  Standard symbols:
##
##  Defined by each Makefile as needed:
##
##  TOP - defined by buildmake, used by make.defs to recurse
##  INCLUDES - used by default .cc->.o and .c->.o rules
##  LIBS - use for libraries to use
##  CFLAGS - should be used for linking as well.
##
##  Available for user to set in environment:
##
##  XCFLAGS - used when compiling .c and .cc files, settable by the user.
##    Executable links should use this as well.
##
##

default : down link

all : init depend down link

init : init_here

down : _down2 local

_down2 :

up : local
@if [ ! -f make.defs.init ]; then \
  echo "cd ..; $(MAKE) up" ; \
  cd .. ; $(MAKE) up; fi

link : link_here

depend : depend_here

clean :
-$(RM) *~ *.bak

.SUFFIXES :
```

```
.SUFFIXES : .o .c .cc .cxx

.c.o :
$(CC) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c $<

.cc.o :
$(C++) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c $<

.cxx.o :
$(C++) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c $<

# BUILDLIBRULE2 is used when one source file builds two .o's
# via differend #defines.

.c.a :
$(CC) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c $<
$(BUILDLIBRULE2)

.cc.a :
$(C++) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c $<
$(BUILDLIBRULE2)

.cxx.a :
$(C++) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c $<
$(BUILDLIBRULE2)

.o.a :
$(BUILDLIBRULE2)

CCOM = $(CC) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c
C++COM = $(C++) $(SYSCFLAGS) $(XCFLAGS) $(CFLAGS) $(INCLUDES) -c
CLINK = $(CC) $(XCFLAGS) $(CFLAGS)
C++LINK = $(C++) $(XCFLAGS) $(CFLAGS)

MAKEDEPEND = $(MAKEDEPEND_PROD)/makedepend -- $(CFLAGS) \
      $(XCFLAGS) $(SYSCFLAGS) -- $(MAKEDEPFLAGS) -i/usr/include

ENDOFLIST=

IHAPI_INC = $(TOP)/include
IHAPI_LIB = $(TOP)/lib
IHAPI_PROD = $(TOP)/./prod
VPAPI_INC = $(TOP)/../VPAPI/include
```

```
VPAPI_LIB = $(TOP)/../VPAPI/lib
VPAPI_PROD = $(TOP)/../VPAPI/./prod
GLOBL_INC = $(TOP)/../GLOBL/include
GLOBL_LIB = $(TOP)/../GLOBL/lib
GLOBL_PROD = $(TOP)/../GLOBL/./prod
```

A.2 Makefile

```
########################################################################
##
##    Cabletron Systems Incorporated
##    Post Office Box 5005
##    Rochester, NH 03867-5005
##
##    Entwurf und Implementierung von Managementszenarien zu verteilten
##    Kommunikationsanwendungen
##    Diplomarbeit
##    Technische Universitaet Muenchen
##
##    Workfile:             Makefile
##    Directory:            /proj/Spectrum3/IHAPI/nwp
##    Original Author:      Jeffrey J. Rodgers
##    Adapted by:           Alexander Keller
##    Date:                 11/3/93
##
########################################################################

.KEEP_STATE:

.PRECIOUS: IHapidemo.a

TOP = ../..

include $(TOP)/IHAPI/make.defs

all debug browse : DemoSS
@$(RM) $(SS_OBJ)
@$(RM) $(MI_NODES)
@echo DemoSS build complete

##  XCFLAGS - used when compiling .c and .cc files, user settable
```

```
#debug := XCFLAGS = -g
#browse := XCFLAGS = -sb -g

CCFLAGS = -DCS_DEBUG $(XCFLAGS)

CPPFLAGS = $(INCLUDES)

LD_OPTIONS = -dp -dc -e start

LDFLAGS = -lm -lc /proj/C++/SC1.0/libC.a
#debug browse := LDFLAGS = -lg -lm -lc

IHAPI_INC = $(TOP)/IHAPI/include
VPAPI_INC = $(TOP)/VPAPI/include
GLOBL_INC = $(TOP)/GLOBL/include

TARGET_ARCH =

INCLUDES = \
        -I$(IHAPI_INC) \
        -I$(GLOBL_INC) \
        -I$(VPAPI_INC)

