INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung und prototypische Implementierung einer graphischen Bedienoberfläche für das VLAN-Management

Bearbeiter	: Anton Pointner
Aufgabensteller	: Prof. Dr. Heinz-Gerd Hegering
Betreuer	: Dipl. Inform. Norbert Wienold

Inhaltsverzeichnis

Abbildungsverzeichnis

Teil I

Grundlagen

Kapitel 1

Einleitung

1.1 Motivation

Lean Management und damit der Wechsel von der Abteilungsstruktur zu projektorientierten kleinen Arbeitsgruppen fordert dem Rechnernetz parallel zum steigenden Bandbreitenbedarf ein Höchstmaß an organisatorischer Flexibilität ab. Umzüge von Mitarbeitern innerhalb des Unternehmens nehmen rasant zu[1] und erfordern somit bei starren Netzstrukturen enorme Kosten[2], da wegen der konventionellen Strukturierung des Netzes mittels herkömmlicher Komponenten, wie etwa Brücken und Routern, der Arbeitsplatz meist auch einem anderen physischen Subnetz zugeordnet werden muß.

Eine wichtige Vorausetzung, um diese Kosten zu drücken, ist eine weit höhere Abstraktion/Virtualisierung der Datenbestände und Rechner-Ressourcen als bisher von der geographischen Lage und den verwendeten Rechnern und Netztopologien. Es soll dabei weder die Überwachung und Administration des Netzes extrem aufwendiger, noch dürfen bestehende Investitionen überflüssig und die Sicherheit im Netz gefährdet werden.

Das Konzept der virtuellen LANs (VLANs) bietet eine geeignete Basis zur Realisierung dieser Virtualisierung, um damit u.a die oben genannten Forderungen nach

- maximaler Flexibilität beim Umkonfigurieren,
- Investitionsschutz und

[1] Zur Zeit (Herbst 1996) soll jeder Mitarbeiter im Durchschnitt alle zwei bis drei Jahre seinen Arbeitsplatz innerhalb des Unternehmens wechseln.

[2] Studien des US-Marktforschungsinstituts Forester Research berechneten die Kosten bei mittelgroßen Netzen (ca. 850 Mitarbeiter) von durchschnittlich 2.1 Millionen Dollar allein für Änderungen und Erweiterungen im Netzwerk. Dataquest/Ledgeway ermittelte, daß 22 Prozent der Netzwerkbetriebskosten auf das Ändern und Hinzufügen von Kommunikationsverbindungen entfallen.

- Sicherheit (Closed User Groups[3], Filtermechanismen,...)

zu erfüllen. Mit diesen drei Forderungen als Grundbestandteil moderner Unternehmen entstand der Begriff des virtuellen LANs.

Ein virtuelles LAN ist zunächst einmal definiert als eine Broadcast-Domäne, d.h. Broadcasts einer Endstation erreichen alle anderen Endstationen, wie es in einem klassischen LAN mit einem *Shared Medium* (z.B. Ethernet und Token Ring) der Fall ist, nur daß der geographische Standort der Endstationen bewußt nicht festgelegt wird. Diese Freiheit, Endstationen eines (weltweiten) Unternehmensnetzes beliebig zu einem LAN zu gruppieren (=VLAN), erleichtert u.a. den erwünschten Übergang von der Abteilungsstruktur zu kleinen projektorientierten Arbeitsgruppen (Workgroups[4]), die sich von Projekt zu Projekt dynamisch ändern können. Wie im Verlauf der Arbeit gezeigt wird, ergeben sich noch weitere Vorteile, wenn die VLANs mit Hilfe der Switching-Technologie realisiert werden.

1.2 Ziel der Diplomarbeit

Das Ziel der Diplomarbeit ist es, Vorschläge für eine graphische Bedienoberfläche für das VLAN-Management zu liefern. Dazu müssen zuerst die Topologien und Strukturierungsmaßnahmen in herkömmlichen LANs und die notwendigen Voraussetzungen für VLANs analysiert werden. Aus diesen Analysen wird der Nutzen virtueller LANs motiviert, d.h. es geht um die Fragestellung: Was leisten VLANs? Wozu braucht man VLANs? Die Beantwortung dieser und ähnlicher Fragen ergeben mögliche Einsatzszenarien für VLANs, die wiederum — wie die Virtualisierung/Abstrahierung vom physischen Netz — weitere Anforderungen an die graphische Bedienoberfläche aufwerfen. Aus diesen Einsatzszenarien werden VLAN-spezifische Aufgaben und Handlungsabläufe eines Netzadministrators extrahiert und festgeschrieben. Diese Ergebnisse bilden den Grundstock für die weiteren Analysetätigkeiten und der daran anschließenden Entwurfs- und Spezifikationsphase (Designphase). Die prototypische Implementierung der graphischen Bedienoberfläche ist dann nur noch eine programmiertechnische Umsetzung der erarbeiteten Vorschläge. In dieser Arbeit werden nur einige der Vorschläge umgesetzt (siehe dazu auch die Bemerkung am Schluß des Kapitels).

Diese Vorgehensweise — von den Aufgaben und Handlungsabläufen als Ausgangsbasis für die Entwicklung auszugehen — läßt sich durch folgende Überlegungen rechtfertigen:

- Die Aufgabenanalyse bildet auch die Ausgangsbasis für viele objektorientierte Software-Entwicklungsmethoden, was der Anlaß war, das Entwicklungskonzept,

[3] CUGs

[4] Auf die Begriffe VLAN und Workgroup wird noch genauer eingegangen.

das im zweiten Teil der Arbeit beschrieben wird, der einer objektorientierten Software-Entwicklungsmethode anzulehnen. Anlehnen deshalb, weil die Entwicklung einer graphischen Bedienoberfläche in den daraufhin untersuchten OO-Entwicklungsmethoden[5] nur am Rande behandelt wird und deshalb ein eigenes Entwicklungskonzept vorgelegt wird.

- Eine gute graphische Bedienoberfläche soll dem Netzadministrator bei seinen Aufgaben, die er durchzuführen hat, optimal unterstützen, d.h. die Aufgaben müssen mit dem VLAN-Managementsystem (kurz System) möglichst effektiv und effizient durchführbar sein und die Bedienung sollte ohne großen Lernaufwand intuitiv beherrschbar und überschaubar sein. Dazu ist es wichtig zu wissen, was der Anwender mit dem System machen will. Genau diese Erkenntnisse liefert die Anwender- und Aufgabenanalyse.

Nach der Festsetzung der Aufgaben, die mit dem System bzw. mit der graphischen Bedienoberfläche durchführbar sein sollen, wird eine konkrete physische Netzumgebung (Ethernet, Ethernetswitches, ATM LAN Emulation) festgelegt, um die Arbeit nicht zu sehr mit Ausnahmebehandlungen, die sich aufgrund der Komplexität und Heterogenität heutiger Rechnernetze ergeben[6], zu überladen.

Die Aufgabenstellung und Zielsetzung dieser Diplomarbeit könnte auch wie folgt umschrieben werden:

> Gegeben sei eine konkrete physische Netzumgebung und eine Menge von typischen VLAN-spezifischen Managementaufgaben. Gesucht ist eine graphische Bedienoberfläche, die eine intuitive Bedienung und eine möglichst effiziente und effektive Bearbeitung der Aufgaben ermöglicht.

Wobei die Netzumgebung und die Aufgaben — in Absprache mit dem Betreuer — eigenständig festgelegt wurden. Bei der Auswahl wurde darauf geachtet, einen Kompromiß zwischen den heute schon und den erst in naher Zukunft realisierbaren Szenarien und den damit verbundenen VLAN-spezifischen Aufgaben zu finden. Kriterien für die Auswahl der Aufgaben waren die Häufigkeit der Ausführung und die Notwendigkeit der Aufgabe im jeweiligen Szenario.

Durch die Virtualisierung/Abstrahierung vom physischen Netz und der Analyse der Aufgaben ergeben sich eine Reihe weiterer Anforderungen an die graphische Bedienoberfläche, die auf den ersten Blick noch nicht absehbar sind und sich erst im weiteren Verlauf der Arbeit als essentiell erweisen.

Anstatt vieler hübscher Grafiken (Hollywood-Effekt) steht die intuitive Bedienung der Bedienoberfläche und ein flüssiger Arbeitsablauf im Vordergrund. Dadurch,

[5] Object-Oriented Software Engineering (OOSE) von Jacobson und Object Modeling Technique (OMT) von Rumbaugh et. al.

[6] Sofern sie keine VLAN-spezifischen Probleme darstellen.

daß sich ein ganzer Zweig der Informatik (Sofware Ergonomie) mit diesen Fragestellungen beschäftigt, zeigt sich, wie wichtig graphische Bedienoberflächen sind, denn in gewisser Weise arbeitet der Anwender mit der graphischen Bedienoberfläche und nicht mit dem funktionalen Teil der Anwendung bzw. Anwendungskern (siehe auch Abschnitt 4.1), der sich *hinter* der Bedienoberfläche verbirgt. Neben dem Aspekt der Software-Ergonomie kommt noch die Tatsache hinzu, daß es bei der Darstellung von VLANs, der Visualisierung der logischen Virtualisierung/Abstraktion vom physischen Netz und den damit verbundenen Tätigkeiten bisher keine Konzepte — geschweige den schlüsselfertige Lösungen — gibt. Z.B. führt die farbliche Hinterlegung der VLANs innerhalb der physischen Netztopologie (am Bildschirm) sehr schnell zu einer unübersichtlichen Darstellung ("Fleckerlteppich") und verdeutlicht in keiner Weise die damit verbundene Virtualisierung des physischen Netzes.

So wie das VLAN-Konzept neue Möglichkeiten der Strukturierung von LANs und den daraus resultierenden Einsatzszenarien ermöglicht, müssen parallel dazu, für die zu entwickelnde graphische Bedienoberfläche, neue Konzepte für das VLAN-Management entwickelt und eingeführt werden. Die Analyse der sich daraus ergebenden Anforderungen an die Bedienoberfläche und die graphische Umsetzung am Bildschirm stellen die Aufgabenstellung dieser Diplomarbeit dar.

1.3 Aufbau der Diplomarbeit

Nach obiger Einleitung und Motivation virtueller LANs, wird in Kapitel 2 näher auf die Grundlagen der VLANs eingegangen. Nach der Analyse der Strukturierung in herkömmlichen LANs wird auf die Notwendigkeit der Einführung einer neuen Klasse von Vermittlungskomponenten, den Switches, geschlossen. Bevor auf die Switching-Technologie und den Switches eingegangen wird, werden deren Einsatz und die daraus resultierenden Vor- und Nachteile, nicht nur für VLANs, erörtert.

Die Switches sind miteinander über Backbones verknüpft. Für die ausgewählte physische Netzumgebung sind beliebige Arten von Backbones einsetzbar (FDDI, Fast Ethernet, ATM, usw.). Über kurz oder lang wird sich wahrscheinlich die ATM-Technologie durchsetzen. Aufgrund fehlender Standards für die Interswitchkommunikation sind heute entweder proprietäre Lösungen oder die Verwendung der ATM LAN Emulation notwendig, um VLANs switchübergreifend einzurichten. Die ATM LAN Emulation ist in einer ersten Version standardisiert und wird von nahezu allen Komponenten-Herstellern, die sich ATM auf die Fahnen geschrieben haben, angeboten. Zudem wird die Migration zu *native* ATM erleichtert, ohne daß bestehende Investitionen gefährdet werden. Diese Vorteile rechtfertigen eine genauere Betrachtung der ATM LAN Emulation in Abschnitt 2.5.

Daran anschließend werden in Kapitel 3 die für das Verständnis der weiteren Arbeit wichtigsten Grundlagen, Begriffe, Konzepte und Werkzeuge zum Thema graphische

Bedienoberflächen besprochen.

Der zweite Teil der Arbeit widmet sich ganz dem methodischen Vorgehen bei der Entwicklung und prototypischen Implementierung einer graphischen Bedienoberfläche. Aus den bereits in Abschnitt 1.2 erwähnten Gründen wird ein eigenes Konzept vorgelegt. Dieses Entwicklungskonzept wird in Kapitel 5 vorgestellt und in Kapitel 6 in die Phasen einer herkömmlichen objekt-orientierten Software-Entwicklungsmethode eingeordnet.

Im dritten Teil der Diplomarbeit wird das Entwicklungskonzept schließlich angewandt, mit dem Ziel, die in der Einleitung geforderte Aufgabenstellung der Diplomarbeit und damit dem Titel der Arbeit gerecht zu werden. Es wird dabei fast der ganze Lebenszyklus *(Lifecycle)* einer Software-Entwicklung, bis hin zur prototypischen Implementierung durchlaufen. Die Implementierung des Gesamtsystems wird aus Gründen, die in der abschließenden Bemerkung am Ende dieses Kapitels angeführt werden, nicht durchgeführt. Da das Entwicklungskonzept einer objekt-orientierten Software-Entwicklung nachempfunden ist, zerfällt es auch in deren drei typischen Phasen. Kapitel 7 beinhaltet die Überlegungen und Ergebnisse der Analysephase, Kapitel 8 die der Entwurfs- und Spezifikationsphase bzw. Designphase[7], wobei auch versucht wird vorhandene Lösungen zu analysieren und die Ergebnisse mit einzubeziehen.
Kapitel 8 untersucht und vergleicht verschiedene verfügbare Werkzeuge für die prototypische Implementierung. Obwohl der Schwerpunkt der Arbeit in dem theoretischen Teil einer Software-Entwicklung (Analyse, Entwurf und Spezifikation) liegt, wird versucht, die Arbeit mit einem Prototypen abzuschließen, der die wesentlichen Ideen der Virtualisierung vom physischen Netz und dem Management virtueller Netze verdeutlicht. Die Beschreibung des Prototypen, der — durch Implementierung einer Teilfunktionalität — zu einem Demonstrator ausgebaut wird, bildet den Abschluß im Kapitel 8.

Kapitel 9 rundet die Arbeit mit einer Zusammenfassung

und einem Ausblick für weitere Studien und Folgearbeiten ab.

Im vierten Teil wird mit dem Quellkode der prototypischen Implementierung diese Arbeit vervollständigt. Den Abschluß bildet das Literatur- und Abkürzungsverzeichnis.

Bemerkung:
Wie in einer Diplomarbeit üblich, liegt der Schwerpunkt im konzeptionellen Bereich. Die prototypische Implementierung sowie die Implementierung des Gesamtsystems spielen eine eher untergeordnete Rolle, da sie im Rahmen eines Einmannprojekts, wie

[7] In dieser Arbeit wird die Bezeichnung Entwurfs- und Spezifikationsphase der Bezeichnung Designphase vorgezogen, weil erstere, neben dem *Designen* der Objekte und Bildschirmdarstellungen, die Tätigkeit des Entwurfs und der Spezifikation der möglichen und notwendigen Interaktions- und Kontrollmechanismen stärker unterstreicht.

es in dieser Arbeit der Fall ist und aufgrund des Umfangs nicht durchführbar sind. Genauso wäre ein vollständiger Entwicklungsprozeß für das VLAN-Managementsystem viel zu tiefgreifend und umfangreich für eine einzelne Diplomarbeit. Bei der Durchführung des Entwicklungskonzepts wird das Augenmerk daher nur auf die am Bildschirm darzustellenden VLAN-relevanten Aspekte gerichtet. Es ist klar, daß damit diese Arbeit an keinem herkömmlichen Software-Entwicklungsprozeß, sowohl an Umfang als auch an Tiefe, heranreicht. Dennoch wurde darauf geachtet, den roten Faden nicht abreißen zu lassen, bei den VLAN-spezifischen Fragestellungen zu bleiben und diese Arbeit mit einem Prototypen, der einige der Vorschläge verdeutlichen soll, abzurunden.

Kapitel 2

Virtuelle LANs

2.1 Einführung in die Problematik

Das Management von (produktiv) genutzten Rechnernetzen gestaltet sich ab einer gewissen Größe umfangreich und komplex. Dies liegt u.a. in der Tatsache begründet, daß Rechnernetze nicht "statisch" sind, sondern es sich dabei um ausgesprochen "dynamische" Gebilde handelt, die ständig weiter ausgebaut, umkonfiguriert und dabei gleichzeitig auch heterogener werden.
Auf der anderen Seite findet in den Unternehmen, wie bereits im ersten Kapitel angesprochen, ein Wechsel von der Abteilungsstruktur hin zu kleinen projektorientierten Arbeitsgruppen statt. Mit zunehmender Komplexität von Rechnernetzen und Unternehmensstrukturen sind Netzkonzepte gefragt, die die Strukturierung herkömmlicher Netze vereinfachen und dabei eine Abbildung der internen Organisation auf die Netztopologie ermöglichen, um so die Bildung dynamischer virtueller LANs (Grundlage für *virtuelle* Workgroups[1]) zu unterstützen. Die Zuordnung von Endgeräten/Mitarbeitern verschiedener Abteilungen zu projektbezogenen VLANs/Workgroups soll dabei möglich sein, ohne das darunterliegende physische Netz umzukonfigurieren und ohne bestehende Investitionen und die Sicherheit im Netz zu gefährden.

Neue Netzkonzepte und Entwicklungen gehen dahin die logische Struktur eines Netzes von der physischen Struktur zu entkoppeln. In einem klassischen LAN sind alle

[1]In der Literatur werden häufig virtuelle LANs mit (virtuellen) Workgroups gleichgesetzt. Diese Gleichsetzung ist nur dann richtig, wenn es eine eindeutige Abbildung zwischen Benutzer und Arbeitsstation gibt. Aufgrund der Forderung nach Mobilität und der Benutzung einer Arbeitsstation durch mehrere Benutzer (Multiuser-Systeme) ist diese Terminologie nicht gerechtfertigt.

In dieser Arbeit ist ein VLAN eine Gruppe von Rechnern, eine Workgroup ist eine Gruppe von Personen. Im Vordergrund stehen die VLANs. Virtuelle Workgroups, als darauf aufbauende zukünftige Szenarien, sind zwar bereits angedacht (z.B. [For95]), aber, ohne Änderungen an bestehender HW, SW vorzunehmen und die ohne Einbeziehung der Anwender, nicht durchführbar.

Ressourcen streng an die pysischen Struktur des Netzes gebunden, die meist nach Gebäuden, Etagen und sogar noch feiner nach Räumen unterteilt sind. Die physische Struktur eines Netzes gibt die logische Struktur vor. Virtuelle Netze brechen diese Struktur auf, die logische Struktur des Netzes wird von der darunterliegenden Architektur weitgehend entkoppelt[2].

Ein VLAN ist zunächst einmal definiert als eine Broadcast-Domäne, d.h. die Mitglieder (Endstationen) werden dieser Broadcast-Domäne unabhängig von ihrem geographischen Standort zugeordnet. Die Domäne defininiert sich also logisch und nicht geographisch. Routing ist für die Kommunikation zwischen den Mitgliedern eines VLANs über mehrere Segmente[3] hinweg nicht mehr erforderlich.

Somit ist ein VLAN nunmehr eine logische Gruppierung von Arbeitsstationen und Serverdiensten, unabhängig von deren geographischen und der verwendeten Technologie. Server(dienste) können in zukünftigen Szenarien[4] so in zentralen Räumen zur besseren Verwaltung (Kühlung, Vorsorge für Stromausfall, usw.) plaziert und dennoch entfernten VLANs und Workgroups zugeordnet werden.

VLANs, realisiert mit Hilfe der Switching-Technologie[5] , sind durch die im folgenden zusammengefaßten Ziele motiviert:

- Entkopplung der logischen von der physischen Netzstruktur
- hohe Flexibilität beim Netzdesign
- flachere Netzstrukturen (unter Verwendung der Switching-Vermittlungstechnik) bei gleichzeitig höherer Bandbreite
- einfacheres Konfigurationsmanagement bei Änderung und Umzügen im Netz (selbständige Netzkonfiguration, usw.)

2.2 Herkömmliche Strukturierung von Netzen

In klassischen LANs sind vernetzte Arbeitsstationen an ein gemeinsames Segment (Shared Medium) angeschlossen. Je weniger Arbeitsstationen sich ein Segment teilen, desto größer ist die verfügbare Bandbreite. Da aber ein Unternehmensnetz sich dynamisch ändert und laufend neue Arbeitsstationen hinzukommen, steigt die

[2]Diese Idee findet sich bereits in jedem "Appletalk"-Netz. Neu ist nur der Begriff virtuell.

[3]Unter einem Segment wird im weiteren Verlauf der Arbeit ein physisches LAN-Teilnetz verstanden, in dem jede teilnehmende Endstation den gesamten Verkehr mithören kann (Schicht 2 Kollisionsdomäne).

[4]Wenn auch einzelne Dienste den VLANs zugewiesen werden können.

[5]Es wird stillschweigend immer vorausgesetzt, daß die Switching-Komponenten VLAN-fähig sind. Die dazu notwendigen Grundlagen und Erweiterungen werden noch ausführlich dargelegt.

Verkehrslast und die Verfügbarkeit des Netzes nimmt ab. Der Administrator ist nun gezwungen das Netz durch Subnetzbildung neu zu strukturieren, indem er zunächst einmal versucht die Netzlast mit Hilfe von Brücken (Bridges) möglichst lokal zu halten. Da Brücken auf der Sicherungsschicht (im ISO-OSI-Referenzmodell) arbeiten, ergeben sich ab einer bestimmten Netzgröße Probleme mit den Schicht-2- und Schicht-3-Broadcasts. Kommt es zu Kapazitätsengpässen, beginnt man üblicherweise, das vorhandene LAN mittels (Multiport-)Router zu segmentieren. Damit werden Broadcasts herausgefiltert und man erreicht wieder eine höhere Verfügbarkeit des Segments. Im Extremfall führt diese Mikrosegmentierung zu einem dedizierten Anschluß, bei der jede Arbeitsstation ein eigenes LAN-Segment erhält und dediziert mit der vollen Bandbreite an den Routerport bzw. Bridgeport angeschlossen wird (*dedicated Ethernet*, wenn es sich um ein Ethernet-LAN handelt). Aber jedes physische Segment wird in der Regel als separates Subnetz betrachtet und benötigt deshalb einen eigenen (Sub-)Adreßraum, der z.B. in der Internet-Welt in absehbarer Zeit erschöpft sein wird. Abgesehen davon sind Routerports relativ teuer. Da jedes Segment nun ein eigenes Subnetz bildet und folglich der Verkehr zwischen Subnetzen jedesmal den Router passieren muß, was aufgrund des Mehraufwands an Rechenzeit zu signifikanten Verzögerungen des Datentransports führt.

Die Strukturierung von Unternehmensnetzen mit Hilfe von Brücken und Routern stößt somit an ihre Grenzen, namentlich im folgenden nocheinmal aufgeführt:

- hohe Strukturtiefen im Netz und die damit verbundenen neu auftretenden Probleme (knappe Adreßräume, usw.),
- Durchsatzprobleme bei Routern,
- steigende Ausfallwahrscheinlichkeit von Verbindungen,
- höhere Administrations- und Konfigurationsaufwand und
- eingeschränkte Flexibilität beim Umkonfigurieren

Die Switching-Technologie verspricht eine Minimierung einiger dieser Probleme, indem sie u.a. den Verkehr zwischen verschieden Segmenten, die aber zum gleichen Subnetz gehören sollen (=VLAN), auf Schicht-2 anstatt auf Schicht-3 abarbeiten. Somit kann sich ein Subnetz wieder über mehrere Segmente erstrecken. Auf die Switching-Technologie wird in einem späteren Abschnitt noch näher eingegangen.

2.3 Voraussetzungen für virtuelle Netze

Zwar sind virtuelle LANs auch mit herkömmlichen Netzkomponenten wie Brücken und Router realisierbar, aber zusätzlich zu den jetzt schon kaum beherrschbaren Administrationsaufwand in einem heterogenen Netz (ab einer gewissen Größe), müssen auch auf einer höheren Abstraktionsstufe die virtuellen LANs verwaltet und auf das physische Netz abgebildet werden. Wie im Abschnitt vorher erläutert ist dieses Vorgehen aufgrund der jetzt schon überlasteten Netze und dem nicht vertretbaren zusätzlichen Aufwand an Zeit und Personal nach betriebswirtschaftlichen Gesichtspunkten nicht mehr durchführbar. Ziel muß es sein die Netzstrukturen zu vereinfachen, bei gleichzeitiger Erhöhung der Bandbreite.

Im folgenden werden die Voraussetzungen für den Aufbau virtueller LANs aufgeführt:

- eine strukturierte Verkabelung (Mikrosegmentierung, sternförmige Topologien),
- der Einsatz von Switching-Komponenten,
- leistungsfähige Backbones (ATM, FDDI, Fast-Ethernet, usw.),
- das durchgängige Netzmanagement aller Netzkomponenten.

Eine strukturierte Verkabelung ist notwendig, um die verfügbare Bandbreite auch optimal zu nutzen. Dies kann durch die bereits angesprochene Mikrosegmentierung und im Idealfall durch ein dediziertes Anschließen der Endstationen an Switching-Komponenten erreicht werden. Die Topologien die sich daraus ergiben sind sternenförmig, mit den Switching-Komponenten als Mittelpunkt. Switching-Komponenten, die im nächsten Abschnitt besprochen werden, sind schnelle Netzkomponenten, konzipiert zur Vereinfachung der Netzstrukturen und durch die VLANs (mit einem vertretbaren Aufwand und annehmbarer *Performance*) erst ermöglicht werden können.

Das Konzept der VLANs unterstellt, daß Hochgewindigkeitsbackbones, zur Verbindung der Switches untereinander, vorhanden sind. Die Einrichtung virtueller LANs darf keine Verschlechterungen noch sich ziehen, d.h. es dürfen keine Verzögerungen, als Folge der Virtualisierung, in den Kommunikationsbeziehungen innerhalb und zwischen den VLANs auftreten. Die Hochleistungsbackbones zusammen mit den Switching-Komponenten sorgen in diesem Konzept für Ortstransparenz. Bei den verwendeten Backbones kann es sich dabei um beliebige Technologien handeln (FDDI, DQDB, Fast Ethernet, ATM, usw.)[6]. Um bestehende Investitionen zu schützen und um damit die Migration zu ATM zu erleichtern, kann die ATM LAN Emulation implementiert werden. Da diese Lösung für die Interswitchkommunikation bereits in einer ersten Version standardisiert ist und praktisch jeder ATM-Komponentenhersteller heute eine LAN Emulation angekündigt hat oder schon anbietet, wird die ATM LAN Emulation in Abschnitt 2.5 besprochen.

Die Forderung nach einem durchgängigen Netzmanagement erscheint selbstverständlich, deshalb wird dieser Bedarf auch von allen Herstellern erkannt. In Hinblick auf

[6] Auf die Probleme die dabei auftauchen, wird in Abschnitt 2.4.2 noch eingegangen.

sich dynamisch ändernde VLANs und der damit eingeschoben logischen Abstraktions-Schicht ist eine Management der Komponenten unabdingbar. Genauso müssen Konzepte gefunden werden, diese neue Abstraktionsschicht der VLANs in das Netz- und Systemmanagement einzuführen. Diese Arbeit soll die Anforderungen an eine graphische Bedienoberfläche für das VLAN-Management erarbeiten und konkrete Vorschläge liefern.

2.3.1 Switching-Technologie

Die Geburtsstunde der Switches ist eng verknüpft mit dem massiven Wachstum der lokalen Netze und deren Leistungseinbruch. Die Folge sind kaum noch beherrschbare Kommunikationsverbunde, die sich nur noch mit enormen Aufwand verwalten ließen. Switching-Komponenten vereinen bewährte Eigenschaften bisheriger (Multiport-)Brücken (Spanning Tree Algorithmus) und Router (Filtermechanismen, Routing im Campus-Bereich) und finden ihren Einsatz hauptsächlich in den unteren drei OSI-Schichten. Ein Switch ist also auf den ersten Blick eine portmäßig und hinsichtlich der Anzahl der Ports erweiterte und hinsichtlich der Koppelfunktionalität (Filter, Paketpuffer) abgespeckte Multiport-Brücke. Dabei ersetzen sie die Brücken vollständig und die Router, bis auf die notwendige Funktionalität für die MAN- und WAN-Kopplung.

Im Gegensatz zu den bereits bekannten Funktionalitäten, die die Switches anbieten, gehört die Architektur einer neuen Gerätegeneration an, da zur Paketverarbeitung ASICs[7] anstelle von den langsameren CISC- und RISC-Prozesoren mit Betriebsfirmware Eingang finden und dadurch die switch-internen Busse (Backplanes) u.a. höhere Durchsatzkapazitäten erreichen.
Was versteht man aber unter Switching?

Switching: Switching ist eine Technologie zur Aufteilung und Regelung von Datenflüssen, um auf MAC-Ebene dedizierten Verkehr zwischen Quelle und Ziel zu optimieren [DKLDSW96].

Nach dieser doch recht allgemeinen Definition, hier eine praktische Umsetzung anhand der Arbeitsweise eines Switches: Switches können, nachdem Header(-anfänge) der Pakete/Zellen/Frames gelesen (z.B. MAC-Adressen) wurden, diese auf einen beliebigen Ausgang schalten. Aufgrund der hohen Verarbeitungs-Kapazität der switch-internen Busse (Backplanes) und ASICs kann die Kommunikation, im Unterschied zu einer klassischen Brücke, für viele Port-Paare (quasi-)parallel durchgeführt werden.

[7] Applikation Specific Integrated Circuit.

2.4 Klassifizierung der Switches

Aufgrund der Vielzahl der Produkte die den Namen Switch enthalten, wird im folgenden Abschnitt versucht die Produktpalette *Switch* zu klassifizieren. Dabei liefern die unten aufgeführten Kriterien verschiedene Ansatzpunkte:

- Art der Paketverarbeitung
- OSI-Schicht Zuordnung

2.4.1 Paketverarbeitung

LAN-Switches lassen sich bzgl. Paketverarbeitung in drei Typen kategorisieren:

- Cut-Through-Switches oder On-The-Fly-Switches
- Store-and-Forward-Switches
- Cell-Oriented-Switches

Am Markt befinden sich bereits hybride Produkte, bei denen der Benutzer selbst bestimmen kann, welche Ports im Cut-Trough- oder Store-and-Forward-Verfahren arbeiten soll.

Cut-Through-Switches (CT-Switches)

Cut-Through-Switches lesen die ankommenden Frames ohne Zwischenspeicher, d.h. sie werden "on the fly" ausgewertet. Nachdem die MAC-Zieladresse gelesen wurde, wird diese im Adreßpuffer zwischengespeichert und ausgewertet. Die Quelladresse wird zum automatischen Aufbau von Adreßtabellen (Selbstlernfunktion) analysiert. Mit Hilfe dieser intern verwalteten Adreßtabelle wird nach dem Lesen der ersten 12 Bytes eine dedizierte Verbindung zwischen Ein- und Ausgangsport durchgeschaltet und die Daten können somit unverzüglich weitervermittelt werden. Gleichzeitig können noch weitere parallele Verbindungen zwischen den Switch-Ports über denn internen Vermittlungsbus geschaltet sein. Pufferspeicher ist nur dann notwendig, wenn gleichzeitig mehrere Pakete zum selben Ausgangsport vermittelt werden sollen.

Ein Nachteil von Cut-Through-Switches ist, daß sich nur Ports mit gleicher Datenrate anschließen lassen. In Client/Server Umgebungen ist das Cut-Through–Verfahren nicht geeignet, da der Server eine weitaus größere Bandbreite benötigt. Ein weiterer Nachteil liegt in der Tatsache begründet, daß fehlerhafte Frames und Kollisionsfragmente weitergeleitet werden, da nur der Frameheader analysiert und keine Fehlererkennung vorgenommen werden kann.

Store-and-Foreward-Switches (SF-Switches)

Store-and-Forward-Switches erinnern stark an herkömmliche Brücken. Im Gegensatz zu den Cut-Through-Switches werden ankommende Frames komplett in eine Eingabewarteschlange eingelesen. Dort können dann Fehlerberechnungen und andere (Filter-)Prozeduren durchgeführt werden. Die Ermittlung des Ausgangsports erfolgt wie bei den Cut-Through-Switches. Die überprüften Frames werden anschließend in die entsprechenden Ausgangswarteschlangen geschoben und schließlich am Ausgangsport ausgegeben.

Mit den Store-and-Forward-Switches werden die Nachteile der Cut-Through-Switches behoben. Durch die Pufferung der Frames können unterschiedliche Portgeschwindigkeiten unterstützt werden und sind damit für Client/Server Kommunikationsbeziehungen geeignet. Außerdem werden fehlerhafte Frames und Kollisionsfragmente erkannt und verworfen.

Der Nachteil der sich durch die Pufferung und der Analyse des Gesamtframes ergibt ist die erheblich größe Durchlaufzeit eines Frames[8]. Da die Durchlaufzeit auch von der tatsächlichen Framelänge abhängig ist, ist dieses Verfahren für einen kontinuierliche Datenfluß, wie er bei der Übertragung von Sprach- und Bewegtbildanwendungen benötigt wird, nur eingeschränkt tauglich.

Eignung von Cut-Through- und Store-and-Forward-Switches zur Bildung von VLANs

VLAN-Funktionalität läßt sich sowohl in bei Cut-Through- als auch bei Sore-and-Forward-Switches finden. Reine Cut-Through-Switches können allerdings nur beschränkt VLANs realisiert werden, da der Frameinhalt bis auf die Adressen nicht weiter ausgewertet wird. Es lassen sich somit nur VLANs anhand der Gruppierung von MAC-Adressen oder Ports bilden. SF-Switches sind für den Aufbau von VLANs bestens geeignet. Die Tatsache, daß sie die Pakete im Ganzen analysieren und dadurch die Frames nach definierbaren Kriterien gefiltert werden können, ermöglicht eine größere Flexibilität bei der VLAN-Bildung. So können VLANs bzgl. MAC-Adressen, Port-Gruppen, Protokolltyp, Schicht-3-Adressen oder Kombinationen davon gebildet werden.

Cell-Oriented-Switches

Cell-Oriented-Switches vereinen die Vorteile von Cut-Through- und Store-and-Forward-Switches. Der Nachrichtenfluß im Netz besteht aus Zellen fester Größe[9]. Da sich die notwendigen Filterinformationen in den ersten Bytes befinden, erhält man

[8]Bei einem Ethernet Frame maximaler Größe beträgt der Unterschied etwa den Faktor 12, um das Adreßfeld zu analysieren.

[9]Bei ATM 53 Bytes.

für den Aufbau von virtuellen Netzen dennoch die gewünschte Filtermöglichkeiten. Die Verzögerungszeit liegt zwischen dem eines Cut-Through- und SF-Switches.

Aufgrund des Transports gleichlanger Zellen ist eine Kontinuität des Datenflusses gewahrt, was zu einer guten Übertragungsqualität von Sprach- und Bewegtbildanwendungen führt.

2.4.2 OSI-Schicht-Zuordnung

Switches werten, je nach verwendeter Technologie, Schicht-2- und/oder Schicht-3-Informationen aus. Je höher die Schicht, die ein Switch interpretieren kann, desto höher der Abstraktionsgrad vom physischen Netz und desto mehr Freiheitsgrade hat man beim Konfigurieren der VLANs.

Port-Switching, Konfigurationsswitching

Die "Virtualisierung" auf physischer Ebene, der OSI-Schicht 1, wird heute von allen Switch- und Hub-Herstellern angeboten. Sie wird mit Portswitching oder seltener Konfigurationsswitching bezeichnet. Dabei werden LAN-Segmente an verschiedenen Ports angeschlossen. Einzelne oder auch Gruppierungen von Ports werden zu einem VLAN zusammengefaßt. Alle Endstationen eines Segments können in der Regel nur einen VLAN zugeordnet sein. Wenn sich die VLAN-Zuordnung ändert, muß allerdings manuell neu konfiguriert werden. Auf diese Weise lassen sich VLANs aufbauen, ohne daß eine Erweiterung der Adreßtabellen notwendig ist. Der *Unicast*-Verkehr zwischen den zusammengefaßten Ports wird durch Auswertung der MAC-Adressen weitergeleitet, der *Broadcast*-Verkehr wird an alle Ports des VLANs geflutet.

Größere Flexibilität wird durch die Virtualisierung auf einer höheren Abstraktionsebene, der MAC-Schicht (OSI-Schicht 2 a), erreicht.

Layer-2-Switches

Layer-2-Switches kann man als durchsatzstarke Multiport-Brücken im neuen Gewand ansehen. Im weiteren Verlauf der Arbeit werden Switches (Layer 2 und 3) vorausgesetzt, die sich zur Bildung von VLANs eignen, d.h. daß die Adreßtabellen erweiterbar sind.

Beliebige MAC-Adressen werden einem VLAN zugeordnet. Somit wird der Verkehr innerhalb eines VLANs mittels MAC-Adressen und dem VLAN Identifikator in den verschiedenen physischen Segmenten weitergeleitet. Broadcasts werden an alle Portausgängen, die mit einem VLAN assoziiert sind, weitergeleitet. Für den Verkehr zwischen verschiedenen VLANs müssen weiterhin Router eingesetzt werden. Dies können seperate Router-Komponenten oder als Steckkarten in den Switches integriert sein.

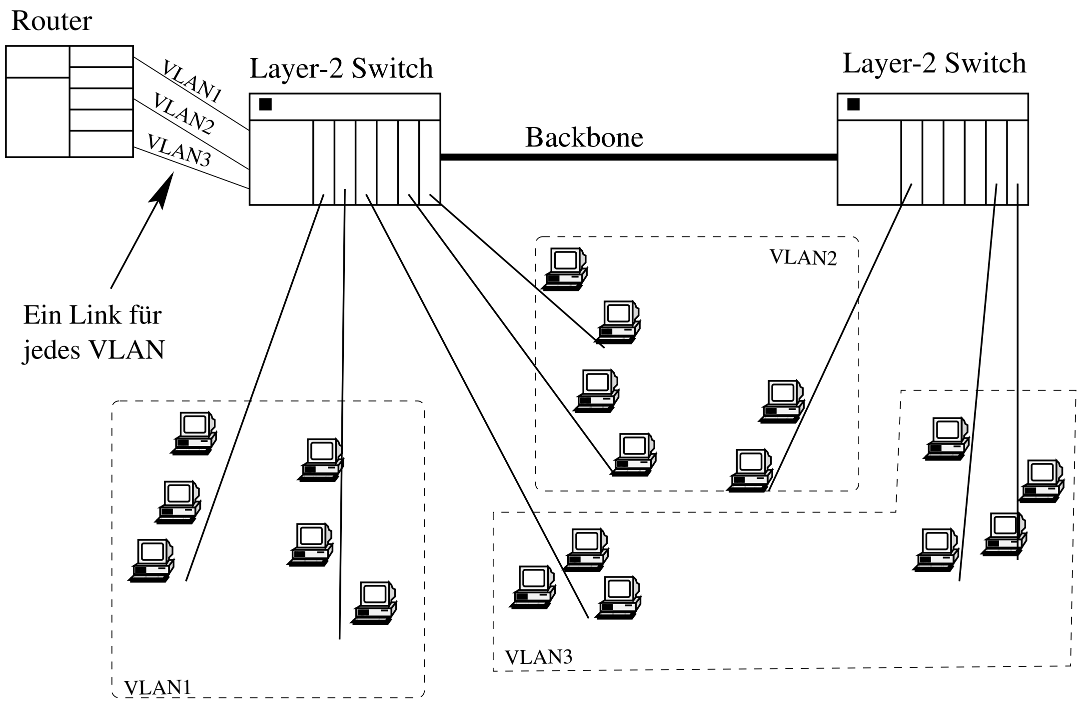


Abbildung 2.1: Layer-2-VLANs

Layer-2-Switches arbeiten protokollunabhängig, was zum einen die Unterstützung nichtroutbarer Protokolle erlaubt und zum anderen aber keine protokollorientierten VLAN-Bildungen ermöglicht. Layer-2-Switches sind einfach zu konfigurieren, weil nur Port-Gruppen für die Vermittlungswege zu definieren sind. Wegen der einfacheren Software/Logik sind Layer-2-Switches günstiger und schneller als Layer-3-Switches.

Probleme entstehen, wenn VLANs switchübergreifend, d.h. mit mehreren Switches, die über leistungsstarke Backbones miteinander verbunden sind, aufgebaut werden. Um die Adreßtabellen der einzelnen Switches konsistent zu halten, müssen sie regelmäßig abgeglichen oder ausgetauscht werden. Für den Austausch der Adreßtabellen benötigt man spezielle Protokolle (Interswitchkommunikation).

Folgende drei Verfahren haben sich bis heute, für die Kommunikation der Layer-2-Switches untereinander, herauskristalisiert:

- Signaling Message
- Time Division Multiplexing (TDM)

- Frame Tagging

Der komplette Austausch von Adreßtabellen wird, wegen des beachtlichen Overheads, von vornherein ausgeschlossen.

Signalling Message Bei dem *Signaling Message* Verfahren tauschen die Switches untereinander spezielle Datenpakete aus, um ihre Adreßtabellen konsistent zu halten. Zieht eine Endstation in ein anderes Subnetz oder wird an ein anderes Ethernetsegment angeschlossen für das ein anderer Switch zuständig ist, so müssen sich der "alte" und "neue" Switch untereinander verständigen, damit Letzterer die Endstation dem richtigen VLAN zuordnen kann.

Dieses Benachrichtigungsverfahren ist einfach zu implementieren, jedoch gibt es hierfür keinen Standard, so daß die Hersteller auch hier wieder eigene Wege gehen. Außerdem wird durch dieses Verfahren Overhead erzeugt, der bei größeren Netzen zu erheblichen Synchronisations- und Überlastproblemen führen kann.

Time Division Multiplexing (TDM) Ein weiterer Ansatz zum Schicht-2 Informationsaustausch unter den Switches ist das *Time Division Multiplexing*. Dazu wird der Switch-Backbone in Time-Slots fester Bandbreite aufgeteilt. Jedem VLAN werden dann ein oder mehrere dieser Time-Slots zur exklusiven Nutzung zugeteilt. Die Switches sind dann zusätzlich bezüglich der Zuweisung von Port-Gruppen zu TDM-Kanälen zu konfigurieren.

Mit diesen Ansatz entfällt jede Notwendigkeit Adreßtabellen untereinander auszutauschen oder die Pakete zu modifizieren (siehe nächten Abschnitt: Frame Tagging). Jegliche Form von Overhead im Switch-Backbone Bereich ist somit vermieden. Einzig die von einem VLAN nicht benutzte Bandbreite kann einem anderen nicht dynamisch zur Verfügung gestellt werden und liegt somit brach. Diese Tatsache erfordert daher ein ständiges Verkehrsbeobachten des Switch-Backbone, um die TDM-Kanäle optimal zu vergeben und effizient auszunutzen.

Frame Tagging Das *Frame Tagging* ist eine weitere Methode, die Switches benutzen um ihre Adreßtabellen konsistent zu halten. Bei dieser Methode wird jedem Datenpaket ein sogenanntes *Tag* vorangestellt, welches u.a. die VLAN-Zugehörigkeit des Pakets beinhaltet. Damit ist der Austausch zur Konsistenzhaltung der Adreßtabellen nicht mehr notwendig, da der VLAN Identifikator in jedem Paket enthalten ist.

Ein unerwünschter Seiteneffekt entsteht, wenn das Paket bereits die maximal zulässige Länge hat und durch das zusätzliche *Tag* die Protokoll-Vorgaben verletzt werden. Brücken würden solche Pakete verwerfen. Switch-Hersteller umgehen dieses Problem, indem sie wieder proprietäre Techniken einbauen, um die Regeln der maximal vorgeschriebenen Paketlängen zu umgehen, was wiederum bedeutet: die einzusetzenden Switches müssen vom selben Hersteller sein oder zumindestens das gleiche Frame Tagging Verfahren verwenden.

Eine weitere Frame Tagging Variante ist die Bildung von VLANs nicht nach Port-Gruppen, sondern nach Feldinformationen (z.B. Subnetz-Adressen, Protokoll-Typ) des Pakets zu richten. Diese Flexibilität erlaubt somit beispielsweise die Zusammenfassung nicht-routfähiger Protokolle zu virtuellen LANs. Nachteilig an dieser Frame Tagging Variante ist die erhöhte Verarbeitungszeit bei der Auswertung und die Administration der Routing-Tabellen, die umfangreiche Protokollkenntnisse erfordern. Dabei ist unverkennbar, daß solche Switches Routing/Schicht-3-Funktionalitäten mit entsprechenden Filtermethoden umfassen.

IEEE 802.10
Eine Möglichkeit zur herstellerübergreifenden Lösung des Frame Taggings bestünde darin Teile des Sicherheitsstandards IEEE 802.10 (Secure Data Exchange) zu implementieren.

IEEE 802.10 beschreibt ein Verfahren zum Anfügen von Verschlüsselungs- und Authentifizierungsinformationen an ein Datenpaket. Dazu werden die MAC-Frames um einen 32 Bit langen Group Identifier erweitert und in einen größeren MAC-Frame eingepackt (Encapsulation). Es ist auch definiert, wie Pakete maximaler Größe zerlegt und später wieder zusammenzusetzen sind, um durch Anfügen der zusätzlichen Informationen die maximal erlaubte Paketlänge nicht zu überschreiten. Dieses Verfahren ist für Ethernet, FDDI, Token Ring und HDLC festgelegt und wird von einigen Herstellern wie Cisco bereits verwendet und eignet sich für das Frame Tagging in Virtuellen Netzen.

Layer-3-Switches

Layer-3-Switches besitzen Basis-Routingfunktionalitäten und ziehen diese bei der Bildung von VLANs mit ein (Stichwort "Layer-3 VLAN"). Im Gegensatz zu Layer-2-Switches sind sie protokollsensitiv; sie können Schicht-3-Informationen auswerten und somit virtuelle LANs nach (Schicht-3) Protokollinformationen zusammenfassen. Dadurch ist es möglich, z.B. daß Subnetzadreßfeld des IP-Protokolls und anderer routbarer Protokolle zur Bildung von VLANs heranzuziehen. In der Regel entspricht eine Subnetzadresse einem VLAN. Ein Port kann mit mehreren Subnetzadressen belegt werden. Somit kann sich ein Subnetz bzw. VLAN über mehrere Segmente erstrecken und mehrere VLANs auf einem Segment eingerichtet bzw. mit einem Port assoziiert sein. Bei nichtroutbare Protokollen wie z.B. NetBIOS und LAT[10] kann ein VLAN nur nach den Protokolltyp gebildet werden. Eine weitere Strukturierung ist nicht möglich. Abbildung 2.2 zeigt schematisch ein Layer-3-VLAN. Die Zuordnung

[10]NetBIOS: Network Basic Input/Output System
LAT: Local Area Transport, ein proprietäres Protokoll von Digital Equipment Corp. (DEC).

der einzelnen Pakete zwischen verschiedenen VLANs erfolgt analog dem klassischen Routing durch Auswertung der Netzadresse, nichtroutbare Protokolle werden weiterhin *"geswitched"*.

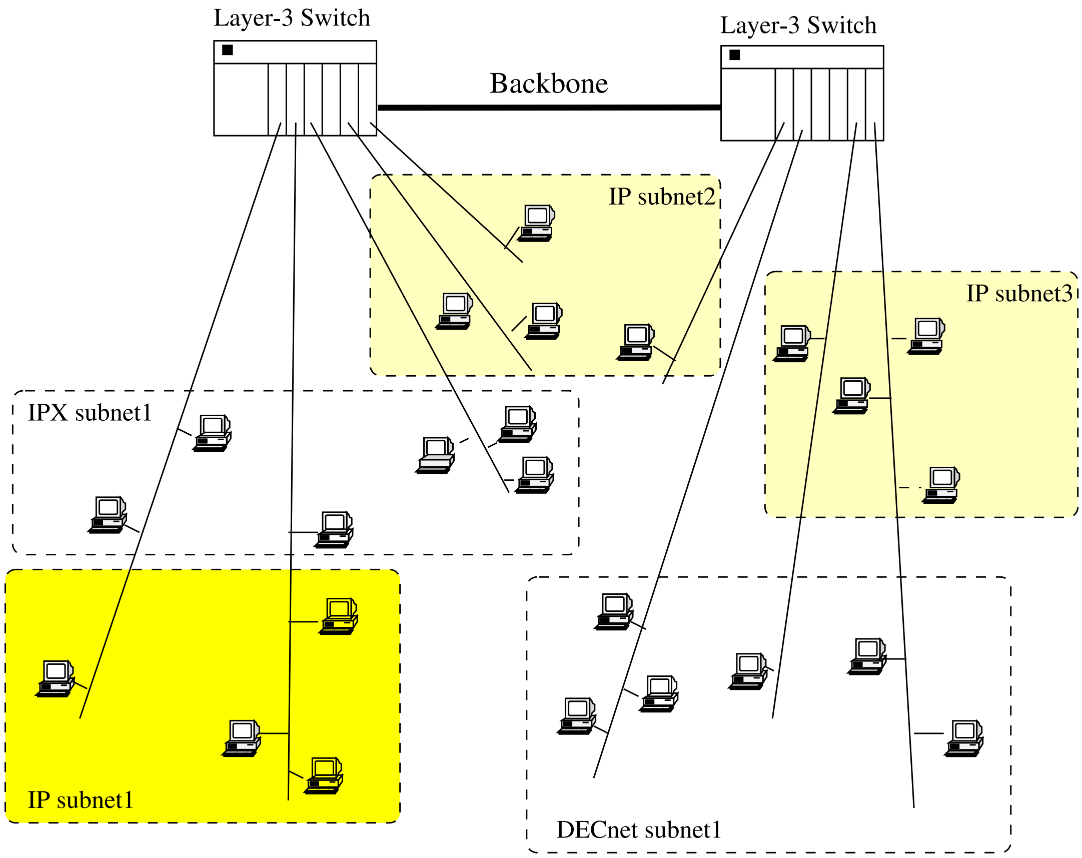


Abbildung 2.2: Layer-3-VLANs

Obwohl diese (Campus-)Routingfunktionalität durch externe Router erbracht oder in Zukunft in den Switches implementiert sein wird, werden trotzdem noch externe Routingkomponenten für die MAN- und WAN-Kopplung benötigt.

Der Datenverkehr innerhalb der VLANs wird weiterhin so behandelt als läge eine Brückentopologie vor; die Routingfunktionen greifen nur beim Datenverkehr zwischen den VLANs. Werden die VLANs mit Sorgfalt konfiguriert, so kann der Netzadministrator damit flachere Netzstrukturen erreichen, d.h. die rechenintensiven Schicht-3-Übergänge werden weniger, obwohl einzelne Endstationen des VLANs an verschiedenen Segmenten angeschlossen sein können. Durch diese Flexibilität besteht aber auch die Gefahr, durch ungeschicktes Konfigurieren der VLANs und der Nichtbeachtung evtl. physischer Besonderheiten (z.B. Voraussetzungen für VLANs nicht erfüllt), die Verfügbarkeit des Netzes zu verschlechtern. Daraus erwächst die

Forderung, die für diese Arbeit auch eine wichtige Rolle spielt, nach einer automatischen Überwachung und Historisierung des Verkehrs.

Die oben beschrieben Mechanismen des Schicht-2-Informationsaustausches sind hier nicht mehr notwendig, da die Layer-3-Switches auf Ebene-3 mit Hilfe der evtl. modifizierten Routingprotokolle RIP oder OSPF[11] (für jedes VLAN, in jedem Switch) die Adreßtabellen selbständig aktualisieren.

2.5 ATM LAN Emulation

Die ATM LAN Emulation (LANE, LE) emuliert herkömmliche lokale Netze (LANs) auf Basis der Breitbandtechnologie ATM. Viele Hersteller sehen darin — trotz unvollständiger Ausnutzung der Vorteile des ATM — die einfachste Möglichkeit, zu ATM zu migrieren und bestehende Hardware- und Softwareinvestitionen zu schützen. In dieser Arbeit wird die LAN Emulation als Backbone zur Verbindung von ATM-fähigen Netzkomponenten bzw. Ethernet-Switches, in denen die LANE implementiert ist, betrachtet. D.h. die LANE sorgt für Ortstransparenz, daß ATM-Netz als solches ist, außer im Fehlerfall oder bei der Analyse und Optimierung des Verkehrs und der Auslastung, nicht sichtbar.

In diesen Abschnitt werden die, für das weitere Verständnis der Arbeit, wichtigsten Grundlagen, Begriffe und Konzepte der ATM LAN Emulation besprochen. Abschnitt 2.5.1 geht auf die Architektur und Komponenten der LAN Emulation ein und Abschnitt 2.5.2 beschreibt, wie sie zur Bildung von VLANs mit einbezogen werden kann.

2.5.1 Architektur und Komponenten der ATM LAN Emulation

Das LANE-Protokoll definiert entsprechend der LANE-1.0-Spezifikation des ATM-Forums [Eea95] die Arbeitsweise eines emulierten LAN (ELAN) und basiert auf den vier Komponenten LANE Client (LEC), LANE Server (LES), Broadcast and Unknown Server (BUS) und LANE Configuration Server (LECS). Als Client-Server-Modell konzipiert stellen die LECs die Klienten *(Clients)* und die restlichen Komponenten die Server dar, die zusammen einen *MAC-Service* anbieten. Der LAN-Typ eines ELANs ist entweder IEEE 802.3 (Ethernet) oder IEEE 802.5 (Token Ring). Es können bis zu 1024 ELANs in der ersten Version des Standards eingerichtet werden, jedes davon kann nur Teilnetze vom gleichen LAN-Typ verbinden. Abbildung 2.3 zeigt den logischen Aufbau der LAN Emulation.

[11] RIP: Routing Information Protokoll; OSPF: Open Shortest Path First.

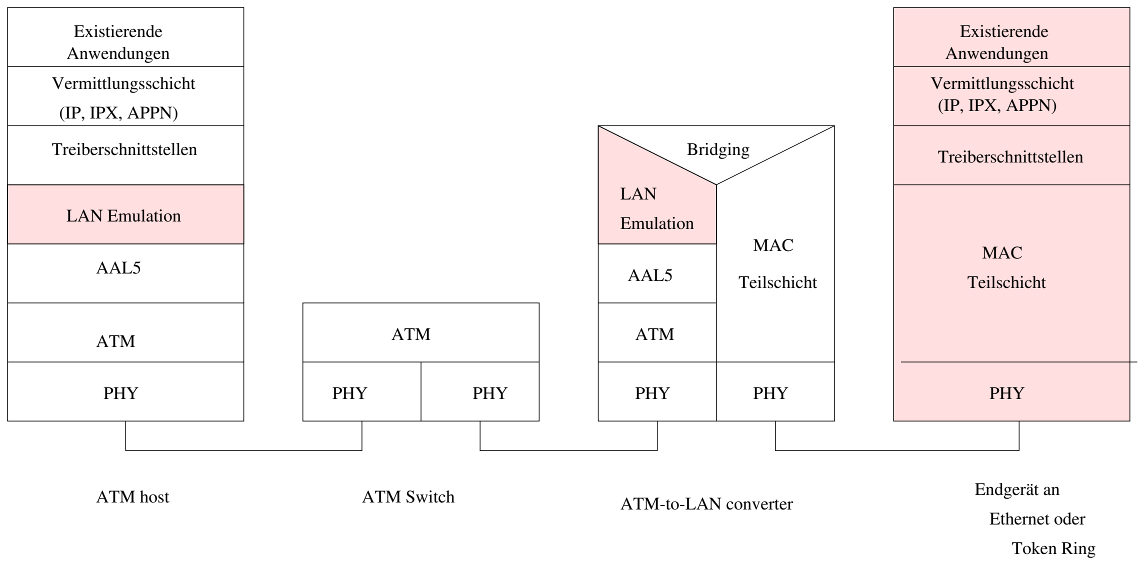


Abbildung 2.3: Logischer Aufbau der LAN-Emulation

Die *LE Clients (LECs)* sind in ATM LANE-fähigen Endstationen oder anderen Vermittlungskomponenten (Hub-Einschubmodule, Brücken, Router oder ATM-Switches) implementiert. Eine solche ATM-LANE-fähige Komponente wird im weiteren Verlauf der Arbeit mit ATM/LAN-Konverter [Kau96] oder *Edge Device* bezeichnet. Es wird davon ausgegangen, daß die Komponenten der LAN Emulation in diesen Konvertern implementiert sind (verteilt oder zentral).
Ein LEC sorgt u.a. für die Umwandlung der (MAC-)Frames[12] in einen ATM-Zellenstrom und umgekehrt. Desweiteren ist er für die Adreßauflösung *(Address Resolution)*, im Normalfall für den Verbindungsauf- und -abbau und andere Kontrollfunktionen zuständig. Er stellt den höheren Schichten ein MAC-Schnittstelle zur Verfügung, genauer: er ist die LUNI[13]-Schnittstelle zum ATM-Netz [Eea95].

Der *LE Server (LES)* kontrolliert und koordiniert ein spezielles ELAN, indem er die zugehörigen LECs registriert und eine Tabelle mit den korrespondierenden ATM/ MAC-Adreßpaaren bereithält. LECs können bei ihrer Instantiierung bei den LES MAC-Adressen von (LAN-)Endstationen registrieren lassen, die sie repräsentieren. Auch besteht für den LEC die Möglichkeit dem ELAN als Proxy beizutreten, d.h. nichts anderes, als daß sich auf der LAN-Seite des LECs noch mehrere (unbekannte) Endstationen befinden können, für die im Augenblick keine Eintragungen in den Adreßtabellen vorgenommen werden sollen oder können[14].

[12]Definiert für Ethernet, und Token Ring. Für diese Arbeit wird stellvertretend, wie später an geeigneter Stelle noch dargelegt wird, Ethernet besprochen.
[13]LAN Emulation User Network Interface.
[14]Vorteilhaft, um bei vielen Endstationen die Adreßtabellen der LECs klein zu halten.

Der LEC kontaktiert den LES, wenn er einen MAC-Frame zu übertragen hat und seine eigene Adreßtabelle die ATM-Adresse des korrespondierenden LECs, der zuständig ist für die Zielstation, nicht enthält. Gleichzeitig mit der LE_ARP *(LAN emulation address resolution protocol)* Aufforderung *(Request)* an den LES sendet er die Unicast Nachricht an den BUS. Der LES wird ihm entweder direkt antworten oder die Anfrage an andere Clients weiterleiten. Die zurückgemeldete Adresse wird dazu benutzt, die Adreßtabelle des anfragenden LECs zu aktualisieren.

Der *Broadcast and Unknown Server (BUS)* behandelt die Broadcast MAC-Adressen. Dieser Server ist nötig, um eine verbindungslosen Dienst, wie es bei einem Ethernet oder Token Ring LAN der Fall ist *(Shared Medium)*, in einem verbindingsorientierten Netz, wie es das ATM-Netz darstellt, anzubieten. Der LEC sendet ein Broadcastframe direkt an den BUS, dieser leitet es dann an alle LECs, die zu dem ELAN gehören, weiter. Auch werden Unicast Nachrichten, dessen MAC-Adresse dem LEC noch nicht bekannt sind, an den BUS gesendet und von diesem an alle Mitglieder des ELANs weitergeleitet.

Der *LAN Emulation Configuration Server (LECS)* ermöglicht die Anbindung einzelner LECs an verschiedenen ELANs[15]. Die Anbindung erfolgt, indem der LECS einen — sich in der Initialisierungsphase befindlichen — LEC die ATM-Adresse des LES bekanntgibt, der für das ELAN zuständig ist, dem er beitreten will. Pro Administrationsdomäne (ATM-Netz) gibt es nur einen LECS.

Abbildung 2.4 zeigt alle Komponenten der LAN Emulation im Zusammenspiel schematisch.

Dennoch kann die LAN Emulation nicht so ohne weiteres eingesetzt werden, da noch wichtige Punkte in der Version 1.0 ungeklärt sind.
Der ATM LAN Emulation Standard beschreibt nur wie ein einzelnes ELAN funktioniert, nicht wie mehrere ELANs implementiert werden. Sollen verschiedene VLANs auf ebensovielen ELANs abgebildet werden, so wird je VLAN ein eigener LES und BUS benötigt. Im Gegensatz dazu reicht ein LECS pro Administrationsdomäne. Zum Austauch von Daten zwischen verschiedenen ELANs muß auf herkömmliche Router-Komponenten mit allen seinen Nachteilen[16] zurückgegriffen werden, da die Kommunikation zwischen ELANs nicht spezifiziert ist. Ebenso ist die Adressierung bei mehreren LECs in einer Komponente nicht definiert und durch die eindeutige Zuordnung (eines LECs) zu genau einem ELAN entsteht wieder eine physische Abhängigkeit, die man durch die VLANs überwunden zu haben schien. In der Arbeit wird diesen Problemen kein allzu großes Gewicht beigemessen, d.h. es werden keine aufwendigen und umständlichen Ausnahmenbehandlungen eingeführt, die in einer der nächsten Versionen des LANE-Standards wieder hinfällig wären. Das dem so ist, zeigt, daß es bereits Ansätze (z.B. [Bor96]) gibt, auf die man sich nur einigen muß.

[15] In der Version 1.0 kann ein LEC nur einen ELAN angehören.
[16] Bottleneck, pro ELAN eine physische Leitung und Port.

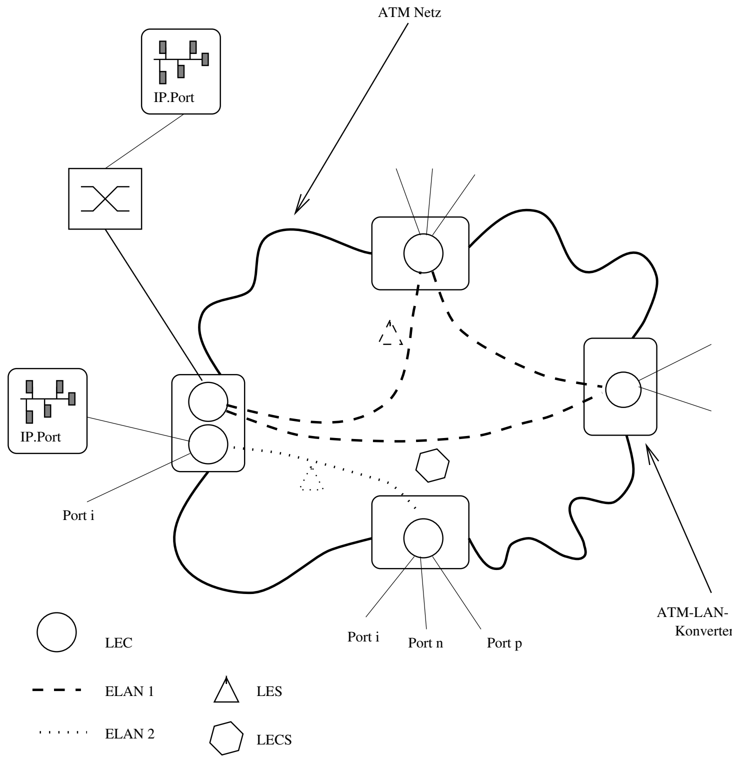


Abbildung 2.4: Schematische Darstellung der LANE Komponenten

2.5.2 VLANs mit der LAN Emulation

Ein ELAN verbindet verschiedene LAN-Teilnetze, die an ATM-LAN-Konvertern angeschlossen sind, zu einem einzigen (logischen) LAN-Segment[17], d.h. *Unicast*-Frames werden vom ELAN zum korrekten und *Broadcasts*-Frames in alle Teilabschnitte weittergeleitet. In dieser Arbeit wird die LAN Emulation für die Interswitchkommunikation, d.h. als Backbone, verwendet. Werden Layer-2-VLANs switchübergreifend eingerichtet und liegt dazwischen ein ATM-Netz mit LAN Emulation, so werden vom zu entwickelnden VLAN-Mangementsystem automatisch in den involvierten ATM-LAN-Konvertern bzw. *Edge Devices* ELANs eingerichtet und damit auch alle LECs und die restlichen Server instantiiert. Die zur Einrichtung eines VLANs instantiierten ELANs bekommen den gleichen Namen wie das VLAN oder, wenn kein Name existiert, den VLAN-Identikator.

Obwohl der LANE-Standard in der Version 1.0 noch einige gravierenden Fragen offenläßt (siehe vorhergehenden Abschnitt), wird in dieser Arbeit davon ausgegangen, daß mehrere LECs in einem Konverter möglich sind und jeder davon eindeutig ein ELAN repräsentiert. Jedem LEC werden jetzt Ports (auf der LAN-Seite) zugewiesen, er leitet dann bei Bedarf den ein und ausgehenden Verkehr wie bereits besprochen weiter. Die LECs können in diesem Szenario als logische Ports aufgefaßt werden, mit den Unterschied zu herkömmlichen LAN-Ports, daß für jeden LEC anstatt einer physischen Leitung eine logische Verbindung zum ATM-Netz vorhanden ist.

Diese Kurzeinführung der LAN Emulation reicht für das Verständnis der weiteren Arbeit. Für tiefergehende Informationen sei auf die einschlägige Literatur verwiesen, insbesondere auf den ATM LAN Emulation Standard des ATM Forums [Eea95], der im ersten Teil einen sehr guten Überblick bietet.

Für die weitere Arbeit sei festzuhalten, daß jedes ELAN für ein VLAN im wesentlichen den Dienst einer (Multiport-)Brücke erbringt. Durch diese Eigenschaft spielt es bei Layer-2-VLANs, die mit ihrer Hilfe eingerichtet werden, keine Rolle mehr, wo sich die Segmente/Endstationen physisch befinden. Somit sind die Voraussetzungen geschaffen, um das ATM-Netz bzw. Backbone für die Switches transparent erscheinen zu lassen. In Idealfall tritt die LAN Emulation und das darunterliegende ATM-Netz, wie bereits erwähnt, nur noch im Fehlerfall oder speziellen bei Managementtätigkeiten bzgl. ATM in Erscheinung.

[17]Schicht 2 Broadcastdomäne; Unter einem (LAN-)Segment wird, wie bereits in Abschnitt 2.1 besprochen, im weiteren Verlauf der Arbeit ein physisches LAN-Teilnetz verstanden, in dem jede teilnehmende Endstation den gesamten Verkehr des Segments mithören kann.

2.6 Klassifizierung der VLANs

In diesen Abschnitt wird versucht die VLANs anhand der erreichten Virtualisierung/Abstraktion vom physischen Netz zu klassifizieren. Dadurch lassen sich auch VLANs in Klassen einteilen bzw. den Schichten des OSI-Referenzmodells zuordnen.

Unter einem *Layer-1-VLAN* wird ein VLAN verstanden, das auf der OSI-Schicht 1 durch Gruppierung der Ports eines Switches (Port-Switching) realisiert ist. Alle Endsysteme die von diesen Ports aus erreichbar sind, gehören zu diesem spezifizierten VLAN.

Mehrere Ports werden in einer Broadcast-Domäne zusammengefaßt. Der *Unicast*-Verkehr wird durch Auswertung der MAC-Adressen weitergeleitet, der *Broadcast*-Verkehr innerhalb des VLANs geflutet.

Ein *Layer-2-VLAN* ist auf der OSI-Schicht 2, durch Gruppierung der relevanten MAC-Adressen der Endsysteme, realisiert.

Endstationen werden mittels der MAC-Adressen über mehrere Switches hinweg als eine Broadcast-Domäne definiert. Der Vorteil dieser Methode liegt in der Unabhängigkeit von der räumlichen Plazierung der LANs. Diese Definition schließt nicht aus, daß Endstationen eines Segments verschiedenen VLANs angehören können.

Ein *Layer-3-VLAN* ist eine Gruppe von Endsystemen, die physisch irgendwo im gesamten Netz verstreut sein können. Die Gruppierung erfolgt auf der Vermittlungsschicht durch zusammenfassen der Netz- bzw. Subnetzadressen der Endstationen oder anderer Protokollinformationen (aus dieser Schicht).

Auf der (logischen) Schicht-3 des OSI-Schichtenmodells erfolgt die Zuordnung von Endstationen auf Basis der Subnetzadresse[18] oder anderer Protokollinformationen als eine Broadcast-Domäne. Ein virtuelles Netz[19] läßt sich mit dieser Methode unabhängig von der physischen Netzstruktur definieren. Räumliche Grenzen spielen keine Rolle mehr.

Abbildung 2.5 verdeutlicht noch einmal die eingeführte Klassen/Schichtenbildung. Wie bei einer Schichtung üblich, nutzt eine Schicht alle darunterliegenden.

[18] In routbaren Protokollen.

[19] VNET=VLANs plus Routingfunktionalität zwischen den VLANs.

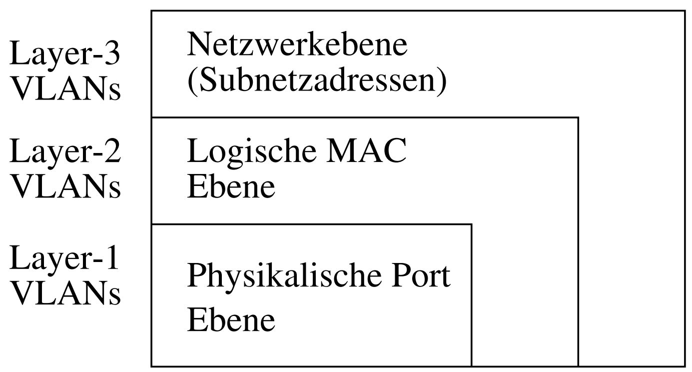


Abbildung 2.5: VLAN-Schichten/Klassen

Kapitel 3

Graphische Benutzerschnittstellen

Der Einsatz des Computers ist nicht [nur] durch seine Rechenleistung begrenzt, sondern in erster Linie durch seine Beschränktheit mit dem Anwender zu kommunizieren. [ISA]

Dieses Zitat aus dem Werbeprospekt eines Softwarehauses zeigt zwei Probleme der heutigen Anwendungssysteme auf. Zum einen werden die Programme, bedingt durch den enormen Preisverfall von Rechenleistung und Speichermedien, immer größer und komplexer. Zum anderen wird es für den Benutzer schwerer, die Anwendung zu beherrschen, sei es wegen der übergroßen Funktionalität oder den komplexen Abläufen.

Um die entstandene Lücke zwischen der angebotenen Funktionalität der Anwendung und der Ausnutzung durch den Benutzer zu verringern, werden zunehmend mehr Anwendungen mit graphischen Benutzerschnittstellen bzw. Bedienoberflächen (Graphical User Interfaces (GUIs)) versehen. Das sieht man am deutlichsten, wenn man die frühen zeilenorientierten Texteditoren mit den heutigen Textverarbeitungssystemen vergleicht. Computer-Neulinge können heute Dokummente erstellen, die vor 10 Jahren nur von Experten erstellt werden konnten. Eine ansprechende graphische Oberfläche stellt heute auch ein wichtiges Verkaufsargument dar, dadurch besteht aber auch die Gefahr den Bildschirm mit unnötigen *Spielereien* zu überladen (Hollywood-Prinzip).
Eine gute graphische Benutzerschnittstelle zeichnet sich dadurch aus, das sie die dahinterliegende Funktionalität den Benutzer optimal anbietet und ein intuitives und flüssiges Arbeiten erlaubt.

3.1 Grundbegriffe

In diesem Abschnitt sollen einige der notwendigsten Grundbegriffe aus dem Bereich der graphischen Benutzerschnittstellen eingeführt werden, die auch für den weiteren Verlauf der Arbeit relevant sind. Mit **Benutzungsschnittstelle, User Interface (abgekürzt mit UI)** wird insgesamt die Kommunikationsschnittstelle zwischen Mensch und Maschine bezeichnet, also diejenigen Komponenten und Aspekte eines Systems mit denen ein Benutzer begrifflich oder über seine Sinne und Motorik in Verbindung tritt [End94]. Der Begriff umfaßt sowohl die Hardware — Bildschirm, Tastatur und Maus — als auch die Software zur Realisierung dieser Schnittstelle. Obwohl die Definition für jede Art von System/Maschine gilt mit der Benutzer interagieren können, wird in dieser Arbeit nur der Softwareanteil (Oberfläche plus Dialogsteuerung) von Interesse sein.

Die **Benutzungsoberfläche, Bedienoberfläche, Benutzerschnittstell** bezeichnet den nach außen sichtbaren Anteil der Benutzungsschnittstelle.

In der Literatur werden die Begriffe Benutzungsoberfläche und Benutzungsschnittstelle oft synomym verwendet, obwohl der eine nur das (Bildschirm-)Layout und der andere aber zusätzlich noch das Dialogverhalten und -ablauf, der "Intelligenz" der Anwendung, mit umfaßt.

Die Begriffe **Applikation, Anwendungskern, Funktionaler Kern** bezeichnen den Teil des gesamten Systems, der die eigentliche Anwendungsfunktionalität realisiert.

Die **Anwendung**, das **Programm** ist alles zusammen, d.h. der Teil der an der Bedienoberfläche gestartet wird.

Mit diesen Begriffen kann später[1], die für eine Softwareentwicklung, notwendige Trennung von Benutzerschnittstelle und Applikation, sowie eine saubere Schnittstelle zwischen diesen beiden Software-Komponenten herausgearbeitet werden.

Die folgenden Begriffe legen in einer idealisierten Weise die am Entwicklungsprozeß beteiligten Personen fest.

Der **Benutzer, Anwender** ist derjenige, der mit dem fertigen Programm arbeiten wird.
Genaugenommen arbeitet er mit der Benutzungsoberfläche des Programms und steuert, wie noch genauer herausgearbeitet wird, durch seine Eingaben den Ablauf des Programms.

Der **Benutzungsoberflächendesigner** gestaltet nach software-ergonomischen Kriterien Layout und Dialogverhalten der Bedienoberfläche. Er benötigt zwar keine Programmierkenntnisse, muß sich aber mit der Problemwelt der Auftraggeber auseinandersetzen.

[1] Abschnitt 4.1: Architekturen für Anwendungssysteme.

Der **Benutzungsschnittstellenprogrammierer** setzt die Vorgaben des Benutzungsoberflächendesigners in Software um. Hat er sich auf eine Systemumgebung (Hardware, Betriebssystem, usw.) einmal festgelegt, so kann er sich auf Werkzeuge wie Window-Systeme, Toolkits und User Interface Builder[2] stützen.

Die **Applikationsprogrammierer** entwerfen und implementieren den funktionalen Kern des Programms.

In dieser Arbeit wird, wie bereits im ersten Kapitel erwähnt, der Schwerpunkt auf die Analyse der Anforderungen an die Bedienoberfläche und dem Design des Bildschirmlayouts gelegt (Kapitel 7 und 8). Der zu implementierende Prototyp soll dann einige der Vorschläge verdeutlichen, demzufolge müssen, neben der schematischen Darstellung der Bedienoberfläche, auch Teile der Dialogsteuerung und des Anwendungskerns ausprogrammiert werden. (Kapitel 9).

3.2 Erwartungen und Anforderungen an graphisch-interaktive Systeme

Mit dem Begriff "graphisch-interaktive Systeme" soll diejenige Art von Software bezeichnet werden, die die Möglichkeit hochauflösender Bildschirme und Mauseingaben nutzen und in ihrem Ablauf wesentlich durch Aktionen des Benutzers "interaktiv" bestimmt sind.

Der hohe Entwicklungsaufwand der Benutzungsschnittstelle von graphisch-interaktiver Anwendungen, der einen Anteil von durchschnittlich 60 [Weg95] bis 80 [Jea94] Prozent — je nach Intensität der Interaktionsmöglichkeiten — am gesamten Quellkode betragen kann, ist im wesentlichen auf die im folgenden charakteristischen Eigenschaften zurückzuführen:

- *Vom Benutzer bestimmte Dialogführung:* Im Gegensatz zum sequentiellen Kontrollfluß in alphanummerischen Anwendungen, zeichnen sich graphisch-interaktive Anwendungen dadurch aus, daß der Benutzer, ohne besondere Vorkehrungen zu treffen, zwischen den verschiedenen Dialogteilen wechseln kann, das heißt aufgrund der ihm angebotenen Interaktionsmöglichkeiten in den verschiedenen Fenstern ist der Programmablauf nichtdeterministisch. Hieraus ergeben sich Probleme mit einer Vielzahl von zu verwaltenden Dialogzuständen und dem nicht sequentiellen Kontrollfluß.

- *Schnelle Reaktion auf Benutzeraktionen:* Um das System für den Benutzer subjektiv annehmbarer und schneller erscheinen zu lassen, sollte jede Benutzeraktion sofort zu einer Reaktion des Systems führen, entweder mit einer kurzen

[2] Siehe Abschnitt 3.6: Überblick Benutzerschnittstellenwerkzeuge.

Meldung oder im Idealfall mit dem Ergebnis der ausgelösten Aktion. Probleme ergeben sich, wenn kurz aufeinanderfolgende Benutzeraktionen jeweils Reaktionen mit umfangreichen anwendungsspezifischen Berechnungen erfordern.

- *Darstellung komplexer Informationen:* Der Hauptgrund für den enormen Entwicklungsaufwand liegt in der Möglichkeit viele Informationen gleichzeitig darzustellen und die dadurch erkauften inhärenten Abhängigkeiten zwischen ihnen. Den je mehr Informationen sichtbar sind, desto schwieriger wird es die dargestellten Daten konsistent zu halten. Der Benutzer erwartet natürlich, daß die Bildschirmdarstellung jederzeit korrekt ist.

- *Graphische Elemente:* Vor allem die vielen Attributszuordnungen für Oberflächenbausteine wie Farbe, Geometrie oder Text treiben die Kodelänge schnell in die Höhe.

3.3 Objektorientierte Bedienoberflächen

Der Arbeitsstil des Menschen ist objektorientiert. Täglich gehen wir mit den verschiedensten Objekten um, wie zum Beispiel der Kaffemaschiene, dem Telefon oder dem Rasenmäher. Objekte haben Eigenschaften, die bestimmen, welche Aktionen mit ihnen sinnvoll ausgeführt werden können. Das Telefon zum Beispiel überträgt analoge Signale (Eigenschaft) und dadurch sind wir in der Lage Nachrichten zu empfangen oder sie zu übermitteln (Aktionen).

Der Begriff der objektorientierten Bedienoberfläche [3] wird häufig mit graphikfähigen Bildschirmen und Positionierhilfe (Maus) in Verbindung gebracht. Diese bilden zwar die Voraussetzung, wesentlich aber ist die Veränderung der *Syntax der Interaktion* gegenüber der konventionellen, funktionsorientierten Oberfläche (siehe nächster Abschnitt). Weiterhin lassen sich graphische Bedienoberflächen durch den Begriff der direkten Manipulation charakterisieren (Abschnitt 3.3.2).

3.3.1 Interaktionssyntax

Bei der *ojektorientierten bzw. graphischen Bedienoberfläche* wird eine Interaktion grundsätzlich mit der Auswahl eines Objektes begonnen. Erst dann wird eine Operation selektiert, die auf dem Objekt ausgeführt wird. Ein typisches Standardbeispiel

[3]Objektorientierte Bedienoberfläche implizieren nicht, daß sie mit einer objektorientierten Programmiersprache implementiert wurde. Diese bieten sich aber geradezu an, da man relativ schnell erkannte, daß sich solche komlexen Systeme mit herkömmlichen funktionalen Programmiersprachen nicht mehr strukturiert beschreiben lassen. Nach Jacobson [Jea94, Seite 111] erlangten die OO-Programmiersprachen ihren tatsächlichen Durchbruch erst durch ihren Einsatz bei der Entwicklung graphischer Bedienoberflächen.

ist die Löschoperation, bei der man ein Dateisymbol (Objekt) anklickt und per *Drag & Drop* über den Papierkorb (Operation) zieht und dabei die Maustaste losläßt.
Konventionelle Anwendungen beginnen eine Interaktion in der Regel mit der Operationsauswahl. Erst danach wird angegeben, worauf die Operation angewendet werden soll. Diese Syntax ist bei funktionssorientierten Oberflächen durchgängig zu finden, gleichgültig ob es sich dabei um *benutzergeführte* (Kommandosprache) oder *systemgeführte* (Menüsystem) handelt.

3.3.2 Direkte Manipulation

Im engen Zusammenhang mit objektorientierten Bedienoberflächen steht das Handlungsprinzip der *direkten Manipulation*, die den Anwender in einer Scheinwelt agieren läßt, in der er fast ausschließlich mit Zeigeoperationen arbeitet. Der Benutzer bekommt den Eindruck, direkt in die dargestellte Welt einzugreifen - gewissermaßen mit der Maus als verlängerte Hand.
Direkt manipulative Oberflächen zeichnen sich durch eine Reihe von Ein- und Ausgabetechniken aus:

- permanente Visualisierung der Gegenstände aus der Welt der Anwendungen am Bildschirm,
- Objektorientiertheit der Interaktionssyntax,
- einfache Zeigehandlung statt komplexer Eingabesyntax,
- Einsatz eines natürlichen Modells (Verwenden von Metaphern[4] bzw. geeigneten Ikonen),
- unmittelbare Ergebnisrückmeldung nach jedem Arbeitsschritt,
- generische Kommandos und
- Funktionsobjekte.

Die nach jedem Arbeitsschritt angezeigte *Ergebnisrückmeldung* erfolgt in Begriffen und Konzepten der gewählten Metaphern.

Um die einfachen Interaktionsformen auch bei sehr großem Funktionsangebot aufrechtzuerhalten, werden die typischen syntaktischen Mittel der direkten Manipulation (Ziehen, Klicken, Doppelklicken) üblicherweise *hochgenerischen Operationen* zugeordnet. Als hochgenerisch werden Objekte dann bezeichnet, wenn sie in jedem

[4]Beispiele für Metaphern sind die Schreibtisch-Metapher, die die Arbeit mit Dateien und Verzeichnissen durch die Begriffe aus der Bürowelt (Akten, Aktenschränke, Dokumente, usw.) darstellt oder der Papierkorb als Metapher für die Löschoperation.

Kontext auf alle Objekte, gleich welchen Objekttyps, angewendet werden können. Aus dem gleichen Grund führt man mit direkt manipulative Oberflächen sogenannte *Funktionsobjekte* ein. Dabei werden den Operationen, die im allgemeinen stark generischen Charakter haben, grafische Repräsentationen auf dem Bildschirm zugeordnet (z.B. Papierkorb und das Druckersymbol).

Im folgenden werden die Vor- und Nachteile der direkten Manipulation aufgeführt:

Vorteile:

- Benutzer finden es leicht, den Umgang damit zu erlernen und diesen zu behalten.
- Anfängern wird die Angst genommen, Fehler zu machen, da praktisch jede Handlung rückgängig gemacht werden kann.
- Die visuell-kognitiven Fähigkeiten der Benutzer werden in straken Maße ausgenutzt.

Nachteile:

- Für erstmalige Benutzer ist eine solche Benutzeroberfläche oft nicht intuitiv handhabbar.
- Es ist oft schwierig, gute Ikonen zu entwerfen.
- Ikonen nehmen oft mehr Platz auf dem Bildschirm ein als Wörter.

Bemerkung: Geübte Benutzer können in manchen Fällen mit Kommandosprachen wesentlich schneller arbeiten als mit direkter Manipulation.

Bei komplexeren Anwendungssystemen ist jedoch die direkte Manipulation nicht ausreichend, so daß man Mischformen einführen muß. Dies kann dann in Form von Formularen geschehen, über die der Anwender die Attribute des selektierten Objekts beeinflußen kann. Mit einem Formular wird auf ähnliche Weise gearbeitet wie mit Formularen auf Papier. Die Daten sind strukturiert und formatiert, und es sind Felder vorhanden, in die der Benutzer Daten eintragen kann. In der Regel hat jedes Feld eine Beschriftung *(Label)*, die dem Benutzer zur Orientierung behilflich sein soll. So wird zum Beispiel die Formatierung von Text (Schriftart, Größe, Fett, usw.) meistens über ein Formular gesteuert.

3.4 Ereignisprinzip

Dieser Abschnitt geht auf eine wesentliche Technik grafischer Bedienoberflächen ein. Es handelt sich dabei um das *Ereignisprinzip*, das für alle *Fenster-Systeme* grundlegenden Charakter hat. Graphische Bedienoberflächen stellen völlig neue Anforderungen an Anwendungsdesigner und Entwickler, aber auch an den Benutzer. Dabei ist der Wechsel von einer alphanummerischen auf eine graphische Oberfläche für den Benutzer am einfachsten, da er nach einer gewissen Einarbeitungszeit feststellen wird, daß die neue Oberfläche seinem natürlichen Arbeiten näher steht. Problematischer vollzieht sich der Wechsel für den Anwendungsdesigner und Entwickler, die sich von den alten, hierarchisch-strukturierten Design-Techniken lösen müssen.

Die zwei wesentlichen Prinzipien, die bei der Entwicklung graphischen Bedienoberflächen mit einbezogen werden müssen, sind:

- der Benutzer steuert die Anwendung und
- die Anwendung reagiert auf Ereignisse.

Der Benutzer steuert den Dialogablauf der Anwendung mit der Maus, der Tastatur und anderen Eingabegeräten. Das Neue dabei ist, daß der Benutzer und nicht das Programm den Dialogablauf steuert, und zwar in einer für die Anwendung nicht vorhersehbaren Reihenfolge. D.h. es gibt keine streng sequentiellen Dialogabläufe mehr, woraus sich die Schwierigkeiten für die Anwendungsdesigner und Entwickler ergeben. Die Aktionen des Benutzers, wie zum Beispiel ein Mausklick, das Eingeben eines Zeichens oder das Minimieren eines Fensters, stellen in der GUI-Terminologie ein sogenanntes Ereignis *(Event)* dar und dienen der Verständigung zwischen Benutzer und Anwendung. Diese elementare Kommunikation zwischen Benutzer und Anwendung ist in allen modernen Fenstersystemen so grundlegend, daß man daraus ein Prinzip, das Ereignisprinzip (auch Prinzip der externen Kontrolle), ableitet. Die Ereignisse werden vom Fenster-System an die Anwendung weitergeleitet, die dann durch Aufruf der entsprechenden Operationen darauf reagiert. Das Ereignisprinzip hat beim Entwurf einer Anwendung erhebliche Konsequenzen:

- entsprechend dem Ereignisprinzip muß die Anwendung neu strukturiert werden: Diese besteht nicht mehr aus einem zusammenhängenden Hauptprogramm, sondern zerfällt in einzelnen Prozeduren, die von der graphischen Oberfläche aus aufgerufen werden,
- alle möglichen Ereignisse sind von vorhinein zu antizipieren und,
- um die Kommunikation zwischen Benutzer und Anwendung so effizient wie möglich zu gestalten, es müssen die vom Fenster-System zur Verfügung gestellten Ereignisarten bekannt sein.

Die Benutzerschnittstelle als Teil der Anwendung[5] wertet die Events, die es vom Fenstersystem bzw. vom Window-Manager bekommt, aus und ruft sogenannte *Callback-Routinen* auf, die die Verbindung zu den eigentlichen Anwendungsoperationen

[5]Siehe Architekturen für Anwendungssysteme.

herstellen. Diese von den Callback-Routinen aufgerufenen Menge von Prozeduren, die in keiner direkten Beziehung zueinander stehen, stellen die eigentliche Funktionalität der Anwendung dar. Sie wurde bereits als *Anwendungskern* eingeführt.

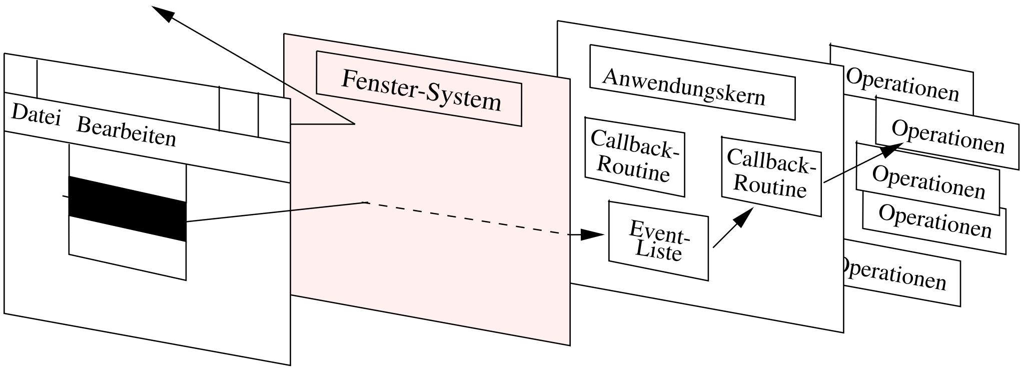


Abbildung 3.1: Die Umsetzung des Ereignisprinzips durch Events und Callbacks

Abbildung 3.1 zeigt schematisch die Umsetzung des Ereignisprinzips durch Events und Callback-Routinen. Dabei werden fast alle Benutzeraktionen in Form von Events an Benutzerschnittstelle der Anwendung weitergegeben. Lediglich die reinen Fensterfunktionen (Verschieben, Vergrößern, Verkleinern, usw.) werden vom Fenster-System selbst ausgeführt. Innerhalb der Anwendung werden entsprechend den Events die Callback-Routinen aufgerufen. Diese wiederum lösen die geforderten Operationen des Anwendungskerns aus.

3.5 Architekturen für Anwendungssysteme

Anwendungssysteme beinhalten heute eine große Zahl von Komponenten, die in einem komplexen Zusammenhang stehen. Um die daraus resultierenden Schwierigkeiten beherrschbar zu machen, wird versucht die Komponenten in kleine und unabhängige Einheiten zusammenzufassen, die separat implementiert und gewartet werden. Eine solche Strukturierung eines Anwendungssystems nennt man Architektur.

Bisherige ASCII-Benutzerschnittstellen wurden in das Anwendungsprogramm integriert und von dort aus aktiviert. Solche Schnittstellen waren anwendunsgetrieben, d.h. die erforderliche Ein- und Ausgaben wurden direkt im Anwendungsprogramm programmiert(Abbildung 3.2 a).

Moderne graphische Benutzerschnittstellen erfordern aufgrund ihrer Komplexität eine andere Architektur. Will man hohe Einarbeitungszeiten und Entwicklungs-

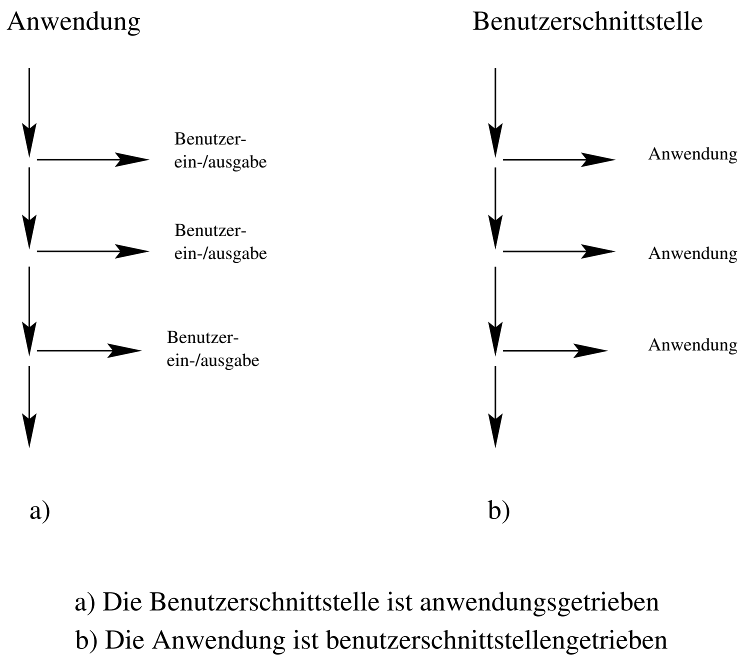


Abbildung 3.2: Benutzerschnittstellenkonzepte

kosten, sowie den erhöhten Wartungsaufwand in Grenzen halten, so müssen Strukturen für die Benutzerschnittstelle angewandt werden, die eine werkzeugunterstützte Software-Entwicklung gestatten. Solche Architekturen laufen im wesentlichen darauf hinaus, daß man graphische Benutzerschnittstellen und die entsprechenden Anwendungsprogramme möglichst vollständig trennt und die Aktivierung der Anwendungsprogramme der Benutzerschnittstelle überläßt (Abbildung 3.2 b). Damit entstehen auf der Basis entsprechender Benutzerschnittstellenmodelle gänzlich neue Softwarekonzepte.

Im nächsten Abschnitt werden Komponenten eines abstrakten Anwendungssystems beschrieben. Anschließend werden zwei weitverbreitete Architekturen vorgestellt, wobei die Komponenten des abstrakten Anwendungssystems auf die Baugruppen der Architektur abgebildet werden.

3.5.1 Komponenten von Anwendungssystemen

Setzt man die Hardware, das Betriebssystem und ein Fenstersystem als gegeben voraus, dann lassen sich bei einer Anwendung mit graphischer Bedienoberfläche folgende *grundlegenden Komponenten* identifizieren:

- die Benutzerschnittstelle[6],
- die Dialogsteuerung und
- der Anwendungskern.

Abbildung 3.3 zeigt alle Komponenten eines Anwendungssystems im Überblick.

Die *Benutzerschnittstelle* ist dabei für die Darstellung und Verwaltung des Inhalts in den Fenstern verantwortlich, d.h. sie ist für die Präsentation zuständig.

Die *Dialogsteuerung* kontrolliert die Aktionen des Benutzers auf ihre Plausibilität, die stark vom jeweiligen Kontext abhängt. Anhand der Benutzeraktionen steuert sie dann die Benutzerschnittstelle und den Anwendungskern.

Der *Anwendungskern* beinhaltet eine Reihe von relativ zusammenhangslosen Prozeduren, die die eigentliche Funktionalität darstellen. Die Prozeduren werden von der Dialogsteuerung aus aufgerufen.

Zwischen Dialogsteuerung und Anwendungskern organisiert ein *Anwendungsinterface* die Verbindung zwischen der Benutzerschnittstelle und den anderen Programmteilen und stellt somit die Nutzersicht auf das Anwendungsprogramm dar.

3.5.2 Standardarchitekturen

Im folgenden Abschnitt werden zwei Standardarchitekturen vorgestellt. Es handelt sich dabei um das Seeheim-Modell und das Model-View-Controller Modell (MVC-Modell) aus der Smalltalk-Welt.

Seeheim-Modell

Das Seeheim-Modell wurde auf einem "User Interface Management Systems" Workshop 1983 in Seeheim (Deutschland) entwickelt und besteht aus den Baugruppen

- Präsentation,
- Dialogkontrolle und
- Applikation.

Dabei entspricht die Präsentation der Benutzerschnittstelle, die Dialogkontrolle der Dialogsteuerung und die Applikation dem Anwendungskern. Besondere Beachtung

[6] Ohne den Anteil, der Infrastruktur, den das Fenstersystem zur Verfügung stellt.

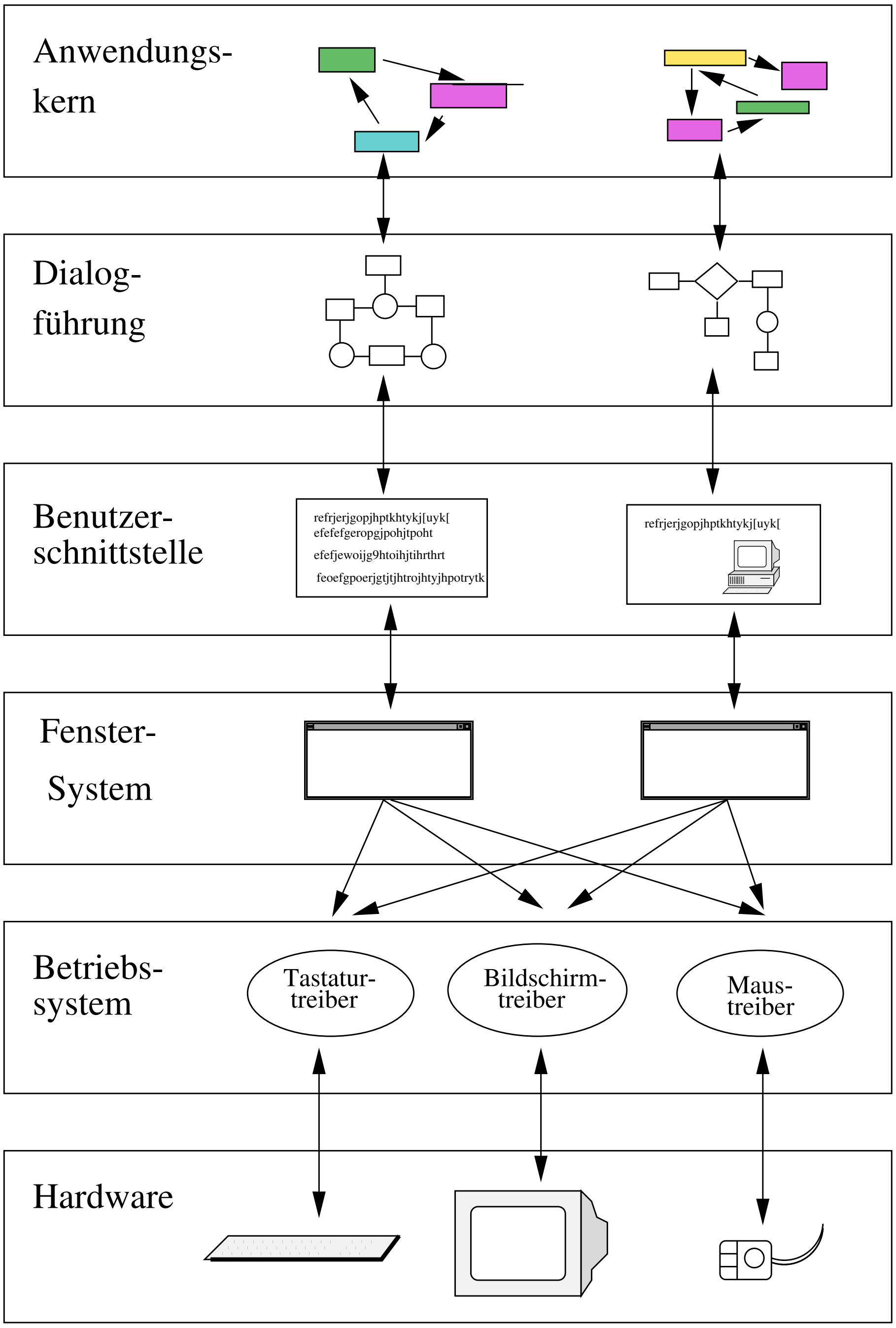


Abbildung 3.3: Komponenten eines Anwendungssystems

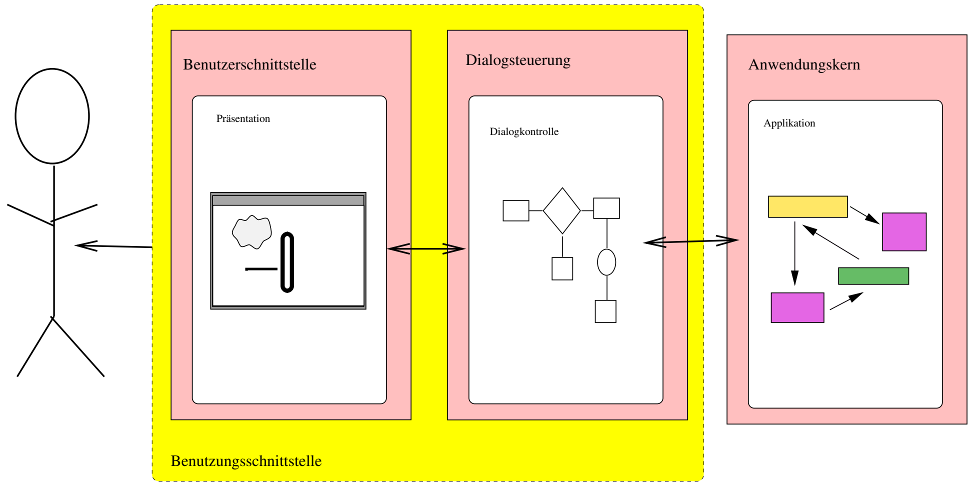


Abbildung 3.4: Das Seeheim Modell

verdient die Trennung zwischen Benutzerschnittstelle und Dialogsteuerung. Vor allem bei graphischen Dialogschnittstellen erscheint diese vorgehensweise auf den ersten Blick redundant, da man die notwendigen Funktionalitäten, die die Verbindung zwischen graphischer Benutzungsoberfläche und Anwendungskern herstellen, intuitiv in eine Baugruppe zusammenfaßt. Der Nachteil solcher unstrukturierter Programmierung wird erst bei Modifikationen deutlich, wenn man mühsam den Dialogablauf aus den einzelnen Baugruppen rekonstruieren muß. Die Dialogsteuerung, die den Dialogablauf "am Stück" beinhaltet, schafft hier Abhilfe.