TK_LIBS = \
$(SS_DIR)/libEPapi.a \
$(SS_DIR)/libIHapi.a \
$(SS_DIR)/libVPapi.a \
$(SS_DIR)/libVWapi.a \
$(SS_DIR)/libGlobl.a \
$(SS_DIR)/libPort.a

## Spectrum object files, explicitly linked

SS_O = $(SS_DIR)/SS.o
SS_OBJ:sh = ar t ${SS_DIR}/ss_obj.a | grep -v SYMDEF ; ar x ${SS_DIR}/ss_obj.a

## mi_node object files must be explicitly linked in

MI_NODES:sh = ar t ${SS_DIR}/mi.a  | grep -v SYMDEF ; ar x ${SS_DIR}/mi.a

DEMO_MI = \
        CsTestMI.o \
$(ENDOFLIST)
```

```
IHFILES = \
CsIHTest.o \
$(ENDOFLIST)

.INIT:\
CsIRM2Hub.h \
CsIHTest.h \
$(ENDOFLIST)

DemoSS: $(IHFILES) $(SS_O) $(SS_DIR)/ss_obj.a $(SS_DIR)/mi.a $(DEMO_MI) \
    $(SS_DIR)/ih.a $(TK_LIBS)
ranlib -t $(TK_LIBS)
$(LD) $(LD_OPTIONS) /lib/crt0.o $(SS_O) $(SS_OBJ) $(MI_NODES) \
    $(DEMO_MI) $(SS_DIR)/ih.a \
    $(TK_LIBS) -o $@ $(LDFLAGS)
@$(RM) $(SS_OBJ)
@$(RM) $(MI_NODES)
/proj/Spectrum3/SS/patch $@
-chmod u+s $@

IHapidemo.a : IHapidemo.a($(IHFILES))

database:
cd Database; $(MAKE)

clean:
cd Database; $(MAKE) clean
sccs clean
@$(RM) *.o IHapidemo.a

.DONE:
@$(RM) $(SS_OBJ) __.SYMDEF
@$(RM) $(MI_NODES)
```

Anhang B

Inference Handler Programmierbeispiel

In diesem Kapitel befindet sich ein vollständiger Abdruck der Quelltextdateien, mit deren Hilfe der in Abschnitt 6.3 entwickelte Inference Handler implementiert wurde.
Jeder Datei und Methode ist ein Informationsblock vorangestellt, der ihre Funktion erläutert.

B.1 CsIRM2Hub.h

```
////////////////////////////////////////////////////////////////////////////
//
//      Entwurf und Implementierung von Managementszenarien zu verteilten
//      Kommunikationsanwendungen
//      Diplomarbeit
//      Technische Universitaet Muenchen
//
//      Workfile:               CsIRM2Hub.h
//      Directory:              /proj/Spectrum3/IHAPI/nwp
//      Original Author:        Alexander Keller
//      Date:                   11/3/93
//
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
//
//      CLASS
//
//      CsIRM2Hub - This is the class for the IRM2Hub Model Type
```

```
//
//      DESCRIPTION
//
//      This class provides the Attribute Id's of the IRM2Hub Models
//
////////////////////////////////////////////////////////////////////////

#if !defined ( __CSIRM2HUB_H__ )
#define __CSIRM2HUB_H__

class CsIRM2Hub
{
        public:
                enum
                {
                  DeviceCRC = 0x10e33
                };
};

// Model Type Handles

#define IRM2_TEST_MTYPE 0x10059
#endif
```

B.2 CsTestMI.cc

```
////////////////////////////////////////////////////////////////////////////
//
//      Entwurf und Implementierung von Managementszenarien zu verteilten
//      Kommunikationsanwendungen
//      Diplomarbeit
//      Technische Universitaet Muenchen
//
//      Workfile:               CsTestMI.cc
//      Directory:              /proj/Spectrum3/IHAPI/nwp
//      Original Author:        Alexander Keller
//      Date:                   11/3/93
//
////////////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////////////////
//
// FILE
//
//    CsTestMI.cc
//
// DESCRIPTION
//
//    This file contains the model intelligence for the Test Model Type.
//    The Inference Handler CsIHTest is attached to the IRM2Hub Model Type
//    and watches its attribute DeviceCRC (=Counter for bad MAC-frames).
//
/////////////////////////////////////////////////////////////////////////////

#if !defined ( __CSMINODE_H__ )
#include "CsMINode.h``
#endif

#if !defined ( __CSIRM2HUB_H__ )
#include ``CsIRM2Hub.h``
#endif

#if !defined ( __CSIHTEST_H__ )
#include ``CsIHTest.h``
#endif

static void CsIRM2HubMI( CsMTypeHandle& m_t_handle )
{

    // Attach CsIHTest IH to IRM2Hub Model Type

    // and if unsuccessful, unattach CsIHTest IH.