Im Gegensatz zum MVC-Modell (nächster Abschnitt) beschäftigt sich das Seeheim-Modell nicht mit der Implementierung einer Benutzerschnittstelle, sondern strukturiert prinzipiell die Funktionalität der Benutzerschnittstelle in ihrer Gesamtheit.

Model-View-Controller Modell

Das MVC-Modell stammt aus der Smalltalk-Welt und dient der Beschreibung von Benutzeroberflächen innerhalb einer objektorientierten Methodik. Es gliedert ein Anwendungssystem in die Module

- *Model*,
- *View* und
- *Controller*.

Dabei entspricht das *Model* dem Anwendungskern, während *View* und *Controller* die Benutzungsschnittstelle bilden. Die Grundkomponente Benutzerschnittstelle wird

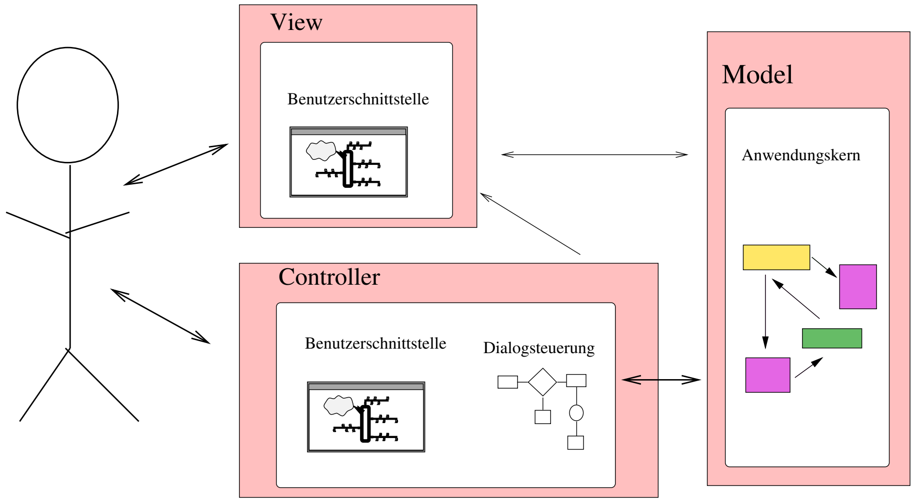


Abbildung 3.5: Das MVC-Konzept

also nach funktionalen Gesichtspunkten auf zwei Baugruppen verteilt. Die Dialogsteuerung ist dagegen mit im Contoller integriert, das heißt diese Baugruppe enthält zwei Komponenten. Der wesentliche Unterschied zur Seeheim-Architektur liegt also in der Aufteilung der Benutzerschnittstelle und der Integration zweier Komponenten im Controller (Abbildung 3.5). Wie bereits erwähnt wird die Benutzerschnittstelle nach funktionalen Gesichtspunkten auf den View beziehungsweise den Controller verteilt. Funktionale Kriterien sind dabei die Ein- und Ausgabe durch die Formulare am Bildschirm, mittels der Oberfläche zu manipulierenden Objekt sowohl ein View- als auch ein Controller-Objekt zugeordnet. Der *View* ist für die Ausgabe zuständig. Bei Änderungen im Modell wird er durch eine "Update" Methode informiert und holt sich die betroffenen Daten direkt aus dem *Model*, um sie in den Formularen darzustellen. Der *Controller* ist für die Eingabe von Daten zuständig. Er empfängt die Aktionen des Benutzers und setzt sie in Methoden um, die die Schnittstelle zum Model darstellen. In gewisser Weise übersetzt der Controller die Benutzereingaben. Wie der View besitzt auch der Controller eine Update Methode, die ihn über Änderungen im Model informiert. Wesentlich ist dabei, daß das *Model* nur über Änderungen informiert, jedoch weder View noch Controller direkt aufruft. Lediglich über deren Update-Methoden wird auf Änderungen hingewiesen. Darauf holen sich diese Komponenten die Daten selbst aus dem *Model*.

3.6 Überblick Benutzerschnittstellenwerkzeuge

Dieser Abschnitt beschäftigt sich mit den zur Oberflächengestaltung und den zur Realisierung der Dialogsteuerung notwendigen Werkzeugen. Dabei werden besonders diejenigen, die für die Implementierung des Prototypen verwendet werden, näher besprochen.

3.6.1 Fenster-Systeme

Grundvoraussetzung zur Entwicklung einer graphischen Oberfläche ist neben der Hardware und dem Betriebssystem ein Fenstersystem, das die gesamte Ein/Ausgabe von meist mehreren Anwendungsprogrammen verwaltet. Eingaben über Tastatur oder Maus, werden wie bereits erwähnt, vom Fenster-System evtl. unter Zuhilfenahme von Puffern an die Anwendung weitergeleitet (Ereignisprinzip). Umgekehrt delegieren Anwendungen alle Bildschirmausgaben an das Fenster-System, wobei jeder Anwendung fest abgegrenzte Bildschirmbereiche (Fenster) zur Verfügung gestellt werden.

X Window System

Das *X Window System*, oder kurz *X*, ist ein System für graphische Bedienoberflächen, das sich zu einem *De Facto* Standard in der Unix-Welt entwickelt hat. Es bietet die Möglichkeit, eine oder mehrere Anwendungen gleichzeitig auf einem graphischen Ausgabegerät ablaufen zu lassen. Wesentliche Kriterien für den Entwurf von X waren die Hardwareunabhängigkeit, die Netztransparenz, die Mehrprozeßfähigkeit und die Freiheit in der Gestaltung der Bedienoberfläche. Diese Anforderungen führten zur Konzeption des X Window Systems als *Client-Server Modell*. Hardwareabhängigkeiten werden durch den *X-Server* versteckt. Er abstrahiert von den jeweiligen Hardwareeigenschaften, indem er in einen hardwareabhängigen und -unabhängigen Teil gegliedert ist (device dependent und independent layer des X-Servers [7]). Seine Dienstleistung stellt er über ein Anwendungsprotokoll, dem *X-Protokoll*, zur Verfügung. Dabei kann er netzweit mehrere Anwendungen (*X-Clients*) gleichzeitig bedienen (Abbildung 3.6).

Die folgende Abbildung zeigt die Kommunikation in einem X Window System.

X-Clients arbeiten ereignisorientiert. Nach einer Initialisierungsphase, in der sie die Verbindung zum X-Server aufbauen, arbeiten die Anwendungsprogramme in einer Ereignisschleife (siehe Abbildung 3.7). Sie warten auf ein beliebiges Ereignis (Event), das vom X-Server geschickt wird, werten es aus und reagieren entsprechend.

[7] Bei einer Portierung des X-Window Systems muß so nur der hardwareabhängige Teil modifiziert werden.

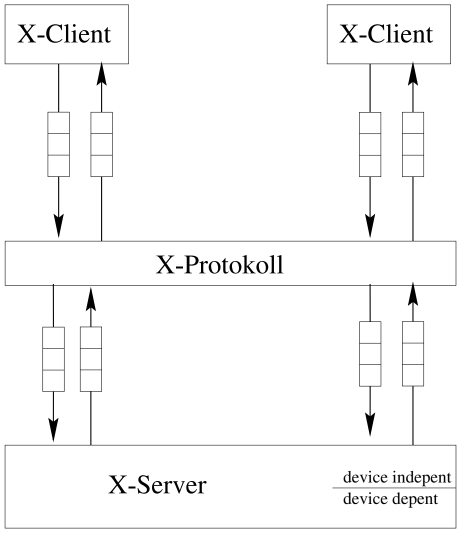


Abbildung 3.6: Kommunikation im X Window System

Die folgende Abbildung zeigt die grundlegende Struktur eines derartigen Klienten.

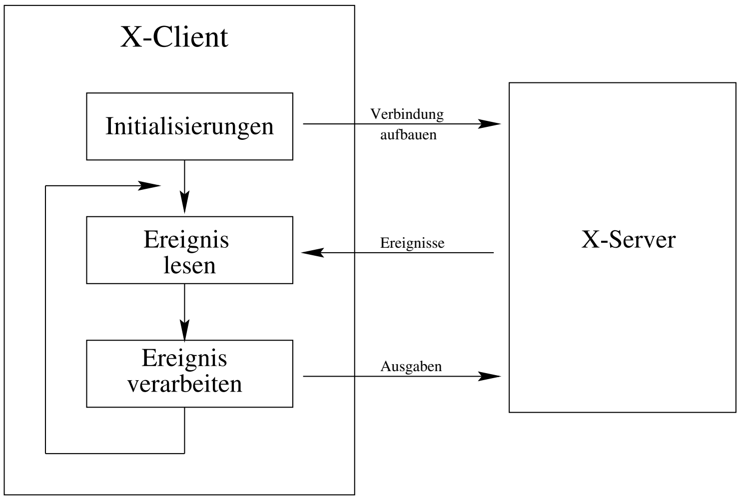


Abbildung 3.7: Struktur von X-Clients

Die Reaktion auf ein Ereignis hängt davon ab, in welchem Fenster und ggf. wo in diesem Fenster das Ereignis eingetreten ist. Dieser Ort ist durch die Position der Maus zum Zeitpunkt des Eintretens bestimmt. Weiterhin muß der aktuelle Programmzustand mit berücksichtigt werden, um so zu prüfen, ob es sich dabei um ein sinnvolles Ereignis handelt.

Die Reaktionen, die durch ein Ereignis ausgelöst werden, lassen sich unterscheiden in:

- Bildschirmausgaben,
- Änderungen an der Struktur der Fenster,
- An- und Abmelden von Interessen,
- Änderung an der Zustandsinformation und
- Ausführen von Anwendungsprozeduren.

X ist ein sogenanntes Basis-Fenster-System, das der Anwendung lediglich "leere" Zeichenflächen zur Verfügung stellt. Es verfügt denoch über eine große Zahl primitiver Operationen, um z.B. Fenster zu zeichnen, in sie zu schreiben oder Eingaben über die Tastatur und Maus zu empfangen. Diese Grundfunktionalität ist im X-Server enthalten. Als Anwender ist es möglich, direkt auf dem X Server aufzusetzen, was einer Programmierung in Maschinensprache gleichkommt.

Um die Entwicklung von Anwendungen zu vereinfachen, wurde eine auf der Schnittstelle des X-Servers aufbauende Bibliothek namens *Xlib* eingeführt. Wenn man die Entwicklung auf Basis des X-Servers mit Maschinensprache vergleicht, ist die Programmierung mit der Xlib-Bibliothek auf der Ebene der Assemblerprogrammierung anzusiedeln.

3.6.2 User Interface Toolkits

Die *X Toolkit Intrinsics*, oder kurz *Xt Intrinsics, X Toolkit*, sind ein Aufsatz der Xlib-Bibliothek. Sie gehören zum Lieferumfang des X Window Systems.

Es handelt sich dabei um eine C-Bibliothek, die Routinen zur Erzeugung und Verwaltung von graphischen Grundobjekten einer Bedienoberfläche, den sogenannten *Widgets*. Alles was darüber hinaugeht, muß weiterhin mit Funktionen der Xlib implementiert werden.

Mit der Einführung der Xt Intrinsics wurde eine weitere Abstraktionseben eingeführt, die einer Programmierung in einer Hochsprache gleichkommt.

Statt eines allgemeinen Fensters, in dem man alle Funktionalitäten und insbesondere alle Widgets selbst implementieren muß, hat man nun die Möglichkeit, vordefinierte Fensterobjekte mit den dazugehörigen Funktionalitäten[8] zu verwenden.

[8] Was z.B. bei einen oder zweifachen Mausklick mit der linken, mittleren oder rechten Maustaste, was bei einem Betreten oder Verlassen eines Objekts zu geschehen hat, usw.

Es gibt aber auch andere vergleichbare Toolkits, die oft auch als *Widget-Sets* bezeichnet werden. Ein Beispiel dafür ist OSF/Motif, das auf den Xt Intrinsics aufbaut und OpenLook von AT&T. Das *Look & Feel* einer Oberfläche wird erst durch das Widget-Set festgelegt, da das X Window System keine Bedienoberfläche festlegt und die Xt Intrinsics nur die Infrastruktur darstellen.

3.6.3 User Interface Builder

Ausgehend von einem Toolkit, das sich dem Benutzungsschnittstellenprogrammierer als Bibliothek von Prozeduren (oder Klassen) darstellt, muß er aus einer Vielzahl vordefinierter Interaktionsobjekten (Widgets)[9], wie Schaltern und Eingabefeldern zunächst ein statisches Abbild der Benutzerschnittstelle entwerfen und diese dann an die eigentliche Anwendung anbinden. Für diese Aufgabe muß er sehr viele Funktionen und deren Parameter kennen, sowie über Kombinationsmöglichkeiten von Bausteinen und Parametern Bescheid wissen. Zur Vereinfachung dieser Aufgabe stehen ihm jedoch unterschiedliche Methoden bzw. Werkzeuge zur Verfügung. Besonders zur Gestaltung des statischen Layouts bieten sich *User Interfacebuilder (UI-Builder)* an, die mit interaktiven Methoden (etwa dem Drag & Drop-Mechanismus) den Zusammenbau der graphischen Oberfläche erleichtern. Die zur Verfügung stehenden Widgets werden z.B. in Form eines Menüs oder einer Palette angeboten und können mit der Maus ausgewählt und in den zu entwickelnden Fensterbereich eingefügt und positioniert werden. Spezielle Editoren erlauben die Bearbeitung der Attribute der Widgets (Farbe, Schriftart, -größe, usw.). Ist das statische Layout der graphische Bedienoberfläche zur Zufriedenheit des Programmierers[10], kann durch den UI-Builder der zu dem Toolkit passende Quellcode generiert werden. Er besteht im wesentlichen aus den Prozueduraufrufen zur Erzeugung der Interaktionsobjekte, die Eventverarbeitung wird durch Dummyfunktionen realisiert und eine Dialogkotrolle bzw. -ablauf gibt es in den wenigsten Fällen.

UI-Builder unterstützen den Entwickler beim Zusammenbau der graphischen Oberfläche. Das dynamische Verhalten (Dialogkontrolle, -ablauf) und die Einbindung des eigentlichen Anwendungskerns muß aber weiterhin durch hand-kodierte Programmteile realisiert werden. Die fertige Anwendung entsteht durch Weiterentwicklung dieses vom UI-Builder generierten Quellkodes, wobei im wesentlichen die Dummy-Prozeduren ausprogrammiert werden.

[9] Erzeugt durch Prozeduraufrufen.

[10] Für diese Tätigkeit benötigt man keine Programmierkenntnisse; diese Aufgabe, die Oberfläche zu entwerfen, könnten auch Designer o.ä erledigen.

3.6.4 User Interface Management Systeme

UI-Builder unterstützen den Programmierer bei der Verwirklichung der statischen Anordnung der Oberfläche, jedoch kaum bei dynamischen Vorgängen, die in der Dialogkontrolle bzw. Dialogablauf[11] zusammengefaßt werden. Diese muß er noch per Hand kodieren — Stichwort "Callback-Prozeduren" (Abschnitt 3.4). User Interface Management Systeme (UIMS)[12] versuchen hier Abhilfe zu schaffen, indem man das Dialogverhalten mit Hilfe geeigneter Beschreibungsmittel, wie z.B. attributtierten Grammatiken [13], Regelsystemen (Event-Response-Systemen) oder Zustandstransitionsnetzen formal spezifiziert und dann vom UIMS diese Spezifikation übersetzen läßt. Als Ergebnis wird der Quellkode der Dialogkontrolle generiert. Die vollständige homogene, rechnerlesbare Beschreibung einer graphischen Benutzerschnittstelle ist aber weiterhin eine Zielvorstellung, die gegenwärtig bei weitem noch nicht realisiert ist. Neuere UIMS setzen auf UI-Toolkits auf und beinhalten i.d.R. auch einen eigenen UI-Builder.

3.6.5 Zusammenfassung der Benutzerschnittstellenwerkzeuge

Ein direkter Vergleich der Benutzerschnittstellenwerkzeuge ist nicht möglich, da man es einmal mit einer Prozedurbibliothek und ein anderes mal mit einem darauf aufbauenden Konstruktionswerkzeug zu tun hat. Eine Bewertung kann aber anhand der dadurch eingeführten Abstraktionsebenen in der Benutzerschnittstellenentwicklung erfolgen. Geht man davon aus, daß man mit der Programmierung auf der Ebene der Xlib und des X-Servers "alle" seine Ideen realisieren kann, so ist klar, daß mit der Einführung einer weiteren Abstraktionsebene zwar die Programmierung vereinfacht wird, aber der Preis dafür eine Einschränkung der bisherigen Möglichkeiten nach sich zieht, die nur durch zusätzlichen manuell erstellten Kode auf Xlib- bzw. auf Basis-Window-System-Ebene ausgeglichen werden kann. Als Veranschaulichung dieser Abstraktionsebenen kann Abbildung 3.8 herangezogen werden. Je höher die Ebene angesiedelt ist, desto geringer die zur Verfügung stehenden Möglichkeiten und Freiheiten bei der Erstellung von graphischen Benutzungsoberflächen.

3.7 Window-Manager

Für das Erscheinungsbild einer X Window Oberfläche ist in erster Linie der sogenannte *Window Manager* verantwortlich. Er sorgt für ein bestimmtes "Look & Feel",

[11]Eingabe- und Ausgabebearbeitung, Prüfung der Eingaben auf ihre Plausibilität, Aufruf der entsprechenden Anwendungsfunktionalität, Fensterinhalte, bzw. den Bildschirm zu jeden Zeitpunkt konsistent halten, usw.

[12]In Anlehnung an Datenbank Management System (DBMS).

[13]Am Lehrstuhl Eickel werden diese Techniken untersucht und angewandt.

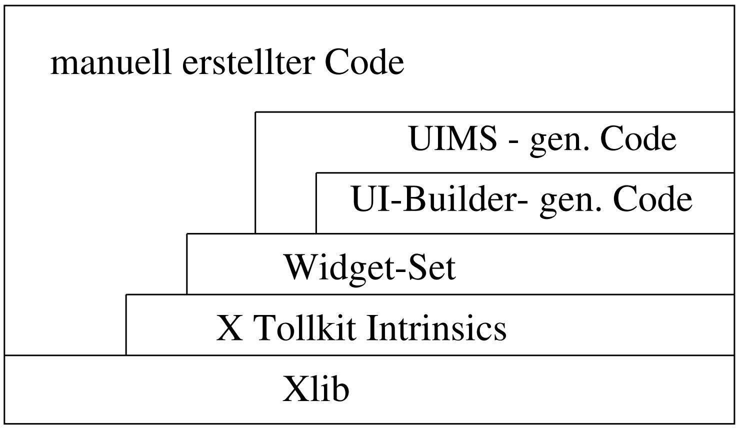


Abbildung 3.8: Architektur von X Programmen

das heißt für ein standardisiertes Layout der Fensterrahmen bzw. Widgets und ein geregeltes Verhalten bei Standardoperationen, wie z.B. für das Verschieben, Vergrößern und Verkleinern des Fensters. Der Window-Manager ermöglicht es effektiv mit mehreren Anwendungen gleichzeitig zu arbeiten, da er die Funktionalität zum beliebigen Anordnen der Fenster bereitstellt. Erst dadurch kann zum Beispiel ein verdecktes Fenster wieder in den Vordergrund geholt werden.

Ein Window-Manager stellt für das X Window System aber nur einen weiteren X Client dar. Dadurch ist es möglich Window-Manager auszutauschen, wobei die Anwendungen in der Regel auch ein anderes Look & Feel erhalten. Das Erscheinungsbild der Oberfläche ändert sich entsprechend.

Es gibt eine ganze Reihe von Window-Managern, wie zum Beispiel den Open Look Window Manager *(olwm)*, den Ultrix Window Manager *(uvm)* und Tom's Window Manager *(twm)*. Am bekanntesten ist jedoch OSF/Motif, das nicht nur ein Toolkit ist, sondern auch einen Window-Manager *(mwm)* enthält. Dadurch eignet es sich besonders gut zur Entwicklung von Anwendungen mit einem einheitlichen und durchgängigen Look & Feel.

Teil II

Entwicklungskonzept

Kapitel 4

Entwicklungskonzept

Die Projektierung einer Benutzerschnittstelle ist mit dem (objektorientierten) Software-Entwicklungsprozeß eines Anwendungsprogramms hinsichtlich Umfang und Ablauf grundsätzlich vergleichbar. Die entsprechenden Konzepte werden deshalb auch hier in einer modifizierten Form angewandt, zumal meist auch noch unterschiedlich große Teile der Benutzerschnittstelle über konventionelle Programmierung erstellt werden müssen.

Abschnitt 4.1 stellt den Ablauf der Aktivitäten bei der Entwicklung einer graphischen Bedienoberfläche in einer kurzen Zusammenfassung vor. Sie werden dann in den darauffolgenden Abschnitten (4.2 bis 4.7) näher besprochen.

4.1 Aktivitäten im Entwicklungsprozeß

Dieser Abschnitt stellt das Konzept bei der Entwicklung einer graphischen Bedienoberfläche vor, die einer objektorientierten Software-Entwicklung nachempfunden ist; die sich anfangs auf herkömmliche Konzepte zur objektorientierten Softwareentwicklung stützt, dabei zusätzliche Gesichtspunkte mit einbringt und danach den Benutzerschnittstellenaspekt stärker herausarbeitet, da dieser in den — dem Autor — bekannten SW-Entwicklungsmethoden[1] sträflich vernachlässigt wird. Die Ergebnisse der Analyse und Entwicklung fließen schließlich in die prototypische Implementierung der Bedienoberflächem ein. Die Gesamtimplementierung des VLAN-Managementsystems (kurz: System) würde den Rahmen einer Diplomarbeit bei weitem überschreiten. Wie Prototypen in die Implementierung des Gesamtsystems einfließen, wird in Abschnitt 4.7 besprochen.

Die Vorgehensweise zerfällt in die drei typischen Phasen einer objektorientierten

[1] Object-Oriented Software Engineering (OOSE) von Jacobson und Object Modeling Technique (OMT) von Rumbaugh et. al.

Software-Entwicklung:

- in eine Analysephase (OO-Analyse, Anwender- und Aufgabenanalyse, konzeptionelles Benutzerschnittstellenmodell), die das zu lösende Problem erfaßt und modelliert, sowie den Funktionsumfang, den das System zu erbringen hat, festlegt,
- in eine Entwurfs- und Spezifikationphase (Designphase), die den abstrakten Objekten Gestalt und Verhalten (Look & Feel) verleiht und in eine
- Implementierungsphase, die sich aber in dieser Arbeit nur auf die Erstellung eines Prototypen und dessen Auswertung beschränkt.

Abbildung 4.1 zeigt den Ablauf der einzelnen Aktivitäten in der Entwicklung einer graphischen (objektorientierten) Bedienoberfläche.

Die Pfeile im Hintergrund zeigen an, daß der Entwicklungsprozeß in der Regel mehrmals durchlaufen wird. Das Vorgehen ist eine Abfolge von einzelnen Schritten, wobei jeweils Verbesserungen und Ergänzungen der vorangegangenen Durchläufe vorgenommen werden.

In den nachfolgenden Abschnitten wird auf jede einzelne Aktivität näher eingegangen und deshalb hier nur kurz zusammengefaßt.

In einem ersten Schritt werden die zentralen Gegenstände aus der Problemwelt ermittelt. Diese dienen als Ausgangsmaterial für die Aufgabenanalyse und der Problemweltmodellierung.

Die **Anwender- und Aufgabenanalyse** untersucht, was ein Benutzer mit dem Programm machen will. Die Erfassung und Zerlegung der Aufgaben legt den Funktionsumfang des Systems in groben Zügen fest.

Für die **Objektorientierte Analyse (OOA)** wird ein Teil der OOA-Methode von Coad/Yourdon verwendet, der die zentralen Gegenstände der Problemwelt und deren Beziehungen zueinander werden modelliert. Das Ergebnis ist das (aufgabenorientierte) Objektmodell. Da es in dieser Arbeit keine Anforderungsspezifikation im herkömmlichen Sinn seitens eines Auftraggebers gibt, werden die Objekte aus der Anwender- und Aufgabenanalyse extrahiert. Das Attribut "aufgabenorietiert" hebt diesen Umstand hervor, außerdem wird dadurch eine Unterscheidung zum nachfolgenden benutzerschnittstellenorientierten Objektmodell ermöglicht.

Die Ergebnisse aus der Anwender- und Aufgabenanalyse und OOA fließen dann ein in das **konzeptionelle Benutzerschnittstellenmodell**. Dieses wird durch Einführung zusätzlicher Objekte und einer anderen Form der Interpretation der Beziehungen konstruiert.

Das konzeptionelle Benutzermodell schließt die Analysephase ab und bildet die Vorgabe für die weitere Software-Entwicklung.

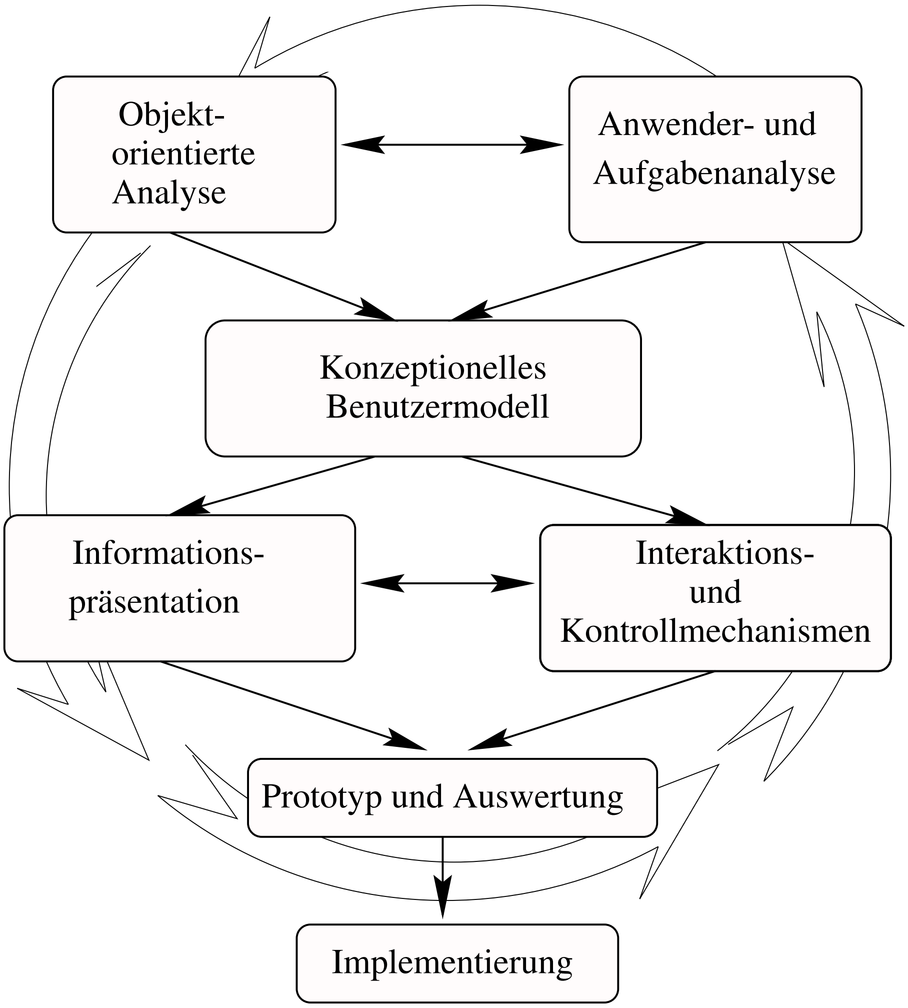


Abbildung 4.1: Aktivitäten im GUI-Entwicklungsprozeß

Erfolgt die Analysephase mehr oder weniger mechanisch, so ist die **Informationspräsentation** ein höchst kreativer Prozeß, der entscheidend den Erfolg der Anwendung mitträgt. Dieser Tätigkeitsbereich befaßt sich mit der Visualisierung der bisher abstrakten Objekte und Sichten (Views), d.h. mit allem was der Anwender am Bildschirm zu Gesicht bekommt.

Die zu entwickelnden und festzulegenden **Interaktions- und Kontrollmechanismen** (Dialogabläufe) erlauben den Anwender mit dem System zu interagieren. Dies können getippte Tastaturkürzel, Menüs oder Aktionen mit der Maus sein, wobei sich diese Techniken nicht gegenseitig ausschließen und durchaus mehrere auf ein Objekt Anwendung finden können.

Die Arbeit schließt mit einem **Prototyp** ab, der als Vorschlag für eine graphische Bedienoberfläche eines VLAN-Managementsystems aufzufassen ist. Ein Prototyp zeigt in groben Umrissen die graphische Bedienoberfläche und Dialogabläufe. Die Schnittstelle zum Anwendungskern ist durch *Dummy*-Funktionen realisiert. In dieser Arbeit wird — um die wesentlichen Ideen zu verdeutlichen — ein Teil der Funktionalität ausprogrammiert. Anhand der **Auswertung** des Prototypen können Erkenntnisse für Verbesserungen in einem erneuten Durchgang der Vorgehensweise — angedeutet durch die Pfeile im Hintergrund — einfließen.

In der **Implementierung des Gesamtsytems** werden die Benutzerschnittstelle und der Anwendungskern zusammengeführt. Sie wurde zur Vervollständigung der Grafik mit aufgenommen. Sie spielt in dieser Arbeit keine Rolle.

4.2 Anwender- und Aufgabenanalyse

Eine Anwender- und Aufgabenanalyse wird durchgeführt, um die Anwender der Bedienoberfläche und die Arbeit, die sie mit dem System ausführen wollen, zu verstehen.

Aufgaben und Anwender, in einem objektorientierten Kontext, werden analysiert, um zu untersuchen, wie Objekte der Problemdomäne benutzt werden. Dies ist sowohl in der Entwicklung des Gesamtsystems als auch für die Entwicklung der Bedienoberfläche hilfreich. Die meisten objektorientierten Methoden benutzen ähnliche Techniken[2]. All diese Techniken sammeln und nutzen Informationen aus den Aufgaben der Anwender.

Die Aufgabenanalyse untersucht die gegenwärtigen Aufgaben der Benutzer und extrahiert daraus Informationen, die helfen die bisherigen Arbeitsabläufe in die neue

[2]Jacobsons "Object-Oriented Software Engineering (OOSE) beinhaltet *Use Cases* [Jea94], Handlungssequenzen, die spezifische Abläufe (Wege), wie der Anwender das System benützt, definieren. Booch empfielt auch eine *Use Case*-Analyse. Die *Object Modeling Technique (OMT)* von Rumbaugh et. al. benutzt *Scenarios*, welche mit den Use Cases zu vergleichen sind.

Systemumgebung so zu übertragen, so daß das Arbeiten effektiver und effizienter durchgeführt werden kann. Die Ergebnisse der Analyse stellen Anforderungen an die vom System zu erbringende Funktionalität.

Aufgaben können von einem *Top-Down-* oder von einem prozeduralen bzw. *Bottom-Up*-Gesichtspunkt aus betrachtet werden. Im ersten Fall besteht die Aufgabe aus einer sinnvollen Arbeitseinheit, die eine abgeschlossene Arbeitsphase (ähnlich einem Use Case) des Systems beschreibt, und im zweiten Fall aus einer Sequenz einzelner Arbeitsschritte auf Objekte. Beide Gesichtspunkte sind in der Benutzerschnittstellenentwicklung wichtig. Die *Top-Down-* und *Bottom-Up*-Betrachtungsweise schließen sich nicht gegenseitig aus; mehr noch: beide ergänzen sich, um bei der Zusammenführung der Ergebnisse den Gesamtumfang der vom System angebotenen Funktionalität vollständig zu erfassen.

Während der *Top-Down*-Analyse sollte man sich immer der einzelnen Arbeitsschritte bewußt sein, um so die Ziele besser zu verstehen und die betroffenen Objekte zu identifizieren. Andererseits ist es genauso wichtig bei der *Bottom-Up*-Analyse der Aufgabenausführung den Sinn und Zweck für den Anwender und den Ablauf der einzelnen Arbeitsschritte im Auge zu behalten.

Die Aufgabenanalyse und andere im folgenden noch zu besprechende Analysetätigkeiten liefern mehrere Ergebnisse:

- Anforderungen an das zu entwickelnde System.
- Ein Analysemodell der Objekte, Klassen und Diensten, die nötig sind, um die Anforderungen am System zu erfüllen.
- Beschreibungen der Aufgaben, deren Ziele und gegenseitigen Beziehungen, sowie der involvierten Objekte und Handlungen.

Die ersten beiden Ausgaben sind Bestandteile jeder objektorientierte Analysephase. Die dritte Ausgabe wird im folgenden Abschnitt besprochen.
Alle drei Ausgaben helfen ein Modell zu konstruieren, das zeigt wie die Anwender das neue System sehen, es benützen und in ihre Arbeit miteinbeziehen. Dieses Modell wird Gegenstand des konzeptionellen Benutzermodells sein, das in Abschnitt 4.4 behandelt wird.

Dokumentation der Aufgabenanalyse
Eine Aufgabenbeschreibung, als Ergebnis der Aufgabenanalyse, sollte folgende Informationen enthalten:

- *Vorbedingungen*, die für die Aufgabe vorausgesetzt werden (z.B. für die Aufgabe "Dokument bearbeiten" muß das Dokument existieren);
- *Ziele* der Aufgabe;

- *(Ausnahme-)Situationen* in denen der Ablauf der laufenden Aufgabe (noch) nicht möglich ist oder irgendein Ziel wegen einer Beschränkung nicht erreicht werden kann. Diese Situationen können Anhaltspunkte liefern, um das System zu verbessern.
- *Nachbedingungen* schließen auch Objektzustände mit ein, die sich aus der Aufgabe ergeben und nicht unbedingt mit dem Ziel der Aufgabe zusammenhängen.
- *Benutzer*, die die Aufgaben ausführen und solche die direkt in der Aufgabe mit einbezogen sind.
- *Objekte*, die von der Aufgabe betroffen sind.
- *Aufgabenschritte* sollten in Aktionen auf die Aufgabenobjekte spezifiziert werden. Einige Schritte können dabei Teilaufgaben von mehreren Aufgaben sein. Allgemeine Teilaufgaben können extra beschrieben werden, um sich dann bei Bedarf darauf zu beziehen.
- *Kritische Besonderheiten* sind Aspekte der Aufgabe die besonders wichtig sind. Sie können Schlüsselobjekte, notwendige Handlungsabläufe und Prozeduren umfassen, die irgendwelche Regeln oder Anforderungen reflektieren.

Da die Aufgabenbeschreibungen die Arbeit des Anwenders wiederspiegeln, dienen sie auch als Grundlage für Bedienungsanleitungen und Trainingskurse.

4.3 Objektorientierte Analyse – Modellierung der Problemwelt

Objektorientierte Analyse setzt bei den Gegenständen der Problemwelt an. Die in der Aufgabenanalyse erfaßten Begriffe werden jetzt zu einem detaillierten Modell — dem aufgabenorientierten Objektmodell — von Objekten und Beziehungen ausgebaut. Die in dieser Arbeit verwendete Analysemethode und Notation erfolgt gemäß der Objektorientierte Analyse (OOA) von Coad und Yourdon [CY90].

Bei der Datenmodellierung in Datenbanksystemen liegt meist das Entity/Relationship-Modell zugrunde, d.h. die Objekttypen des Problembereichs werden als sogenannte Entitätstypen modelliert. Diese stehen untereinander in Beziehung (Relationship). Sowohl Entitäten als auch Beziehungen können Attribute haben, die deren Eigenschaften näher beschreiben. Coad und Yourdon bauten das reine Datenmodell zu einem Objektmodell in der Weise aus, daß Entitätstypen auch Funktionalität in Form von Methoden zugeordnet werden. Hierdurch entstehen Klassen im Sinne der Objektorientierung, die auch Vererbungsbeziehungen haben können. Zusätzlich wurden einige *Higher-Level*-Konzepte, wie z.B. zusammengesetzte Objekte, eingeführt.

Das Ergebnis der OOA ist das *aufgabenorientierte* Objektmodell. Das Attribut aufgabenorientiert wurde eingeführt, da die zentralen Objekte in dieser Arbeit zum Großteil aus der Anwender- und Aufgabenanalyse extrahiert werden und um das Objektmodell besser von dem Ergebnis des darauf aufbauenden konzeptionellen Benutzerschnittstellenmodells, den *benutzerschnittstellenorientierten* Objektmodell, zu unterscheiden. Im aufgabenorientierten Objektmodell werden die Objekt(klassen) aus dem Anwendungsbereich des Benutzers modelliert, zusammen mit ihren Attributen und Methoden bzw. Funktionen (die in der Regel — um die graphische Darstellung nicht zu überladen — gesondert festgeschrieben werden). Abbildung 4.2 zeigt als Beispiel einen Teilausschnitt eines Auftragssystems.

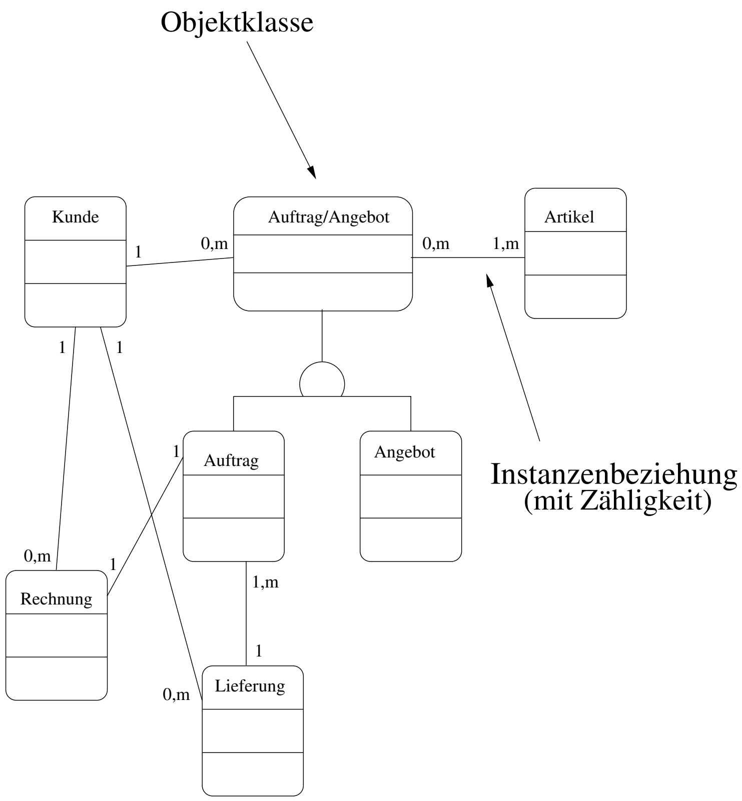


Abbildung 4.2: Aufgabenorientiertes Objektmodell

Die Zähligkeit der Beziehungen wird durch Beschriftung der Kanten angegeben. Der Halbkreis zeigt Vererbungsbeziehungen an und das Dreieck (im Beispiel nicht vorhanden), dessen Spitze zum Ganzen zeigt, drückt eine *besteht-aus*-Beziehung aus. Für die Zwecke des Benutzungsschnittstellenentwurf spielt Klassenbildung in bezug auf Methoden sowie die Vererbung von Methoden eine eher untergeordnete Rolle, da Benutzer im allgemeinen keine explizite Klassenhierarchie wahrnehmen.

Ob man mit der Problemweltmodellierung oder Aufgabenanalyse beginnt, hängt im Grunde nur davon ab, ob zu Beginn der Analysephase die Gegenstände (OOA) oder die Tätigkeiten (Anwender- und Aufgabenanalyse) wichtiger sind. Durch die parallele Anordnung in Abbildung 4.1 soll dieser Sachverhalt unterstrichen werden. In der Regel werden beide Tätigkeiten parallel ausgeführt, da sie sich gegenseitig beeinflussen und ergänzen. Die Sequentialität des Textes erfordert aber eine Reihenfolge in der Aufschreibung.

4.4 Das Konzeptionelle Modell der Benutzungsschnittstelle

Die Ergebnisse der beiden vorangehenden Analysetätigkeiten, der OO-Analyse und der Aufgabenanalyse, fließen in das **konzeptionelle Benutzerschnittstellenmodell** ein, welches wiederum als Basis und Rahmen für die nachfolgenden Schritte dient und diese dadurch steuert.

Gegenüber dem aufgabenorientierten Objektmodell aus der OO-Analyse, ein Gebilde aus Klassen, die zum Teil mit beschrifteten Kanten verbunden sind, die die Beziehungen verdeutlichen und eventuell erläuternden Text, kommen beim konzeptionellen Entwurf der Benutzungsschnittstelle weitere Objekte hinzu, die sich aus der Aufgabenanalyse, der OOA und den daraus resultierenden Anforderungen an die Benutzerschnittstelle ergeben.

Neben den bisher reinen *Datenobjekten* werden *Mengenobjekte* eingeführt, die die Gesamtheit der aktuell vorhandenen Objekte einer Klasse repräsentieren und Selektionsmechanismen bereitstellen. *Werkzeugobjekte* stellen Funktionen bereit, die auf andere Objekte angewendet werden können. (z.B. Drucker). Als Verfeinerung der Objektstruktur können außerdem im Entwurf neue *Teilobjekte* entstehen, sofern diese Strukturierung für den Benutzer bedeutsam ist. Wie auch im aufgabenbezogenen Objektmodell werden im benutzungsschnittstellenorientierten Objektmodell Beziehungen zwischen Objekten gezeigt. Es liegt jedoch eine andere Form von Beziehungen vor. Diese sind gerichtet und geben bereits Zugriffspfade an, die später in Dialogabläufe umgesetzt werden. Die umgekehrte Aufrufrichtung muß nicht zwingend vorhanden sein und wird ggf. als gesonderte Beziehung dargestellt. Abbildung 4.2 und Abbildung 4.3 zeigen Beispiele für aufgaben- bzw. benutzungsschnittstellenbezogene Objektmodelle für einen Teilausschnitt eines Auftragssystems.

Die Strukturierung des benutzungsschnittstellenorientierten Objektmodells sollte so erfolgen, daß die Aufgabenbearbeitung mit möglichst wenigen Zugriffen, also mit möglichst wenigen Navigationsschritten im späteren Dialogsystem, ermöglicht wird.

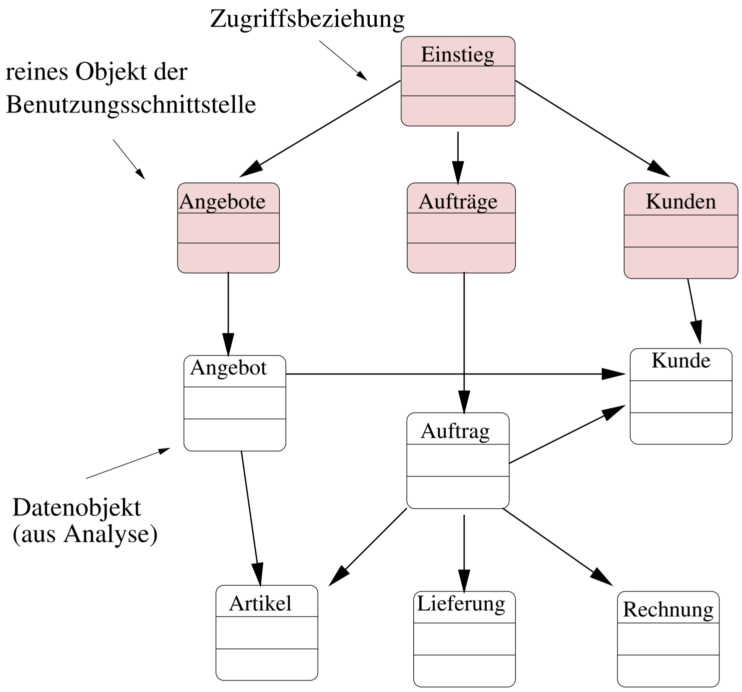


Abbildung 4.3: Benutzungsschnittstellenorientiertes Objektmodell

Im Einstiegsdialog werden daher diejenigen Objekte direkt angeboten, für die im Rahmen der Aufgabenbearbeitung ein primärer Zugriff erfolgt.

Für das Beispiel wurde angenommen, daß dieses *Angebote, Kunden* und *Aufträge* sind. Die sekundären Zugriffsbeziehungen ergeben sich dann aufgrund der Struktur des aufgabenbezogenen Objektmodells, unter Berücksichtigung der Arbeitsabläufe. Alle Zugriffsbeziehungen werden graphisch als Aufrufbeziehungen zwischen den Objekten dargestellt.

Die nächsten zwei Tätigkeiten bestehen in der Entwicklung der Darstellung, der Informationspräsentation, d.h. wie das System die Informationen dem Benutzer am Bildschirm präsentiert, und in der Entwicklung von Mechanismen, die den Benutzern erlauben, mit dem System zu interagieren.

Der Entwurf und die Spezifikation der Informationspräsentation und der Interaktions- und Kontrollmechanismen *(Look & Feel)* laufen teilweise parallel und werten die Ergebnisse des konzeptionellen Benutzermodells aus, wobei sich durchaus, in einem späteren Durchlauf des Entwicklungsprozesses, Änderungen ergeben können.

4.5 Entwurf und Spezifikation der Informationpräsentation

Zur Bearbeitung der konzeptionellen Objekte der Benutzerschnittstelle sind Bildschirmdarstellungen erforderlich, aus denen der Benutzer den Objektzustand ersehen und mit denen er interagieren kann. Solche Darstellungen werden Sichten genannt. Dieser Tätigkeitsbereich besteht in der Entwicklung und Spezifikation der Repräsentation der bisher abstrakten[3] Objekte und Sichten (Views). Sichten im Zusammenhang mit der GUI-Entwicklung stellen jede Art der Repräsentation von Objekten am Bildschirm dar[4]. Die Spezifikation einer Sicht umfaßt die Angabe der Objekte und Attribute, die in der Sicht dargestellt werden und je nach Kontext Funktionen, die zugänglich sein sollen. Ferner müssen für eine Sicht geeignete Interaktions- und Kontrollobjekte ausgewählt, visualisiert und angeordnet werden.

Objekte, die als Sichten dargestellt werden, fallen in zwei Kategorien:

- (Informations-)Objekte, die Bestandteil des konzeptionellen Benutzermodells sind und
- Interaktions- und Kontrollobjekte (Buttons, Scrollbars, usw.), die den Anwender erlauben mit den Objekten aus der ersten Kategorie zu interakieren.

Interaktions- und Kontrollobjekte sind in der Regel bereits in sogenannten *Style Guides* festgelegt. Sichten können einfache (Informations-)objekte repräsentieren oder aus mehreren Sichten zusammengesetzt sein. Sichten werden zusammen mit Mechanismen entwickelt, die den Benutzer erlauben mit den am Bildschirm dargestellten Informationen zu interagieren. Abbildung 4.1 zeigt die parallele und verzahnte Ausführung der Entwicklung der Informationspräsentation mit der Entwicklung der Interaktion- und Kontrollmechanismen (waagerechter Pfeil). Der Grund dafür, daß diese beiden Tätigkeiten nicht zusammengeführt werden, liegt darin, daß viele Darstellungen von den Interaktionsmöglichkeiten unabhängig sind und hier eine Trennung zwischen dem *Feel* und *Look* vorgenommen werden kann, um sich so eine Option zum Ändern der graphische Darstellung oder der Reaktion auf Benutzereingaben offenzuhalten. Andererseits sind Kontrollelemente, wie z.B. Scrollbars oder Buttons, mit ihrer graphischen Darstellung und den Mechanismus, den sie repräsentieren, eine untrennbare Einheit.

[3]Das Ziel eines GUI-Entwicklers ist nicht die abstrakte Natur der Objekte aus der Analysephase als solche zu verneinen, sondern sinnvolle und konkrete Darstellungs- und Interationsmöglichkeiten zu liefern.

[4]Formaler ausgedrückt: Sichten (Views) sind nach bestimmten Kriterien (funktional, organisatorisch, verfahrensorientiert) definierte Sichtweisen, die eine Abstraktion von bestimmten Details in der Betrachtung darstellen [Seg95]. Bekannteste Vertreter im Netz- und Systemmanagement sind die als "Plattform-Views" bezeichneten, von Managementplattformen angebotenen, mehr technisch orientierten Sichtweisen auf Kommunikationsresourcen.

Abhängig vom Objekttyp kann es unterschiedliche Sichtentypen geben. Eine *Komplettsicht* ist eine Gesamtdarstellung eines Objektes, bei der — wenn überhaupt — nur wenige Einzelheiten (Attribute) gezeigt werden (Piktogramme bzw. Ikonen oder Listeneinträge). Teilobjekte oder Attribute von Objekten werden in *Datail-Sichten*, meist in eigenen Fenstern, dargestellt.