    CsIHTest * test_ih = new CsIHTest( m_t_handle ) ;
    if ( test_ih→get_error() == CsError::FAILURE )
    {
        CsDelete( test_ih );
    }

}
```

```
static CsMINode mi_node ( IRM2_TEST_MTYPE, CsIRM2HubMI ) ;
```

B.3 CsIHTest.h

```
///////////////////////////////////////////////////////////////////////////////
//
//      Entwurf und Implementierung von Managementszenarien zu verteilten
//      Kommunikationsanwendungen
//      Diplomarbeit
//      Technische Universitaet Muenchen
//
//      Workfile:               CsIHTest.h
//      Directory:              /proj/Spectrum3/IHAPI/nwp
//      Original Author:        Alexander Keller
//      Date:                   11/3/93
//
///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////
//
// FILE
//
//    CsIHTest.h
//
// DESCRIPTION
//
//    This file defines the CsIHTest class including functionality
//    for measuring the bad MAC-frames.
//
// MODIFICATION HISTORY
//
//
///////////////////////////////////////////////////////////////////////////////

#if !defined ( __CSIHTEST_H__ )
#define __CSIHTEST_H__

#if !defined ( __CSCHANGEN_H__ )
#include "CsChangeN.h"
#endif
```

```
#if !defined ( __CSIHBASE_H__ )
#include ''CsIHBase.h''
#endif


////////////////////////////////////////////////////////////////////////////////
//
// CLASS
//
//      CsIHTest
//
// DESCRIPTION
//
//      This class holds the IH CsIHTest which will be attached to the
//      CABLETRON IRM2 Hub Model Type. The methods will watch the DeviceCRC
//      attribute and report about changes.
//
// PUBLIC METHODS
//
//   CsIHTest( CsMTypeHandle& m_t_handle )
//
//      This is the constructor for CsIHTest which will register for
//      CsIRM2Hub::DeviceCRC.
//      CsIHTest is derived from CsIHBase
//
//   trig_attr_change( const CsModelHandle& mh,
//                     const CsChangeNode * change )
//
//
//
// PRIVATE METHODS
//
//
////////////////////////////////////////////////////////////////////////

class CsIHTest : public CsIHBase
{
        public:

        CsIHTest( CsMTypeHandle& );

        void trig_attr_change( const CsModelHandle& ,
                               const CsChangeNode * );
```

```
};
#endif
```

B.4 CsIHTest.cc

```
////////////////////////////////////////////////////////////////////////////////
//
//      Entwurf und Implementierung von Managementszenarien zu verteilten
//      Kommunikationsanwendungen
//      Diplomarbeit
//      Technische Universitaet Muenchen
//
//      Workfile:               CsIHTest.cc
//      Directory:              /proj/Spectrum3/IHAPI/nwp
//      Original Author:        Alexander Keller
//      Date:                   11/3/93
//
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//
// FILE
//
//      CsIHTest.cc - File containing the implementation of CsIHTest
//
// DESCRIPTION
//
//      CsIHTest is attached to the CABLETRON-IRM2-Hub Model Type.
//      It monitors the changes in the counter for broken MAC-frames and
//      reports them to the user.
//
///////////////////////////////////////////////////////////////////////////////

#ifndef MY_DEBUG
#define MY_DEBUG 1
#endif

#if !defined ( __CSIRM2HUB_H__ )
#include "CsIRM2Hub.h"
#endif
```

```
#if !defined ( __CSIHTEST__ )
#include ``CsIHTest.h``
#endif

#if !defined ( __CSATTRVALL_H__ )
#include ``CsAttrValL.h``
#endif

#if !defined ( __CSATTRVALDEF_H__ )
#include ``CsAttrValDef.h``
#endif

#if !defined ( __CSATTRDESC_H__ )
#include ``CsAttrDesc.h``
#endif

#if !defined ( __CSDEBUG_H__ )
#include ``CsDebug.h``
#endif

unsigned long CsIHBase::debug_flags = CsDebug::ALL_ON;