Die Sichten sind so zu gestalten, daß aufgabenangemessene und benutzbare Darstellungen der Objekte entstehen. Zum Beispiel sind Attribute in Detailsichten so auszuwählen, daß die dargestellten Informationen dem Informationsbedarf für die jeweils zu bearbeitende(n) Aufgabe(n) entsprechen. In der Regel wird über eine Sicht eine Reihe von Aufgaben bearbeitet, so daß interne Mechanismen (Dialogkontrolle) bereitgestellt werden müssen, die eine, zu jedem Zeitpunkt, konsistente Darstellung der Informationen am Bildschirm gewährleisten. Je mehr Informationen sichtbar sind, desto schwieriger und komplexer sind die zu entwickelnden Mechanismen (siehe auch Abschnitt 3.2). Bei der Gestaltug der Fenster für die Sichten ist zu berücksichtigen, daß weder zu viele Informationen dargestellt noch zu viele Fenster für eine Aufgabenbearbeitung gleichzeitig benötigt werden. Hier ergeben sich natürlich bei komplexen Anwendungen Zielkonflikte. Solange noch nicht zu viele Fenster vorhanden sind, können neue Teilobjekte eingeführt werden, für die dann Sichten in eigenen Fenstern dargestellt werden.

Abbildung 4.4 zeigt, aus dem Kontext des ganzen Entwicklungsprozesses, den Tätigkeitsbereich des Entwurfs und der Spezifikation der Informationspräsentation im Überblick. Dabei zerfällt der Ablauf in den Entwurf der Objektsichten (Detail- und Komplettsichten) und deren Zusammensetzung (Anordnung am Bildschirm und zeitlicher Ablauf bzw. Aufrufreihenfolge, -hierarchie der einzelnen Fenster) zur vollständigen Bedienoberfläche.

Diese Schritte sind wieder iterativ (angedeutet durch die Pfeile im Hintergrund) und eng mit dem Entwurf der Interaktions- und Kontrollmechanismen verbunden.

4.6 Entwurf und Spezifikation der Interaktions- und Kontrollmechanismen

Interaktions- und Kontrollmechanismen erlauben den Anwender mit dem System zu interagieren. Es ist durchaus möglich und üblich mehrere Interaktionstechniken auf einem Informationsobjekt zu realisieren. Typische Beispiele, um mit dem System zu interagieren, sind die Eingabe von "getippten" Kommandos, das Drücken bestimmter Funktionstasten oder Tastenfolgen, mit der Maus auf Objekte zeigen oder Aktionen über Menüs auswählen.

Obwohl viele Merkmale der Präsentation und Interaktion voneinander unabhängig sind, gibt es denoch Abhängigkeiten, wodurch deren parallele Entwicklung notwen-

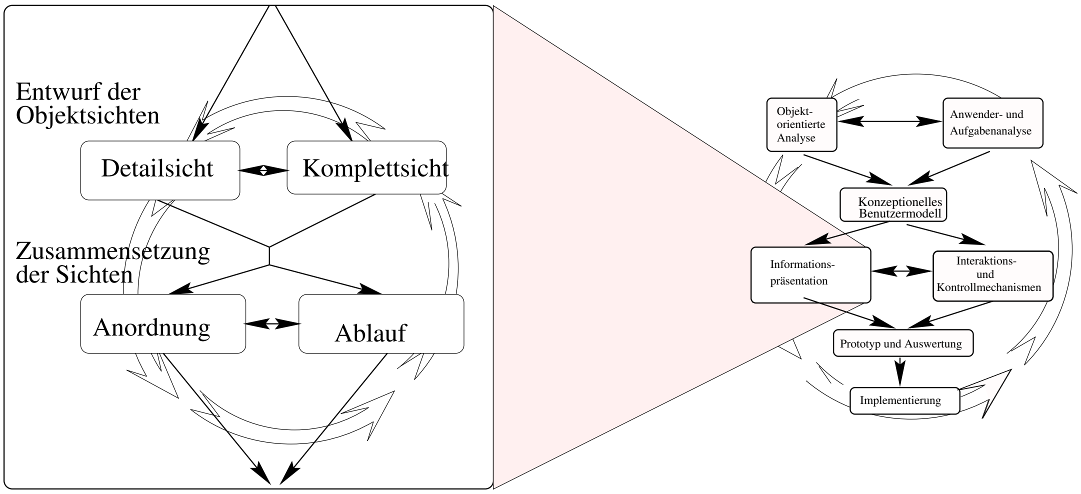


Abbildung 4.4: Entwurf und Spezifikation der Informationspräsentation

dig ist. Kontrollelemente sind typischerweise am Bildschirm sichtbar, womit viele Kontrollelemente sowohl ein *Look* als auch ein *Feel* besitzen. Z.B. ein Button ist am Bildschirm sichtbar und bewirkt beim Anklicken, daß irgendetwas ausgelöst oder gestartet wird. Die Entwicklung der Interaktions- und Kontrollmechanismen läuft, wie in Abbildung 4.5 schematisch dargestellt, im wesentlichen darauf hinaus, daß man ausgehend von den bereits erwähnten Grundtechniken, die i.d.R. durch *Style-Guides* definiert sind, diese auf Detail- und Komplettsichten überträgt, neue einführt und schließlich noch Überlegungen für das Gesamtverhalten und den Gesamtablauf anstellt.

Typische Fragestellungen in diesem Tätigkeitsbereich lauten z.B.: Was soll z.B. beim einfachen oder zweifachen Anklicken mit der linken, mittleren oder rechten Maustaste, beim Betreten oder Verlassen eines Widgets, eines Buttons oder Fensters passieren? Diese Aufzählung läßt sich fast beliebig fortsetzen. Für die Spezifikation haben sich sogenannte Objekt/Objekt- oder Objekt/Aktion-Matrizen bewährt. In den Feldern der Matrix werden Reaktionen festgeschrieben, wenn ein Objekt (Zeilenbeschriftung) auf ein anderes Objekt (Spaltenbeschriftung) plaziert oder wenn auf einem Objekt (Zeilenbeschriftung) eine Aktion (Spaltenbeschriftung) ausgeführt wird. In dieser Arbeit werden die Interaktions- und Kontrollmechanismen im Zusammenhang mit der Informationspräsentation nur informell besprochen. Auf eine Spezifikation mit Hilfe der Objekt/Objekt- oder Objekt/Aktion-Matritzen wird zugunsten des besseren Leseflusses und der Platzersparnis verzichtet, da viele Interaktions- und Kontrollmechanismen trivial sind[5].

[5]Z.B. Doppelklick auf eine Ikone (Komplettsicht) öffnet i.d.R. ein Fenster (Detailsicht), oder das Anklicken eines Buttons startet irgendetwas, usw.

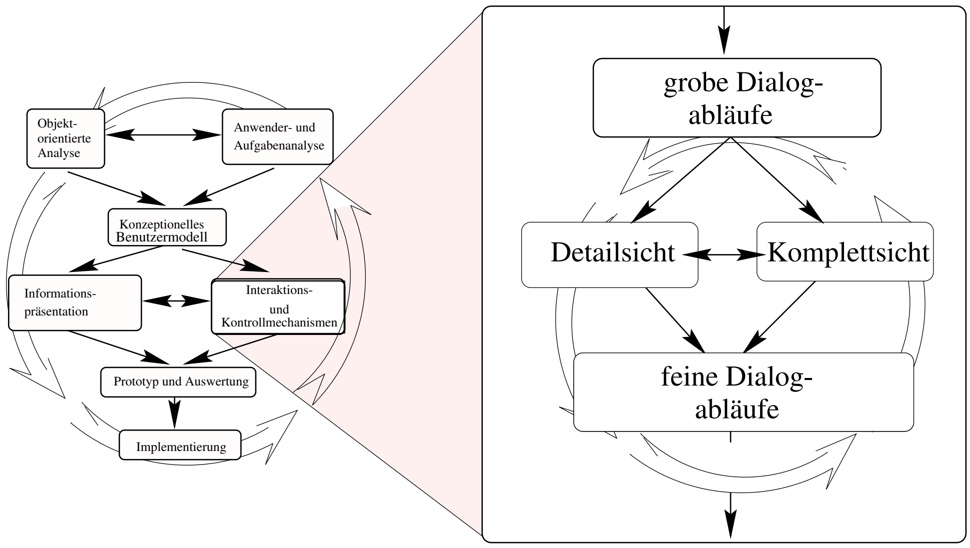


Abbildung 4.5: Entwurf und Spezifikation der Interaktions- und Kontrollmechanismen

4.7 Prototyp, Evaluation und Rückkopplung

Die bisher beschriebenen Modelle liefern Darstellungsmöglichkeiten und eine methodische Grundlage für das Vorgehen beim objektorientierten Benutzungsschnittstellenentwurf. Die Ergebnisse (z.B. Objektmodelle, Objekt/Objekt-, Objekt/Aktion-Matrizen, usw.) sind aber hauptsächlich für die Software-Entwickler als Spezifikation und Dokumentation interessant. Zur Bewertung des Entwurfes zusammen mit dem Benutzer müssen zyklisch Prototypen erstellt werden. Prototyping erlaubt dem Entwickler die geforderte Funktionalität zu prüfen und liefert ihm dabei wertvolle Rückkopplung, bevor teure Ressourcen zur Implementierung gebunden werden.

Wichtig für die Bewertung des Entwurfs sowie für die Validierung der Analyse-Ergebnisse ist also die Konstruktion und Evaluierung (Auswertung) eines lauffähigen Prototypen für die Benutzerschnittstelle. Die Evaluierung des Prototypen erfolgt anhand möglichst realistischer Aufgabenbearbeitungen durch die späteren Endbenutzer. Sie liefern nicht nur die fachlichen Informationen in der Analysephase und beim Entwurf des verfeinerten Objektmodells, sondern sind i.d.R. auch am Entwurf der Benutzungsschnittstelle beteiligt, entweder als Mitglieder des Entwurfsteams, oder zumindest aber als Testpersonen bei der Evaluierung. Hierdurch können Fehler und Unvollständigkeiten des Entwurfs frühzeitig erkannt und verbessert werden.

Die rückwärts gerichteten Pfeile in Abbildung 4.6 machen deutlich, daß hier nicht ein streng sequentielles Phasenmodell vorliegt. Die Ergebnisse der Anwender- und Aufgabenanalyse und der objektorientierten Analyse fließen in das konzeptionel-

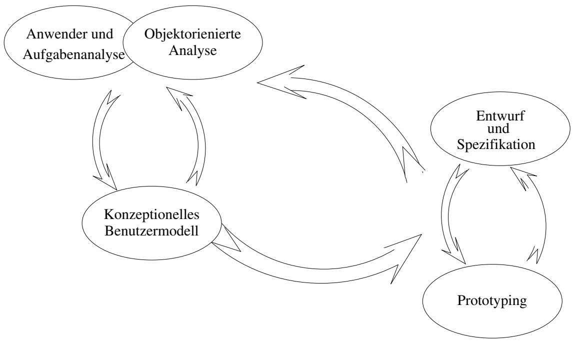


Abbildung 4.6: Rückkopplung

le Benutzermodell ein, wobei eine Rückkopplung, angedeutet durch den rückwärts gerichteten Pfeil, durchaus möglich und sinnvoll ist. Ähnlich ergänzen sich die Entwurfs- und Spezifikationsphase und die Erstellung des Prototypen. Im Gesamtzusammenhang beeinflussen sich, auf der linken Seite, die Analysephase und, auf der rechten Seite, die Realisierungsphase (Entwurfs- und Spezifikationsphase, *Prototyping*) durch Rückkopplung. Die sequentielle Anordnung in Abbildung 4.1 zeigt vielmehr den logischen Zusammenhang und den idealisierten Ablauf zwischen den verschiedenen, aufeinander aufbauenden Tätigkeiten und deren Ergebnissen, die im Verlauf der Entwicklung erstellt werden, auf. Das eigentliche (idealisierte) Vorgehen ist iterativ, es läuft in Zyklen, in denen jeweils am Ende neue Versionen oder Prototypen erstellt und bewertet werden, ab. Als Ergebnis der einzelnen Evaluierungen werden dann jeweils die erstellten Modelle korrigiert.

Wie tief die Modelle jeweils ausgearbeitet werden und wieviele Zyklen durchlaufen werden, hängt stark von dem jeweiligen Projekt ab. So kann es sein, daß bei gut strukturierten und im Prinzip bekannten Problemen die Analyseergebnisse weitgehend vollständig ausgearbeitet und kaum korrekturbedürftig sind. Bei weniger vertrauten Problemen werden dagegen die Prototypen ein entscheidender Faktor bei der Auswertung der vorhergehenden Phasen sein. Von der Erfahrung der Projektteilnehmer in bezug auf die Gestaltung graphischer Oberflächen hängt es außerdem ab, wieviele Entwurfszyklen durchlaufen werden müssen, um zu einer endgültigen Benutzungsschnittstelle zu kommen.

Zum Prototyping gibt es verschiedene Auffassungen. Im Bereich des klassischen Softwareengineering soll der Prototyp nur als Hilfsmittel zur Diskussion über Anforderungen oder als Grundlage für die Einschätzung der Machbarkeit genutzt werden. Beim Einsatz objektorientierter Technologien[6] wird der Prototyp Bestandteil

[6]Objektorientierte Softwareentwicklung, objektorientierte Bedienoberflächen, objektorientierte Programmierung und objektorientierte Datenbanken.

des Zielsystems. Er ist kein Wegwerf-Modell mehr und bleibt auch nach Fertigstellung des Produkts als Referenzsystem für Änderungen erhalten. Abhängig vom eingesetzten Projektierungswerkzeug bieten sich für eine Benutzerschnittstellen-Entwicklung beide Varianten an. Nachfolgend sind einige Vor- und Nachteile von Prototypen aufgeführt.

Vorteile:

- Prototypen erlauben etwas auszuwerten, daß aussieht wie das Endprodukt und liefern deshalb ein vernünftiges *Feedback*.
- Prototypen entlarven früh im Entwicklungszyklus Fehler in der Anforderungsanalyse und Entwurf.
- Prototypen dokumentieren den Entwurf besser als es Papier je könnte.
- Prototypen liefern Ansatzpunkte für Diskussionen, sowohl für die Entwickler und Projektmanager als auch für Testpersonen.

Nachteile:

- Realistische Prototypen einer komplexen Benutzerschnittstelle erfordern einen hohen Aufwand und der Ressourcen bindet, die zur Implementierung des Produkts benötigt werden.
- Prototyping kann auch dazu verleiten die Anforderungsanalyse und den Entwurf nicht gründlich genug durchzuführen.
- Prototypen unterscheiden nicht zwischen notwendigen Merkmalen des Entwurfs und die von den dazu benutzten Werkzeugen zufällig eingeführten Merkmalen.
- Prototypen können auch zu Mißverständnissen führen. Manager und Endanwender, die nicht in den Entwicklungsprozeß mit eingebunden sind, sehen einen Prototyp und glauben, daß das Projekt kurz vor der Vollendung steht, was in Wirklichkeit nicht der Fall ist.

Kapitel 5

Einordnung des Entwicklungskonzepts in den Entwicklungsprozeß des Gesamtsystems

Eine gute graphischen Benutzerschnittstelle kann nicht isoliert entwickelt werden. Sie ist Teil eines Gesamtsystems und muß deshalb mit den anderen Teilen des Systems entwickelt und implementiert werden. Dieses Kapitel beschreibt, wie das Entwicklungskonzept in den Entwicklungsprozeß des Gesamtsystems einzuordnen ist.

Abbildung 5.1 zeigt die bei großen Projekten übliche Abspaltung des Benutzerschnittstellenentwurfs vom Entwurf des funktionellen Kerns nach der Analysephase; beide werden dann in der Implementierungsphase wieder zusammengeführt.

Abbildung 5.2 zeigt, wie die einzelnen Aktivitäten des Entwicklungskonzepts in den Gesamtentwicklungszyklus (Planung, Analyse, Design (Entwurf- und Spezifikation), Implementierung) einzuordnen sind. Dabei handelt es sich nur um eine grobe Zuordnung, welche stark von der jeweiligen Projektgröße abhängt.

5.1 Planung

Die Planung geht der Entwicklung voraus, um die Problemwelt des Projekts und die Anwender zu identifizieren. Das zu lösende Problem wird erfaßt und beschrieben, sowie festgelegt was das zu entwickelnde Programm leisten soll. Es geht in erster Linie um eine Bestandsaufnahme der Problemwelt, dann um eine Festlegung des Funktionsumfangs des Systems. Der große Entscheidungsspielraum bei der Gestaltung der Bedienoberfläche und die Auswirkungen von guter oder schlechter Ge-

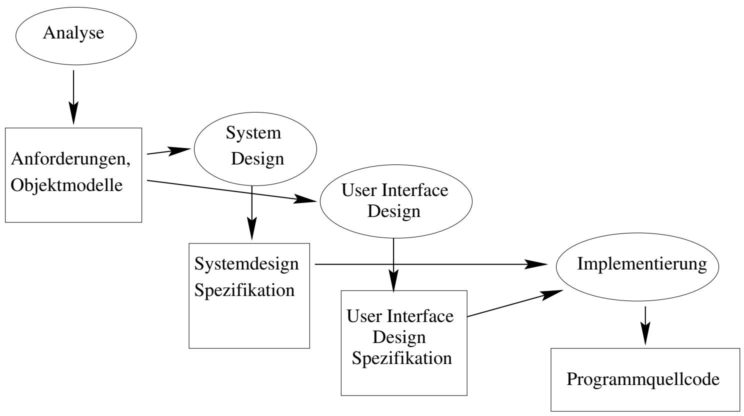


Abbildung 5.1: Entwicklungsphasen und deren Ergebnisse

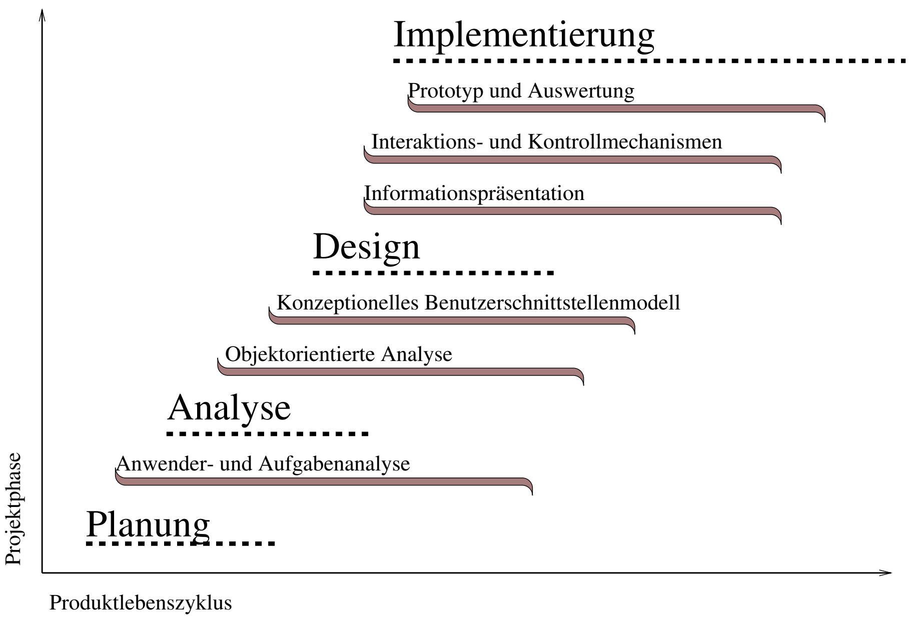


Abbildung 5.2: Aktivitäten des Entwicklungskonzeptes im *Software Life Cycle*

staltung auf die Bedienbarkeit und damit der Akzeptanz und Erfolg eines Programms erfordern aber, daß zumindest in groben Zügen Anforderungen an die Bedienoberfläche in der Plannungsphase aufgenommen werden. Die Auftraggeber des Systems bzw. des Programms beginnen mit dieser Phase und ziehen die Entwickler an einer geeigneten Stelle mit ein. Darin liegt auch der Grund, daß der Beginn der Anwender- und Aufgabenanalyse in dieser Phase eingeordnet werden kann, um so, in Zusammenarbeit mit den Auftraggebern, Entscheidungen zu vermeiden, die später nur sehr schwer durchführbar sind.

5.2 Analyse

Die Arbeit in der Analysephase hat zwei Aspekte. Der Erste ist die in der Planungsphase begonnene Anwender- und Aufgabenanalyse fortzusetzen. Diese Arbeit wird Eingaben für die weiteren Tätigkeiten liefern. Der zweite Aspekt ist die Einbeziehung der Arbeit der Systemanalysten, die ein Analysemodell erstellen. Es sollte alle Szenarien unterstützen, die in den Systemanforderungen spezifiziert sind. Der Gebrauch von Anwendungsszenarien, basierend auf der Aufgabenanalyse, hilft bei der Erstellung und Validierung des Analysemodells, und zur Konstruierung von sinnvollen Testfällen für das System. Die mit den Benutzerschnittstellenentwickler und dem Analyseteam in Zusammenarbeit entwickelten Szenarien haben viele Vorteile. Die Analysten bekommen ein besseres Produkt für weniger Eigenleistung und die UI-Entwickler können auf bereits geleistete Mühen der Anaysten zurückgreifen und stellen somit sicher, daß ihre Entwicklung dem gleichen Modell zugrunde liegt, welches, wie bereits mehrfach erwähnt den ganzen (System-)Entwicklungsprozeß steuert. Das Analysemodell selbst sollte strukturell gleich sein wie das konzeptionelle Benutzermodell. Wenn nicht, so ist es in diesem Stadium einfacher Differenzen auszubügeln als zu einem späteren Zeitpunkt, wenn die Anwender den Mißstand aufdecken. Der Prozeß der Angleichung wird zu einem besseren konzeptionellen Benutzerschnittstellenmodell und Benutzerschnittstellenentwurf führen.

5.3 Entwurf und Spezifikation — Design

Das Aufgabenmodell sollte bei Beginn der Designphase bereits vorhanden sein und das Analysemodell sollte bereits die Aufgaben des Anwenders und deren Vorstellungen vom System reflektieren. Das Entwicklung des konzeptionellen Benutzermodells sollte zu diesen Zeitpunkt bereits vorangeschritten sein, wenn mit der Entwicklung der Präsentation und den Interaktions- und Kontrollmechanismen begonnen wird. Erste Versionen eines Prototypen sind in dieser Phase nicht unüblich.

5.4 Implementierung

Abbildung 5.2 zeigt, daß die Aktivitäten im Entwurf der Informationspräsentation und der Interaktions- und Kontrollmechanismen über die Systemdesignphase hinausreichen. Endgültige Verfeinerungen am Entwurf in diesen Bereichen müssen die Implementierung nicht aufhalten. Sind erst einmal das konzeptionelle Benutzerschnittstellenmodell, die grundlegenden Sichten und die Interaktions- und Kontrollmechanismen *(Look & Feel)* festgelegt, kann die Implementierung der Benutzerschnittstelle und der restlichen Bestandteile des Systems nahezu parallel erfolgen. In Abschnitt 4.7 wurde bereits besprochen, wie die Realisierung eines Prototypen zwischen der Design- und Implementierungsphase eingeordnet werden kann.

Teil III

Projektierung

Kapitel 6

Analyse

In diesem Kapitel wird die Analysephase des Entwicklungskonzepts angewandt, um die Anforderungen an die graphische Bedienoberfläche zu identifizieren und festzuschreiben.

In Abschnitt 6.1 werden die Rahmenbedingungen, in denen das VLAN-Managementsystem eingesetzt wird und was es in groben Zügen leisten soll, festgelegt. Um den Prototypen (der Bedienoberfläche) nicht zu sehr mit Ausnahmebehandlungen, die sich aufgrund der Komplexität und Heterogenität heutiger physischer Netze ergeben und daher im allgemeinen keine VLAN-spezifischen Probleme darstellen, zu überladen, wird in Abschnitt 6.1.1 eine physische Netzumgebung/Szenario festgelegt, das dieser Arbeit dann zugrundeliegt. Dadurch soll erreicht werden, daß der Schwerpunkt der Betrachtungen bei den VLAN-spezifischen Fragestellungen bleibt. In Abschnitt 6.1.2 wird festgelegt, welche VLAN-spezifischen Managementaufgaben mit der graphischen Bedienoberfläche durchführbar sein sollen. Die Analyse der Aufgaben ergibt die notwendigen Funktionalitäten und bilden mit den grundlegenden Objekten der Problemwelt die Grundlage weiterer Analysetätigkeiten.

Abschnitt 6.1.3 beschreibt die für das VLAN-Management erforderlichen (Grund-) Sichten[1] grob und benennt sie. Damit werden zwar Ergebnisse der Entwicklungstätigkeiten vorweggenommen. Rechtfertigen läßt sich diese Vorgehensweise damit, daß der Lesefluß nicht mit unnötigen Umschreibungen strapaziert und, als Folge davon, eine Aufblähung des Textes vermieden wird. Im Verlauf der Durchführung des Entwicklungskonzepts kommen noch weitere Sichten hinzu.

Bevor mit der Analyse der Anwender und deren Aufgaben begonnen wird, werden in Abschnitt 6.2 bereits vorhandene Komponenten und Lösungen auf ihre Wieder-

[1] Sichten, zur Erinnerung, sind, im Zusammenhang mit der GUI-Entwicklung, jede Art der Repräsentation von Objekten am Bildschirm. Objekte, die mehrere gleiche oder verschiedene Objekte enthalten, stellen wieder Objekte dar, deren Repräsentation am Bildschirm wieder eine Sicht (Abschnitt 4.5). Einfachste Beispiele sind die Fenster einer graphischen Bedienoberfläche. Vergl. auch Viewbegriff im Netz- und Systemmanagement und in Datenbanken.

verwendbarkeit hin untersucht.

Die erste Phase im Rahmen des Entwicklungskonzepts, die Analysephase, beginnt in Abschnitt 6.3, mit der Anwender- und Aufgabenanalyse und der objektorientierten Analyse (Abschnitt 6.4), die beide dann in das konzeptionelle Benutzerschnittstellenmodell (Abschnitt 6.5) einfließen.

Bemerkung: Da eine vollständige Systemanalyse, die daran anschließende Entwurfs- und die Spezifikationsphase des Systems (Designphase) und deren Implementierung, kurz der ganze Lebenszyklus *(Lifecycle)* eines Software-Systems, den Umfang einer Diplomarbeit bei weitem übersteigt, ist es notwendig nur die im Titel der Arbeit genannten Aspekte zu verfolgen und wenn es sich anbietet verschiedene Tätigkeiten/Phasen des Entwicklungskonzepts im Text gleichzeitig einzuarbeiten. Es muß also eine Abwägung zwischen der strikten Einhaltung des Entwicklungskonzepts (vollständige Spezifikation) und den Betrachtungen der wesentlichen Aspekten, die das VLAN-Management charakterisieren, bei gleichzeitiger Gewährleistung des Leseflusses, gefunden werden. Konkret wirkt sich das z.B. in der Anwender- und Aufgabenanalyse aus. Dort tauchen, wenn es sich anbietet, bereits Interaktions -und Kontrollmechanismen auf, die erst in der Designphase erarbeitet und spezifiziert werden sollten, genauso werden bereits visuelle Anforderungen an den Fensterinhalten gestellt, obwohl sie, in diesem Stadium der Analysephase, im Grunde nur (abstrakte) Datenbankobjekte darstellen.
Andererseits werden die objektorientierte Analyse und das konzeptionelle Benutzerschnittstellenmodell nur am Rande behandelt, da erstere bereits in den Arbeiten eines Kollegen [Vuk96] und meines Betreuers erarbeitet wird und beide, OOA und konzeptionelles Benutzerschnittstellenmodell, den Schwerpunkt in der Datenmodellierung setzen und mehr für die Implementierung des Gesamtsystems eine Rolle spielen. Die Tätigkeit beschränkt sich in dieser Arbeit somit nur in der Identifizierung der für die Benutzerschnittstelle relevanten Objekte und deren Beziehung zueinander. Auf eine genauere Spezifikation der Datenstrukturen, Attribute und Methoden wird verzichtet.

6.1 Anforderungsspezifikation

Dieser Abschnitt legt die Anforderungen an das VLAN-Managementsystem (oder auch kurz System), insbesondere die Anforderungen an die graphische Bedienoberfläche, in groben Zügen fest. Die Anforderungen sind relativ knapp gehalten, sie werden im Verlauf des Entwicklungsprozesses weiter konkretisiert und verfeinert.

Am Anfang eines Entwicklungsprozesses steht in der Regel die Anforderungsspezifikation der Auftraggeber, die auch zusammen mit den Systementwicklern erstellt werden kann. Da für diese konzeptionelle Arbeit keine *Auftraggeber* in diesem Sinne vorhanden sind, ist es wichtig im Rahmen der Diplomarbeit festzulegen, was das

System respektive dessen Bedienoberfläche leisten und in welcher physischen Netzumgebung, welchem Szenario es eingesetzt werden soll.

Typische Aufgaben aus dem VLAN-Management, die mit dem System durchführbar sein sollen, bilden den Ausgangspunkt der Entwicklungstätigkeiten, da daß Entwicklungskonzept für die Benutzerschnittstelle einer Software-Entwicklungsmethode nachempfunden ist. Bei der Auswahl der Aufgaben und der zugrundeliegenden konkreten Netzumgebung wurde darauf geachtet, nur diejenige zu betrachten die heute oder zumindest in naher Zukunft realisierbar sind.

6.1.1 Zugrundeliegende konkrete Netzumgebung

Endstationen sind an Ethernetsegmente angeschlossen. Die Segmente sind ihrersets über Ports an den Ethernet-Switches[2] angeschlossen.
Die Switches sind über leistungsfähige Backbones miteinander verbunden. Als Backbone kommt entweder ein ATM-Netz mit LAN Emulation zum Einsatz oder, im Falle mehrerer zueinander kompatibler Switches, beliebige andere Lösungen/Techniken, die mit proprietären Protokollen betrieben werden. Proprietär deshalb, weil es für die Interswitchkommunikation keine Standards gibt und deshalb die Switches vom gleichen Hersteller sein oder, wie angedeutet, ein gemeinsames Protokoll benützen müssen. Die ATM LAN Emulation wurde bereits in Abschnitt 2.5 besprochen und mit IEEE 803.10 in Abschnitt 2.4.2 eine Variante für die Interswitchkommunikation.

Bemerkung: Diese Netzumgebung impliziert nicht, daß jede Endstation an jedem VLAN teilnehmen kann. Das zu entwickelnde System hat zwar den *Gesamtüberblick*, kann aber unter Umständen ein VLAN, daß sich über mehrere zueinander inkompatibler Vermittlungskomponenten (Switches) erstreckt, nicht bilden.

6.1.2 Auswahl VLAN-spezifischer Aufgaben

In diesen Abschnitt wird festgelegt, welche VLAN-spezifischen Managementaufgaben mit dem System bzw. mit der Bedienoberfläche durchführbar sein sollen. Bei den Aufgaben handelt es sich um eine Auswahl aus dem Bereich des Leistungs-, Konfigurations- und Fehlermanagements. Zu beachten ist, daß es sich hier nur um eine Auswahl VLAN-spezifischer Managementaufgaben handelt, die im Rahmen eines Einmannprojekts, wie es diese Diplomarbeit darstellt, bearbeitet werden kann. Andererseits wird mit dieser Auswahl bereits ein großer Bereich der häufigsten und grundlegensten Tätigkeiten eines (VLAN-)Administrators abgedeckt und mit dem zu entwickelnden System eine solide Grundbasis für Verfeinerungen und Erweiterungen geschaffen.

[2] Im weiteren Verlauf der Arbeit mit Switches oder allgemeiner mit Switching-Komponenten bezeichnet.

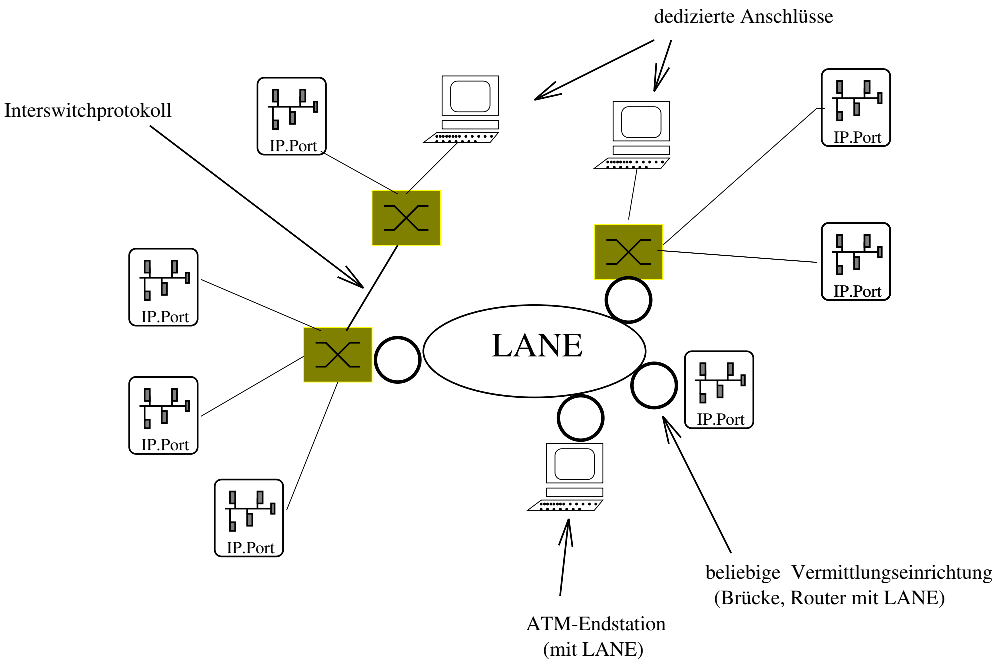


Abbildung 6.1: Zugrundeliegende Netzumgebung

Ein Großteil der Managementaufgaben, die ein Administrator mit dem System zu erledigen hat, sind Tätigkeiten aus dem Konfigurationsmanagement. Kreieren, Modifizieren und Löschen von VLANs, Einfügen und Herausnehmen von VLAN-Mitgliedern sind dabei mit Abstand die häufigsten Tätigkeiten im Hinblick auf VLANs.

Weiterhin sind typische Fragestellungen aus dem Leistungsmanagement interessant. Man ist darauf bedacht eventuelle Verkehrsengpässe *(Bottlenecks)* möglichst früh zu erkennen, um so rechtzeitig mit geeigneten Maßnahmen reagieren zu können.

Konkret sind Mechanismen zum Beobachten und Historisieren der Verkehrslast (auf Backbones und Segmenten), der Verteilung der VLANs über einem Backbone und des Verkehrs innerhalb und zwischen VLANs außerordentlich wichtig, um die Nachteile klassischer LANs (hohe Netzstrukturen, Kollisionen, Verkehrsengpässe, usw.) nicht zu vergrößern, sondern zu verringern.

Als Backbone kann ein ATM-Netz mit LAN Emulation eingesetzt werden. Dieses soll aber weitgehend im Hintergrund bleiben, d.h. bei der Einrichtung eines VLANs unter Zuhilfenahme der LAN Emulation, werden die notwendigen Instantiierungen der Clients und Server (der LAN Emulation) automatisch vom System eingerichtet. Mit der Bedienoberfläche des Systems soll ein auftretender Fehler der LAN Emulation angezeigt, der Fehler lokalisiert und die Fehlerursache festgestellt werden können.

Desweiteren sind für den Administrator der Standort und Status der einzelnen Komponenten interessant. Weitere Betrachtungen in bezug zur LAN Emulation geht schon zu sehr in Richtung ATM-*Device*-Management und eines eigenständigen LAN Emulation Managers, dessen Darlegung für das VLAN-Management auf dieser hohen Ebene und in dieser Arbeit zu weit führen würde. In einer konkreten Realisierung sind natürlich die entsprechenden (Element-)Manager oder die notwendigen Funktionalitäten implementiert und aufrufbar. In Abschnitt 6.2.1 wird eine funktionelle Schichtung eingeführt, die versucht genau diese Aufgabenbereiche bereits existierender Lösungen gegeneinander abzugrenzen, um sich so der bereits vorhandenen Funktionalitäten zu bedienen und das Augenmerk auf die VLAN-spezifischen Anforderungen gerichtet zu lassen.

Die Anforderungen an die Bedienoberfläche werden zusammenfassend durch folgende Aufgaben[3] indirekt festgelegt:

- **Aufgabe: Kreieren, Modifizieren und Löschen von VLANs**
- **Aufgabe: Hinzufügen und Herausnehmen von Mitgliedern**

Bei den nächsten Tätigkeiten des Administrators wird in dieser Diplomarbeit der Schwerpunkt in die Visualisierung der vom System angebotenen Informationen am Bildschirm gelegt.

- **Aufgabe: Management der ATM LAN Emulation**
  Die LAN Emulation sorgt zwar für Ortstransparenz, d.h. das ATM Netz erscheint als *Black Box*, trotzdem sind folgende Aspekte für das VLAN-Management von Interesse:
  - Visualisierung beteiligter Komponenten (ATM-Hardware und Struktur und Komponenten der LAN Emulation)
  - Feststellen von Fehlern, Lokalisieren der fehlerhaften Komponente(n) und Ursache für Fehler bestimmen
- **Aufgabe: Verkehr analysieren und auswerten**.
  Für das VLAN-Management sind folgende Aspekte besonders interessant:
  - Gesamtdarstellung der Auslastung der Backbones und Segmente
  - Verkehr und Verteilung von VLANs über einen Backbone
  - Verkehr und Verteilung der Protokolle auf einem Segment
  - Verkehr innerhalb eines VLANs

[3] Zur Erinnerung: Ausgehend von diesen Aufgaben wird im wesentlichen die vollständige Bedienoberfläche im Rahmen des Entwicklungskonzepts erarbeitet.

- Verkehr zwischen VLANs

Bei diesen Tätigkeiten und der zugrundeliegenden konkreten Netzumgebung handelt es sich um heute schon realisierbare Szenarien, d.h. ohne das Änderungen an Endsystemen vorgenommen werden müssen. Viele darauf aufbauende und angedachte Einsatzszenarien für VLANs ziehen Erweiterungen oder Änderungen in den Endsystemen[4] und sogar, in einen weiteren Schritt, die Einbeziehung des Anwenders[5] nach sich.

6.1.3 Sichten auf das Systems

Nachdem die zugrundeliegende physische Netzumgebung und die Aufgaben, die mit der Bedienoberfläche durchführbar sein sollen, festgelegt wurden, ergeben sich bereits Sichten auf das System, die (ohne große Analysetätigkeiten) von der Bedienoberfläche angeboten werden muß. Um den Text, bis zum Entwurf und Spezifikation der Informationspräsentation (im Entwicklungskonzept), nicht mit unnötigen Umschreibungen zu strapazieren, werden im folgenden die vier grundlegenden Sichten kurz eingeführt und benannt.

Ein Administrator, der ein VLAN einrichten muß, hat in der Regel schon eine Vorstellung davon, aus welchen Endsystemen es bestehen soll. Das kann eine willkürliche Menge von Endsystemen (eindeutig charakterisiert durch Adressen, logische Namen, usw.) oder durch ein Kriterium (Abteilungs-, Workgroup-Zugehörigkeit, bestimmtes Protokoll, usw.) festgelegt sein.
Was ihn interessiert ist zunächst einmal eine Gesamtübersicht aller existierender VLANs. Diese Sicht wird in dieser Arbeit

- logische (Gesamt-)VLAN-Sicht

genannt. Parallel dazu interessiert er sich für eine physische Gesamtdarstellung des Netzes, wobei nur VLAN-relevante Komponenten (Endsysteme, Ethernetsegmente, Switches, evtl. Router, Backbones, usw., aber keine Repeater, Brücken, usw.) gezeigt werden sollen. Diese

- physische Gesamtsicht

kann optional sein. In dieser Arbeit wird die physische und logische Sicht in einem Fenster nebeneinander dargestellt.

[4]Verwendete Protokolle, Betriebssystem, Hardwarekomponenten, usw.
[5]Anwender ist Mitglied in mehreren VLANs/Workgroups gleichzeitig.

Selektiert er ein existierendes VLAN oder will ein VLAN einrichten, erscheint in beiden Fällen ein gleiches Fenster, mit dem Unterschied, daß letzteres nur Standardwerte (u.a. einen vom System vergebenen fortlaufenden VLAN Identikator enthält. Diese

- VLAN-spezifische-Sicht

enthält in der

- logischen VLAN-spezifischen-Sicht

die aktuellen Mitglieder (Endstationen, Server) des VLANs. Parallel dazu zeigt die

- physische VLAN-spezifische-Sicht

den entsprechenden VLAN-relevanten Ausschnitt des physischen Gesamtnetzes. Die Bestandteile und Anordnung der Funktionalitäten und Sichten auf die darin enthaltenen Objekte wird im Rahmen der Durchführung des Entwicklungskonzepts herausgearbeitet und genauer spezifiziert.

Diese (Grund-)Sichten bilden die Basis des Systems. Von hier aus werden dann die weiteren Verzweigungsmöglichkeiten realisiert. Dieser Abschnitt diente nur zur Begriffsbildung, um im weiteren Verlauf des Textes die notwendigen *Termini Technici* parat zu haben.

6.2 Analyse bestehender Komponenten und Lösungen

Nachdem in die Anforderungen an die Bedienoberfläche in groben Zügen festgeschrieben sind, wird in Abschnitt 6.2.1 versucht diese zu strukturieren, um so die Komplexität besser in den Griff zu bekommen.
Als Hilfsmittel der Strukturierung dient das Prinzip der Schichtenbildung. Als Ergebnis wird eine Zerlegung der Gesamtfunktionalität in aufeinander aufbauenden Schichten geliefert. Diese Schichtung ist notwendig, um zu differenzieren, welche Funktionalitäten und — für diese Arbeit besonders wichtig — welche Darstellungsmöglichkeiten, Repräsentationen, Sichten auf aus bisherigen Managementlösungen klassischer Netze übernommen werden können, welche nur modifiziert und welche noch konzipiert werden müssen.

Daran anschließend werden in Abschnitt 6.2.2 zwei vorhandene zwei Elementmanager für ATM-Switches des jeweiligen Herstellers mit implementierter LAN Emulation, untersucht, um so evtl. bereits existierende Realisierungen der Anforderungen aus der Analysephase zu extrahieren.

6.2.1 Funktionelle Schichtung des Systems

Um die VLAN-spezifischen Aspekte von bereits existierenden Managementlösungen zu extrahieren, wird, wie in der Informatik oft angewendet, das Prinzip der Schichtenbildung angewandt. In diesem Fall handelt es sich um eine funktionelle Schichtung, die, wie in Kapitel 4 angedeutet wurde und im folgenden näher dargelegt wird, in der Gestaltung der graphischen Bedienoberfläche einen direkten Einfluß hat.

Um VLANs als oberste Schicht/Abstraktionsstufe effektiv und effizient zu managen, muß man in einem nächsten Schritt auch Hersteller-VLANs[6] und die ATM-LAN Emulation managen können, dazu ist wiederum ein Management des ATM-Netzes oder der einzelnen klassischen LANs notwendig, das seinerseits wieder auf das Mangagement der einzelnen Komponenten beruht. Jede einzelne Schicht dieser Abstufung benutzt die Dienste der darunterliegenden. Die dadurch eingeführte Schichtenbildung zeigt Abbildung 6.2.

Abbildung 6.2: Funktionelle Schichtung

Die unterste Schicht bildet die Komponenten-Schicht. Das ist die Domäne der Elementmanager. Sie können durch Zugriffe auf nicht standardisierte MIB-Werte Informationen liefern, die sonst nicht erreichbar wären, so daß die Politik der Hersteller aufgeht, indem die physischen Komponenten zusammen mit den jeweiligen Elementmanagern gekauft werden müssen und den Entwicklern nichts anderes übrigbleibt, als darauf aufzusetzen. Ähnliches gilt auch für Komponenten der anderen Schichten (ATM Komonenten, Switches, usw.).

Die (Gesamt-)Netz-Schicht setzt die einzelnen (VLAN-relevanten) Komponenten und Leitungen in einem zusammenhängenden Netz in Beziehung.

[6]Hersteller-VLAN: VLAN-fähige und zueinander kompatible Switching-Komponenten eines Hersteller (siehe auch Abschnitt 6.3.6. Werden meist mit einem eigenen Elementmanager ausgeliefert.

Diese beiden unteren Schichten sind durch ihre physischen Komponenten geprägt. Die darauf aufsetzenden Schichten abstrahieren durch Software, die zum Teil auch durch ASICs realisiert sein kann, von den physischen Gegebenheiten und ziehen so jeweils eine weitere Abstraktionsebene/Schicht ein. Eine Hersteller-VLAN-Manager-Domäne realisiert (innerhalb mehrerer Hersteller-VLANs) auf dieser Ebene die geforderte Virtualisierung vom physischen Netz und das ATM-Netz zusammen mit der LAN Emulation erscheint als Black-Box, die nach außen hin eine Brückenfunktionalität suggeriert. Die nächste Abstraktionsstufe bilden die virtuellen LANs, die sich auf den Diensten der einzelnen Hersteller-VLAN-Manager und den MAC-Service der LAN Emulation stützen. Das VLAN-Managementsystem realisiert im Idealfall die vollständige Virtualisierung von dem physischen Gesamtnetz. Das Workgroupkonzept könnte als weitere Abstraktionsstufe den Abschluß am oberen Ende dieser Schichtenbildung einnehmen.

Um VLANs effektiv managen zu können, muß also auch jede darunterliegende Schicht effektiv gemanagt werden können. Für jede Schicht müssen Schnittstellen zu den tieferliegenden (Schichten-)Werkzeugen/Managementanwendungen bereitstehen oder zumindest die entsprechende Funktionalität direkt im VLAN-Managementsystem implementiert sein. Da ein VLAN-Managementsystem in der Regel nicht als *Stand Alone*-Anwendung konzipiert wird, sondern z.B. auf eine Managementplattform aufsetzt, ist zu diskutieren, ob die Gesamtfunktionalität der einzelnen (Schichten-)Manager, außer den VLAN-relevanten Teil, *nocheinmal* durch das VLAN-Managementsystem angeboten werden soll. Der volle Funktionsumfang der einzelnen darunterliegenden (Schichten-)Manager wird nur in Ausnahmefällen (Fehler, usw.) benötigt.

Die Frage ist also: parallele Anordnung (an der Bedienoberfläche) der einzelnen Managementanwendungen oder eine lineare, hierarchische Aufrufstruktur, ausgehend vom VLAN-Managementsystem durch die einzelnen Schichten. Wie so oft muß ein Kompromiß zwischen diesen beiden Extremen gefunden werden.

Da jeder Funktionalität eine Sicht zugeordnet wird, sei es ein Button oder ein eigenes Fenster, besteht ein enger Zusammenhang zwischen den diskutierten funktionellen Schichten und den graphischen Sichten/Darstellungen/Repräsentationen der Objekte am Bildschirm. In der Realisierung verschwimmen die an der Bedienoberfläche angebotenen Funktionalitäten und Sichten der einzelnen Schichten-Manager zunehmend, in dem Sinne, daß dem Administrator z.B. eine Komponenten- bzw. Portsicht des Switches auf gleicher Ebene zur VLAN-Sicht angeboten werden muß, ohne daß er sich dabei durch die einzelnen Schichten bzw. Fenster des Systems zu hangeln hat.
Eine lineare Anordnung der Aufrufbeziehung durch die einzelnen Schichten(-Manager) ist somit nicht aufrecht zu erhalten.

Die Schichtenbildung erfolgt wie bereits erwähnt zur Strukturierung des VLAN-Managementsystems. Um nicht für jedes schichtenspezifisches Problem das Rad neu erfinden zu müssen, versucht man weitgehend auf bereits Vorhandenes zurückzugreifen. Anhand dieser Strukturierung lassen sich jetzt bereits vorhandene Lösungen, z.B. die von Elementmanager, Discovery-Funktionen, usw. angebotenen Dienste, identifizieren.

Die Fragestellungen lauten jetzt:

- worauf kann aufgesetzt, was kann vorausgesetzt werden?
- wo gibt es Problemlösungen?
- was kann übernehmen werden?
- was muß evtl. nur modifiziert werden?
- was muß neu konzipiert werden?

Um im weiteren Verlauf der Analyse und der Entwurfs- und Spezifikationsphase bei den VLAN-relevanten Aspekten zu bleiben, werden bei Bedarf bereits existierende Lösungen nur kurz erwähnt und eingearbeitet. Es wird z.B. vorausgesetzt, daß eine graphische Darstellung des physischen Netzes existiert; auf die dazu notwendigen Mechanismen (z.B. Discovery-funktionalität, Element-Manager) wird genausowenig eingegangen, wie auf die erforderliche Funktionalität den VLAN-relevanten Anteil davon zu extrahieren und am Bildschirm darzustellen. Durch die Schichtenbildung ist man — im Sinne der Objektorientierung — in der Lage auf diese Funktionalitäten zurückzugreifen und in seine Problemlösungen (beliebig) ein- und auszubauen.

6.2.2 Analyse vorhandener VLAN-Managementsysteme

In diesen Abschnitt werden zwei konkret vorhandene VLAN-Manager auf verwendbare Lösungen bzgl. der graphischen Bedienoberflächengestaltung hin untersucht. Es handelt sich dabei um den Transcend ATMvLAN Network Manager von 3Com und den Virtual Network Manager von Newbridge Networks. Beide sind proprietäre Elementmanager, d.h. Manager von ATM LANE-fähigen Komponenten eines Herstellers, und stützen sich bei der Bildung von VLANs auf die Dienste der LAN Emulation, was zur Folge hat, daß deren Verständnis von VLANs mit dem Begriff der ELANs nahezu identisch ist.

Die Transcend ATMvLAN Network Manager von 3Com bestehen aus vier Werkzeugen, von denen jedes eine der im vorangehenden Abschnitt 6.2.1 eingeführten, funktionellen Schichten abdeckt.
Der **ATM Device Manager**, liefert eine geräteorientierte und das **ATM Network Tool** eine verbindungsorientierte Sicht der involvierten 3Com-Switches. Beide Werkzeuge sind für das Management der physischen Komponenten zuständig. Die beiden

folgenden abstrahieren nun von den physischen Komponenten und scheinen für diese Arbeit ergiebiger zu sein. Das **LAN Emulation Tool** bietet eine Sicht auf die LES/LEC-Verbindungen und den daran beteiligten ATM-Komponenten. Von dieser Sicht ausgehend können die Operationen der Clients und Server verwaltet werden. Und schließlich wird mit dem **Virtual LAN Tool** eine Sicht auf die Verbindungen der Ethernet-Ports in den verschiedenen *vLANs*[7] angeboten. Sie erlaubt ein Management der logischen Verbindungen der Endsysteme durch die VLANs.

Diese vier Werkzeuge werden parallel, durch Ikoken symbolisiert, an einer Management-Plattform[8] zugänglich gemacht. Mit beiden letzteren Werkzeugen wird im Grunde nur ein Management der LAN Emulation realisiert. Der Transcend ATMvLAN Manager scheint dennoch im Gegensatz zum nachfolgenden Virtual Network Manager der Vielversprechendere zu sein.

Der Virtual Network Manager von Newbridge Networks ist ein Modul für deren VIVID System Manager. Die in diesem Szenario gebildeten VLANs basieren auf der LAN Emulation und Portswitching mit ein paar proprietären Besonderheiten.

Bei den untersuchten VLAN-Managern handelte es sich um eine Werbe-CD-ROM (Transcend ATMvLAN Network Manager) und eine Beta-Version (Virtual Network Manager), die beim Test ohne Hardware auskommen mußte. Beide könnten somit nur nach optischen Gesichtspunkten und nach den in den Bedienungsanleitungen beschriebenen Leistungsmerkmalen beurteilt werden. Auf eine Beurteilung wird aber aus diesem Grund — und wegen der Einschränkung auf das Management der LAN Emulation — verzichtet und an dieser Stelle nur festgehalten, daß sich durch das Studium der graphischen Bedienoberflächen der untersuchten VLAN-Manager, speziell für diese Arbeit, keine neuen Erkenntnisse ergaben. Dadurch wird auch die nicht tiefergehende Analyse beider Manager gerechtfertigt.

6.3 Anwender- und Aufgabenanalyse

Die Anwender des Managementsystems für VLANs (kurz: Administratoren) rekrutieren sich aus den Netzverantwortlichen und sind somit schnell bestimmt. Handelt es sich dabei um mehrere Personen, so sind die üblichen Fragen der Rechtevergabe und Partitionierung des zu verwaltenden Bereichs abzuklären. Ob nun mehrere oder nur einer, ob alle gleichberechtigt sind oder eine hierarchische Verteilung der Rechte vorliegt, ist kein VLAN-spezifisches Problem und soll hier deshalb nicht weiter verfolgt werden.
Im weiteren Verlauf wird nur von einem Administrator als solchen gesprochen.

[7] Eine 3Com Terminologie für ELANs.
[8] Verfügbar für HP Open View und SunNet Manager.

6.3.1 Aufgabe: Kreieren eines VLANs

Das System soll an der Bedienoberfläche einen Überblick über alle vorhandenen VLANs anbieten (logische VLAN-Sicht). Diese können nach verschiedenen Kriterien geordnet sein (alphabetisch, nach dem Erstellungsdatum, nach dem VLAN Identifikator, alle Kriterien sowohl aufsteigend als auch absteigend). Beim Kreieren eines VLANs sollen geeignete Bildschirmmasken und -formulare[9] bereitgestellt werden, die ein übersichtliches Arbeiten ermöglichen. Eine Undo-Funktion soll bei der Rücksetzung versehendlich gelöschter Einträge bzw. VLANs zur Verfügung stehen.

Der typische Arbeitsablauf, beim Kreieren eines VLANs, sieht wie folgt aus:

(1) Der Administrator wählt den Menüpunkt/Button *Create VLAN*

(2) Das System vergibt einen laufenden VLAN-Identifikator, es erscheint ein leeres VLAN-spezifisches-Fenster. Folgende Eingaben (*Group Parameter*) sind möglich:

   - ein beliebig gewählter, innerhalb des Systems eindeutiger VLAN-Name
   - ein Feld zur textuellen Beschreibung (Kommentar, Zweck des VLANs usw.)
   - Kontaktperson(en) (Name, Telefonnummer, Raum, EMail, ...)
   - Soll das VLAN durch ein bestimmtes Schicht 3 Protokoll oder Subnetzadresse, definiert sein, so muß das dem System explizit mitgeteilt werden. (Layer-3 VLAN[10]). In einem extra Fenster werden das Protokoll und evtl. die Subnetzadresse noch näher spezifiziert.
   - in zukünftigen Versionen wären im VLAN-spezifischen-Fenster noch Konfigurationsmöglichkeiten für *Filters, Quality of Traffic, Security Level, Accounting Policies, usw.* denkbar.

(3) Weiter mit Abbrechen *(Chancel)* oder Eingaben bestätigen *(Apply)* oder Eingaben bestätigen und Fenster schließen *(OK)* oder weiter mit **Hinzufügen von Endstationen** (Abschnitt 6.3.4).

6.3.2 Aufgabe: Modifizieren eines VLANs

Hierunter fallen Tätigkeiten wie die Änderung von Group Parametern und das Ändern der Mitgliedschaft einer Endstation bzw. Veränderungen in der Zusammensetzung der VLANs. In späteren Versionen sind weitere Modifikationen denkbar.

[9] Zur Begriffbildung von Formularen siehe auch Abschnitt 3.3.2.

[10] Standardeinstellung: Layer-1- oder Layer-2 VLAN; ergibt sich automatisch nach Identifizierung der im VLAN involvierten Switchingtechnologie und der Endstationen.

Die Modifikationen werden in den entsprechenden VLAN-spezifischen-Fenstern vorgenommen. Das System prüft die Eingaben und Änderungen des Administrators auf ihre Plausibilität und gibt entweder positive oder negative Rückkopplung und evtl. Hilfestellungen.

Der Ablauf, um die Mitgliedschaft einer Endstation zu ändern, könnte so ablaufen: Endstation aus der

a) physischen oder logischen Gesamtsicht oder einer konkreten

b) physischen oder logischen VLAN-spezifischen-Sicht

in eine andere VLAN-spezifische-Sicht ziehen. Das System prüft im Fall

a) ob die Endstation bereits Mitglied in einem anderen VLAN ist. Wenn nein, dann liegt ein **Hinzufügen von Endstationen** vor. Wenn ja, muß geprüft werden, ob eine Teilnahme der Endstation an mehreren VLANs möglich ist *(Single vs. Multiple Membership).* Die entsprechende Rückmeldung gibt Aufschluß darüber;

im Fall

b) handelt es sich — datenbanktechnisch gesehen — um einen reinen Kopiervorgang mit anschließendem Löschen. Auch hier gibt das System Rückmeldung.

Ähnlich liegt der Fall beim Verschieben von Segmenten, Ports, Unter-VLANs[11] oder sogar ganzen VLANs (Ikone aus logischer VLAN-Gesamtsicht in das VLAN-spezifische-Fenster schieben; Wirkung: Verschmelzung zweier VLANs zu einem einzigen VLAN).

6.3.3 Aufgabe: Löschen eines VLANs

Der Administrator zieht die entsprechende VLAN-Ikone per Drag & Drop über die Mülltonnen-Ikone. Es erfolgt eine Rückfrage, die entweder positiv oder negativ bestätigt werden muß. Um eine Löschung rückgängig zu machen, steht eine Undo-Funktion zur Verfügung oder durch Anklicken der Mülltonne erscheint deren Inhalt in einem Fenster; von dort aus kann die VLAN-Ikone wieder in eine der beiden Sichten[12] verschoben werden.

[11]Ein VLAN enthält u.a. alle Mitglieder eines anderen VLANs. So lassen sich ganze Hierarchien bilden.

[12]Logische oder physische VLAN-spezifische-Sicht.

6.3.4 Aufgabe: Hinzufügen von Endstationen

Der Administrator kann durch Verschieben der Endstationen, in der Regel von der physischen Gesamtsicht in die logische oder physische VLAN-spezifische-Sicht, die Mitglieder eines VLANs bestimmen. Weiterhin sollen Bildschirmmasken bereitstehen, die ein manuelles Eingeben der Mitglieder ermöglicht. Denkbar wäre auch Mechanismen, die Namen oder Adressen (logische, symbolische oder physische) der Endstationen eines einzurichtenden VLANs aus einer Datei lesen, bereitzustellen.

Das System unterstützt Layer-1, Layer-2 und Layer-3 VLANs. Es stellt selbständig fest, welche Technologie bei den involvierten Switches vorliegt. Per Default werden Layer-2 VLANs gebildet. Zur Bildung von Layer-3 VLANs müssen weitere Parameter (Protokolltyp oder Protokoll und Subnetzadresse) angegeben werden.

Hat der Administrator ein neues VLAN einzurichten, so hat er meist schon eine Vorstellung davon, aus welchen Endstationen es bestehen bzw. welches Kriterium für die Zugehörigkeit einer Endstation zu einem VLAN gelten soll. Mindestens folgende Kriterien sollen vom System unterstützt werden:

(K1) alle Endstationen, die an einem oder mehreren Segmenten bzw. Ports hängen (Portswitching).

(K2) beliebige Endsysteme (durch Angabe von MAC-, Netzadressen, logischen oder vollständigen Namen oder durch Anklicken in der physischen Gesamtsicht).
Alle Endstationen an Segmenten, die mit Hilfe der ATM LAN Emulation eingebunden werden und ein ELAN bilden, können nur einem VLAN angehören, d.h. innerhalb eines ELANs ist eine Strukturierung in mehreren VLANs nicht möglich.

(K3) Endstationen, die ein gemeinsames Protokoll oder Protokoll und Subnetzadresse benützen.

Um im Text auf die einzelnen Kriterien besser verweisen zu können, wurden sie durchnummeriert. Diese Kriterien der Zugehörigkeit von Stationen zu einem VLAN werden *Membership Rules* genannt. Aufgabe einer graphische Bedienoberfläche ist es dem Administrator eine intuitive Umsetzung dieser Membership Rules zu ermöglichen.

Der Arbeitsablauf bein Hinzufügen von Endstationen zu einen VLAN könnte wie folgt aussehen:

(1) Der Administrator wählt ein VLAN aus, das erweitert werden soll. Falls es sich um ein Neues handelt, muß es vorher mit dem Befehl *Create VLAN* kreiert werden (siehe auch Abschnitt 6.3.1: Kreieren eines VLANs).

Das erscheinende Fenster zeigt die *Group Parameter*, die aktuellen Mitglieder des VLANs in einer logischen VLAN-spezifischen-Sicht (Endstationen, evtl. explizit benannte Server (Name + Dienst)[13] und weitere (Unter-)VLANs) und alle involvierten VLAN-relevanten Komponenten (Endstationen, Segmente, Switches, Router, Backbones, usw.) in einer physischen VLAN-spezifischen-Sicht.

(2) Die Kriterien (K1) und (K2) für die Zugehörigkeit können durch Anklicken der Endstationen, Segmente oder Ports in der physischen Gesamtsicht und Verschieben in die logische oder physische VLAN-spezifische-Sicht realisiert werden. Wird ein Segment oder Port angeklickt und verschoben, so sind automatisch alle Endstationen, die an dem Segment hängen, Mitglieder des VLANs. Das gilt auch für alle Endstationen eines Segments, das *direkt* mit Hilfe der LAN Emulation eingebunden werden. Um mehrere Stationen gleichzeitig zu selektieren, müssen geeignete Mechanismen bereitstehen (z.B. Kombination von Drücken der SHIFT-Taste und Anklicken des Objekts mit der Maus, oder einen Rahmen mit der mittleren Maustaste um die betreffenden Stationen aufziehen, usw.)[14]. Eine Menge von Segmenten kann auch von Hand eingegeben werden. Ein Segment ist durch die Adresse des Switches und der Portnummer, an dem es angeschlossen ist, eindeutig bestimmt.
Die explizite Eingabe von Schicht-3 Adressen, MAC-Adressen oder logischen Namen kann auch unterstützt werden.

(3) Soll ein Layer-3 VLAN gebildet werden (Kriterium K3), so ist der Administrator angehalten ein Schicht-3 Protokoll anzugeben. Zwei Fälle sind zu unterscheiden (vorausgesetzt das Protokoll wird vom System unterstützt):

a) es handelt sich um ein routbares Protokoll, dann genügt die zusätzliche Angabe einer Subnetzadresse. Das System bildet die Eingabe entsprechend auf die beteiligten Komponenten (Switches, ATM/LAN-Konverter) ab.

b) das Protokoll ist nicht routbar, d.h. die Kriterien für die Zusammensetzung der VLANs *(Membership Rules)* sind jetzt, neben dem verwendeten Protokolltyp, die gleichen, wie sie bei Layer-2 VLANs verwendet werden. Die Umsetzung der Kriterien erfolgt analog, wie bereits besprochen. Wird nur ein nichtroutbares Protokoll angegeben, so sind alle Endstationen innerhalb der Administrationsdomäne, die dieses Protokoll verwenden, Mitglieder des VLANs. Das System muß den Administrator darauf hinweisen.

[13] Heute noch nicht möglich.

[14] Diese Mechanismen sollten laut Entwicklungskonzept erst im nächsten Kapitel 7 spezifiziert werden. Die Gründe warum sie bereits hier erfolgen, sind in der ersten Bemerkung am Anfang dieses Kapitels dargelegt.

6.3.5 Aufgabe: Herausnehmen von Mitgliedern eines VLANs

Endstationen, Segmente bzw. Ports werden in einer der beiden VLAN-spezifischen-Sichten selektiert und über die Mülltonnen-Ikone gezogen. Dabei repräsentiert ein Segment bzw. Port wieder eine Menge von Endsystemen, die an diesem Segment hängen. Wird eine Switch-Ikone aus der physischen VLAN-spezifischen-Sicht über die Mülltonnen-Ikone gelegt, so werden alle Endstationen, die mit einem Port des Switches assoziiert sind und zu dem betrachteten VLAN gehören, aus dem VLAN entfernt. Wie beim Löschen von VLANs, steht auch hier beim Herausnehmen/Löschen von Mitgliedern eine Undo-Funktion zur Verfügung, um versehentliche Löschungen rückgängig zu machen.

Abbildung 6.3 zeigt die Abläufe für die Aufgaben Kreieren von VLANs, Hinzufügen und Herausnehemen von Mitgliedern eines VLANs anhand eines Flußdiagramms:

6.3.6 Einschub: Hilfssichten, bei heterogener Netzumgebung

Dieser Abschnitt führt Hilfssichten ein, die den Administrator, bei seiner Arbeit mit einer heterogenen Netzumgebung, unterstützen.

Begriffsbildung

Wir schon bereits mehrfach angeklungen, unterscheiden sich die Switches verschiedener Hersteller bzgl. der Interswitchkommunikation erheblich. Da so ziemlich jeder Hersteller sein eigenes proprietäres Protokoll verwendet, ist eine Kommunikation zwischen den Switches verschiedener Hersteller, ohne die Verwendung gemeinsamer Standards, nicht möglich.
Fakt ist, daß virtuelle LANs, basierend auf Switching-Komponenten, nur reine homogene Lösungen sind, d.h. Lösungen die aus VLAN-fähigen Komponenten eines Herstellers, unter Verwendung einer Switching-Technologie[15] und meist nur für ein physisches Medium[16] konzipiert sind.

Ein Hersteller-VLAN sind physische zueinander kompatible VLAN-fähige Switching-Komponenten, die meist mit einem eigenen Elementmanager[17] ausgeliefert werden.

Das Gesamtnetz besteht unter Umständen aus mehreren Hersteller-VLANs, die — im schlechtesten Fall — nicht zueinander kompatibel sind und daher VLANs nicht über diese Hersteller-VLANs hinaus eingerichtet werden können. Um dennoch solche übergreifende VLANs einzurichten, sollte das VLAN-Managementsystem mit Hilfe

[15]Portswitching, Layer-2-, Layer-3-Switches.
[16]Z.B. Ethernet oder Token Ring. In dieser Arbeit wird, wegen den in Abschnitt 6.1.1 erwähnten Gründen, stellvertretend Ethernet herausgegriffen.
[17]In dieser Arbeit mit Hersteller-VLAN-Manager bezeichnet.

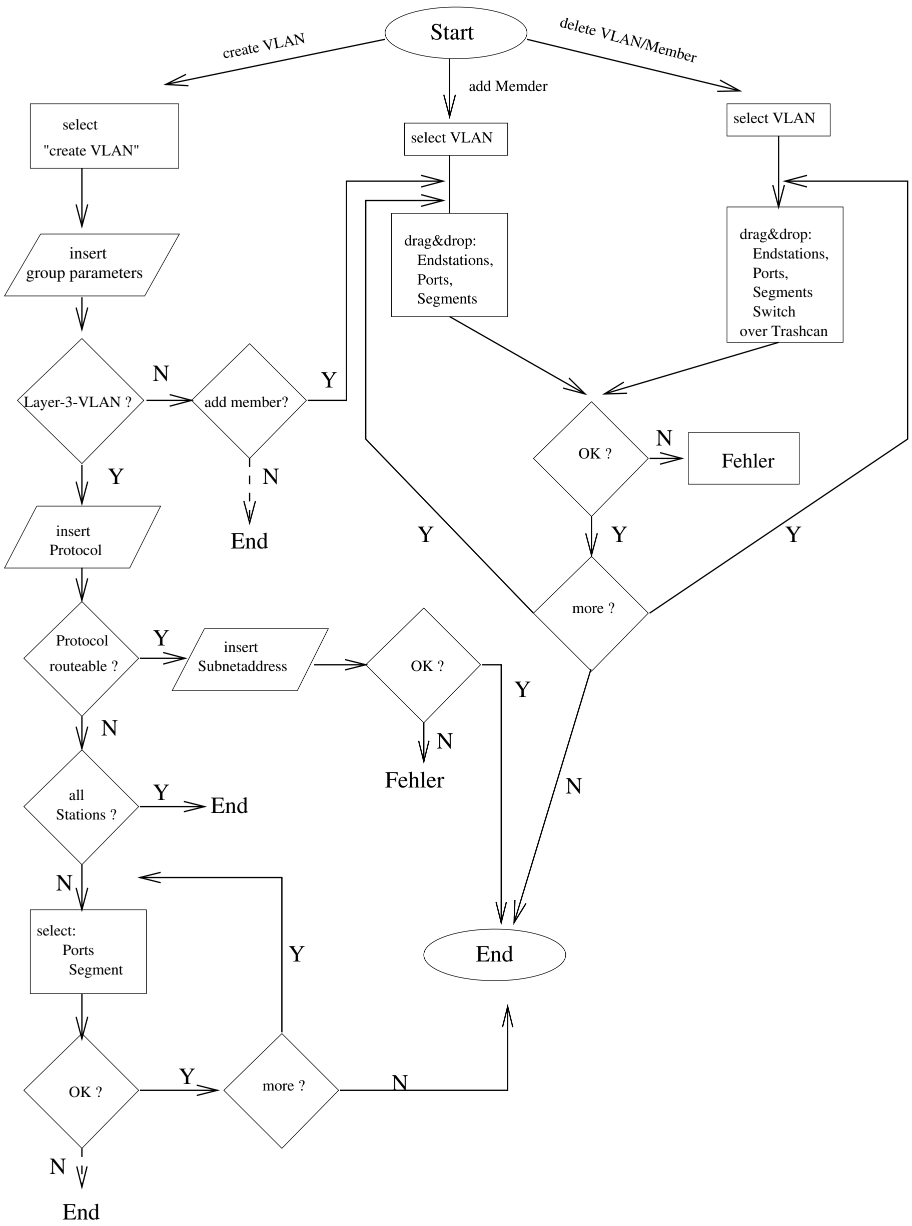


Abbildung 6.3: Flußdiagramm

von Abbildungsfunktionen, wenn keine Standards für die Interswitchkommunikation vorhanden sind, das VLAN auf die involvierten Bereiche der Hersteller-VLANs abbilden können.

Ein **Sub-VLAN** ist in dieser Arbeit der Teil eines Hersteller-VLANs, das in einem konkreten VLAN enthalten ist. In einer (zukünftigen) heterogenen Umgebung besteht nun ein VLAN evtl. aus mehreren Sub-VLANs, die, wie bereits erwähnt, mit Hilfe von gemeinsamen Standards oder Abbildungsfunktionen des VLAN-Mangementsystems miteinander in Beziehung gesetzt werden müssen. Abbildung 6.4 macht die Unterscheidung zwischen Hersteller- und Sub-VLANs nocheinmal deutlich.

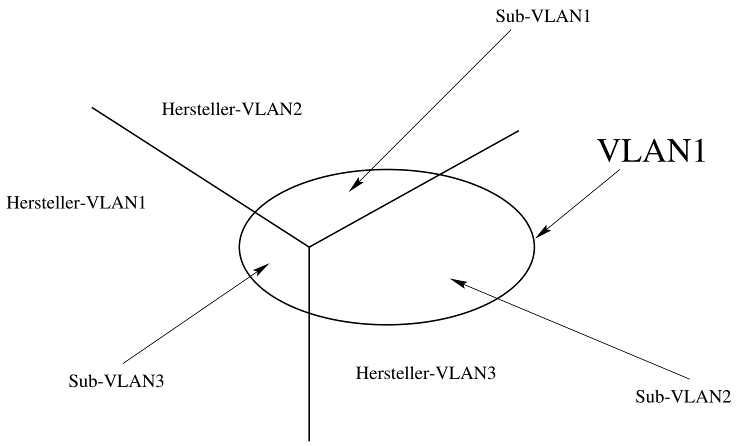


Abbildung 6.4: VLANs, Hersteller- und Sub-VLANs

Ein Hersteller-VLAN sind physische zueinander kompatible VLAN-fähige Switching-Komponenten, die meist mit einem eigenen Elementmanager[18] ausgeliefert werden.

Analyse der Hilfssichten

Der häufigste auftretende Fehler bei der switchübergreifenden Konfiguration eines VLANs dürfte die nicht mögliche Teilnahme einer/mehrerer Endstation(en) an einem VLAN sein. Ein Grund dafür könnten zueinander inkompatible Switching-Komponenten sein. Als Hilfestellung für den Administrator kann die physische Gesamtsicht dahingehend modifiziert werden, daß z.B. durch Anklicken einer Endstation, eines Segments bzw. Ports in einem bestimmten Modus alle Endstationen die theoretisch noch dem aktuellen VLAN zugeordnet werden können, in der physischen Gesamtsicht hervorgehoben werden.

Die Idee hierbei ist, daß die, relativ zur angeklickten Komponente, entsprechenden

[18]In dieser Arbeit mit Hersteller-VLAN-Manager bezeichnet.

Endstationen, die auf der VLAN-Managementsystem-Ebene ein "homogenes" VLAN bilden könnten, als Ganzes hervorgehoben werden. Dieser Bereich soll im weiteren VLAN-Management-Area und die daraus resultierende Sicht **VLAN-Domänen-Sicht** genannt werden. Im Idealfall ist das ganze Netz eine einzige VLAN-Management-Area, d.h. beliebige Kombinationen von Endstationen können zu einem VLAN zusammengefaßt werden[19].

Zunächst einmal bestehen Netze in dieser Arbeit aus evtl. mehreren verschiedenen Hersteller-VLANs. Diese sind wiederum durch VLAN-fähigen-Netzkomponenten geprägt, die sich durch das Tripel (Hersteller, Technologie, Medium) charakterisieren lassen. Der Begriff Hersteller-VLAN umfaßt auch die daran angeschlossenen Endgeräten. Es handelt sich dabei um ein homogenes Subnetz, meist mit eigenen proprietären Managementwerkzeugen. Diese Hersteller-VLAN-Manager können häufig mehrere Hersteller-VLANs managen (gleicher Hersteller, aber auf anderen Technologien der Switches und/oder Übertragungsmedien basierend). Diese, durch einen proprietären Hersteller-VLAN-Manager gemanagten Hersteller-VLANs, sollen im weiteren Verlauf der Arbeit als Hersteller-VLAN-Manager-Domänen bezeichnet werden.

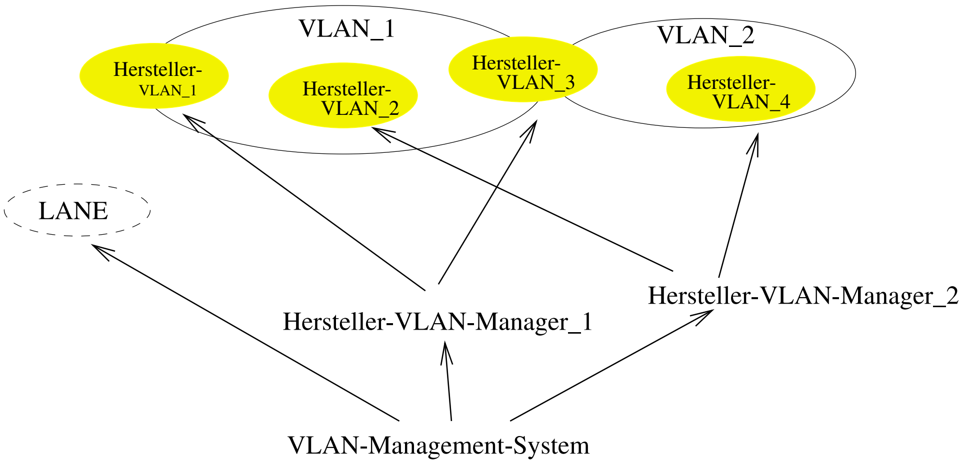


Abbildung 6.5: Aufrufstruktur involvierter Managementwerkzeuge

VLANs, die Mitglieder aus mehreren solcher — meist nicht zueinander kompatiblen — Hersteller-VLAN-Manager-Domänen enthalten, können nur unter bestimmten Kriterien eingerichtet werden. Das wichtigste Kriterium ist die Fähigkeit des VLAN-Managementsystems mit Hilfe von geeigneten Abbildungsfunktionen und Schnittstellen die Vorgaben des Administrators auf die, für die Hersteller-VLANs zuständigen, Hersteller-VLAN-Manager zu delegieren, und so diese Hersteller-VLAN-Manager für den Administrator transparent erscheinen zu lassen. Das VLAN-Management-

[19] Siehe dazu auch die Bemerkung am Ende des Abschnitts.

system setzt auf die Hersteller-VLAN-Manager (und evtl. einem LAN Emulation Manager) auf. Diese notwendige “Symbiose” wird in Abbildung 6.5 schematisch angedeutet. In Abschnitt 6.2.1 wurde bereits diese Problematik herausgearbeitet. Diese Sicht, aus der die maximal größtmöglichen zusammenhängenden VLANs im Administrationsbereich ersichtlich sind, wurde bereits weiter oben als VLAN-Management-Area-Sicht eingeführt. Es handelt sich dabei um eine oder mehrere Hersteller-VLAN-Manager-Domänen, die vom VLAN-Managementsystem, also eine Abstraktionsstufe höher, mit Hilfe der oben genannten Abbildungsfunktionen und Schnittstellen, verschmolzen werden können. Diese neue Sicht wurde bereits weiter oben als VLAN-Management-Area-Sicht eingeführt. Konfiguriert der Administrator die VLANs innerhalb dieser Hersteller-VLAN-Manager-Domäne oder VLAN-Management-Area, so wird es zu keiner Fehlermeldung kommen, die Hilfssichten werden nicht benötigt.

Letztendlich wird vom System noch eine (VLAN→Sub-VLAN)-Sicht angeboten, aus der, wie der Name schon sagt, die im aktuellen VLAN involvierten Sub-VLANs gezeigt werden und erst in einem zweiten Schritt, analog zur logischen VLAN-Sicht, die Mitglieder bzw. Endstationen.

Bemerkung:
Diese, neben der ursprünglichen logischen VLAN-Sicht, zusätzlich eingeführten logischen und physischen Sichten ergeben sich allein aus der Heterogenität und den Inkompatibilitäten heutiger Produkte und Realisierungen. Der Mangel an Standards und die unterschiedlichen Vorstellungen der Hersteller von dem Begriff VLAN[20] führt zu der Notwendigkeit dieser “Klimmzüge”. Aufgabe der Bedienoberfläche — auf einer höheren Abstraktionsebene — sollte es sein, den Administrator davon zu entlasten.

Mit dieser Exkursion, in die zur Einrichtung eines VLANs notwendigen “Hilfssichten”, endet die Anforderungsanalyse der vom System angebotenen Hilfssichten. Diese Hilfssichten werden im Abschnitt 6.3.9 noch einmal mit den anderen Sichten stichpunktartig zusammengefaßt. In Kapitel 7 werden dann Vorschläge für deren Visualisierung gemacht.

Wie bereits erwähnt liegt der Schwerpunkt der nachfolgenden Aufgabenbereiche in der Visualisierung der vom System angebotenen Informationen. Die üblichen Interaktions- und Kontrollmechanismen treten in den Hintergrund und werden nur besprochen, wenn sie VLAN-spezifischen Charakter haben.

[20] Die Definition eines VLANs seitens der Hersteller richtet sich stark nach den technischen Möglichkeiten ihrer Produkte.

6.3.7 Aufgabe: ATM LAN Emulation managen

Das System versucht den Administrator weitgehend vom Management der LAN Emulation zu entlasten. Bei der Einrichtung eines VLANs werden die nötigen Clients und Server der LAN Emulation automatisch instantiiert, wenn sich das VLAN über ein ATM-Backbone erstreckt. Das emulierte LAN (ELAN) erhält den gleichen Namen wie das VLAN; wenn kein Name definiert ist, den entsprechenden VLAN Identifikator. In der Regel wird der Administrator nur im Fehlerfall mit der LAN Emulation konfrontiert. Um einen Fehlerfall abzuarbeiten, ergibt sich automatisch die Notwendigkeit einer geeigneten Darstellung der Struktur und Komponenten der LAN Emulation und natürlich des Fehlerfalls. Die Tätigkeiten beschränken sich hier im wesentlichen auf die Anzeige und Prüfung der Struktur und Komponenten der LAN Emulation und in der Feststellung und Lokalisierung von Fehlern. Für tiefergehende Managementtätigkeiten stehen die vorausgesetzten LANE- und ATM-Device/Network-Managementapplikationen zur Verfügung, auf dessen Dienste das System aufsetzt (siehe dazu auch Abschnitt 6.2.1).

Wird ein VLAN in der *physischen Gesamtsicht* hervorgehoben, so sind auch die involvierten ATM/LAN-Konverter *(Edge Devices)* markiert, daß ATM-Netz wird in dieser Darstellung z.B. durch eine Wolke, symbolisiert. Komponenten, Leitungen und logische Verbindungen *(Virtual Channels)* innerhalb des ATM-Netzes werden wegen der besseren Übersicht vorerst nicht gezeigt. In der Regel sind in diesen Konvertern die Dienste der LAN Emulation *(MAC-Services)* implementiert (zentral oder verteilt), so daß für das LAN Emulation Management nur diese betrachtet werden müssen. Der Administrator gelangt, von den physischen Sichten aus, durch Anklicken der ATM-LANE-Wolke in die **LAN Emulation Sicht**. Dieses Fenster dient als Ausgangspunkt für weitere LAN Emulation Managementtätigkeiten, wobei in dieser Arbeit nur Vorschläge für einen kleinen aber grundlegenden Ausschnitt aus diesem Bereich gemacht werden. Um geeignet auf einen Fehler reagieren zu können, müssen in weiteren Fenstern, zusätzlich zu den bereits oben erwähnten ATM/LAN-Konvertern, noch alle Clients und Server (LECs, LES, BUS, LECS) und die Verbindungen (optional) zwischen ihnen, sowohl physisch als auch virtuell (*Virtual Channels (VCs))*, dargestellt werden, aus der neben der Struktur der LAN Emulation auch der Status und Standort einzelner LANE-Komponenten ersichtlich ist. Diese Anforderungen an die Bedienoberfläche werden, neben der Möglichkeit auf Fehler reagieren zu können, in Abschnitt 7.6 visualisiert.

6.3.8 Aufgabe: Verkehr analysieren und auswerten

Einer der Gründe, warum man VLANs einrichtet, ist die Kanalisation der Verkehrsströme, um so Verkehrsengpässe zu vermeiden. Ein gewisser Prozentsatz des Verkehrs sollte innerhalb eines herkömmlichen LANs bzw. VLANs lokal bleiben.

Hier hat sich als Faustregel die sogenannte 80/20 Regel bewährt, die als Kenngröße aussagt, daß für einen normalen Betrieb mindestens 80 Prozent des Verkehrs lokal gehalten und nur etwa 20 Prozent über (V)LAN-Grenzen hinweg abgewickelt werden sollten. Wird diese Regel nicht eingehalten, so ist es erforderlich das LAN (klassisch oder virtuell) umzustrukturieren. Auf der anderen Seite besteht durch die relativ hohen Freiheitsgrade bei der Konfiguration von VLANs auch die Gefahr, daß sich die Verkehrssituation verschlechtern kann (siehe dazu auch Kapitel 2). Daher ist es notwendig Mechanismen und Sichten anzubieten, die es erleichtern die Verkehrsströme zu beobachten und zu historisieren, zu analysieren und die Ergebnisse in eine Umstrukturierung der (V)LANs einfließen zu lassen.

Die Anforderungen an der darzustellenden Information am Bildschirm, die sich aus dem Aufgabenbereich "Verkehr analysieren und auswerten" ergeben, sind durch folgende Fragenstellungen indirekt gegeben. Die Sichten, die daraus resultieren, sind in Klammern angegeben.

- Wo treten Verkehrsengpässe *(Bottlenecks)* auf?
  (Gesamtdarstellung der Auslastung)
- Verkehr auf einer Backboneleitung, einem Segment (Auslastung)?
  (Backbone-, Segment-Sicht)
- Verteilung der Protokolle auf einem Segment?
  (Segment-Sicht)
- Anzahl der konfigurierten VLANs und deren beanspruchte Bandbreite bzgl. einer Backboneleitung?
  (Backbone-Sicht)
- Wie ist das Netz ausgelastet?
  (Gesamtdarstellung der Auslastung)
- Verkehr innerhalb eines VLANs?
  (Inter/Intra VLAN Verkehr)
- Verkehr zwischen VLANs?
  (Inter/Intra VLAN Verkehr)

6.3.9 Zusammenfassung der Sichten

Dieser Abschnitt faßt die notwendigen Sichten zusammen, die sich aus der Analyse der Aufgaben ergaben

Bemerkung: Im Verlauf dieser Arbeit wurde bereits ausgiebig von Sichten gesprochen. Mit den eingeführten Sichten ist in einigen Fällen bereits ein bestimmtes

Aussehen assoziiert worden, d.h. welche Attribute und Funktionalitäten am Bildschirm wie auszusehen haben (z.B. Mülltonnen-Ikone als Delete-Funktion, ATM-Netz und LANE dargestellt als Wolke). Dieses "Sichtbar machen" ist genaugenommen die Aufgabe in der Designphase. In der Analysephase sind Sichten — datenbanktechnisch gesehen — nichts weiter als Objekte mit einer Menge von Attributen und Methoden.

In dieser Arbeit werden in den nachfolgenden Tätigkeiten des Entwicklungskonzepts Vorschläge für die Visualisierung für folgende Sichten gemacht:

1) **logische VLAN Gesamtsicht**
   Zeigt alle im System vorhandenen VLANs.

2) **physische Gesamtsicht**
   Zeigt das VLAN-fähige physische Netz. Dabei kann es sich um einen oder mehrere Teilausschnitte der Gesamtbetrachtung eines(beliebigen) physischen Netzes handeln.

3) **VLAN-spezifische-Sicht** (für jedes VLAN)

   3a) logische VLAN-spezifische-Sicht (VLAN→Mitglieder)

   3b) physische VLAN-spezifische-Sicht (zusammenhängender Ausschnitt der Komponenten aus physischer Gesamtsicht).

4) aus 3a folgt als Umkehrung eine **Mitglieder→VLAN-Sicht**, d.h. in welchen VLAN(s) ist eine Endstation Mitglied (da heute nur *Single Membership* möglich und sinnvoll ist, besteht diese Sicht aus einer eindeutigen Zuordnung — im Gegensatz zu *Multiple Membership* — von Endstationen zu höchstens einem VLAN).

5) **physische Switch/Port-Sicht** der Switching-Komponenten (Portstatus, Portbelegung, Auslastung, Nachbarsysteme, ...)

6) **logische Switch/Port-Sicht** der Switching-Komponente (wie 5)

7) **Geasamtdarstellung der Auslastung**

8) **Segmentsicht** (Auslastung, Verteilung der Protokolle, ...)

9) **Backbonesicht** (Auslastung, Anzahl der konfigurierten VLANs über Backbone, ...)

10) **Inter/Intra VLAN Verkehr**

11) **ATM LAN Emulation** (Komponenten, Zustand, Fehlerdarstellung,....)

12) Hilfssichten, bei heterogener Netzumgebung:
Zusammenfassend benötigt der Administrator Hilfssichten, die sich aus den folgenden Fragestellungen ergeben:

12 a) aus welchen Sub-VLANs besteht ein VLAN?
→ logische und physische (VLAN→Sub-VLAN)-Sicht.

12 b) aus welchen Hersteller-VLANs besteht das Gesamtnetz?
→ logische und physische (Gesamtnetz→Hersteller-VLAN)-Sicht; Verfeinerung von (c).

12 c) welche Hersteller-VLANs werden vom gleichen Hersteller-VLAN-Manager gemanagt?
→logische und physische Hersteller-VLAN-Manager-Domäne-Sicht; Verfeinerung von (d).

12 d) welche Hersteller-VLAN-Manager-Domänen lassen sich vom VLAN-Managementsystem zu einem zusammenhängenden VLAN (Layer-1, Layer-2, oder Layer-3) verschmelzen bzw. welche größtmöglichen zusammenhängenden Bereiche des Gesamtnetzes könnten ein VLAN bilden.
→VLAN-Management-Area-Sicht.

Bevor auf diese Sichten näher eingegangen werden kann, müssen zuerst die grundlegenden Objekte identifiziert[21], in Beziehung zueinander gesetzt und geeignete Repräsentationen (Sichten, Visualisierungen) gefunden werden. Die ersten beiden Aspekte liefert die objektorientierte Analyse (OOA), die Visualisierung ist Gegenstand der Designphase. Im konzeptionellen Benutzerschnittstellenmodell werden die oben geforderten Sichten mit den zentralen Gegenständen aus der OOA in Beziehung gesetzt.

Damit wären die Anforderungen an die graphische Bedienoberfläche des System, die sich aus den Aufgaben ergaben, analysiert und festgeschrieben.

6.4 Objektorientierte Analyse

In diesen Abschnitt wird parallel zur Anwender- und Aufgabenanalyse die Problemwelt modelliert. Sind in der Anwender- und Aufgabenanalyse die Tätigkeiten bzw. Funktionalitäten wichtig, so setzt die OOA bei den Gegenständen der Problemwelt an.

Wie bereits in Abschnitt 4.3 erwähnt, ist das Ergebnis der OOA das *aufgabenorientierte* Objektmodell. In dieser Arbeit wird die Notation der OOA-Methode von Coad/Yourdon angewandt, wobei aus Platzmangel die Felder für Attribute und

[21] Indirekt in der Anwender- und Aufgabenanalyse bereits geschehen.

Methoden freigelassen werden.
Das aufgabenorientierte Objektmodell wird dargestellt als ein Graph, dessen Knoten Objekte (Klassen) sind. Die benannten Kanten repräsentieren Beziehungen zwischen Objekten. Den Graph kann man sich als ein erweitertes Entity-Relationshipmodell vorstellen. Die Entitäten werden dabei um Methoden/Funktionalitäten erweitert. Für weitere Einzelheiten sei auf Abschnitt 4.3 verwiesen. Abbildung 6.6 zeigt das aufgabenorientierte Objektmodell.

Das zentrale Objekt ist das VLAN, das durch eine Menge von Endstationen, die eine Broadcast-Domäne bilden, definiert ist. Die Endstationen sind an Ethernetsegmenten angeschlossen, diese sind wiederum über Ports mit den Switches verbunden.
Andererseits kann ein VLAN durch die enthaltenen Sub-VLANs definiert werden. Diese Sub-VLANs sind durch die vorhandenen Switching-Komponenten geprägt. Ein Switch ist evtl. über Ports an einem oder mehreren Backbones angeschlossen. ATM/LAN-Konverter sind in dieser Arbeit, ohne Beschränkung der Allgemeinheit, ATM-fähige LAN-Switches (Vererbung), in denen die Komponenten der LAN Emulation, entweder zentral oder verteilt, implementiert[22] sind. Das *native* ATM-Netz ist für diese Arbeit von zweitrangiger Bedeutung. Wichtiger sind die ATM/LAN-Konverter und die dadurch realisierten Dienste der LAN Emulation.

Da dieses aufgabenorientierte Objektmodell für die Datenmodellierung und somit mehr für die Implementierung des Gesamtsystems von Interesse ist, wird es in dieser Arbeit nicht weiter verfeinert. Dies geschieht bereits in einer eigenen Diplomarbeit [Vuk96].
Wichtig für diese Arbeit ist die Identifizierung der grundsätzlichen Objekte der Problemwelt und deren Beziehung zueinander. Das Ergebnis der OOA fließt, zusammen mit den Anforderungen aus der Anwender- und Aufgabenanalyse, in das konzeptionelle Benutzerschnittstellenmodell mit ein.

6.5 Konzeptionelles Benutzermodell

In der Anwender- und Aufgabenanalyse ließen sich bereits die grundlegenden Funktionalitäten und notwendigen Sichten ableiten, die das System bzw. die graphische Bedienoberfläche anbieten soll.
Die objektorientierte Analyse ermittelte die zentralen Gegenstände der Problemwelt und deren Beziehungen. In den konzeptionellen Entwurf der Benutzerschnittstelle werden die Ergebnisse der beiden Analysetätigkeiten zusammengeführt. Das Ergebnis ist das *benutzerschnittstellenorientierte* Objektmodell. Dieses wird um weitere Objekte erweitert und eine neue Art der Beziehung eingeführt. Diese Beziehungen

[22] Der Standard des ATM Forums läßt diesen Implentierungsaspekt bewußt offen.

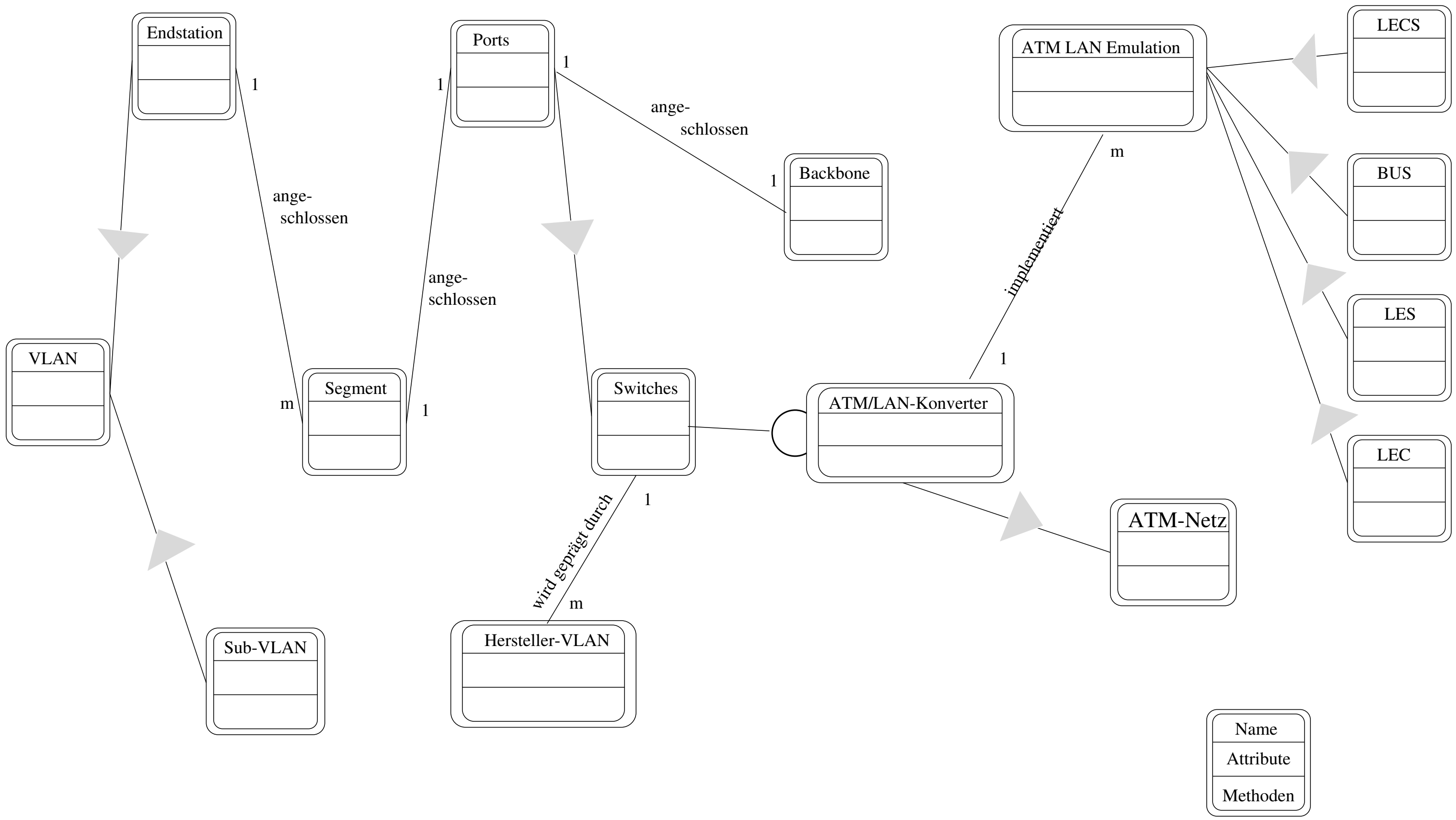


Abbildung 6.6: Aufgabenorientiertes Objektmodell

sind gerichtet und modellieren Zugriffe des Anwenders. Im konzeptionellen Benutzerschnittstellenmodell werden neben den bisher reinen Datenobjekten zusätzlich noch Mengenobjekte und Werkzeugobjekte eingeführt. Mengenobjekte repräsentieren die Gesamtheit der aktuell vorhandenen Objekte einer Klasse und stellen Selektionsmechanismen bereit (z.B. VLAN-Sicht). Werkzeugobjekte repräsentieren Funktionen, die auf andere Objekte angewendet werden können (z.B. Drucker oder Mülltonne, sind Metaphern für Funktionen, die keiner Erklärung bedürfen). Mengenobjekte stellen, wie der Name schon sagt, wieder Objekte dar und werden meist in eigenen Fenstern oder in Teilbereichen davon visualisiert. Die am Ende der Anwender- und Aufgabenanalyse zusammengefaßten Sichten, sind zum Großteil solche Mengenobjekte[23]. Der konzeptionelle Entwurf der Benutzerschnittstelle erfolgt so, daß, neben den Zugriffsbeziehungen, bereits die Hierarchie der zu visualisierenden Fenster ersichtlich wird. Abbildung 6.7 zeigt das benutzerschnittstellenorientierte Objektmodell. In dieser Abbildung wurde aus Platzmangel auf die übliche Repräsentation eines Objekts im Sinne von Coad/Yourdon verzichtet und statt dessen die Felder für die Attribute und Methoden weggelassen. Außerdem wurden, um die Übersichtlichkeit zu gewährleisten, nicht alle Beziehungen (Pfeile) zwischen den Objekten eingezeichnet.

Die logische VLAN-Sicht und die physische Gesamtsicht bilden den Einstieg in das System. Es liegt also nahe beide Sichten in einem Fenster zusammenzufassen. Die logische VLAN-Sicht bietet einen Zugriff auf die Objekte VLAN und den darin enthaltenen Mitgliedern. Die physische Gesamtsicht dient zur Visualisierung derselben in der physischen Gesamtdarstellung und als Ausgangspunkt für die Visualisierungen der physischen Hilfssichten (VLAN-Management-Areas, Hersteller-VLAN-Manager-Domänen, Hersteller-VLANs, Sub-VLANs) und Managementtätigkeiten (LAN Emulation, logische und physische Switch/Port-Sichten, Inter/Intra-VLAN Verkehr, diverse Elementmanager[24], usw.). Für jedes VLAN ist, von der logischen VLAN-Sicht ausgehend, die VLAN-spezifische-Sicht erreichbar. In dieser Sicht werden wieder in einer logischen VLAN-spezifischen-Sicht die Endstationen, die Mitglieder im VLAN sind, und in einer physischen VLAN-spezifischen-Sicht der relevante Teilausschnitt des Netzes dargestellt. Gründe für die Aufnahme des physischen Teilausschnitts in die VLAN-spezifische-Sicht können erst, nachdem vorher die Inhalte einiger Grundsichten (log. VLAN-Sicht, phys.Gesamtsicht und wesentliche Teile der VLAN-spezifischen-Sicht) entworfen und spezifiziert wurden, in Kapitel 7 (Entwurf- und Spezifikation byw. Design) bei der Besprechung der VLAN-spezifischen-Sicht angeführt werden.

Sowohl die physische Gesamtsicht als auch die physischen VLAN-spezifischen-Sichten dienen als Ausgangspunkt für weitere Managementtätigkeiten, für die wiederum eigene Fenster konzipiert werden. Durch Anklicken einer Switching-Komponente wird

[23]Die logische VLAN-Sicht besteht aus den aktuell vorhandenen VLANs, die physische Gesamtsicht aber ist genaugenommen eine Komposition von Objekten und Mengenobjekten.

[24]Nicht relevant für diese Arbeit.