///////////////////////////////////////////////////////////////////////////////
//
// FUNCTION
//
//    CsIHTest::CsIHTest()
//
// SYNOPSIS
//
//    CsIHTest( CsMTypeHandle& in_m_t_handle)
//      : CsIHBase(in_m_t_handle)
//
// DESCRIPTION
//
//      This is the constructor for CsIHTest. It will verify that the
//      model type to which CsIHTest is attached contains the DeviceCRC
//      attribute. It will register to receive attribute changes for DeviceCRC.
//
//////////////////////////////////////////////////////////////////////////////

CsIHTest::CsIHTest( CsMTypeHandle& in_m_t_handle )
```

```
    : CsIHBase ( in_m_t_handle )
{
    tout( CsIHTest::CsIHTest() m_t_handle=("
<< hex << in_m_t_handle << dec << ‘‘) constructor activated."
<< endl );

    CsVnmMTypeHandle *vmth = new CsVnmMTypeHandle( in_m_t_handle );

    // Verify that the model type contains the attribute DeviceCRC.


    if ( ! vmth->has_attr ( CsIRM2Hub::DeviceCRC ))
    {
        eout( No such attribute: DeviceCRC"
<< endl );
          set_error( CsError::FAILURE );
    }
    else
    {
        // Register to receive changes in DeviceCRC


        if (!reg_attr_change( CsIRM2Hub::DeviceCRC ))
        {
           eout( Couldn’t register for changes in DeviceCRC."
<< endl);
           eout( m_t_handle = "
<< hex << m_t_handle << dec << endl );
           set_error( CsError::FAILURE );
           return;
        }
        else
        {
           tout( ‘‘Registered for CsIRM2Hub::DeviceCRC change"
<< endl);
        }
    }
    tout( CsIHTest::CsIHTest() ending."
<< endl);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//
// FUNCTION
//
//    CsIHTest::trig_attr_change()
//
// SYNOPSIS
//
//    trig_attr_change( const CsModelHandle& mh,
//                      const CsChangeNode * change )
//
// DESCRIPTION
//
//    This method will receive changes in DeviceCRC.
//
// RETURNS
//
//      void
//
// ERRORS
//
//      none
//
// CAVEATS
//
//      none
//
//
// SEE ALSO
//
////////////////////////////////////////////////////////////////////////////////

void CsIHTest::trig_attr_change( const CsModelHandle& mh,
                  const CsChangeNode * change )
{
        tout(CsIHTest::trig_attr_change() mh=("
<< hex << mh << dec << ‘‘) activated."
<< endl );

        int errors = * (int *)(change->get_cur_value());
        int prev_errors = * (int *)(change->get_prev_value());
```

```
        // if DeviceCRC has not changed, do nothing

        if (errors ≠ prev_errors)
                {
            switch( change→get_attr_id())
                        {
                        case CsIRM2Hub::DeviceCRC:
                                {
                                 iout(CsIHTest:trig_attr_change() mh=("
≪ hex ≪ mh ≪ dec ≪ ‘‘) Errors changed from "
≪ prev_errors ≪ to "
≪ errors ≪ endl);