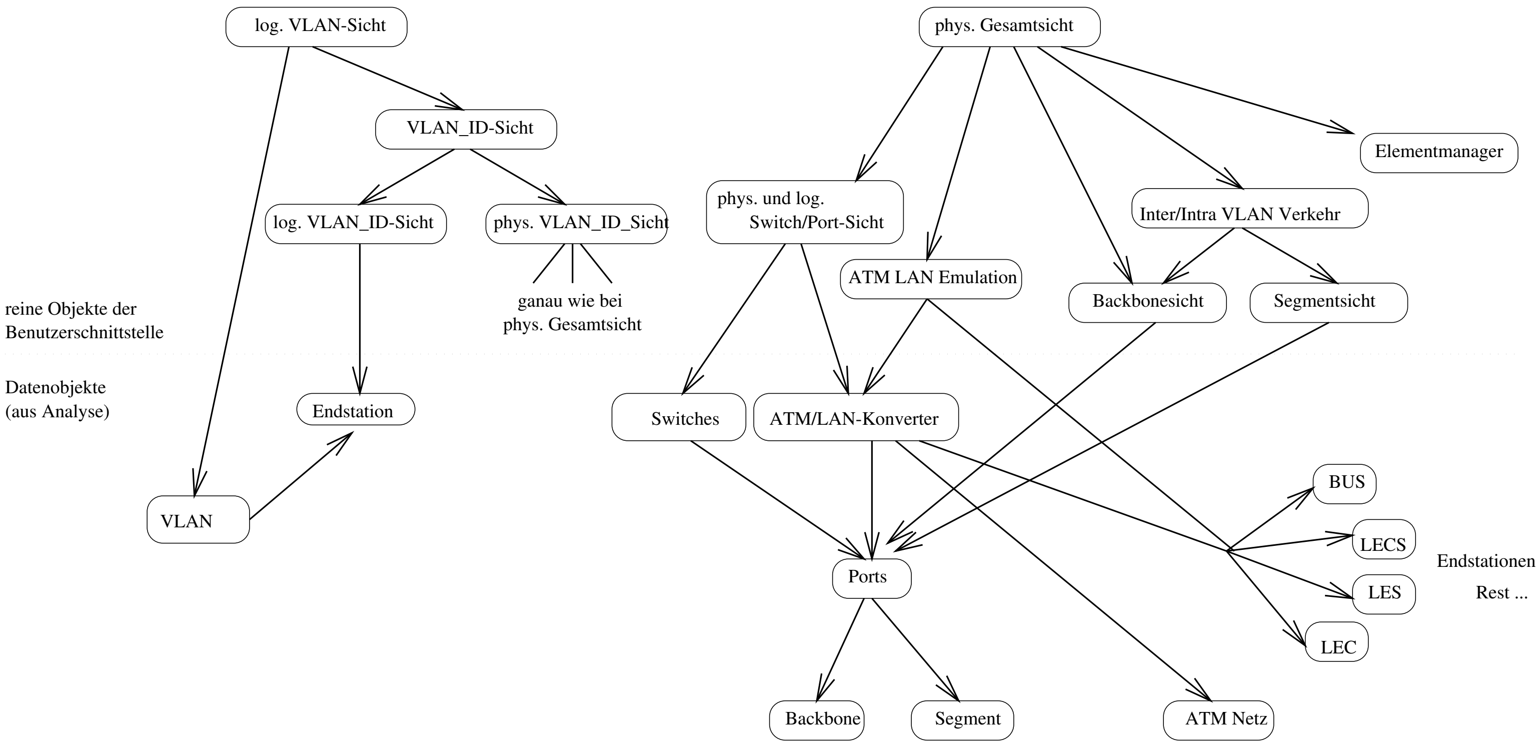


Abbildung 6.7: Benutzerschnittstellenorientiertes Objektmodell

ein Fenster geöffnet, die, je nach Einstellung, eine physische oder logische Sicht auf die Komponente zeigt. Die ATM-LANE-Ikone symbolisiert die LAN Emulation in Komplettsicht. Die korrespondierende Detailsicht dient als Ausgangspunkt für das Management der LAN Emulation. In einem weiteren Schritt können von dort aus ATM-Komponenten gemanagt werden[25]. In dieser Arbeit ist aber nur das Management der LAN Emulation von Interesse.

Verschiedene Analysetätigkeiten, wie etwa die in dieser Arbeit zu besprechenden Verkehrsanalysen (Gesamtauslastung, Inter/Intra VLAN Verkehr, Verkehr auf einem Backbone/Segment) werden auch von den physischen Sichten ausgehend oder durch Anklicken entsprechender Buttons initiiert.

Zusammnenfassend ist jetzt anhand des konzeptionellen Benutzerschnittstellenmodells für das VLAN-Managementsystem folgende grobe Fensterhierarchie vorgegeben. Das Hauptfenster besteht aus einer logischen und einer physischen Sicht, genauso jedes VLAN-spezifische Fenster. Von den physischen Sichten ausgehend wird in die restlichen Fenster verzweigt.

Ziel des konzeptionellen Benutzerschnittstellenmodells ist u.a. diese Aufrufbeziehungen zu modellieren. Für die weitere Arbeit reicht obige Modellierung vollkommen aus, auf eine weitere Verfeinerung des Modells wird verzichtet, um nun das Augenmerk weg von den datenbanktechnischen Aspekten[26] auf die eigentlichen, für diese Arbeit interessanteren, Visualisierungen bzw. Repräsentierungen der Objekte und Sichten und einigen Interaktions- und Kontrollmechanismen zu richten.

[25] Durch Anklicken einer ATM-Komponente in einer der physischen Sichten, kann der zugehörige Elementmanager direkt aufgerufen werden.

[26] Wichtig für die Entwicklung des Gesamtsystems.

Kapitel 7

Entwurf und Spezifikation

In diesen Kapitel werden im Rahmen des Entwicklungskonzepts die Tätigkeiten aus der Entwurf- und Spezifikationsphase (Designphase) durchgeführt, die darin münden den bisher abstrakten Objekten und Sichten aus der Analysephase Gestalt und Verhalten *(Look & Feel)* zu verleihen. Gleichzeitig werden auch die wichtigsten Interaktions- und Kontrollmechanismen besprochen.

Sowohl für die Komplettsichten als auch für die Detailsichten eines jeden Objekts und Kompositionen von Objekten, müssen geeignete graphische Repräsentationen entworfen und spezifiziert werden, d.h. welche Attribute, Funktionalitäten usw., im folgenden auch als die darzustellenden Informationen bezeichnet, wann und wo sichtbar sind. Dabei ist ein Kompromiß zwischen Informationen, die einerseits absolut notwendig und andererseits — in einem bestimmten Kontext — überflüssig sind, zu finden, so daß der begrenzte Bildschirm (Fenster) optimal im Zusammenspiel mit anderen Objekten, die wiederum aus gleichen oder verschiedenen Objekten zusammengesetzt sein können, ausgenutzt wird. Beispiele für zusammengesetzte Objekte sind Fenster, die, wie bereits mehrfach erwähnt, im Rahmen der (objektorientierten) Benutzerschnittstellenentwicklung, auch als Objekte aufgefaßt werden.

Die Bedienoberfläche soll anfangs ein möglichst einfaches und intuitives Arbeiten erlauben. Nur die notwendigsten Funktionalitäten und Sichten für ein VLAN-Management sind vorhanden. Treten Probleme auf oder soll eine Feinabstimmung erfolgen, muß man in der Regel einen Menüpunkt auswählen, einen Button drücken und/oder in ein anderes Fenster verzweigen. Abbidung 7.1 zeigt schematisch die Ausgangsfenster der VLAN-Managementanwendung, wie sie auch schon im konzeptionellen Benutzerschnittstellenmodell angedeutet wurde. Jedes dieser Fenster dient als Ausgangspunkt weiterer, tiefergehender Managementtätigkeiten.

Die Aufrufstruktur der Fenster(-klassen) ist hierarchisch[1] konzipiert. An der Wur-

[1]Was nicht ausschließt, daß bei der Instantiierung der Fenster (Objekte zur Laufzeit) Zyklen auftreten können.

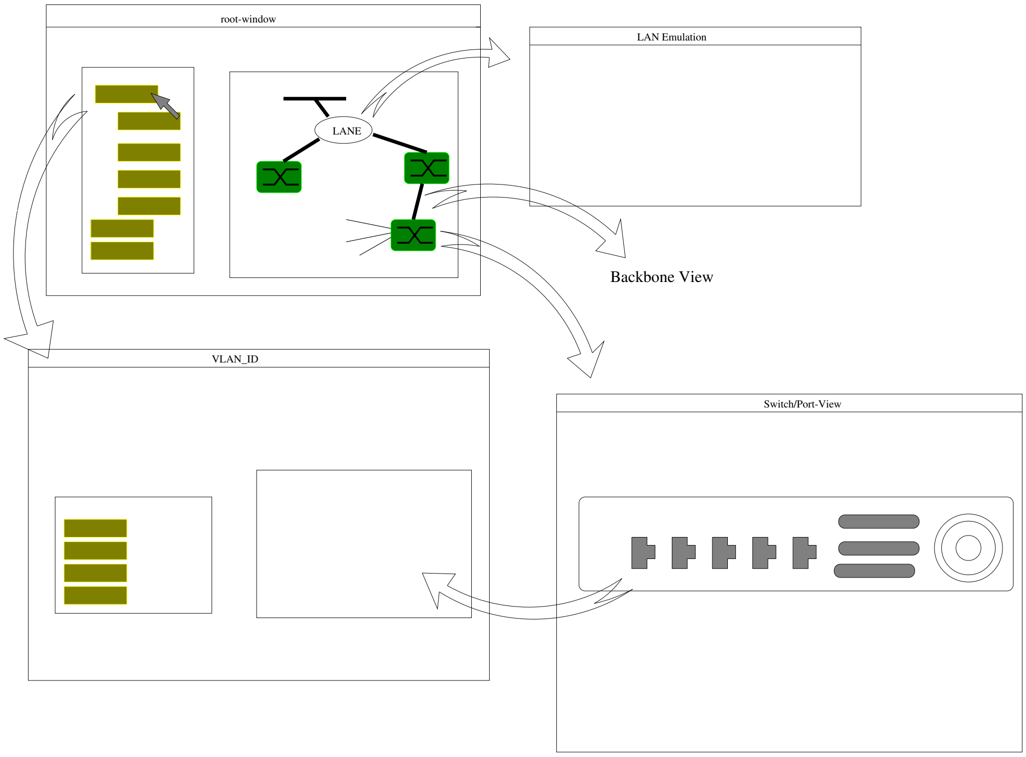


Abbildung 7.1: Fensterhierarchie

zel sind möglichst abstrakte, die einfachsten und notwendigsten Informationen für das VLAN-Management angeordnet. Die mehr in die Tiefe gehenden, komplexeren Informationen oder nicht VLAN-spezifischen Funktionen/Sichten werden tiefer in Richtung Blattebene plaziert.
Die Anordnung in den Fenstern und die hierarchische Aufrufstruktur der Fenster wird durch die Gewichtung der Funktionalitäten und Sichten nach der Häufigkeit der Verwendung vorgegeben. In gewisser Weise entspricht diese Überlegung, was häufig benötigt wird, wird leicht zugänglich angeordnet, was nicht so oft gebraucht wird, kann tiefer im System realisiert werden, einem *Top-Down*-Vorgehen.

Im Verlauf der Analysephase wurden die einzelnen Arbeitsschritte zerlegt und so der Funktionsumfang in groben Zügen, die Objekte und Sichten respektive die Repräsentation der Objekte identifiziert. In einem *Bottom-Up*-Vorgehen muß alles geeignet zusammengefügt werden, sowohl räumlich (2D-Bildschirm) als auch zeitlich (Aufrufreihenfolge/zeitliche ABfolge der Fenster), bis beide Vorgehensweisen sich treffen und zum gleichen Ergebnis kommen.

7.1 Logische und physische VLAN-(Gesamt-)Sicht

Als wichtigstes Objekt des Systems, bekommt das VLAN neben der Komplettsicht (Ikone, Icon) ein eigenes Fenster (Detailsicht). Eine Ikone aus Abbildung 7.3 a) zeigt ein Objekt VLAN[2] in Komplettsicht.

Neben den Attributen VLAN-Identifikator und VLAN-Name, gibt die Hintergrundfarbe Auskunft über den Zustand.
Eine farbliche Repräsentation der einzelnen Zustände könnte folgendermaßen aussehne:

- Grün: Status OK,
- Gelb oder Orange: Status anormal, Fehler aber nicht schwerwiegend (z.B einzelnes Segment nicht erreichbar, Endstation *down*, usw.)
- Rot: schwerwiegender Fehler: (z.B. Switchport, Switch *down*, usw.)

Diese farbliche Kennzeichnung findet auch bei der Beschreibung der Zustände der Komponenten der LAN Emulation Anwendung. Ein weiteres sichtbare Attribut in dieser Darstellung könnte die Art des VLANs (Layer-1, -2, -3 VLAN) sein. Der Versuch mehr Informationen unterzubringen, würde die Ikone überladen. Der Name des VLANs und/oder eine fortlaufende Nummer sollte aussagekräftig genug sein.
Die bereits in Kapitel 6 eingeführte VLAN-spezifische-Sicht entspricht der Detailsicht des Objekts VLAN, desse Besprechung in Abschnitt 7.2 erfolgt.

[2]Als Instanz der Klasse VLAN.

In der Regel existieren mehrere VLANs zur gleichen Zeit[3] und daher ist es notwendig geeignete Werkzeuge zu deren Verwaltung bereitzustellen. Die in dieser Arbeit erarbeiteten Sichten können als Vorschläge für die Bedienoberfläche solcher Werkzeuge aufgefaßt werden.

Im konzeptionellen Benutzermodell wurde bereits das Mengenobjekt Logische und physische VLAN-Sicht eingeführt, das die Gesamtheit der aktuell vorhandenen Objekte der Klasse VLAN repräsentiert und Selektionsmechanismen bereitstellt. In manchen akademischen Abhandlungen, in Zeitschriften, in Büchern und bereits in (angekündigten) konkreten Realisierungen [Red96] findet man Darstellungen der logischen VLAN-Sicht ähnlich derer in Abbildung 7.2.

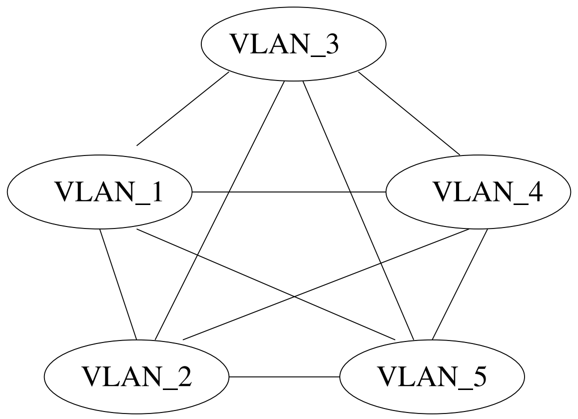


Abbildung 7.2: Alternative logische VLAN-Sicht

Andere Visualisierungen unterscheiden sich nur in der Anordnung (kreis-, sternförmig, usw.). Diese Darstellungen eignet sich nach Meinung des Autors nur für eine begrenzte Anzahl von VLANs (etwa <=20-30). Durch diese graphische Darstellungen wird relativ will Platz in Anspruch genommen. Ab einer gewissen Größe, sobald der zur Verfügung stehende beschränkte Platz am Bildschirm nicht mehr ausreicht, wird sie unübersichtlich und unhandlich, auch dadurch daß man gezwungen ist in **zwei** Dimensionen zu *scrollen*.

Selbstgestellte Kriterien für die zu entwickelnde logische VLAN-Sicht waren die übersichtliche Darstellung beliebig vieler VLANs und ein schnelles Wieder-/Auffinden, die in der obigen Darstellung, ab einer gewissen Größe, nicht mehr gegeben ist. Zusammen mit der inhärenten Hierarchie — VLAN→ evtl. UnterVLAN → End-

[3]Swich-Hersteller sprechen von Tausenden (theoretisch möglichen); die LAN Emulation in der Version 1.0 erlaubt 1024 ELANs. Es drängt sich unweigerlich die Frage auf, wer so viele VLANs/ELANs braucht, geschweige den, bei der ungeheueren Dynamik von VLANs, den nötigen Überblick bewahren kann.

stationen[4] — resultierte daraus der in Abbildung 7.3 a) visualisierte Vorschlag für das Mengenobjekt "Logische VLAN-Sicht", eine Menge von VLAN-Ikonen, geordnet nach alphabetischen oder anderen Kriterien, von denen jede ein VLAN repräsentiert.

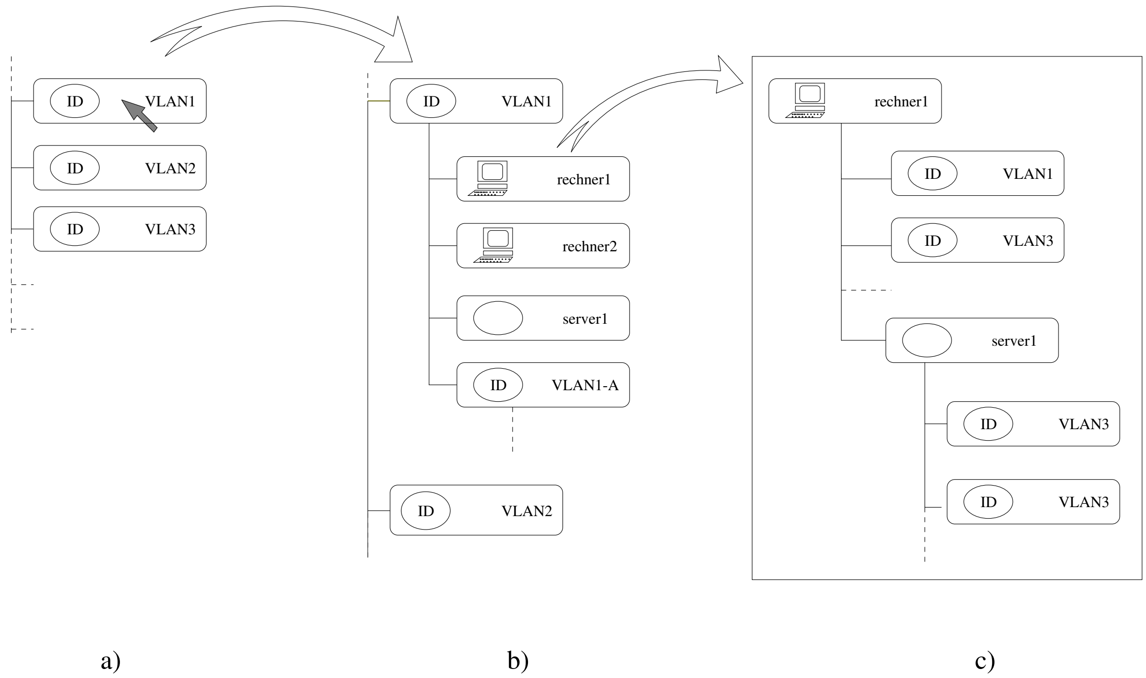


Abbildung 7.3: Logische VLAN-Sicht und (Member→VLAN)-Sicht

Neben der Darstellung müssen Selektionsmechanismen bereitgestellt werden. Hier bietet sich eine Technik an, wie sie zur Verwaltung in Datei-/File-Systemen verwendet wird. In gewisser Weise entsprechen in dieser Metapher die VLANs den Verzeichnissen und die Mitglieder des VLANs den darin enthaltenen Dateien; VLANs sind Mengen von Endsystemen[5], die entweder disjunkt zueinander sind oder sich überschneiden können *(Single vs. Multiple membership)*. Mit Hilfe eines (File-)Browsers, als Werkzeug zur Verwaltung der aktuell vorhandenen VLANs, wird dem Administrator ein intuitives Arbeiten ermöglicht. Befindet sich der Mauszeiger über einer VLAN-Ikone, so wird der entsprechende Ausschnitt in der physischen Gesamtsicht und die VLAN-Ikone, z.B. durch farbliche Hinterlegung und/oder durch Blinken, hervorgehoben. Durch Drücken der Shift-Taste und Anklicken einzelner VLAN-

[4] Auch für *Multiple Menmership* anwendbar. Da die Teilnahme einer Endstation an mehreren VLANs *(Multiple Membership)* noch nicht befriedigend gelöst ist, wird dieser Aspekt in dieser Arbeit ausgespart. Bietet sich aber eine diesbezügliche Erweiterung der Vorschläge geradezu an, so werden sie dennoch kurz angesprochen.

[5] In einem zweiten Schritt können die involvierten Leitungen und Vermittlungskomponenten ermittelt werden.

Ikonen mit der linken Maustaste ist eine gleichzeitige Darstellung mehrerer VLANs möglich. Die farbliche Hinterlegung wird automatisch durch eine Legende in einem extra, verschiebbaren Fenster erläutert. Zu diskutieren wäre, ob die farbliche Kennzeichnug eines VLANs, z.B. als weiterer *Group Parameter* bei der Definition eines VLANs, dem Anwender überlassen werden sollte. Um mit der farblichen Kodierung der Zustandssemantik nicht in Konflikt zu geraten, dürfen die dafür definierten Farben für die Hinterlegung von VLANs nicht verwendet werden. Oder, um diese Restriktion zu umgehen, kann die Anzeige der Fehler von der physischen Gesamtsicht in die logische VLAN-Sicht verlagert werden. Durch Blinken einer Ikone wird ein Fehler signalisiert und in einem zweiten Schritt kann dann die betroffene Komponente in der physischen Gesamtsicht gefunden werden.

Bei einem einfachen Mausklick auf eine VLAN-Ikone werden die Mitglieder aufgelistet (Abbildung 7.3 b). Die Darstellung kann wie folgt interpretiert werden: Das VLAN mit dem Identifikator x bzw. dem Namen *abc* hat als Mitglieder die Endstationen *foo23*, die Endstation *foo56*, usw. VLAN x enthält — als Unter-VLAN — das VLAN y.
Zu diskutieren wäre, ob ein VLAN aus weiteren (Unter-)VLANs bestehen kann und ob Endstationen, die als Server fungieren explizit als solche gekennzeichnet werden sollen[6]. Ein Server ist durch die Adresse der Endstation und den Dienst (Portnummer oder Protokoll[7] charakterisiert. Unterstützt der Server mehrere Dienste, so ist er mehrmals aufzuführen.

Befindet sich der Mauszeiger über einer Mitglieder-Ikone oder über einer Endstation in der physischen Gesamt–Sicht, kann sich der Administrator optional (nur *Multiple Membership*) in einer (Member→VLAN)-Sicht angezeigen lassen, in welchen VLANs sie Mitglied ist (Abbildung 7.3 c)). Die Abbildung kannn so gelesen werden: Die Endstation *foo1* ist Mitglied in den VLANs *abc*, *ijk*, usw., sie ist *Mail*-Server für die VLANs *aab*, usw.

7.2 Logische und physische VLAN-spezifische-Sicht

Wie bereits erwähnt entspricht die VLAN-spezifische-Sicht der Detailsicht des Objekts VLAN.

Ein Doppelklick auf eine VLAN-Ikone in der logischen VLAN-Sicht öffnet das zugehörige VLAN-spezifische-Fenster, d.h. — um mit den in Kapitel 6 eingeführten *Termini Technici* zu sprechen — er bewirkt den Übergang von der Komplettsicht in die Detailsicht.
Abbildung 7.4 zeigt schematisch ein Objekt VLAN in Detailsicht.

[6] Heute noch nicht realisierbar.
[7] Ftp, SMTP, usw.

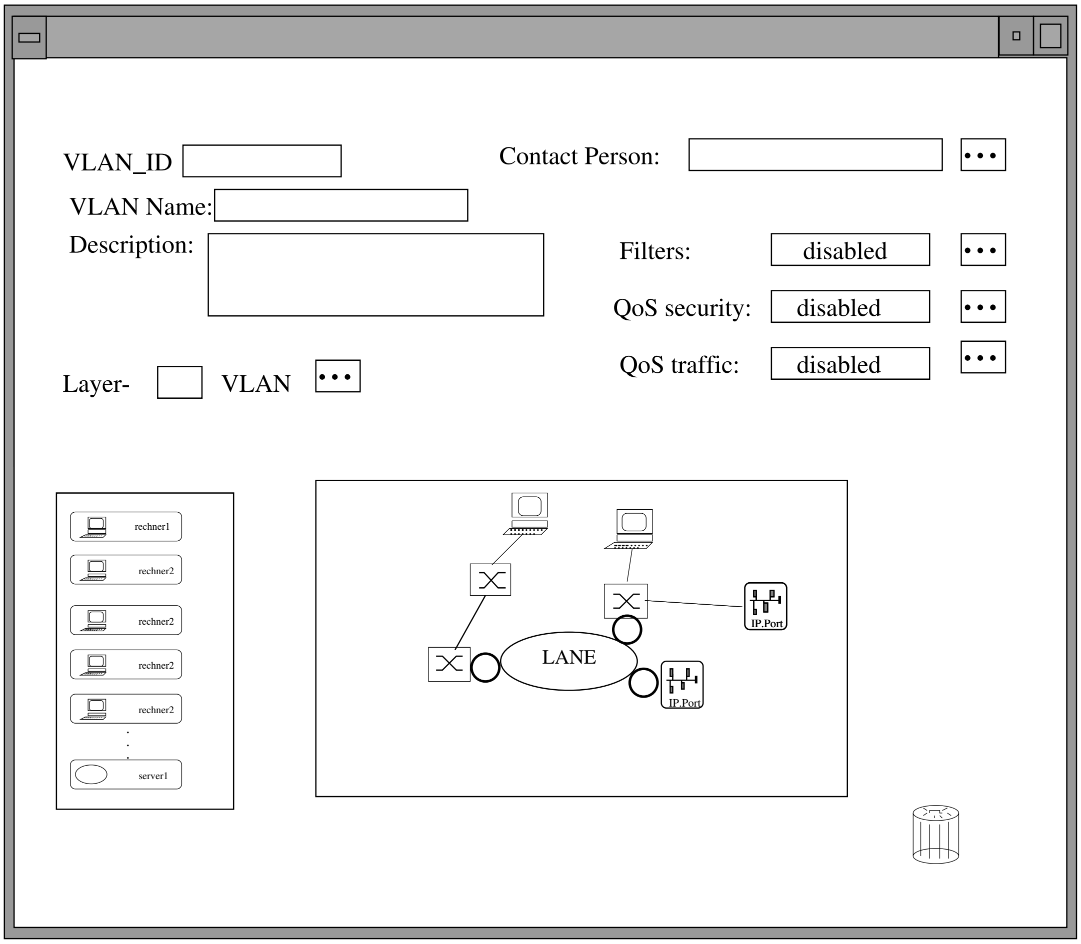


Abbildung 7.4: VLAN-spezifische-Sicht

Das Fenster gliedert sich in drei Hauptfelder. Im oberen Feld ein Formular mit den *Group Parametern*, links darunter die **logische VLAN-spezifische-Sicht** mit den einzeln aufgeführten Mitgliedern, rechts davon der entsprechende physische Ausschnitt aus der Gesamtdarstellung des Netzes, die **physische VLAN-spezifische-Sicht**. Dieser Teilausschnitt ist bei einem geöffneten VLAN-spezifischen-Fenster auch in der physischen Gesamtsicht hervorgehoben. Die physische VLAN-spezifische-Sicht wurde in die VLAN-spezifische-Sicht mit aufgenommen, um bei mehreren gleichzeitig geöffneten VLAN-spezifischen-Sichten/Fenstern (etwa >3) zumindest in den VLAN-spezifischen-Sichten noch den geeigneten Überblick, der in der physischen Gesamtsicht nicht mehr gegeben ist, zu gewährleisten. Eine (verschiebbare) Legende verdeutlicht die Zuordnung VLAN↔Farbe. Die Farben sind durch Schieberegler individuell einstellbar. Bei mehreren gleichzeitig in der physischen Gesamtsicht hervorgehobenen VLANs kann es schnell unübersichtlich werden ("Fleckerlteppich"), außerdem werden dann die Farben immer schwerer unterscheidbar. Eine erweiterte Hinterlegung/Abbildung VLAN↔(Farbe+Ziffer oder Name) reicht zur eindeutigen Identifizierung, ist aber weiterhin sehr unübersichtlich.

Abhilfe könnte hier vielleicht folgende Idee schaffen: Befindet sich der Mauszeiger in einem VLAN-spezifischen-Fenster oder in der logischen VLAN-Sicht über einer VLAN-Ikone oder in der physischen Gesamt-Sicht über einer Endstation, dann wird zusätzlich zur farblichen Hinterlegung das entsprechende VLAN in der physischen Gesamtsicht und in der Legende durch blinken hervorgehoben.
Zu diskutieren wäre, ob dem Administrator nicht die Möglichkeit gegeben werden soll, die physischen Darstellungen des Netzes, zumindest nach dem Einrichten der VLANs, ein- und auszublenden, um so, die doch beträchtlichen Verzögerungszeiten durch den Aufbau der graphischen Darstellung zu umgehen und dadurch ein flüssigeres Arbeiten zu gewährleisten.

Das System vergibt bei der Einrichtung eines neuen VLANs einen eindeutigen VLAN Identifikator (VLAN_ID). Dem Administrator obliegt es nun zusätzlich einen logischen, fast beliebigen Namen zu definieren. Desweiteren ist ein (Text-)Feld zur informellen Beschreibung des VLANs vorgesehen. Die Daten der für das VLAN zuständigen Kontaktperson(en) können in einem weiteren Feld eingetragen werden. Soll ein Layer-3 VLAN gebildet werden, muß das dem System explizit mitgeteilt werden. Das darauf erscheinende Fenster zeigt die vom System unterstützten Schicht-3 Protokolle an. Wie bereits in Abschnitt 6.3.1 und 6.3.4 besprochen wählt der Administrator ein Protokoll aus und ist bei einem routbaren aufgefordert eine Subnetzadresse anzugeben. Bei einem nichtroutbaren Protokoll wird wie bei einen Layer-2 VLAN weiter verfahren.

Bevor nun die Mitglieder des VLANs bestimmt werden, können in zukünftigen Versionen, weitere *Group Parameter* definiert werden. Abhängig von der zugrundeliegenden Software und Hardware, wäre eine explizite Angabe der geforderten *Quality of Services (QoSs)* bzgl. geforderter Mindestbandbreite *(Traffic)*, Sicherheitsanforderungen *(Security)* inklusive Filter[8] oder andere Parameter denkbar.

Als nächstes können die Mitglieder des VLANs, wie bereits in der Anwender- und Aufgabenanalyse besprochen, bestimmt werden. Dort wurden auch die dazu notwendigen Interaktions- und Kontrollmechanismen bei der Selektion von Komponenten besprochen. Das System prüft, angestoßen durch Aktionen des Administrators, dessen Eingaben auf ihre Plausibilität und gibt als Ergebnis eine geeignete positive oder negative Rückkopplung *(Feedback)*, in Form von Eingabebestätigungen oder Fehlermeldungen bei gleichzeitiger Hilfestellung.
Solche Hilfen vom System können die in der Anwender- und Aufgabenanalyse gefolgerten Hilfssichten darstellen. Die Visualisierung der Hilfssichten erfolgt, nach der Besprechung der physischen Gesamtsicht, in Abschnitt 7.3.

Physische Gesamtsicht
In Abschnitt 6.1.1 wurde bereits die für diese Arbeit zugrundeliegende physische Netzumgebung festgelegt.
Die Repräsentation des Objekts physische Netzumgebung durch die **physische Gesamt-Sicht** am Bildschirm stellt sich im wesentlichen so wie in Abbildung 6.1 dar. Zusätzlich müssen Router, wenn die relevanten Teile von deren Funktionalität (noch) nicht in den Switching-Komponenten implementiert sind *(Virtual Router)*, mit berücksichtigt werden. Eine möglichst realitätsnahe topologische Darstellung ist genauso gefordert, wie bei der Visualisierung klassischer Netze. Die Anforderungen sind demnach nicht nur VLAN-spezifisch und werden deshalb im folgenden nur kurz aufgezeigt.
Es genügt nicht das Netz durch einen beliebigen Graphen zu symbolisieren, es ist auch eine intuitive Zuordnung der Komponente zu deren geographische Standort wünschenswert. Dem Administrator muß eine Möglichkeit bereitstellt werden, den Hintergrund mit Lageplänen (2D, 3D) etc. frei zu gestalten und die Darstellung evtl. manuell, durch Verschieben einzelner Komponenten, anzupassen, sofern das nicht maschinell geschieht. Das man davon noch weit entfernt ist, wird ersichtlich, wenn man den dazu nötigen Aufwand betrachtet. Als Stichworte seien Discovery-Funktionen und Netzdokumentationen genannt. In [Poi95] und natürlich [HA93] wird näher auf die Problematik eingegangen.

Die physische Gesamtsicht und die physische VLAN-spezifische-Sicht dienen als Ausgangspunkt für Visualisierungen, z.B. aller physischen Hilfssichten, die im Anschluß besprochen werden, der Gesamtauslastung (Abschnitt 7.8), und für weitere Managementtätigkeiten. Durch Anklicken einer Switching-Komponente gelangt man je nach Einstellung in die logische (Abschnitt 7.5) oder phys Switch/Port-Sicht (Abschnitt 7.5), durch Anklicken eines Backbones bzw. Segments in die Backbone-Sicht (Abschnitt 7.10) bzw. Segment-Sicht (Abschnitt 7.9) oder durch Anklicken der ATM-Wolke in ein Fenster für das ATM LAN Emulation Management (Abschnitt 7.6).

[8]Etwa in der Art (Protokoll, Endstation oder VLAN, Richtung)

7.3 Visualisierung der Hilfssichten, bei heterogener Netzumgebung

In der Analysephase wurde bei einer heterogener Netzumgebung auf die Notwendigkeit geeigneter Hilfssichten geschlossen, die im Fehlerfall den Administrator bei der Zusammenstellung der VLANs unterstützen. Sie ermöglichen ihm u.a. festzustellen, welche Endstationen an dem aktuellen, gerade in Arbeit befindlichem VLAN teilnehmen können. Zur Erinnerung sei im folgenden nocheinmal eine kurze Zusammenfassung der geforderten Hilfssichten aufgeführt:

a) log. und phys. (VLAN→Sub-VLAN)-Sicht, d.h. aus welchen Sub-VLANs besteht ein VLAN?

b) log. und phys. (Gesamtnetz→Hersteller-VLAN)-Sicht; d.h. aus welchen Hersteller-VLANs besteht das Gesamtnetz (Verfeinerung von c)?

c) log. und phys. Hersteller-VLAN-Manager-Domänen-Sicht; d.h. welche Hersteller-VLANs werden vom gleichen Hersteller-VLAN-Manager gemanagt[9](Verfeinerung von d)?

d) log. und phys. VLAN-Management-Area-Sicht; d.h. welche Hersteller-VLAN-Manager-Domänen lassen sich vom VLAN-Managementsystem zu einem zusammenhängenden VLAN (Layer-1, Layer-2, oder Layer-3) verschmelzen bzw. welche größtmöglichen zusammenhängenden Bereiche des Netzes können ein VLAN bilden? Im Idealfall ist das ganze Netz eine einzige VLAN-Management-Area, die evtl. vorhandene Heterogenität des Netzes hat bei der Konfiguration der VLANs keine Auswirkungen.

7.3.1 Logische und physische (VLAN→Sub-VLAN)-Sicht

Zur Visualisierung der logischen (VLAN→Sub-VLAN)-Sicht wird die logische VLAN-Sicht erweitert; Abbildung 7.5 zeigt das Ergebnis.

Durch explizites Drücken eines *Radio*buttons[10] im Hauptfenster erfolgt der Übergang der Darstellung von der logischen VLAN-Sicht (Standardeinstellung) in die logischen (VLAN→Sub-VLAN)-Sicht ((VLAN→Sub-VLAN)-Modus). Die Sub-VLAN-Ikonen sind zwischen den VLAN- und den Endstationen-Ikonen eingeschoben. Zur Verwaltung dient wieder ein Browser. Durch Klicken auf eine VLAN-Ikone werden die involvierten Sub-VLANs gezeigt, durch Klicken auf ein Sub-VLAN-Ikone werden

[9] In der Regel kann ein Hersteller-VLAN-Manager die ganze Produktpalette eines Herstellers managen. Durch die Einführung dieser Sicht wird Vorsorge getroffen, wenn dem nicht so ist.

[10] *Radio Button*: Exclusives oder; im Gegensatz zu *Command Button*

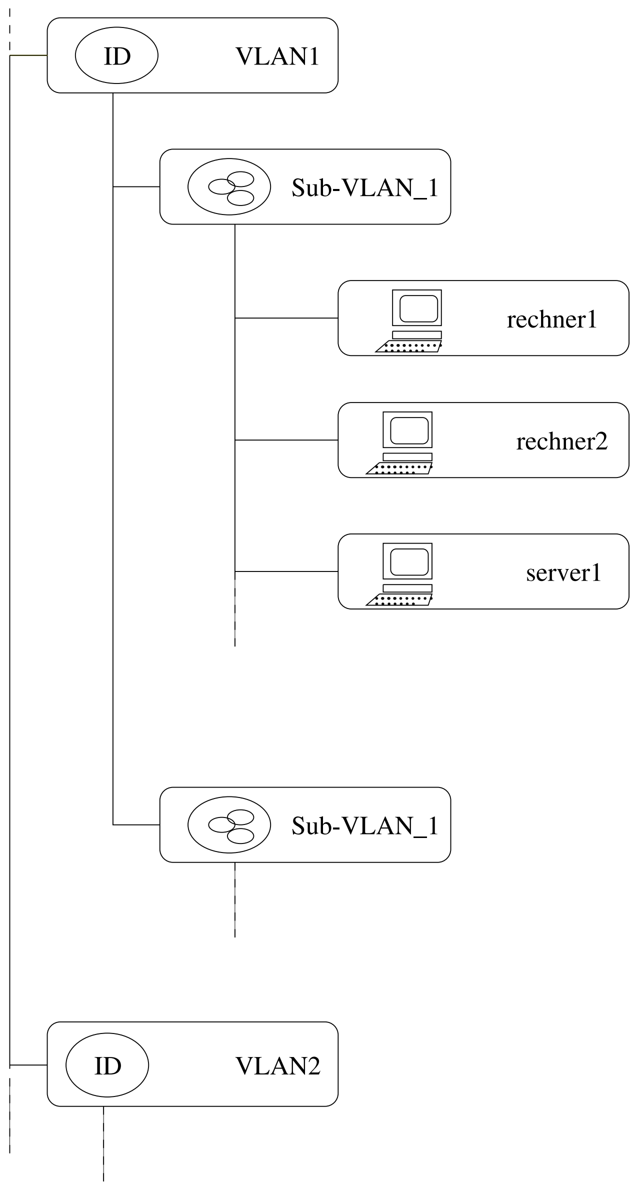


Abbildung 7.5: (VLAN→Sub-VLAN)-Sicht

die Mitgliedern gezeigt, die im selektierten Sub-VLAN enthalten sind. Durch einen Schalter können alle Komponenten des VLANs und/oder des Sub-VLANs automatisch angezeigt werden.

Die Realisierung der **physischen (VLAN→Sub-VLAN)-Sicht** erfolgt in der physischen Gesamtsicht. Alle Komponenten, außer die des betrachteten VLANs, werden in der physischen Gesamtsicht schemenhaft dargestellt bzw. ausgeblendet und die verbleibenden entsprechend ihrer Sub-VLAN-Zugehörigkeit eingefärbt.

7.3.2 Logische und physische (Gesamtnetz → Hersteller-VLAN)-Sicht

Durch Drücken des Radiobuttons **ALL→Vendor VLAN** wird das Gesamtnetz als ein einziges VLAN interpretiert und alle im System vorhandenen Hersteller-VLANs und bei Bedarf deren Endstationen in der **logischen (Gesamtnetz→Hersteller-VLAN)-Sicht** gezeigt ((ALL→Vendor VLAN)-Modus). Das Ergebnis ist der Abbildung 7.5 ähnlich, mit dem Unterschied das nur eine VLAN-Ikone mit dem Namen ALL Vendor VLAN vorhanden ist und die Sub-VLAN-Ikonen den Hersteller-Label tragen.

Eine Darstellung der **physischen (Gesamtnetz→Hersteller-VLAN)-Sicht** innerhalb der physischen Gesamtsicht könnte durch Einfärbung des Hintergrunds oder der Komponenten realisiert werden. Das Ergebnis im ersteren Fall wäre eine einem Voronoi-Diagramm nicht unähnliche Wabenstruktur. Auch für die physischen Sichten in den Abschnitten 7.3.3 und 7.3.4 wird die gleiche Strategie verwendet.

7.3.3 Logische und physische Hersteller-VLAN-Manager-Domänen-Sicht

Durch Drücken eines Radiobuttons kann die logische Sicht des Hauptfensters die **logische Hersteller-VLAN-Manager-Domänen-Sicht** darstellen. Es werden dann alle Hersteller-VLAN-Manager-Domänen aufgelistet, jede durch eine Ikone repräsentiert. Anhand des *Labels* sollte der Name des Hersteller-VLAN-Managers und evtl. der Standort (Adresse der Endstation(en)), auf der/denen er installiert ist, ersichtlich sein. Die Aufnahme weiterer Informationen in die Ikone wäre denkbar. Ein einfacher Mausklick auf die Ikone zeigt die Hersteller-VLANs, die in der Hersteller-VLAN-Manager-Domäne enthalten sind, gleichzeitig werden in der physischen Gesamtsicht die korrespondierenden Bereiche hervorgehoben. Abbildung 7.6 zeigt — ausgehend von den Hersteller-VLAN-Manager-Ikonen — diese logische Sicht. Ein Doppelklick auf eine Ikone startet den zugehörigen Hersteller-VLAN-Manager. Eine andere gängigere Möglichkeit ihn zu starten, wäre in den physischen Sichten die entsprechende Switching-Komponente anzuklicken.

Die Realisierung der **phys. Hersteller-VLAN-Manager-Domänen-Sicht** erfolgt in der physischen Gesamtsicht ähnlich wie in Abschnitt 7.3.2, mit den Unterschied das benachbarte Hersteller-VLANs verschmolzen (gleichfarbig) dargestellt werden, sofern sie vom gleichen HerstellerVLAN-Manager gemanagt werden.

7.3.4 Logische und physische VLAN-Management-Area-Sicht

Analoges zu den in den Abschnitt 7.3.2 und 7.3.3 entwickelten Sichten gilt auch hier bei der Visualisierung der **logischen und physischen VLAN-Management-Area-Sicht**, da eine hierarchische "ist Teil von" Beziehung vorliegt; eine VLAN-Management-Area besteht aus einer oder mehreren Hersteller-VLAN-Manager-Domänen, eine Hersteller-VLAN-Manager-Domäne wiederum aus einen oder mehreren Hersteller-VLANs und schließlich ist ein Hersteller-VLAN durch seine physischen Komponenten charaktierisiert. Abbildung 7.6 zeigt diese hierarchische Beziehung, wie sie im VLAN-Managementsystem dargestellt werden könnte.

Im Idealfall besteht die logische Darstellung der VLAN-Management-Area-Sicht aus einer Ikone, d.h. VLANs können (theoretisch) beliebig, über das ganze Netz verstreut, gebildet werden. Das System verdeckt die evtl. darunterliegende Heterogenität.

Die physische Darstellung der VLAN-Management-Area faßt Hersteller-VLAN-Manager-Domänen zusammen, die vom VLAN-Managementsystem in Beziehung zueinander gesetzt werden können. Im wesentlichen gilt für die physische Darstellung das bereits in den Abschnitten 7.3.2 und 7.3.3 Gesagte.

Bemerkung: Zwischen den logischen und physischen Darstellungen/Sichten besteht ein enger Zusammenhang. Befindet sich der Mauszeiger über einer VLAN-, Hersteller-VLAN-, Endstation-Ikone oder wird eine Ikone einmal oder zweimal angeklickt, wird/werden die Komponente(n) in der physischen Sicht hervorgehoben. Bisher wurde nur diese Richtung (logische Sicht → physische Sicht) besprochen; umgekehrt gilt Ähnliches: Befindet sich der Mauszeiger in der physischen Gesamtsicht über einer Endstation oder wird diese einmal oder zweimal angeklickt, so wird/werden die Komponente(n), je nach Kontext VLANs, Hersteller-VLANs, Hersteller-VLAN-Manager-Domänen oder VLAN-Management-Areas, sowohl in der physischen Gesamtsicht als auch in der logischen Sicht hervorgehoben bzw. gezeigt. Beispiel: Wird eine Endstation in der physischen Gesamtsicht einmal angeklickt, so wird z.B. im VLAN-Modus die logische Sicht so weit "aufgeklappt" bis die Endstation sichtbar ist. In der physischen Sicht wird gleichzeitig das VLAN hervorgehoben, daß die Endstation enthält. Ähnliches gilt für die restlichen Modi.

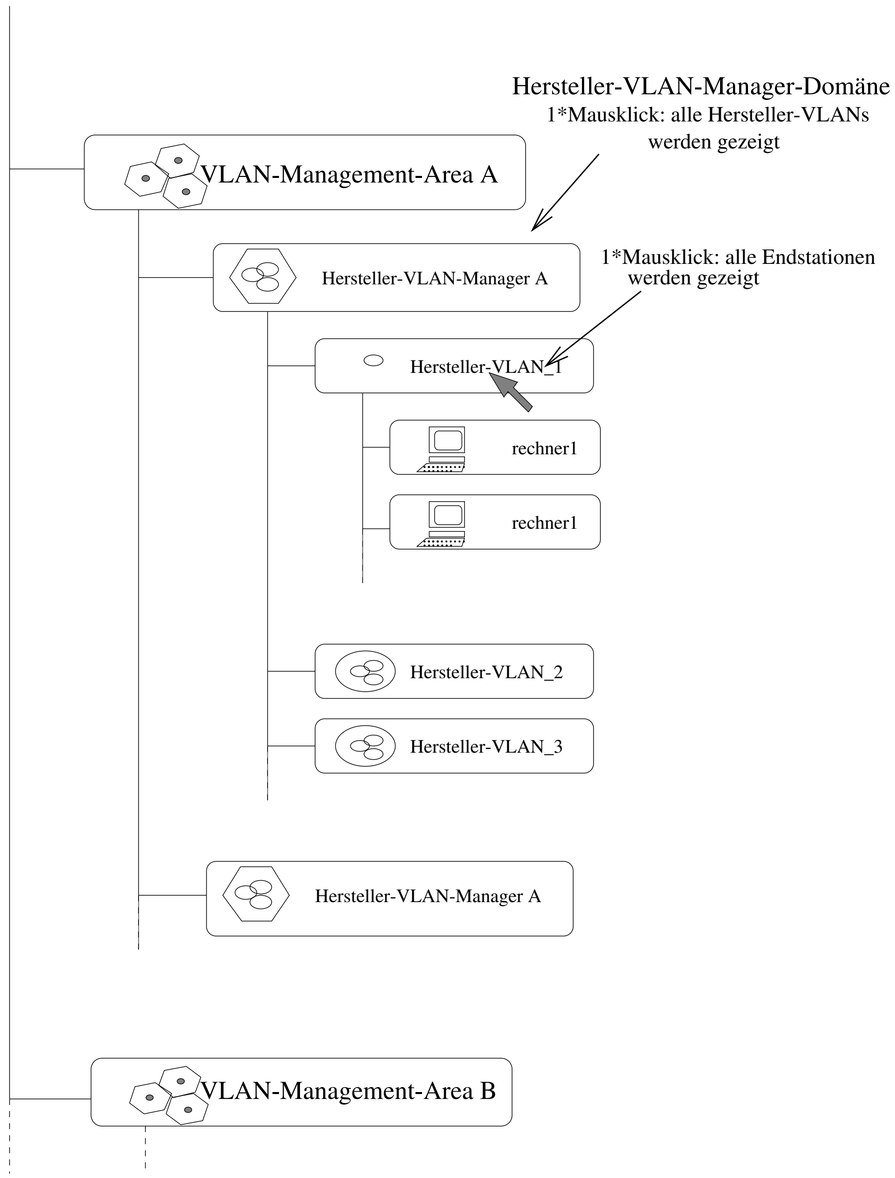


Abbildung 7.6: Hierarchie der Hilfssichten

7.4 Physische Switch/Port-Sicht

Von der physischen Darstellung ausgehend, gelangt der Administrator durch zweifaches Klicken auf eine Switch- oder ATM/LAN-Konverter-Ikone in die physische Switch/Port-Sicht (Abbildung 7.7). Das geöffnete Fenster zeigt u.a. die vorhandenen Anschlußmöglichkeiten und LEDs der betreffenden Komponente in einer realitätsnahen Darstellung. Anhand der farblichen Einfärbung der Anschlüsse (Stromversorgung, AUI, usw.) und der Ports ist der gegenwärtige Status ersichtlich. Die Farben Gelb und Rot sind wie gehabt reserviert, um Fehler zu signalisieren. Die Farbe Grün impliziert bei Ports keinen (feststellbaren) Fehler und keine Belegung. Die Zusammengehörigkeit einzelner Ports zu einem VLAN ist durch die Hinterlegung mit der gleichen Farbe ersichtlich (Portswitching), wobei die reservierten Farben der Zustandssemantik nicht benützt werden. Fehlerhafte Ports, die belegt sind, werden durch Blinken hervorgehoben. In einem yweiten Schritt kann dann der auftretende Fehler, im Rahmen der Fehlerbehandlung, festgestellt werden. Der *Label* eines Ports besteht aus der Portnummer und den VLAN-Namen. Ist die Zuordnung der Ports zu einem VLAN nicht eindeutig, so ist der Label entsprechend erweitert und der Port mit einer Farbe, die diesen Sachverhalt kodiert, hinterlegt.

All diese Informationen können optional in einem extra Fenster anhand einer übersichtlichen Tabelle/Legende abgerufen werden. Abbildung 7.7 zeigt schematisch die "Rückseite" einer Switching-Komponente.