                                 break;
                                }

                        default:
                                {
                                eout( Model with mh=("
≪ hex ≪ mh ≪ dec ≪
                                          ‘‘) is only registered for
DeviceCRC "
≪ endl );
                                }
                        }
                }
        tout( CsIHTest::trig_attr_change() ending."
≪ endl );
}
```

Anhang C

Liste der verwendeten Abkürzungen

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ASN.1 Abstract Syntax Notation One (1)

BDT Bulk Data Transfer

BMW Bayerische Motorenwerke

BSC Binary Synchronous Communication

CMIP Common Management Information Protocol

CMIS Common Management Information Services

CMS Conversational Monitoring System

CP Control Program

CSCW Computer Supported Cooperative Work

DCM Device Communication Manager

DDCMP Digital Data Communication Protocol

DEC Digital Equipment Corporation

DNS Domain Name System

EPAPI Extended Protocol Interface Application Programming Interface

FTP File Transfer Protocol

GMOC Generic Managed Object Classes

ICMP Internet Control Message Protocol

IEEE Institute of Electrical and Electronics Engineers

IHAPI Inference Handler Application Programming Interface

IP Internet Protocol

IMT Inductive Modeling Technology

ISO International Organisation of Standardization

LME Layer Management Entity

MAC Media Access Control

MI Model Intelligence

MIB Management Information Base

MNM-Team Münchner Netzmanagement Team

MO Managed Object

MS-DOS Microsoft Disk Operating System

MTE Model Type Editor

MVS Multiple Virtual Storage

NFS Network File System

NIDAT Netwerk Informations Datei

NJE Network Job Entry

NM Netzmanagement

NWP NetzWerk–Programm

ODP Open Distributed Processing

OSF Open Software Foundation

OSI Open Systems Interconnection

PC Personal Computer

RFC Request for Comments

RSCS Remote Spooling Communication Subsystem

SAP Service Access Point

SDLC Synchronous Data Link Control

SMFA Specific Management Functional Area

SMI Structure of Management Information

SNA Systems Network Architecture

SNMP Simple Network Management Protocol

SSAPI SpectroSERVER Application Programming Interface

TCP Transmission Control Protocol

TSO Time Sharing Option

VM Virtual Machine

VMS Virtual Memory System

VPAPI VnmParmBlock Application Programming Interface

VWAPI View Application Programming Interface

Literaturverzeichnis

[ABEC 92] Sebastian Abeck, „The Knowledge Aspect in an Extended Management Information Model“, Arbeitspapier, Februar 1992.

[ABLE 93] Sebastian Abeck und Martin Leischner, „Einsatz der Inductive Modeling Technology zur Netz- und Komponentenmodellierung im Netzmanagement“, In *Kommunikation in verteilten Systemen*, GI/ITG–Fachtagung, März 1993.

[ALSE 92] Sebastian Abeck, Martin Leischner und Peter Segner, „Applying the Inductive Modeling Technology to Tackle the Problem of Integrated Network Management“, IFIP/IEEE Workshop on Distributed Systems: Operations and Management 1992, Oktober 1992.

[ASVA 92] Sebastian Abeck, Peter Segner und Robert Valta, „Der Informationsaspekt im Fehlermanagement für Kommunikationsnetze – Eine Gegenüberstellung von Ergebnissen aus Forschung und Praxis“, In *Praxis der Informationsverarbeitung und Kommunikation*, Heft 2/92, März 1992.

[ASWI 93] Sebastian Abeck, Peter Segner und Norbert Wienold, „Managing Hubs in a Heterogeneous Environment: An Integrated Approach“, Cracow International Workshop on Requirements and Techniques for Network Management, März 1993.

[BLAC 92] Uyless Black, *Network Management Standards - The OSI, SNMP and CMOL Protocols*, McGraw-Hill, 1992.

[CCITT X.400] CCITT, *Message Handling Systems: System and Service Overview*, August 1987.

[CCITT X.500] CCITT, *Data communication networks: directory. Fascicle VIII.8*, November 1988.

[DEV 92] Roger Dev, „Managing the Enterprise Network: Command and Control“, Technischer Bericht, Cabletron Systems, Inc., Rochester, NH 03867-0505, November 1992.