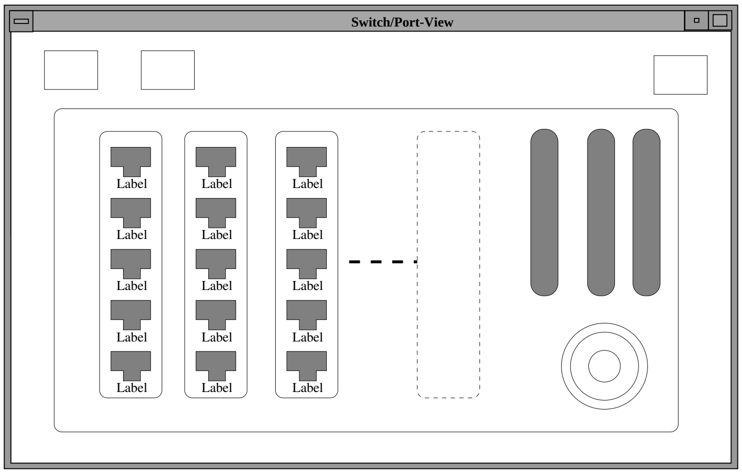


Abbildung 7.7: Physische Switch/Port-Sicht

Von diesen Fenster ausgehend, kann der Administrator einzelne Ports in ein VLAN-spezifisches-Fenster ziehen *(Drag & Drop)*. Alle Endstationen, die von diesem Port aus erreichbar sind, sind dann automatisch Mitglieder des VLANs (Portswitching).

Die Wirkung ist dieselbe, als wenn ein Segment bzw. Segmentsymbol aus der physischen Gesamtsicht in ein VLAN-spezifisches-Fenster gezogen wird (siehe auch Abschnitt 6.3.4). Das System prüft jede der Aktionen auf ihre Plausibilität und gibt geeignete Rückmeldungen.

Die physische Switch/Port-Sicht wird in der Regel von einem Elementmanager zur Verfügung gestellt. Durch diesen sind auch die üblichen Diagnostik- und Statistikfunktionen, die sich auf MIBs (OSI oder SNMP) und den RMON-MIBs (für jeden Ports) stützen, abrufbar. Diese Informationen beziehen sich aber nur auf die jeweilige Komponente. Der globale Zusammenhang ist nicht ersichtlich. Das System muß deshalb eine Möglichkeit bereitstellen, Informationen mehrerer Elementmanager (auch die der LAN Emulation Managementanwendung) in Beziehung zueinander zu setzen und in einem Gesamtzusammenhang darzustellen. VLAN-relevante Beispiele sind die Gesamtdarstellung der Auslastung (Abschnitt 7.8), die Backbone-Sicht (Abschnitt 7.10), die Segment-Sicht (Abschnitt 7.9) und die Sicht auf den Inter/Intra VLAN-Verkehr (Abschnitt 7.11).

7.5 Logische Switch/Port-Sicht

Von der physischen Switch/Port-Sicht oder, bei entsprechender Schalterstellung, von der physischen Gesamtsicht durch Anklicken der Switching-Komponente, ist die logische Switch/Port-Sicht erreichbar. Sie zeigt die Komponente und zusätzlich ihre Nachbarsysteme (Switching-Komponenten, ATM-Switches, Segmente, Endstationen, usw.) in einer logischen Darstellung. Weitere Verzweigungsmöglichkeiten, etwa zu den zugehörigen Elementmanager oder verschiedenen Diagnostik- und Statistik-Sichten, sind vorgesehen.

Die logische Switch/Port-Sicht löst sich von der starren physischen Darstellung, die durch die Bauart und dem Design der physischen Komponente vorgegeben ist. Dadurch wird die Möglichkeit geschaffen mehr Informationen, als in der physischen Switch/Port-Sicht, am Bildschirm anzuordnen, da die oben genannten Schranken wegfallen und das Augenmerk auf das Design der Informationsaufbereitung und -präsentation gerichtet werden kann. Zudem werden die angeschlossenen Nachbarkomponenten in die Sicht mit einbezogen (Abbildung 7.8). Es wäre denkbar — bei kleinen Netzen – diese Sichtweise auf das ganze Netz auszuweiten.

Das Zentrum der logischen Switch/Port-Sicht zeigt die relativ zur Komponente konfigurierten VLANs. Es gibt drei Klassen von Schnittstellen zur Außenwelt: Ports an denen Segmente und Ports an denen Backbones angeschlossen sind und die durch einen ATM-Adapter zur Verfügung gestellten Schnittstellen der LAN Emulation[11].

Die Ports sind nach ihrer VLAN-Zugehörigkeit farblich gekennzeichnet und grup-

[11]Genauer: Die von den LECs realisierte LUNI-Schnittstelle zum ATM-Netz.

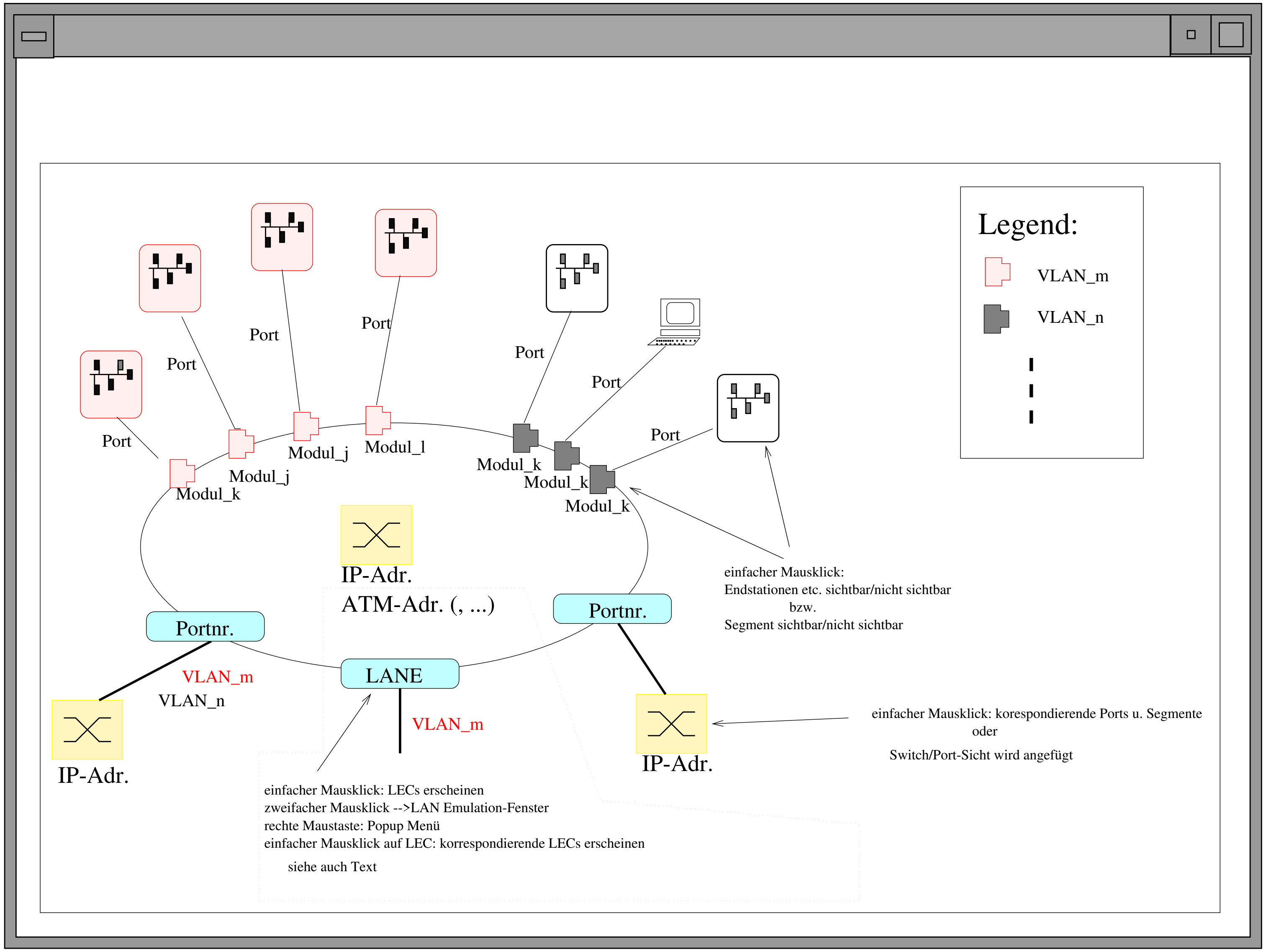


Abbildung 7.8: Logische Switch/Port-Sicht

piert. Die Ports können nach Steckkarten oder VLANs gruppiert werden (im Unterschied zur physischen Switch/Port-Sicht)[12]. Eine Legende gibt wieder Hilfestellung. Die Zustandssemantik, die farblichen Kodierungen sind in vorhergehenden Abschnitten bereits besprochen und können übernommen werden. Ports, die noch nicht konfiguriert sind, werden auch aufgeführt und, wenn kein feststellbarer Fehler vorhanden ist, grün hinterlegt. Die an den Ports angeschlossenen Segmente[13] werden durch Segment-Ikonen symbolisiert und durch die Switchadresse und Portnummer sind sie eindeutig identifizierbar innerhalb des Systems. Die Darstellung einer Endstation-Ikone impliziert, daß nur eine Endtation an den Port angeschlossen ist (*dedicated* Ethernet). LANE-fähige ATM-Endstationen sind durch die LANE Schnittstelle angebunden, genauso Segmente, für die ein LEC zuständig ist. Durch Klicken auf eine Segment-Ikone wird das Subnetz gezeigt. Der Administrator kann diesen Vorgang automatisieren, indem er das System explizit dazu veranlaßt, anstatt der Segmentsymbole alles anzuzeigen. Das Ergebnis ist im Grunde genommen ähnlich einem Ausschnitt der physischen Gesamtsicht, nur daß jetzt die Darstellung der Switches eine andere ist. Da das Hauptinteresse in der logische Switch/Port-Sicht die Belegung der Anschlüsse ist, dient das Segmentsymbol einerseits der bereits erwähnten Repräsentation klassischer *Kollisionsdomänen* und andererseits zur Platzersparnis und Wahrung der Übersichtlichkeit.

Ausgehend von den Ports sind die über ein Backbone erreichbaren Switches ersichtlich. Klickt man auf einen solchen Switch, so wird dessen logische Switch/Port-Sicht gezeigt. Es ist hier zu entscheiden, wie bereits erwähnt, ob sukzessive so weiter verfahren werden soll, um so mehrere logische Switch/Port-Sichten bis hin zum ganzen Netz in einem Fenster zu haben, oder jede Switch/Port-Sicht in einem eigenen Fenster darzustellen. Die Beschriftung der Backbones gibt Aufschluß, welche VLANs switchübergreifend konfiguriert sind.

Die vorhandene LANE-Ikone suggeriert dem Administrator, daß der betrachtete Switch ein ATM/LAN-Konverter oder Edge Device (und somit LANE-Fähig) ist. VLANs die sich auf die Dienste der LAN Emulation stützen, sind durch Angabe ihrer Namen (VLAN-Name=ELAN-Name[14]) in der entsprechenden Farbe unterhalb des LANE-Ikone angedeutet.

Durch Klicken auf die LANE-Ikone werden die LECs und evtl. die in diesem ATM/LAN-Konverter instantiierten Server der LAN Emulation gezeigt, durch Klicken auf einen LEC[15] werden die korrespondierenden LECs[16] angezeigt. Ein weiterer Klick auf einen der LECs zeigt den ATM/LAN-Konverter als Ikone (Komplettsicht) und

[12] In gewisser Weise sind die Ports in der physischen Switch/Port-Sicht **nur** nach den Steckkarten bzw. Modulen, auf denen sie sich befinden, gruppiert.

[13] Siehe auch Abschnitt 3.1: Segment=Kollisionsdomäne.

[14] Siehe auch Abschnitt 2.5.2 und 7.6.

[15] Für jedes VLAN gibt es im ATM/LAN-Konverter höchstens einen LEC.

[16] Eindeutig durch Kombination von ATM-Adresse(en) oder IP-Adresse des ATM/LAN-Konverters und einer laufender Nummer.

mit einem weiteren Klick auf die Komponente, die zu dem VLAN gehörigen Segmente oder/und evtl. Backbones hin zu anderen Switching-Komponenten. Der Administrator kann sich so sukzessive *entlang* der einzelnen VLANs weiterklicken, bis schließlich die ganzen Komponenten eines oder mehrerer VLANs "aufgeklappt" und sichtbar sind[17].

Ein Doppelklick auf das LANE-Ikone öffnet das LAN Emulation Fenster, das im nächsten Abschnitt vorgestellt wird.

7.6 LAN Emulation

Das System verbirgt beim VLAN-Management die Existenz der LAN Emulation so lange wie möglich. Die Instantiierungen der einzelnen Clients und Server der LAN Emulation (LEC, LES, BUS und LECS) werden automatisch durch das System veranlaßt. Bei auftretenden Fehlern werden dem Administrator diese durch eine farbliche Kennzeichnung der LANE-Ikone signalisiert (Abschnitt 7.7).

Die intuitivste Möglichkeit in das LAN Emulation Fenster (Abbildung 7.9) zu gelangen, ist die, das in der physischen Gesamtsicht durch eine Ikone repräsentierte Objekt LAN Emulation anzuklicken. Die Ikone entspricht der Komplettsicht des Objekts LAN Emulation, das entsprechende Fenster der Detailsicht.

Im folgenden werden zunächst Vorschläge einer Benutzerschnittstelle für das LANE-Management, die für das VLAN-Management von Interesse sind, vorgestellt. Gleichzeitig wird aber die Einbettung der Vorschläge in die physische Gesamtsicht nicht aus dem Auge gelassen, so daß die Darstellung der Komponenten der LANE, der ATM/LAN-Konverter und die der physischen Gesamtsicht fließend ist. Obwohl es sich einerseits um Software-Komponenten und andererseits um physische Komponenten handelt, findet kein Strukturbruch statt. Bei der Besprechung der LANE-Managementschnittstelle wird weitgehend auf eine nähere Besprechung der von den ATM-Device-Elementmanagern angebotenen Sichten und Funktionalitäten verzichtet, da sie zum einen auf dieser Ebene (VLAN) nicht relevant sind und es andererseits bereits konkrete Lösungen gibt, und somit deren Darlegung für diese Arbeit keine Eigenleistung mehr bedeuten würde.

Das LAN Emulations Fenster ist in einen logischen und einen graphischen Teil[18] gegliedert (Abbildung 7.9).

[17] Die obige Beschreibung soll nur als ein Vorschlag aufgefaßt werden. Es sind durchaus andere Szenarien denkbar oder besser: Das System sollte Möglichkeiten anbieten, sich seine eigenen Sichten/Views zu definieren, d.h. Möglichkeiten die es erlauben, die Komponenten, die einen interessieren, sich automatisch, beim Eintreffen einer vorher festgelegten Aktion, anzeigen zu lassen.

[18] Neben den physischen Komponenten werden noch die in der Regel durch Software realisierten Instantierungen der Clients und Server der LAN Emulation gezeigt. Deshalb wird das Adjektiv "graphisch" anstatt "physisch" verwendet.

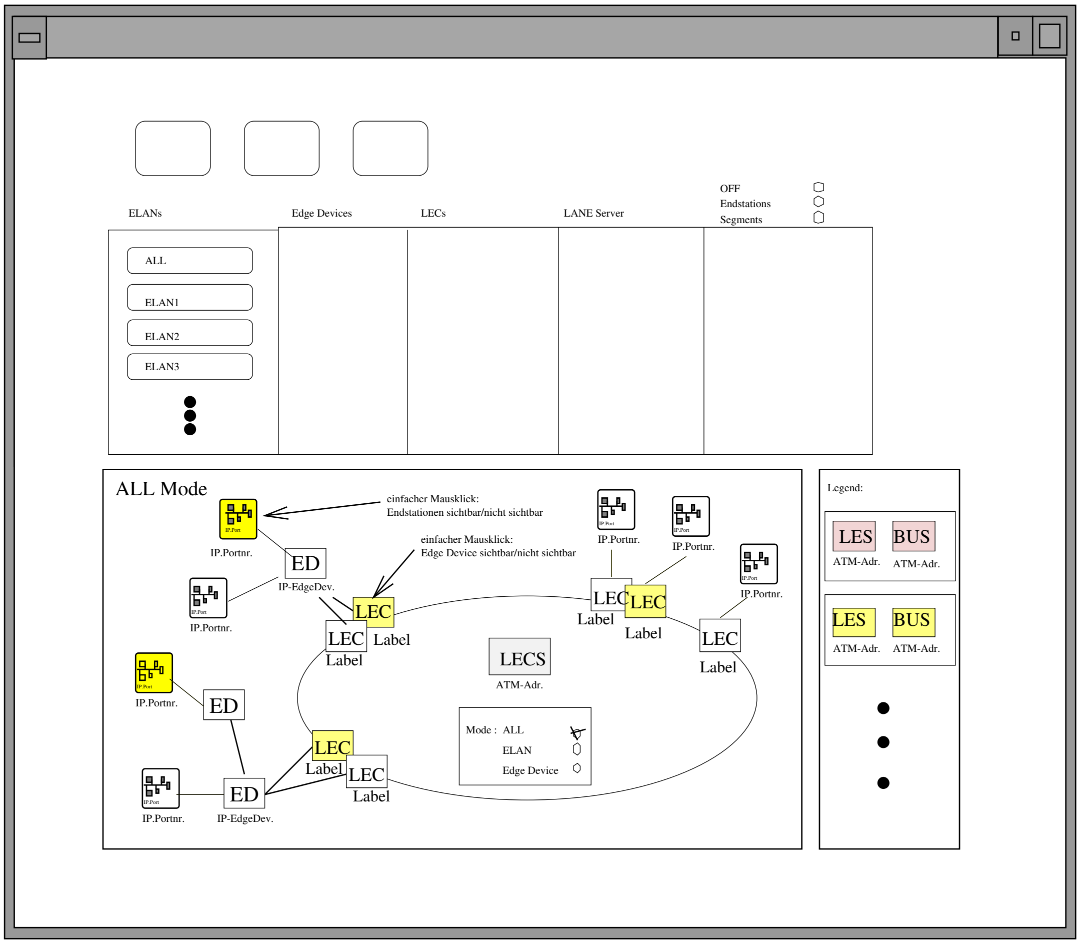


Abbildung 7.9: LAN Emulation Fenster (ALL Mode)

Die logische Sicht auf die LAN Emulation wird wieder mit Listen und deren Handhabung mit Hilfe eines Browsers realisiert. Die Listenelemente (Ikonen) sind bezüglich der farblichen Kodierung des Zustandes und der Zusammengehörigkeit, wie bereits festgelegt, eingefärbt. Das erste Feld, das **ELAN-**Feld, enthält eine Auflistung aller vorhandener ELANs. Das Listenelement ALL ELAN repräsentiert dabei alle ELANs. Innerhalb des Systems bekommen die ELANs die gleichen Namen, wie die VLANs, d.h. ein VLAN, das sich der Dienste der LAN Emulation bedient, gibt dem involvierten ELAN seinen Namen. Das zweite Feld, das **Edge Device-**Feld, listet die ATM/LAN-Konverter auf. Das dritte und vierte Feld, das **Client-** und **Server-**Feld, zeigt die Clients und Server der LAN Emulation. Das fünfte und letzte Feld, das **Segment-**Feld, listet die Endstationen und Segmente (Kollisionsdomänen) auf, die von einem LEC bedient werden. Segmente werden in der Regel als Tupel mit der Adresse des (ATM-)Switches und der Portnummer als Label aufgeführt. In jedem dieser fünf Felder gibt es ein ALL-Listenelement.

Der Inhalt der logischen Felder ist abhängig von den Interaktionen des Administrators. Ausgehend von dem Feld, in dem die Interaktion erfolgte, wird der Inhalt der restlichen Felder angeglichen, so daß alle Felder als Ganzes eine (konsistente) Interpretation der Struktur und der Komponenten der LAN Emulation erlauben. In der graphischen Sicht werden, wie noch an geeigneter Stelle besprochen wird, die in der logischen Sicht selektierten Elemente (ATM/LAN-Konverter, Komponenten der LANE, Endstationen oder Segmente) — und umgekehrt — hervorgehoben, evtl. wird der Modus der graphischen Sicht gewechselt.

Klickt man eine oder mehrere Ikonen im ELAN-Feld an, so werden die in den ELANs involvierten ATM/LAN-Konverter im Edge-Device-Feld, die Instantiierungen der Clients und Server im Client- und Server-Feld und die Segmente und Endstationen im Segment-Feld aufgelistet. Klickt man auf einen oder mehrere ATM/LAN-Konverter, so werden die darin konfigurierten ELANs im ELAN-Feld, die Instantiierungen der Clients und Server im Client- und Server-Feld und die Segmente oder Endstationen im Segment-Feld aufgelistet. Analoges gilt für die Client-, Server- und Segment-Felder.

Bis jetzt wurde nur die logische Darstellung der LAN Emulation besprochen. Im unteren Teil des LAN Emulation Fensters wird zusätzlich eine graphische Sicht auf die Komponenten der LAN Emulation eingeführt. Diese ist wie gewohnt eng mit der logischen Darstellung verknüpft.

Es werden innerhalb der graphischen Sicht drei verschiedenen Modi eingeführt:

(a) **ELAN Mode:** gezeigt werden alle in einem ELAN involvierten ATM/LAN-Konverter (*Edge Devices (EDs)*), LECs und die korrespondierenden Server: LES, BUS und LECS (Abbildung 7.10 a).

(b) **ALL Mode:** wie ELAN-Mode, nur alle ELANs werden gleichzeitig dargestellt.

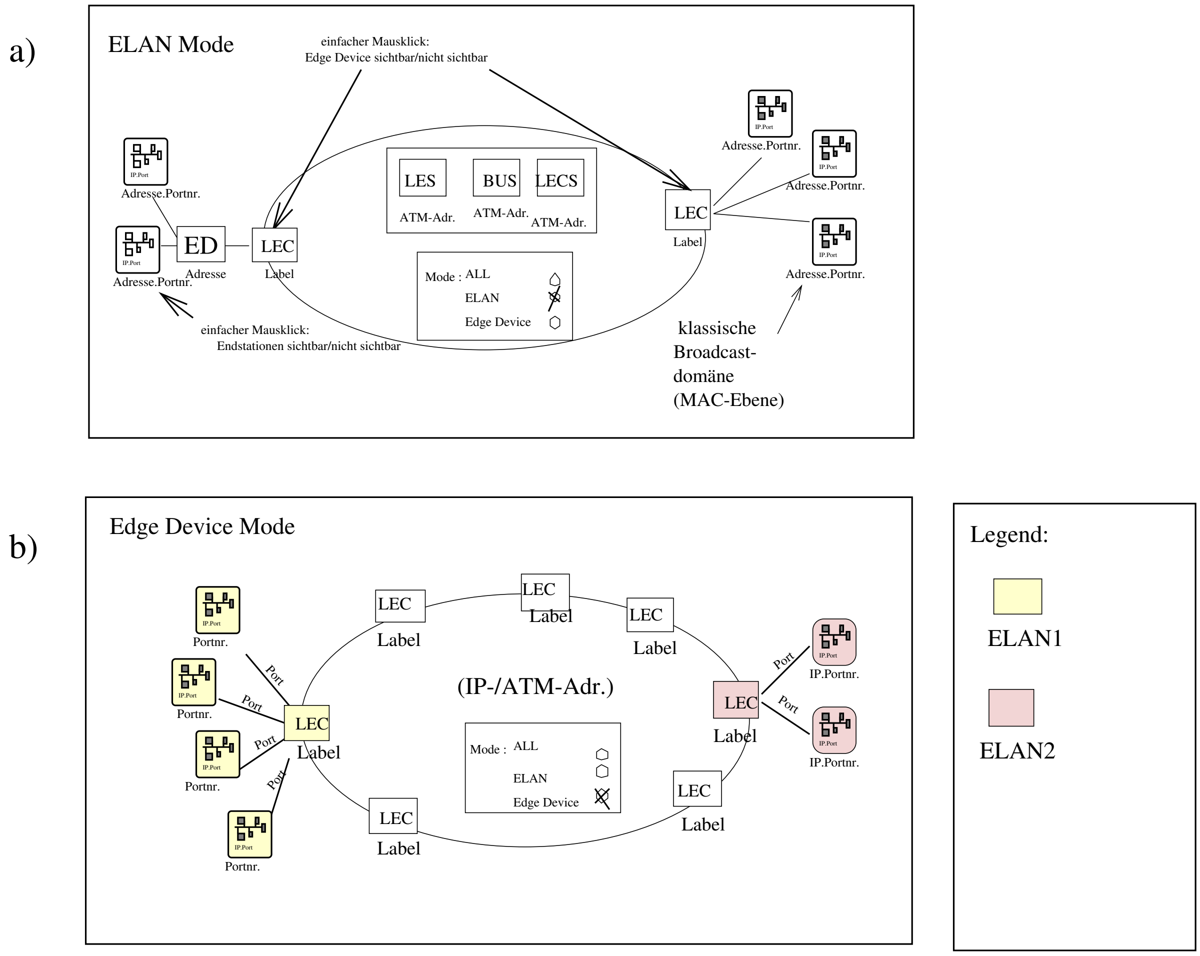


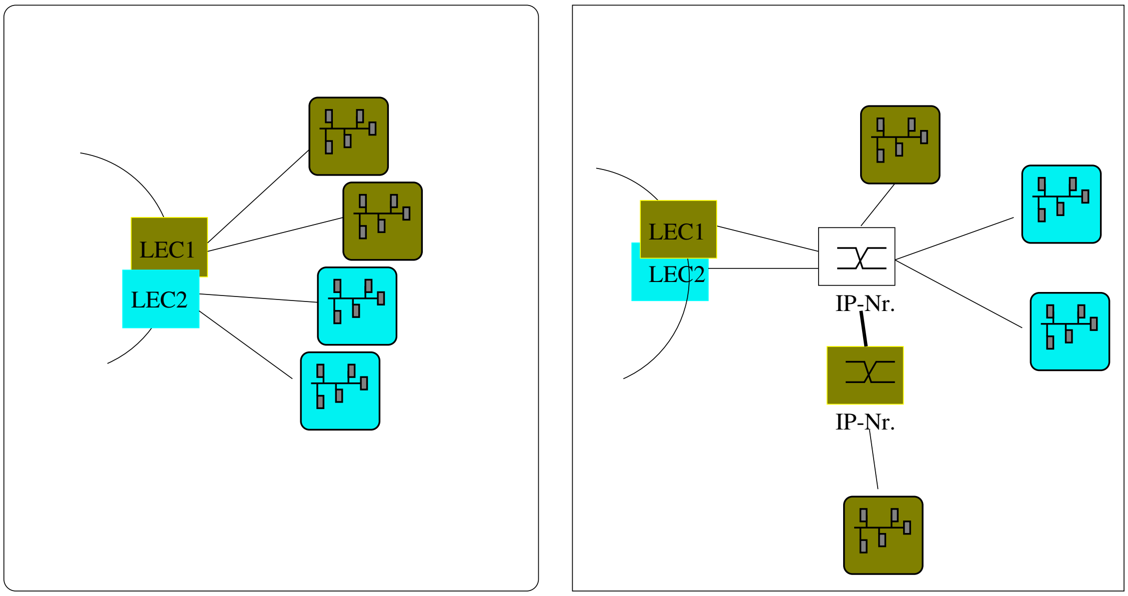
Abbildung 7.10: ELAN und Edge Device Mode

ATM/LAN-Konverter die (noch) zu keinem ELAN gehören werden auch gezeigt (Abbildung 7.9).

(c) **Edge Device Mode:** In dieser Sichtweise bildet ein ATM/LAN-Konverter den Mittelpunkt (Abbildung 7.10 b), ähnlich der Darstellung eines Ethernet-Switches in der logischen Switch/Port-Sicht.

Das Zentrum der graphische Sicht — in jedem Modus — bildet ein Oval auf dem die LECs angeordnet werden. Je nach Modus symbolisiert es einmal das ganze ATM-Netz (ALL-Mode, Abbildung 7.9), ein anderes Mal ein ELAN (ELAN-Mode, Abbildung 7.10 a) oder einen ATM/LAN-Konverter (Edge Device Mode, Abbildung 7.10 b). In der Standardeinstellung hängen an den LECs Segmet-Ikonen oder, bei dedizierten Anschlüssen, Endstationen. Es handelt sich dabei um die in der logischen

Sicht aufgelisteten Endstationen und Segmente. Die Segment-Ikone symbolisiert wieder eine physische Kollisionsdomäne und dient in erster Linie der Platzersparnis. Zu beachten ist, daß sie — in der graphischen Sicht — logisch mit dem LEC verbunden sind und nicht direkt an dem Port des ATM/LAN-Konverters angeschlossen sein müssen, in der der zugehörige LEC instantiiert ist, sondern — bildlich gesprochen — können in der aufgeklappten graphischen Sicht "auf dem Weg vom LEC zur Endstation, zum Segment" noch weitere Backbones und Switching-Komponenten (z.B. auch Ethernet-Switches) liegen. Daher ist es offensichtlich, daß eine Verknüpfung — zumidest in einem zweiten Schritt — mit den physischen Sichten notwendig ist. Abbildung 7.11 a) und b) zeigt einen solchen Fall.



a) b)

Abbildung 7.11: Einbettung der LAN Emulation in die physische Gesamtsicht

Ein Klicken auf einen LEC, in der Standardeinstellung[19] (Abbildung 7.11 a), verursacht ein "Aufklappen" der physischen Sicht und zeigt die dazwischenliegenden Backbones und Switching-Komponenten (Abbildung 7.11 b). Ein Klicken auf ein Segmentsymbol zeigt schließlich das physische Netz (Kollisionsdomäne) mit allen Endstationen und den restlichen klassischen (LAN-)Vermittlungseinrichtungen. Um diese zu erkennen, ist aber die Zuhilfenahme von anderen Managementanwendungen (Discovery-Funktionen, usw.) notwendig[20]. Die Tabellen der LECs können aufgrund

[19]Der Administrator kann diese Standardeinstellung ändern und sich jeweils automatisch alle involvierten Komponenten zeigen lassen.

[20]Siehe auch Abschnitt 6.2.1: Funktionelle Schichtung des Systems.

ihrer möglichen Proxyeigenschaft[21] nicht herangezogen werden, da sie in der Regel nicht alle MAC-Adressen der Endstationen enthalten, die sie bedienen.

Zu diskutieren wäre, ob im ELAN- und ALL-Mode die physischen ATM/LAN-Konverter in den Standardeinstellungen überhaupt gezeigt werden sollen. Ein LEC ist durch eine fortlaufende Nummer, die ihn innerhalb der physischen Komponente eindeutig kennzeichnet, und der IP- und/oder ATM-Adresse(n) der Komponente eindeutig identifizierbar. Durch einen Verzicht auf die Darstellung der ATM/LAN-Konverter und anschließender Gruppierung der LECs nach gleicher Adresse[22] kann vielleicht sogar eine bessere Übersicht erreicht werden. Andererseits werden im nächsten Abschnitt verschiedene Fehlerarten innerhalb der LAN Emulation (tolerierbar bis schwerwiegend) besprochen. Um Fehler innerhalb der LAN Emulation zu lokalisieren, ist u.a. eine Darstellung der physischen Komponente (bei Hardware-Fehler) erforderlich. Wie so oft sollte der Administrator selbst entscheiden können, ob er eine minimale oder eine ausgiebigere Darstellung für geeigneter hält; das System jedenfalls sollte die Mechanismen zu deren Umsetzung zur Verfügung stellen.

7.7 Fehlerbehandlung in der LAN Emulation

Der Administrator sollte beim Managment der VLANs von der Existenz der LAN Emulation im Idealfall nichts mitbekommen. Das System gibt z.B. bei der Konfiguration eines VLANs, unter Zuhilfenahme der Dienste der LAN Emulation über ein ATM-Netz hinweg, entsprechende Rückkopplung. Die beteiligten ATM/LAN-Konverter werden, wie die anderen Komponenten des VLANs (Switches, Backbones, Endsysteme, Segmente, Segmentsymbole), bei positiver Bestätigung durch das System mit der gleichen Farbe hinterlegt (Ikonen in den logischen und physischen Sichten).
Ein Managementwerkzeug für die LAN Emulation benötigt der Administrator erst dann, wenn er sich näher für dessen Zusammensetzung interessiert oder wenn ein Fehler aufgetreten ist. Zunächst muß dem Administrator das Vorhandensein eines Fehlers angezeigt werden. Seine Aufgabe ist es dann ihn zu lokalisieren, die Ursache für den Fehler zu bestimmen und schließlich Gegenmaßnahmen einzuleiten. Letztere Tätigkeit wird in dieser Arbeit nicht besprochen. Sie ist der typische Einsatzbereich der Element-/*Device*-manager bei Hardware-Fehlern oder einer *native* LAN Emulation Managementanwendung[23].

Der gegenwärtige Zustand der LAN Emulation ist wieder aus den Farben der Ikonen ersichtlich:

[21] Siehe auch Abschnitt 2.5.1.
[22] Nur sinnvoll im ALL Mode; im ELAN Mode ist ein Sortieren nicht möglich, da es nur einen LEC für jedes ELAN pro ATM/LAN-Konverter gibt.
[23] Siehe wieder Abschnitt 6.2.1.

LAN Emulation Status	
Farbe	Zustand
Grün	Normaler Betrieb
Gelb	Warnung (Endstation *down*, Segment unreachable, ...)
Rot	Kritischer Zustand (LANE-Client, -Server oder ATM-Komponente *down*)

Im Normalzustand ist die LANE-Ikone in den physischen Sichten (Gesamtsicht und VLAN-spezifische-Sicht) grün eingefärbt. Sobald ein Fehler auftritt, färbt sie sich gelb oder rot ein, je nach Art des Fehlers. Um den Fehler zu lokalisieren, muß der Administrator in das LAN Emulation Fenster wechseln. Dort wird dem Administrator die fehlerhafte Komponente (LANE oder Hardware), sowohl in der logischen als auch in der graphischen Sicht, hervorgehoben präsentiert. Nachdem der Fehler lokalisiert ist, wird durch Anklicken der Komponente in die Fehlerbearbeitung verzweigt.

7.8 Gesamtdarstellung der Auslastung

Um sich einen schnellen Gesamtüberblick der aktuellen Leitungsauslastung zu verschaffen, kann die physische Gesamtsicht dahingehend erweitert werden (optional), daß die in ihr dargestellten Segmente und Backbones mit einer Prozentangabe versehen werden, die die Auslastung im letzten Intervall angibt (Abbildung 7.12). Das Intervall kann vom Administrator vorgegeben werden.

Der Administrator sieht so auf den ersten Blick die Gesamtauslastung der Leitungen und kann Verkehrsengpässe *(Bottlenecks)* schon im Ansatz erkennen. Durch Anklicken eines Segments oder eines Backbones in einer der physischen Sichten gelangt er in die Segment- (Abschnitt 7.9) oder Backbone-Sicht (Abschnitt 7.10).

7.9 Segment-Sicht

Die Segment-Sicht wird in dieser Arbeit nicht visualisiert, da sie im Grunde genommen nur aus Tabellen und "hübschen" Diagrammen besteht. Wichtig ist daher nur stichpunktartig aufzuzählen, was sie enthalten sollte. Segmente sind an den Ports der Switching-Komponenten angeschlossen, jeder dieser Ports wird in der Regel mit einer RMON-MIB ausgestatten. Es liegt also nahe diese Informationen auszuwerten, in Beziehung zueinander zu setzen und geeignet darzustellen, wie z.B. die gegenwärtige Auslastung oder die Verteilung und Art der verwendeten Protokolle auf dem Segment. Weitere wichtige darzustellende Informationen sind die Anzahl der Broadcast, die Anzahl und Richtung der Frames (Port in/out), die Anzahle der

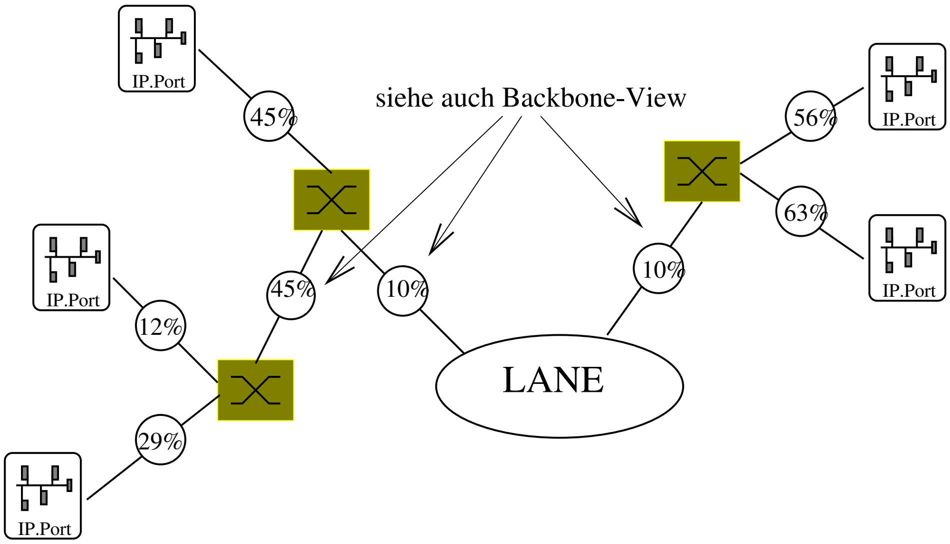


Abbildung 7.12: Gesamtauslastung

fehlerhaften Frames (Port in/out) in Paketen pro Sekunde, Anzahl der Kollisionen, usw. Genauso sollten alle Daten, soweit wie möglich und sinnvoll, historisiert dargestellt werden, d.h. der zeitliche Verlauf soll anhand eines Graphen ersichtlich sein.

7.10 Backbone-Sicht

In der Backbone-Sicht werden vom VLAN-Managementsystem die Daten zweier Switchingkomponenten ausgewertet, um die VLAN-relevanten Informationen für den Administrator ableiten zu können. Bei der Darstellung der Backbone-Sicht in Abbildung 7.13 handelt es sich um eine Tabelle in der die VLANs aufgeführt werden, die über den Backbone (switchübergreifend) konfiguriert sind.

Der durch ein VLAN verursachte Verkehr über den Backbone wird dabei in beiden Richtungen differenziert betrachtet, wobei die Verkehrslast in Mbit pro Intervall[24] und der prozentuale Anteil an der Gesamtauslastung die wichtigsten Informationen sind. Durch Anklicken eines *VLANname*-Feldes, werden an den Ikonen der Switching-Komponenten die Ports angegeben, die von dem selektierten VLAN benutzt werden. Denkbar wäre auch diese Ports mit der prozentualen Angabe der Auslastung zu versehen. Zudem werden die Ausgangsports für jede Richtung in die Tabelle mit aufgenommen. Die Aufnahme weiterer Informationen in die Tabelle wäre denkbar. Am Ende der Tabelle werden die Anzahl der VLANs und alle Werte — sofern sinnvoll

[24]Intervall wieder vom Administrator einstellbar.

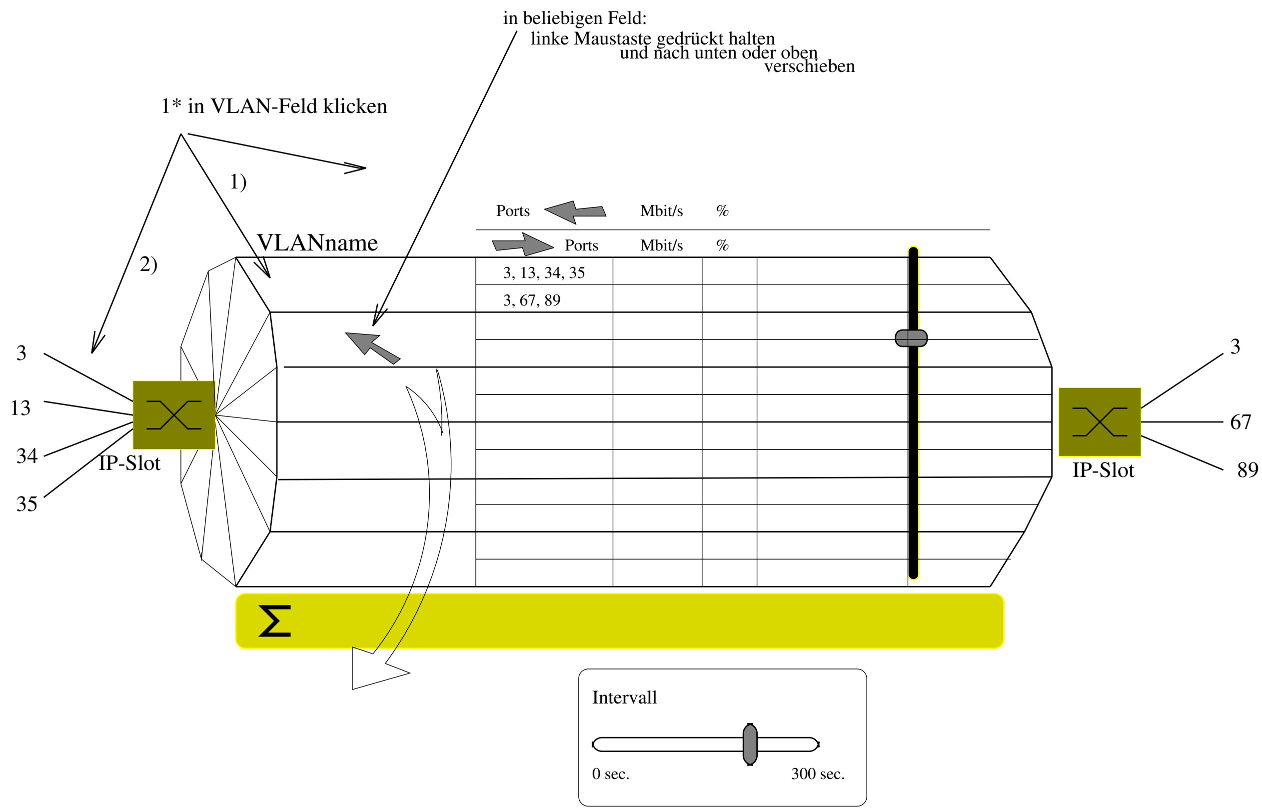


Abbildung 7.13: Backbone-View

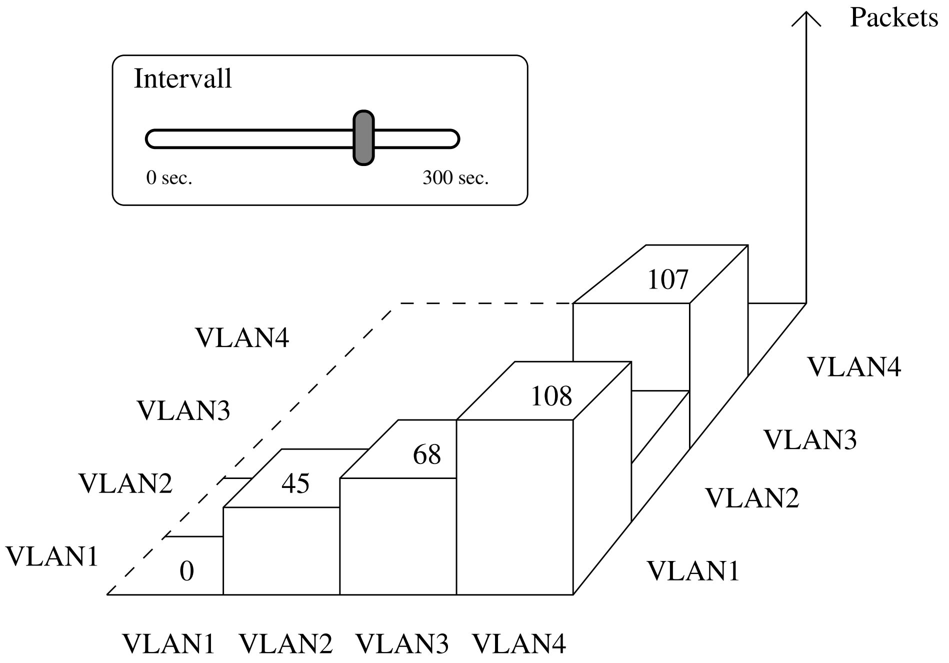


Abbildung 7.14: Inter/Intra-VLAN-Sicht

— aufsummiert.

7.11 Verkehr innerhalb und zwischen VLANs

Wie bereits öfters erwähnt, werden VLANs auch eingesetzt, um den Verkehrsfluß zu separieren und zu steuern, mit dem Ziel, eine höhere Verfügbarkeit des Netzes und damit eine bessere *Performance* als in klassischen LANs zu erreichen und der Umgehung von Verkehrsengpässen. Aufgrund der Flexibilität, die dem Administrator bei der Konfiguration eines VLANs zur Verfügung steht, kann es durchaus auch zu ungewollten Leistungseinbrüchen kommen. Es ist daher erforderlich mit Hilfe der bereits besprochenen und der nachfolgenden Sicht den Verkehr ständig zu kontrollieren. Die Intra/Inter-VLAN-Sicht (Abbildung 7.14) kann dem Administrator Anhaltspunkte liefern, welche VLANs konkret einer Umstrukturierung bedürfen.

An der Diagonalen ist der Verkehr innerhalb eines VLANs abzulesen, die restlichen Säulen visualisieren den Verkehr zwischen verschiedenen VLANs. Der Verkehr wird in Pakete pro Intervall angegeben. Der Intervall kann wieder vom Administrator individuell vorgegeben werden. Die Werte werden historisiert, die graphische Darstellung erfolgt in einem extra Fenster. Zeigt die Inter/Intra-VLAN-Sicht ungewöhnliche Spitzen, so hat man ein Indiz dafür, daß eine Umkonfiguration notwendig ist. Mit großer Wahrscheinlichkeit werden sich dann auch in den anderen Sichten Verkehrsengpässe abzeichnen.

Kapitel 8

Prototypische Implementierung

In diesem praktischen Teil der Arbeit werden einige der in den vorangegangenen Kapiteln gemachten Vorschläge umgesetzt. In Abschnitt 8.1 wird festgelegt, was aus der Vielzahl der Vorschläge wichtig ist, um die wesentlichen Ideen für das VLAN-Management zu verdeutlichen und dabei gleichzeitig den Realisierungsaufwand in Grenzen zu halten. Bevor das Ergebnis vorgestellt wird, werden in Abschnitt 8.2 Werkzeuge auf ihre Eignung hin untersucht, die prototypische Implementierung möglichst rationell zu realisieren. Abschnitt 8.3 beschreibt in groben Zügen die Struktur des Quellkodes.

8.1 Ziel der prototypischen Implementierung

Der Prototyp der Bedienoberfläche soll in einem Hauptfenster die logische und physische Gesamtsicht enthalten. Innerhalb dieser Sichten werden auch die Hilfssichten angeboten.
Damit der Prototyp nicht nur aus "gezeichneten" Bildschirmmasken besteht und, was wichtiger ist, um die wesentliche Idee, das *Feeling* beim Management virtueller LANs zu vermitteln, wird er zu einem "Demonstrator" ausgebaut, d.h. es werden Teile der Dialogkontrolle und Teile des Anwendungskerns ausprogrammiert. Dieser Übergang von der bloßen Bildschirmmaskendarstellung zum Demonstrator hat gravierende Auswirkungen auf die Auswahl des Implementierungswerkzeuges (Abschnitt 8.2).

So gibt es im Demonstrator bereits vorkonfigurierte VLANs in der logischen und ein existierendes Beispielnetz in der physischen Sicht, das VLAN-relevante Komponenten im Zusammenhang zeigt. Abbildung 8.1 zeigt das Hauptfenster des Demonstrators.

Beim einfachen Anklicken einer VLAN-Ikone in der logischen VLAN-Sicht werden

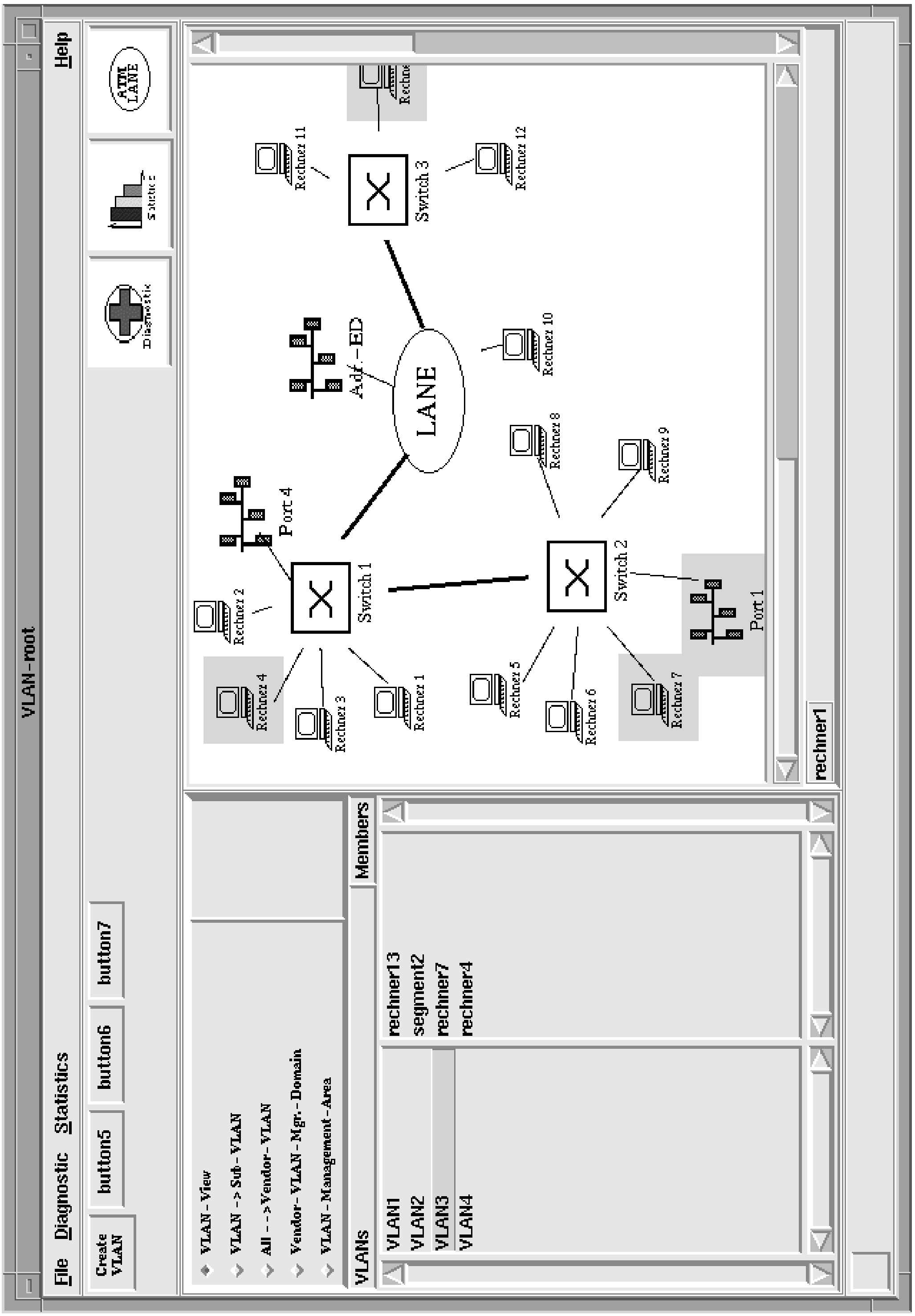


Abbildung 8.1: Hauptfenster des VLAN-Managementsystems

die Mitglieder aufgelistet und parallel dazu in der physischen Sicht hervorgehoben (in Abbildug 8.1 ist VLAN3 selektiert). Durch doppeltes Anklicken einer VLAN-Ikone wird das zugehörige VLAN-spezifische-Fenster geöffnet. Abbildung 8.2 zeigt die VLAN-spezifische-Sicht für das in Abbildung 8.1 selektierte VLAN.