[FORUM 90] OSI / Network Management Forum, *Forum Library of Managed Object Classes, Name Bindings and Attributes*, 1990.

[GEIH 91] Kurt Geihs, „The Road to Open Distributed Processing“, Kommunikation in verteilten Systemen, März 1991.

[GIES 85] E. Giese, K. Goergen, E. Hinsch, G. Schulze und K. Truöl, *Dienste und Protokolle in Kommunikationssystemen*, Springer Verlag, 1985.

[GMD 89] M. Tschichholz, M. Behrendt, A. Consael, A. Dittrich, N. Jantzen, O. Schittko, S.Waßerroth und C. Wieschiolek, „Management für verteilte Anwendungen im ISDN-B: Anforderungsanalyse und Stand der Arbeiten im Bereich Management“, GMD-Bericht Version 1.0, November 1989.

[GORA 90] Walter Gora und Reinhard Speyerer, *ASN.1 Abstract Syntax Notation One*, Datacom Verlag, 1990.

[HAVA 93] Heinz-Gerd Hegering, Sebastian Abeck und Robert Valta, „Integriertes Netzmanagement: Architekturansätze und Werkzeuge“, In *Kommunikation in verteilten Systemen*, Tutoriumsbeitrag, März 1993.

[HEAB 93] Heinz-Gerd Hegering und Sebastian Abeck, *Integriertes Netz- und Systemmanagement*, Addison-Wesley, März 1993.

[HEGE 88] Heinz-Gerd Hegering, „Open Systems Interconnection - Eine kritische Würdigung“, GI–Jahrestagung, Oktober 1988.

[HEGE 89] Heinz-Gerd Hegering, „Auf dem Weg zu offenen Netzmanagement-Architekturen“, *Computerwoche*, August 1989.

[ISO 10040] ISO, *Information Processing Systems - Open Systems Interconnection - Systems Management Overview*, September 1990.

[ISO 10164] ISO, *Information Processing Systems - Open Systems Interconnection - Systems Management, Systems Management Functions: Part 1 – 15*, Oktober 1992.

[ISO 10165] ISO, *Information Processing - Open Systems Interconnection - Management Information Services - Structure of Management Information, Part 1 – 6*, Oktober 1992.

[ISO 10165-1] ISO, *Information Processing - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 1: Management Information Model*, September 1991.

[ISO 10165-5] ISO, *Information Processing - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 5: Generic Management Information*, Mai 1992.

[ISO 7498] ISO, *Information Processing Systems - Open Systems Interconnection - Basic Reference Model*, 1984.

[ISO 7498-4] ISO, *Information Processing Systems - Open Systems Interconnection - Basic Reference Model Part 4: Management Framework*, 1989.

[ISO 9595] ISO, *Information Processing Systems - Open Systems Interconnection - Common Management Information Service Definition*, November 1991.

[ISO 9596-1] ISO, *Information Processing Systems - Open Systems Interconnection - Common Management Information Protocol - Part 1: Specification*, November 1991.

[ISO N4888-2] ISO, *Basic Reference Model of Open Distributed Processing Part 2: Descriptive Model*, 1992.

[ISY IK] IAS Gmbh, „ISYKON Entity Relationship Modell", IV-Konzept, IAS Gesellschaft für EDV-Training, Beratung, Entwicklung mbH, Februar 1992.

[KERN 92] Helmut Kerner, *Rechnernetze nach OSI*, Addison–Wesley, 1992.

[MNM ARB] Sebastian Abeck, Martin Leischner und Peter Segner, „Arbeiten und Entwickeln mit SPECTRUM", Projektbericht, März 1992.

[MNM HUB] Münchner Netzmanagement Team, *Anforderungen an die Integration von Hubs in Spectrum*, November 1992.

[MNM IMT] Sebastian Abeck und Martin Leischner, „Die Inductive Modeling Technology im Netzmanagement", Projektbericht, April 1992.

[MNM UEB] Sebastian Abeck und Martin Leischner, „SPECTRUM-Projekte im Überblick", Projektbericht, Juni 1992.

[NWP ALL] BMW AG, „NetzWerk-Programm Allgemeine Information", Technischer Bericht, Dezember 1990.

[NWP BEN] BMW AG, „NetzWerk-Programm Benutzerhandbuch", Technischer Bericht, Dezember 1990.

[NWP DOK] K.Micka und M.Ryan, „NWP Dokumentation und Fehlersuche", Technischer Bericht, 1992.

[NWP FK] Jürgen Lohrmann, „Management der NWP-Knoten auf SNMP-Basis", Fachkonzept, ICS Intelligent Communication Software GmbH, Klugstrasse 58, 8000 München 19, September 1992.

[NWP NMa] V. Heymer, „NWP Netzmanagement", Fachkonzept, UBS GmbH, Dezember 1991.

[NWP NMb] V. Heymer, H.P. Kopp und S. Befort, „NWP Netzmanagement“, DV-Konzept, UBS GmbH, Februar 1992.

[NWP PRO] BMW AG, „NetzWerk-Programm Programmierer Handbuch“, Technischer Bericht, Dezember 1990.

[ORAM 91] Andrew Oram und Steve Talbott, *Managing Projects with make*, O‘Reilly & Associates, Inc., Second edition, Oktober 1991.

[RFC 1155] M.T. Rose und K. McCloghrie, „Structure and identification of management information for TCP/IP-based internets“, RFC 1155, IAB, Mai 1990.

[RFC 1156] K. McCloghrie und M.T. Rose, „Management Information Base for network management of TCP/IP-based internets“, RFC 1156, IAB, Mai 1990.

[RFC 1157] J.D. Case, M. Fedor, M.L. Schoffstall und C. Davin, „Simple Network Management Protocol (SNMP)“, RFC 1157, IAB, Mai 1990.

[RFC 1212] eds. K. McCloghrie und M.T. Rose, „Concise MIB definitions“, RFC 1212, IAB, März 1991.

[RFC 1213] eds. K. McCloghrie und M.T. Rose, „Management Information Base for network management of TCP/IP-based internets: MIB-II“, RFC 1213, IAB, März 1991.

[RFC 1215] ed. Rose, M.T., „Convention for defining traps for use with the SNMP“, RFC 1215, IAB, März 1991.

[RFC 1270] ed. Kastenholz, F., „SNMP communications services“, RFC 1270, IAB, Oktober 1991.

[RFC 1271] S. Waldbusser, „Remote network monitoring management information base“, RFC 1271, IAB, November 1991.

[RFC 1303] K. McCloghrie und M. Rose, „A Convention for Describing SNMP-based Agents“, RFC 1303, IAB, Februar 1992.

[RFC 1351] J. Davin, J. Galvin und K. McCloghrie, „SNMP Administrative Model“, RFC 1351, IAB, Juli 1992.

[RFC 1353] K. McCloghrie, J. Davin und J. Galvin, „Definitions of Managed Objects for Administration of SNMP Parties“, RFC 1353, IAB, Juli 1992.

[ROSE 91] Marshall T. Rose, *The Simple Book: An Introduction to Management of TCP/IP-based internets*, Prentice Hall, 1991.

[SPEC BAS] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Basic Extension Guide*, Mai 1992, Order Number: 9030453E2.

[SPEC CON] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Concepts Guide*, Mai 1992, Order Number: 9030647E1.

[SPEC EPI] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM MSAP-EPI Developers Guide*, Juli 1991, Order Number: 9030487.

[SPEC EXT] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Extension Integration Toolkit Developer's Guide*, Mai 1992, Order Number: 9030623E1.

[SPEC GLO] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Global Classes Reference*, Dezember 1992.

[SPEC IHA] Cabletron Systems, Inc., Rochester, NH 03867-0505, *Inference Handler Application Programming Interface Developer's Guide*, Dezember 1992, Order Number: 9030489E5.

[SPEC IHC] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Developer's Toolkit Inference Handler Course*, Dezember 1992, Revision 1.0.

[SPEC KBG] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Knowledge Base Guide*, Mai 1992, Order Number: 9030663E1.

[SPEC MTE] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM Model Type Editor Guide*, April 1992, Order Number: 9030659E2.

[SPEC SSA] Cabletron Systems, Inc., Rochester, NH 03867-0505, *Asynchronous SpectroSERVER Application Programming Interface Developer's Guide*, März 1992, Order Number: 9030486E2.

[SPEC SSS] Cabletron Systems, Inc., Rochester, NH 03867-0505, *Synchronous SpectroSERVER Application Programming Interface Developer's Guide*, März 1992, Order Number: 9030624E3.

[SPEC VPA] Cabletron Systems, Inc., Rochester, NH 03867-0505, *SPECTRUM VnmParmBlock Developer's Guide*, Dezember 1992.

[STRO 91] Bjarne Stroustrup, *The C++ Programming Language*, Addison–Wesley, Second edition, 1991.

[TANE 89] Andrew S. Tanenbaum, *Computer Networks*, Prentice–Hall International Editions, 1989.

[WIED 93] Klaus Wiedemann, „Integration eines Netz- und Fehlerdokumentationssystems in eine Management-Plattform als Basis für ein Enterprise Management“, Mai 1993, Diplomarbeit der Technischen Universität München.

[WIEN 93] Norbert Wienold, „Anforderungsanalyse und Lösungsvorschläge für ein unternehmensweites Komponentenmanagement“, November 1993, Diplomarbeit der Technischen Universität München.