Es enthält in einer logischen Sicht die Mitglieder des VLANs und in der physischen Sicht die relevanten Komponenten in einer zusammenhängenden Darstellung.

Durch Anklicken von Radiobuttons im Hauptfenster des Demonstrators wird der Modus gewechselt, die logische VLAN-Sicht wird zur logischen VLAN-Management-Area-, zur Hersteller-VLAN-, zur Hersteller-VLAN-Manager-Domänen- oder zur (VLAN→Sub-VLAN)-Sicht und die physische Gesamtsicht zur jeweiligen korrespondierenden physischen Sicht. Es werden nach Selektion einer Ikone in den logischen Sichten die Mitglieder und Komponenten sowohl in der logischen als auch in der physischen Sicht gezeigt.

Durch Anklicken des *Create VLAN*-Buttons im Hauptfenster erscheint eine leere VLAN-spezifische-Sicht mit einen vom System vergebenen eindeutigen fortlaufenden VLAN-Identifikator.

Dies sind die wesentlichen selbstgestellten Vorgaben, was die prototypische Implementierung bzw. der Demonstrator zeigen soll. Ziel ist es, die Idee der Vorschläge geeignet "rüberzubringen", schematisch darzustellen, anzudeuten, ohne dabei auf irgendwelchen anderen Restriktionen (Geschwindigkeit, Wiederverwendbarkeit, Kompatibilität, usw.) Rücksicht zu nehmen.

8.2 Vergleich der verfügbaren Werkzeuge

Um die Vorgaben aus den letzten Abschnitt möglichst rationell zu implementieren, müssen geeignete Werkzeuge gefunden und benützt werden. Die Betriebssystem- und Fensterumgebung fiel auf Unix und dem darauf aufsetzenden X Window System, aufgrund der weiten Verbreitung im universitären Umfeld. Mit dem Standardwerkzeug OSF/Motif (kurz Motif) für X und dem freien Tcl/Tk-Paket von John Ousterhout [Ous94] als User Interface Toolkits[1] standen schnell die beiden Kandidaten für die Endauswahl fest. Nachfolgende Tabelle versucht die beiden Toolkits kurz gegenüberzustellen und zu vergleichen.

[1]Zur allgemeinen Klassifikation und Beschreibung der Benutzerschnittstellenwerkzeuge siehe auch Abschnitt 3.6.2.

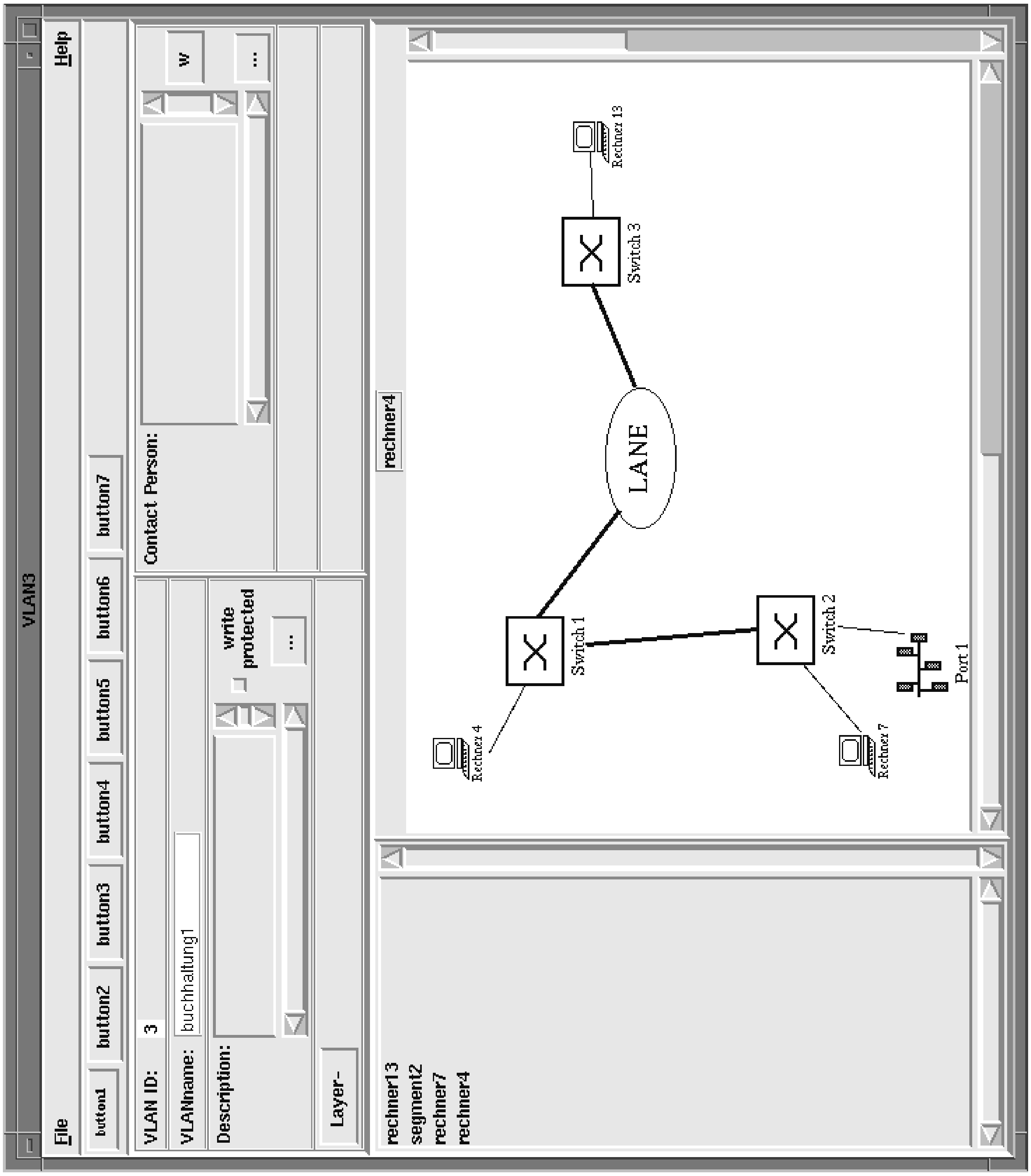


Abbildung 8.2: VLAN-spezifische-Sicht

Vergleich: OSF/Motif — Tk		
User Interface Toolkit	OSF/Motif	Tk
Fenstersysteme	X Window	X , MS Windows 3.1/95/NT, Mac
Programmiersprache, -umgebung	C	Tcl (Skriptensprache)
Programmkode wird	kompiliert	interpretiert
Verfügbarkeit	kommerziell	Public Domain
Einsatzbereich	Profibereich	Hobbybereich
UI Builder	ja, aber nur kommerziell	ja, aber unausgereift
Erlernbarkeit	aufwendig	leicht

Mit UIMX/Move stand ein UI-Builder für Motif zur Verfügung, so daß die aufwendige und schwer erlernbare Realisierung der **statischen** Bedienoberfläche mit Motif kein nachteiliges Argument mehr darstellt. Die Entscheidung trotzdem gegen C + Motif + UIMX/Move und zugunsten Tcl + Tk wird durch folgenden Überlegungen motiviert:

- Wenn die Lizenz für UIMX/Move abläuft, ist die Bedienoberfläche des Demonstrators nicht mehr pflegbar und erweiterbar, da ein manuelles Nachbearbeiten des durch den UI-Builder generierten Programmkodes praktisch nicht mehr möglich ist.

- Das interaktive zusammensetzen der Bedienoberfläche (Baukasten-Prinzip) mit dem UI Builder UIMX/Move ist nach kurzer Einarbeitung unübertroffen einfach. Ähnlich die Durchführung der Modifikationen der Widget-Attribute (Größe, Farbe, usw.). Was aber darüber hinausgeht, die ganze "Intelligenz" (Dialogkontrolle und Anwendungskern), muß in C per Hand kodiert werden.

  Die zusammen mit dem UI Toolkit Tk entwickelte Skriptensprache Tcl, erlaubt zwar kein so bequemes Zusammensetzen der statischen Bedienoberfläche, aber zusammen mit der Implementierung der geforderten Funktionalität, ist der Gesamtkodierungsaufwand unter dem Strich ungleich geringer, sowohl im Umfang des *Listings* als auch in der investierten Zeit.

- Zur Darstellung der physischen Sichten, des physischen Netzes[2], d.h. zum Zeichnen innerhalb einer vom Window Manager zur Verfügung gestellten "leeren" Fläche *(Canvas)*, sind in Tcl/Tk — im Gegensatz zu Motif — einfache Zeichenfunktionen (Linien, Kreise, usw.) vorhanden. Ebenso einfach ist es (selbstgezeichnete) Bitmaps von Rechnern, Switching-Komponenten, Hintergrundplänen, usw. darin einzubinden. Jedes Element innerhalb der Zeichen-

[2]Es handelt sich dabei um einen Graphen; die physischen Komponenten stellen die Knoten und die verbindenden Linien die Leitungen dar.

fläche kann mit einfachen Befehlen auf Ereignisse *(Events)* reagieren. Die Reaktion auf Events wird mit Prozeduren in Tcl abgearbeitet.

- Der *Sourcecode*[3] des Demonstrators in der Skriptensprache Tcl/Tk ist auf den in der Tabelle genannten Fenstersystemen ohne Anpassungen[4] ablaufbar, sofern ein Tcl/Tk-Interpretier installiert ist.

Fazit:
Im Vergleich mit Motif und C müßte für das gleiche Ergebnis ein sehr viel höherer Aufwand (um einige Größenordnungen) betrieben werden. Für die Zielsetzung dieser Arbeit (Schwerpunkt: theoretischer Teil, optional: praktischer Teil) und für die Realisierung der Vorgaben (Abschnitt 8.1) fällt die Entscheidung zugunsten von Tcl/Tk als Werkzeug für die Implementierung des Demonstrators aus.

8.3 Übersicht der prototypischen Implementierung

Nachdem die Anforderungen an den Demonstrator in Abschnitt 8.1 beschrieben und in Abschnitt 8.2 mit Tcl/Tk ein Werkzeug für dessen Realisierung festgelegt wurde, dient dieser Abschnitt der groben Beschreibung der Struktur des im Anhang abgedruckten Quellkodes. Zusammen mit den bereits darin eingebauten Erläuterungen und Kommentaren, wird versucht eine Dokumentation zu liefern, die es erleichtern soll, den Demonstrator respektive den Quellkode zu verstehen und ihn z.B. im Rahmen eines Fortgeschrittenenpraktikums (Fopra), für das Rechnernetzepraktikum oder anderen Demonstrationszwecken, auszubauen.

Der Demonstrator besteht aus zwei Fenstern. Das Hauptfenster (`VLAN-root`) und den VLAN-spezifischen-Fenstern (`ShowWindow.top0`). Letzteres wird für jedes vorkonfigurierte VLAN beim Anklicken einer VLAN-Ikone in der logischen Sicht instantiiert, genauso wird beim Drücken des *Create-VLAN*-Buttons in der Hauptbuttonleiste eine "leere" Instanz des Fensters bzw. der Sicht angeboten.

Die Realisierung der statischen Oberfläche (Fensterrahmen, Buttons, Menüleisten, Scrollbar, usw.) nimmt — wie erwartet — viel Platz in Anspruch. Diese Bereiche sind durch Kommentare ausreichend gekennzeichnet und, da sie selbstsprechend sind, wird darauf nicht näher eingegangen.

Der Übergang vom Prototypen zum Demonstrator kommt durch die implementierte Funktionalität, die hier im wesentlichen in den logischen und physischen Sichten in beiden Fenstern realisiert ist, zum Ausdruck.

Die logischen und physischen Sichten im Hauptfenster werden ab dem Kommentar "`Filebrowser + Canvas`" spezifiziert. Die logische Sicht besteht aus zwei *Listboxes*.

[3] ASCII-Text.
[4] Solange keine (Betriebs-)Systemaufrufe etc. notwendig sind.

In der Linken werden — je nach Modus — die vorhandenen VLANs, Hersteller-VLANs, Hersteller-VLAN-Manager-Domänen und die VLAN-Management-Area-Sichten, repräsentiert durch Ikonen, aufgelistet, in der Rechten, nach Anklicken eines solchen Listenelements (=Ikone), deren Mitglieder und Komponenten. Gleichzeitig werden sie in der physischen Sicht hervorgehoben. Die physische Sicht besteht aus einer "leeren" Zeichenfläche (in Tcl/TK: Canvas), in die, ab dem Kommentar "`Canvas (phys. Sicht)`", die statische Anordnung (Größe, Scrollbars, usw.) und anschließend der Graph respektive das physische Netz aufgebaut wird. Ab dem Kommentar "`Standard-Bindings für phys. Sicht`" wird die Grundreaktion (Eventbehandlung) der einzelnen Komponenten festgelegt. Befindet sich der Mauszeiger über einer Komponente, so wird diese blau hervorgehoben und gleichzeitig der Name eingeblendet. Das "Verlassen" einer Komponente (mit dem Mauszeiger) bewirkt, daß sie wieder schwarz (Standardeinstellung) eingefärbt wird.

Ein doppeltes Anklicken eines Listenelements im VLAN-Modus, wird die VLAN-spezifische-Sicht öffnen, dadurch daß die Tcl-Prozedur `ShowWindow.top0` abgearbeitet wird. In dieser Prozedur sind alle Vereinbarungen und Funktionalitäten für die VLAN-spezifische-Sicht zusammengefaßt. Die grobe Struktur dieser Prozedur ist ähnlich der des Hauptfensters. Eine dennoch erwähnenswerte Besonderheit ergibt sich aus der Tatsache, daß gleichzeitig mehrere Instanzen der VLAN-spezifische-Sicht dargestellt werden können. Die dazu notwendigen Vorkehrungen werden am Anfang der Prozedur getroffen.

Ab dem Kommentar "`Hilfssichten (Radiobuttons)`" werden die Hilfssichten im Hauptfenster realisiert. Durch Anklicken eines Radiobuttons (im Hauptfenster) wird der Modus gewechselt, die Prozeduren die dabei ausgeführt werden[5], sind mit der `-command` Option als Parameter für jeden Radiobutton angegeben. Anhängig vom selektierten Radiobutton, wird die logische Sicht, wie bereits erwähnt, mit Ikonen (Listenelementen), die vorhandenen VLANs, Hersteller-VLANs, Hersteller-VLANs-Manager-Domänen oder VLAN-Management-Areas repräsentieren, in der linken Listbox aufgefüllt. Die rechte Listbox bleibt leer, wenn mehrere Listenelemente vorhanden sind, ist nur eines vorhanden werden die Bestandteile gezeigt. Wird ein Listenelement angeklickt, tritt die Eventbearbeitung, die ab `''Bindings Hilfssichten''` spezifiziert ist, in Aktion. Die wesentliche Funktionalität die dadurch ausgelöst wird, ist ab dem Kommentar "`Funktionen: Hilfssichten`" wieder durch Prozeduren realisiert. Diese Prozeduren listen die einzelnen Bestandteile des selektierten Listenelements in der rechten Listbox auf, gleichzeitig werden alle Hervorhebungen in der physischen Sicht gelöscht, d.h. die physische Sicht wird einem *Refresh* unterzogen, und danach die aufgelisteten Bestandteile hervorgehoben.

Den Abschluß des Quellkodes bilden zwei Prozeduren die eine Dialogbox (`YesNoBox`) und eine Textbox (`TextBox`) realisieren. Beide öffnen ein extra Fenster. Die Dialogbox gibt eine Frage aus, die mit Ja oder Nein beantwortet werden muß. Der Text

[5]Prozeduren können wieder Prozeduren aufrufen.

wird der Prozedur als Paramenter übergeben. Abhängig von der Antwort, werden unterschiedliche Aktionen ausgeführt. Die Textbox gibt einen Text, der, wie bei der Dialogbox, als Paramenter mit übergeben wird, aus.

Kapitel 9

Zusammenfassung und Ausblick

Die vorliegende Arbeit hat versucht Vorschläge für eine graphische Bedienoberfläche für das VLAN-Management zu liefern, und zwar für Szenarien die heute oder in naher Zukunft schon realisierbar sind. Die in dieser Arbeit gelieferten Vorschläge können als Ausgangsbasis für Verfeinerungen und als Grundstock für Erweiterungen herangezogen werden. Mit dem Demonstrator wurde versucht einige der wesentlichen Ideen daraus skizzenhaft anzudeuten.

Während der Abfassung der Diplomarbeit konnten viele Aspekte aus dem Bereich der VLANs, aufgrund der inhärenten Dynamik, nicht berücksichtigt werden. Von den vielen bereits angesprochenen Fragestellungen, die VLANs mit sich bringen, wurde versucht für einige Lösungsvorschläge (bzgl. der graphischen Darstellung) zu liefern und andere wurden übergangen, da sie im allgemeinen auf fehlenden Standards und der Inkompatibilität zueinander beruhen und im Laufe der "Evolution" mit großer Wahrscheinlichkeit gelöst werden, so daß sie für diese Arbeit keine richtigen VLAN-spezifischen Fragestellungen darstellen.

In dieser Arbeit war jede Endstation höchstens einem VLAN zugeordnet *(Single Membership)*. In naher Zukunft wird es aber erforderlich sein, daß eine Endstationen mehreren VLANs angehört *(Multiple Membership)*. Im Idealfall soll der Endbenutzer selbst entscheiden können welchen VLANs — auch mehreren gleichzeitig — er in einer Sitzung angehören will oder zumindestens bei Sitzungsbeginn die Möglichkeit hat, einem VLAN seiner Wahl beizutreten. In jedem der Fälle sind aber Modifikationen der Betriebssysteme und der Hardware erforderlich. Die in dieser Arbeit gemachten Vorschläge sind, wie bereits im Text an entsprechenden Stellen bereits angedeutet, in dieser Hinsicht einfach erweiterbar.

Durch die Einbeziehung der ATM LAN Emulation wurde versucht der vielzitierte Verbindung von VLANs mit der ATM-Technologie gerecht zu werden. Zur Vervollständigung sei dennoch erwähnt, daß neben der betrachteten LAN Emulation auch die Ansätze der IETF mit "Classical IP over ATM" [cla94], die des ATM Forums mit

"Multi Protocol over ATM (MPOA)" [mpo93] und natürlich *native* ATM, mit seinen virtuellen Kanälen und Pfaden geradezu prädestiniert, zur Bildung von VLANs eine wichtige Rolle spielen werden. All diesen Ansätzen ist gemeinsam, daß sie, wie viele weiterer offener Aspekte aus den Bereich des VLAN-Managements, an der Bedienoberfläche in irgendeiner Weise eingearbeitet und angeboten werden müssen.

Nach diesem Ausblick auf die Aspekte, die in die Bedienoberfläche noch einfließen können, werden im folgenden nun die in dieser Arbeit gemachten Vorschläge und der Demonstrator als Ausgangsbasis für weiterführende Tätigkeiten und Berachtungen herangezogen.

Durch die Vermeidung von Ausnahmebehandlungen, die sich aus der Heterogenität heutiger Produkte und fehlender Standards ergeben, und die Betrachtung der häufigsten Tätigkeiten eines Netzadministrator bezüglich VLANs, bliebt der Schwerpunkt der Betrachtungen in dieser Arbeit bei den VLAN-relevanten Fragestellungen und dadurch wurde, wie bereits erwähnt, eine solide Ausgangsbasis für Verfeinerungen[1] und Erweiterungen geliefert.

Der Demonstrator könnte dahingehend erweitert werden, daß er zu Lehrzwecken, z.B. in dem Rechnernetzepraktikum am Lehrstuhl, oder zu anderen Demonstrationszwecken Verwendung findet.

Zwar widerspricht eine reine Softwarelösung — wie es der Demonstrator, aufgrund fehlender Hardwarekomponenten, ist und nur sein kann — der Philosophie eines Praktikums, aber dennoch sei an dieser Stelle nocheinmal seine Ausbaufähigkeit, genauso wie die Ausbaufähigkeit aller in dieser Arbeit gemachten Vorschläge, hervorgehoben.

[1]Z.B. für die ausgesparten Ausnahmebehandlungen.

Teil IV

Anhang

Anhang A

Listing

```
#!/usr/local/bin/wish -f
#
# Systemvoraussetzung: Tcl 7.5
#                      Tk  4.0
#
  tk_bisque
  wm positionfrom . user
  wm maxsize . 940  770
  wm minsize . 1 1
  wm title . {VLAN-root}

# build widget .frame4 Menueleiste
  frame .frame4 -borderwidth {2} -relief {groove}
  pack configure .frame4  -fill x -anchor n

# build widget .frame4.menubutton1
menubutton .frame4.menubutton1 -menu {.frame4.menubutton1.m} \
      -padx {4} -pady {3} -text {File} -underline {0}

# build widget .frame4.menubutton1.m
   menu .frame4.menubutton1.m -postcommand {} -tearoff {0}

# .frame4.menubutton1.m add command
  .frame4.menubutton1.m add command \
    -command { if {[ YesNoBox "Really Quit?"] == 1} {
                     destroy .
                   }
              } \
-label {Exit} -underline {0}

##########################################
# Diagnostic Menue
##########################################

menubutton .frame4.menubutton3 \
```

```
   -menu {.frame4.menubutton3.m} -padx {4} -pady {3} -text {Diagnostic} \
   -underline {0}

# build widget .frame4.menubutton3.m
   menu .frame4.menubutton3.m -postcommand {} -tearoff {0}

# .frame4.menubutton3.m add command
  .frame4.menubutton3.m add command -label {VLAN} -underline {0}
  .frame4.menubutton3.m add command -label {ATM LANE} -underline {0}
 .frame4.menubutton3.m add command -label {...} -underline {0}

###############################################################
# Statistics Menue
###############################################################

# build widget .frame4.menubutton4
  menubutton .frame4.menubutton4 \
      -menu {.frame4.menubutton4.m} \
      -padx {4} -pady {3} -text {Statistics} -underline {0}

# build widget .frame4.menubutton4.m
   menu .frame4.menubutton4.m -postcommand {} -tearoff {0}

# .frame4.menubutton4.m add command
  .frame4.menubutton4.m add command -label {Inter VLAN} -underline {0}
  .frame4.menubutton4.m add command -label {Intra VLAN} -underline {0}
  .frame4.menubutton4.m add command -label {Backbone} -underline {0}
  .frame4.menubutton4.m add command -label {...} -underline {0}
#################################################################

menubutton .frame4.menubutton2 \
    -menu {.frame4.menubutton2.m} \
    -padx {4} -pady {3} -text {Help} -underline {0}

# build widget .frame4.menubutton2.m
  menu .frame4.menubutton2.m

# pack master .frame4
  pack configure .frame4.menubutton1  -side left
  pack configure .frame4.menubutton3  -side left
  pack configure .frame4.menubutton4  -side left
  pack configure .frame4.menubutton2   -side right

tk_menuBar .frame4 .frame4.menubutton1 .frame4.menubutton2

###############################################################
### end menuleiste
###############################################################

###############################################################
### Button Leiste
```

```
###############################################################

# build widget .frame6
  frame .frame6 -borderwidth {2} -height {30} -relief {groove} -width {30}

# pack .frame6
  pack configure .frame6 -expand 1 -fill both  -anchor n

   button .frame6.button4 \
      -command {ShowWindow.top0} \
      -padx {9} -pady {3} -text "Create\nVLAN" -font  *times-bold-r-normal*10*

# build widget .frame6.button5
  button .frame6.button5 -padx {9} -pady {3} -text {button5}

# build widget .frame6.button6
  button .frame6.button6 -padx {9} -pady {3} -text {button6}

# build widget .frame6.button7
  button .frame6.button7 -padx {9} -pady {3} -text {button7}

  image create photo ATMLANE -format gif -file ATMLANE.gif
# build widget .frame6.button8
  button .frame6.button8 -padx {9} -pady {3} -image ATMLANE

  image create photo Statistics -format gif -file Statistics.gif
# build widget .frame6.button9
  button .frame6.button9 -padx {9} -pady {3} -image Statistics

image create photo RotKreuz -format gif -file RotKreuz.gif
# build widget .frame6.button10
  button .frame6.button10 -padx {9} -pady {3} -image RotKreuz

######################################################################
#  pack master .frame6 menubar
######################################################################

    pack configure .frame6.button4  -anchor nw -side left
    pack configure .frame6.button5  -anchor nw -side left
    pack configure .frame6.button6  -anchor nw -side left
    pack configure .frame6.button7  -anchor nw -side left
    pack configure .frame6.button8  -anchor nw -side right
    pack configure .frame6.button9  -anchor nw -side right
    pack configure .frame6.button10 -anchor nw -side right

#########################################################
## end  button Leiste
########################################################

########################################################
### Logische und physische Sichte
```

```
######################################################

# build widget .frame1
  frame .frame1 -borderwidth {2} -height {30} -relief {groove} -width {30}

# pack filebr + canvas
  pack configure .frame1  -expand 1  -fill both -anchor n

# build widget .frame1.frame2
  frame .frame1.frame2 -borderwidth {3} -height {30} -relief {groove}\
             -width {30}

# build widget .frame1.frame3
  frame .frame1.frame3 -borderwidth {3} -height {30} -relief {groove} \
    -width {30}

# pack master .frame1
   pack configure .frame1.frame2 -expand 1 -fill both -side left

  pack configure .frame1.frame3 -expand 1 -fill both -side right

#built widget .frame5
   frame .frame5 -borderwidth {2} -relief {groove}

# pack .frame5
  pack configure .frame5 -expand 1 -fill both -anchor n

  button .frame5.button1
  pack .frame5.button1 -side left

#####################################################
## Canvas (phys. Sicht)
#####################################################

  set can0 .frame1.frame3
  global can0

# build widget .frame0.scrollbar3
  scrollbar $can0.scrollbar3 \
    -command {$can0.canvas2 xview} \
    -orient {horizontal} \
    -relief {groove}

# build widget $canv0.scrollbar1
  scrollbar $can0.scrollbar1 \
    -command {$can0.canvas2 yview} \
    -relief {raised}

# build widget $can0.canvas2
   canvas $can0.canvas2 \
     -confine 1 \
```

```
     -height {400} \
     -borderwidth {2} \
     -relief {raised} \
     -scrollregion {0c 0c 30c 30c} \
     -width {500} \
     -xscrollcommand {$can0.scrollbar3 set} \
     -yscrollcommand {$can0.scrollbar1 set} \
     -background white

    frame $can0.frame0 \
     -bd 2\
     -relief raised \
     -height {30}\
     -width {30}

label $can0.frame0.label0 \
     -textvariable labelcanvas \
     -justify left     \
     -bd 2 \
     -relief {groove}

pack configure $can0.frame0.label0 -side left

pack append $can0 \
    $can0.frame0  {bottom frame n expand fill} \
    $can0.scrollbar1   {right frame center filly} \
    $can0.canvas2   {top frame center expand fill} \
    $can0.scrollbar3   {top frame center fillx}

#######################################################################

set top $can0.canvas2 ;# physische Gesamtsicht

# Variablen f"ur Komponenten, Leitungen, ...

set buildswitch1  {create bitmap 135 100 \
                    -bitmap @Switches/Switch1.xbm -tags switch1 }
set buildrechner4 { create bitmap  60 40 \
                    -bitmap @Rechner/Rechner4.xbm -tags rechner4 }
set buildrechner2 { create bitmap 120 24  \
                    -bitmap @Rechner/Rechner2.xbm -tags rechner2 }
set buildrechner3 { create bitmap 45 95 \
                     -bitmap @Rechner/Rechner3.xbm  -tags rechner3 }
set buildrechner1 { create bitmap 60 150 \
                    -bitmap @Rechner/Rechner1.xbm -tags { rechner1 rechner }}
set buildsegment1 {  create bitmap  190 50\
                     -bitmap @Segmente/Segment4.xbm -tags segment1  }
set buildline2    { create line 121 46 130 74 -tags line2 }
set buildline4    { create line 59 61 108 87 -tags line4  }
set buildline3    { create line  56 95 108 93 -tags line3  }
set buildline1    { create line 72 140 108 100  -tags line1 }
```

```
set buildsegline1 { create line 178 50 141 74   -tags segline1 }
set buildlane1    { create line 156 96 232 154  -width 3  -tags lane1 }
set buildswitch2  { create bitmap  150 279 \
                    -bitmap @Switches/Switch2.xbm -tags {switch2 switch}}
set buildrechner5 { create bitmap  69 217 -bitmap @Rechner/Rechner5.xbm -tags rechner5 }
set buildrechner6 { create bitmap  50 270  -bitmap @Rechner/Rechner6.xbm \
                   -tags rechner6 }
set buildrechner7 { create bitmap 62 330  \
                    -bitmap @Rechner/Rechner7.xbm  -tags rechner7 }
set buildrechner8 { create bitmap 243 245  \
                    -bitmap @Rechner/Rechner8.xbm  -tags rechner8 }
set buildrechner9 { create bitmap 233 321  \
                    -bitmap @Rechner/Rechner9.xbm  -tags rechner9 }
set buildsegment2 {  create bitmap  120 380 \
                    -bitmap @Segmente/Segment1.xbm -tags segment2 }
set buildbackbone1 { create line 136 130 147 253   \
                    -width 3 -tags backbone1 }
set buildline5     { create line 73 235 123 263  -tags line5  }
set buildline6     { create line 61 269 123  274  -tags line6  }
set buildline7     { create line 74 326 123 284  -tags line7  }
set buildline8     { create line 171 266 230 245 -tags line8 }
set buildline9     { create line 172 277 222 317 -tags line9  }
set buildsegline2  { create line 149 309 144 357  -tags segline2 }
set buildsegment3  { create bitmap 299 99  \
                    -bitmap @Segmente/SegmentED.xbm  -tags segment3 }
set buildsegline3  { create line 294 111 279 145   -tags segline3 }
set buildrechner10 { create bitmap 310 240 \
                     -bitmap @Rechner/Rechner10.xbm  -tags rechner10 }
set buildrechner11 { create bitmap 440  67  \
                     -bitmap @Rechner/Rechner11.xbm  -tags rechner11 }
set buildrechner12 { create bitmap 440 221 \
                     -bitmap @Rechner/Rechner12.xbm  -tags rechner12 }
set buildrechner13 { create bitmap 500 140 \
                     -bitmap @Rechner/Rechner13.xbm  -tags rechner13 }
set buildswitch3   { create bitmap  420 140 \
                     -bitmap @Switches/Switch3.xbm -tags switch3 }
set buildline10    { create line 300 190 309 222 -tags line10  }
set buildline11    { create line 433 86 416 113   -tags line11 }
set buildline12    { create line 426 171 438 201 -tags line12 }
set buildline13    { create line  442 135 488 133 -tags line13 }
set buildlane2     { create line  320 165 394 134  -width 3  -tags lane2 }
set buildlaneicon  { create bitmap  270 170 \
                     -bitmap @LANE-Icon.xbm  -tags laneicon}

bind $top  <Motion> { puts " %x %y"} ;# gibt Koordinaten in Standardausgabe aus

### Zusammenfassen der Komponenten und Linien;
global ALLKomp ALLLine

set ALLKomp  {rechner1 rechner2 rechner3 rechner4 \
          rechner5 rechner6 \
```

```
          rechner7 rechner8 rechner9 rechner10 \
          rechner11 rechner12 \
          rechner13 switch1 switch2 switch3 \
          segment1 segment2 segment3 \
          laneicon }

set ALLLine {line1 line2 line3 line4 \
          line5 line6 line7 line8 \
          line9 line10 line11 line12 line13 \
          lane1 lane2  \
          segline1 segline2 segline3 \
           backbone1 }

###################################################
### physische Gesamtsicht zeichnen
###
### Warum geht das nicht ?
###
###  foreach i  $ALL   {
###       eval
###       }

eval $top  $buildrechner1 ; eval $top  $buildrechner2
eval $top  $buildrechner3 ; eval $top  $buildrechner4
eval $top  $buildrechner5 ; eval $top  $buildrechner6
eval $top  $buildrechner7 ; eval $top  $buildrechner8
eval $top  $buildrechner9 ;  eval $top  $buildrechner10
eval $top  $buildrechner11 ;  eval $top  $buildrechner12
eval $top  $buildrechner13 ;  eval $top  $buildline1
eval $top  $buildline2   ;  eval $top  $buildline3
eval $top  $buildline4  ;   eval $top  $buildline5
eval $top  $buildline6  ; eval $top  $buildline7
eval $top  $buildline8  ; eval $top  $buildline9
eval $top  $buildline10 ; eval $top  $buildline11
eval $top  $buildline12  ; eval $top  $buildline13
eval $top  $buildsegment1 ;  eval $top  $buildsegment2
eval $top  $buildsegment3 ; eval $top  $buildlaneicon
eval $top  $buildbackbone1 ; eval $top  $buildlane1
eval $top  $buildlane2  ;
eval $top  $buildswitch1  ; eval $top $buildswitch2
eval $top  $buildswitch3
 eval $top  $buildsegline1 ; eval $top $buildsegline2
 eval $top  $buildsegline3

################################################################
### Standard-Bindings  f"ur phys. Sicht
################################################################

proc BindTags {} {
global top Selected
global ALLKomp ALLLine
```

```
foreach i  $ALLKomp  {
             $top bind $i <Enter> {
               $top itemconfigure current -foreground blue
                set j [$top gettags current]
                set labelcanvas [lindex $j 0]
                 }
             }

foreach i  $ALLLine  {
             $top bind $i <Enter> {
               $top itemconfigure current -fill blue
                set j [$top gettags current]
                set labelcanvas [lindex $j 0]
                }
             }

foreach i  $ALLKomp {
          $top bind "$i" <Leave> {
              parray Selected *
              set j [$top gettags current]
             puts "Selected([lindex $j 0])"
     #      if {$Selected([lindex $j 0]) == 0} {
             $top itemconfigure current  -foreground black ;#}
              }
            }

foreach i  $ALLLine  {
             $top bind $i <Leave> {
                 set j [$top gettags current]
                 if { $Selected([lindex $j 0]) == 0} {
                        $top itemconfigure current -fill black
                        }
                   }

                  }
}

BindTags  ;# start bindings

#####################################################################
### Logische Sicht, incl. Hilfssichten (Radiobuttons)
#####################################################################

 frame .frame1.frame2.frame0 -borderwidth {2} -relief {raised}
 frame .frame1.frame2.frame0.frame0 -borderwidth {2} -relief {raised}
 frame .frame1.frame2.frame0.frame1 -borderwidth {2} -relief {raised}
 pack .frame1.frame2.frame0.frame0  -expand 1 -side left -fill both
 pack .frame1.frame2.frame0.frame1 -expand 1 -side left -fill both

 global fil
 set fil .frame1.frame2 ;# Abk"urzung definieren
```

```
 ### Radiobuttons

 radiobutton $fil.frame0.frame0.radio1 \
                       -font {*times-bold-r-normal*10*} \
                       -text "VLAN-View" \
                       -variable {Viewmodus} \
                       -value Default \
                       -command { set Viewmodus Default
                                  ListDefaultModus }

 radiobutton $fil.frame0.frame0.radio2 \
                       -font  *times-bold-r-normal*10* \
                       -text "VLAN -> Sub-VLAN" \
                       -variable Viewmodus \
                       -value  VLAN->Sub-VLAN \
                       -command { TextBox "(VLAN-->SubVLAN)-Sicht noch \
                                   nicht implementiert"
                                   $fil.frame0.frame0.radio1 invoke
                                 }

radiobutton $fil.frame0.frame0.radio3 \
                        -font  *times-bold-r-normal*10*  \
                        -text "All -->Vendor-VLAN"  \
                        -variable Viewmodus \
                        -value  ALL->Vendor-VLAN \
                        -command {  set Viewmodus ALL->Vendor-VLAN
                                     ListSubVLANs
                                   }

radiobutton $fil.frame0.frame0.radio4  -font  *times-bold-r-normal*10*  \
                        -text "Vendor-VLAN-Mgr.-Domain" \
                        -variable Viewmodus \
                        -value Vendor-VLAN-Mgr \
                        -command { set Viewmodus Vendor-VLAN-Mgr
                                   ListMgr
                                 }

radiobutton $fil.frame0.frame0.radio5  -font  *times-bold-r-normal*10*  \
                         -text "VLAN-Management-View" \
                         -variable Viewmodus   \
                         -value VLAN-Domain-View \
                         -command { set Viewmodus VLAN-Domain-View
                                     ListDom
                                   }

set rad $fil.frame0.frame0.radio

# pack radiobuttons
  foreach  i { 1 2 3 4 5 } {
  pack $rad$i  -side top -anchor w
```

```
  }

#######################
### Frames, Listboxen, Scrollbars etc.
#####################

# build widget $fil.frame1
  frame $fil.frame1  -relief {raised}

# build widget $fil.frame1.label4
  label $fil.frame1.label4 \
    -anchor {w} -padx {2} -relief {raised} \
    -text {/home/garfield/development/xf/demos}

# build widget $fil.frame1.label5
  label $fil.frame1.label5 -padx {2} -relief {raised} -text {}

# pack widget $fil.frame1
  pack append $fil.frame1 \
    $fil.frame1.label4 {left frame center expand fillx} \
    $fil.frame1.label5 {right frame center fillx}

# build widget $fil.frame2
  frame $fil.frame2 -relief {raised}

# build widget $fil.frame2.frame
  frame $fil.frame2.frame

# build widget $fil.frame2.frame.scrollbar2
  scrollbar $fil.frame2.frame.scrollbar2 \
    -command {$fil.frame2.frame.listbox1 yview} \
    -relief {raised}

# build widget $fil.frame2.frame.scrollbar3
  scrollbar $fil.frame2.frame.scrollbar3 \
    -command {$fil.frame2.frame.listbox1 xview} \
    -orient {horizontal} \
   -relief {raised}

# build widget $fil.frame2.frame.listbox1
  listbox $fil.frame2.frame.listbox1 \
    -relief {raised} \
    -xscrollcommand {$fil.frame2.frame.scrollbar3 set} \
    -yscrollcommand {$fil.frame2.frame.scrollbar2 set}

# pack widget $fil.frame2.frame
  pack append $fil.frame2.frame \
    $fil.frame2.frame.scrollbar2 {left frame center filly} \
    $fil.frame2.frame.listbox1 {top frame center expand fill} \
    $fil.frame2.frame.scrollbar3 {bottom frame center fillx}
```

```
# build widget $fil.frame2.frame6
  frame $fil.frame2.frame6

# build widget $fil.frame2.frame6.scrollbar2
  scrollbar $fil.frame2.frame6.scrollbar2 \
    -command {$fil.frame2.frame6.listbox1 yview} \
    -relief {raised}

# build widget $fil.frame2.frame6.scrollbar3
  scrollbar $fil.frame2.frame6.scrollbar3 \
    -command {$fil.frame2.frame6.listbox1 xview} \
    -orient {horizontal} \
    -relief {raised}

# build widget $fil.frame2.frame6.listbox1
  listbox $fil.frame2.frame6.listbox1 \
     -relief {raised} \
    -xscrollcommand {$fil.frame2.frame6.scrollbar3 set} \
    -yscrollcommand {$fil.frame2.frame6.scrollbar2 set}

# pack widget $fil.frame2.frame6
  pack append $fil.frame2.frame6 \
   $fil.frame2.frame6.scrollbar2 {right frame center filly} \
    $fil.frame2.frame6.listbox1 {top frame center expand fill} \
    $fil.frame2.frame6.scrollbar3 {bottom frame center fillx}

# pack widget $fil.frame2
  pack append $fil.frame2 \
    $fil.frame2.frame {left frame center filly} \
    $fil.frame2.frame6 {left frame center expand fill}

# pack widget .frame1.fram2
  pack append .frame1.frame2 \
    $fil.frame0 {top frame center fillx} \
    $fil.frame1 {top frame center fillx} \
    $fil.frame2 {top frame center expand fill}

################################
### Variablen, Arrays  und Abk"urzungen
#################################

  global {curDir}
  set {curDir} {/Tk}
  global can0
  global {symbolicName}
  set {symbolicName(contentsList)} \
      {.frame1.frame2.frame2.frame6.listbox1}
  set {symbolicName(dirList)} {.frame1.frame2.frame2.frame.listbox1}
  set {symbolicName(dirName)} {.frame1.frame2.frame1.label4}
  set {symbolicName(fileName)} {.frame1.frame2.frame1.label5}
  set {symbolicName(root)} {.}
```

```
##########################################################################
### Hilfsvariablen f"ur vorkonfigurierte VLANs, Dom"anen, ...;
### logisch und physisch
##########################################################################

global VLAN1 VLAN2 VLAN3 VLAN3 VLAN4
set VLAN1 { rechner1 rechner2 rechner6 segment1 }
set VLAN2 { rechner12 rechner3 rechner9 }
set VLAN3 { rechner13 segment2  rechner7 rechner4 }
set VLAN4 { rechner5 rechner8 rechner10 segment3 }

set VendorVLAN1 { rechner1 rechner2 rechner3 rechner4 switch1 segment1}
set VendorVLAN1Lin { line1 line2 line3 line6 segline1 lane1  }
set VendorVLAN2 { rechner5 rechner6 rechner7 segment2 rechner8 rechner9 \
                     switch2 }
set VendorVLAN2Lin  { line5 line6 line7 line8 line9 segline2 }
set VendorVLAN3 { rechner11 rechner12 rechner13 switch3 }
set VendorVLAN3Lin   { line11 line12 line13 }
set VendorVLANs { VendorVLAN1 VendorVLAN2 VendorVLAN3 }

set Manager { Manager1 Manager2}
set Manager1 { rechner1 rechner3 rechner4 rechner2 segment1 switch1 \
                rechner11 rechner12 rechner13 switch3 }
set Manager1Lin { line1 line2 line3 line6 segline1 \
                     line5 line6 line7 line8 line9 segline2  }
set Manager2 { rechner5 rechner6 rechner7 segment2 rechner8 rechner9 \
                     switch2    }
set Manager2Lin { line5 line6 line7 line8 line9 segline2 }

set Domain  { Domain1 }
set Domain1 { rechner1 rechner2 rechner3 rechner4 \
              rechner5 rechner6 \
              rechner7 rechner8 rechner9 rechner10 \
              rechner11 rechner12 \
              rechner13 switch1 switch2 switch3 \
              segment1 segment2 segment3 \
              laneicon  }
set DomLin  { line1 line2 line3 line4 \
              line5 line6 line7 line8 \
              line9 line10 line11 line12 line13 \
              lane1 lane2  \
              segline1 segline2 segline3 \
              backbone1 }

#####################################################################
### Selected: global array: if
#####################################################################
global Selected
foreach i  $ALLKomp  {
      set Selected($i) 0
```

```
  }
foreach i  $ALLLine  {
      set Selected($i) 0
  }

######################################################################
### Hersteller-VLANs
######################################################################
proc markSubVLAN { fname } {
global  symbolicName VendorVLAN1 VendorVLAN2 VendorVLAN3
global   VendorVLAN1Lin VendorVLAN2Lin VendorVLAN3Lin Selected top
ResetCanvas
foreach i $fname {
        $top itemconfigure $i -background yellow
         }
}


proc ListSubVLANs {} {
global VendorVLANs symbolicName
$symbolicName(dirList)          delete 0 end
$symbolicName(contentsList) delete 0 end
$symbolicName(dirName)  configure -text "SubVLANs"
$symbolicName(fileName) configure -text ""
ResetCanvas
   foreach i { VendorVLAN1 VendorVLAN2 VendorVLAN3 } {
      $symbolicName(dirList) insert end $i
      }
}

proc ListMemberSubVLAN { fname } {
global  symbolicName VendorVLAN1 VendorVLAN2 VendorVLAN3
global       VendorVLAN1Lin VendorVLAN2Lin VendorVLAN3Lin Selected
$symbolicName(contentsList) delete 0 end
foreach i $fname {
        $symbolicName(contentsList) insert end $i
        $symbolicName(fileName) configure -text "Components"
        }
markSubVLAN $fname
}

########################################################################
### List Vendor-VLAN-Manager
########################################################################

proc ListMgr {} {
global  Manager symbolicName
$symbolicName(dirList)          delete 0 end
$symbolicName(contentsList) delete 0 end
$symbolicName(dirName)  configure -text "Vendor-VLAN-Manager-Domains"
$symbolicName(fileName) configure -text ""
```

```
ResetCanvas
   foreach i { Manager1 Manager2 } {
      $symbolicName(dirList) insert end $i
      }
}

proc ListMemberHersVLAN { fname } {
global  symbolicName Manager Manager1 Manager1Lin Manager2 Manager2Lin Selected
$symbolicName(contentsList) delete 0 end
foreach i $fname {
        $symbolicName(contentsList) insert end $i
        $symbolicName(fileName) configure -text "Components"
        }
MarkHerstMgr $fname
}

############################################################################
###  Markiert Hersteller-VLAN-Mgr.-Dom. in phys. Sicht
############################################################################
proc MarkHerstMgr { fname } {
global  symbolicName Manager Manager1 Manager1Lin Manager2
global Manager2Lin Selected top
ResetCanvas
foreach i $fname {
        $top itemconfigure $i -background brown
         }
switch $fname {
      Manager1 { foreach i  $Manager1Lin  {
                         $top itemconfigure $i  -fill brown
                     }
               }
      Manager2 { foreach i  $Manager2Lin  {
                         $top itemconfigure $i  -fill brown
                         }             }
               }
}

############################################################################
### VLAN-Dom"ane
############################################################################
proc ListDom {} {
global Domains Domain1 symbolicName
$symbolicName(dirList)           delete 0 end
$symbolicName(contentsList) delete 0 end
$symbolicName(dirName)  configure -text "VLAN-Domains"
$symbolicName(fileName) configure -text ""
ResetCanvas
   foreach i { Area1 } {
      $symbolicName(dirList) insert end $i
      }
  ListMemberDom $Domain1
```

```
}

########################################################################
proc ListMemberDom { fname } {
global  symbolicName Domain Domain1 Selected
$symbolicName(contentsList) delete 0 end
foreach i $fname {
        $symbolicName(contentsList) insert end $i
        $symbolicName(fileName) configure -text "Components"
        }
MarkDom $fname
}
#######################################################################
proc MarkDom { fname } {
global  symbolicName Domain Domain1 DomLin Selected top
ResetCanvas
foreach i $fname {
        $top itemconfigure $i -background pink
         }

#foreach i  $DomLin  {
#           $top itemconfigure $i  -fill pink
#           }
}
#######################################################################
### Phys. Sicht-Refresh
#######################################################################
proc ResetCanvas { } {
global top ; global ALLKomp ; global ALLLine
global Selected

foreach i  $ALLKomp   {
           $top itemconfigure $i  -foreground black
           $top itemconfigure $i  -background white
           set Selected($i) 0
           }
foreach i  $ALLLine  {
           $top itemconfigure $i  -fill black
           set Selected($i) 0
           }

}
###########################################################################
### VLAN
###########################################################################
proc markVLAN  { fname  } {
global top
global VLAN1 VLAN2 VLAN3 VLAN4
global Selected
ResetCanvas
foreach i $fname {
```

```
        puts "$i"
        $top itemconfigure $i -background yellow
        set Selected($i) 1
         }
}

proc markLine { fname } {
global top  symbolicName Selected
global VLAN1 VLAN2 VLAN3 VLAN
global VendorVLAN1Lin VendorVLAN2Lin VendorVLAN3Lin
foreach i $fname {
       # puts "$i"
       # puts "$fname"
        $top itemconfigure $i -fill blue
        set Selected($i) 1
         }
}

proc ListDefaultModus {} {
global  symbolicName
$symbolicName(dirName)  configure -text VLANs
$symbolicName(dirList)          delete 0 end
$symbolicName(contentsList) delete 0 end
ResetCanvas
    foreach i { VLAN1 VLAN2 VLAN3 VLAN4 } {
      $symbolicName(dirList) insert end $i
      }
   }

proc ListMemberVLAN {  fname  } {
global  symbolicName
global VLAN1 ; global VLAN2 ; global VLAN3 ; global VLAN4
global Selected
$symbolicName(contentsList) delete 0 end
  foreach i $fname {
        $symbolicName(contentsList) insert end $i
        $symbolicName(fileName) configure -text "Members"
        }
markVLAN   $fname
}

$fil.frame0.frame0.radio1 invoke

##############################################################################

proc ReturnViewModus {} {
global Viewmodus
 switch $Viewmodus {
    Default          ListDefaultModus
    VLAN->Sub-VLAN    -
    ALL->Sub-VLAN     -
```

```
    Sub-VLAN-Mgr      -
    VLAN-Domain-View  -
    }
}

# Procedure: TkBroOpen
 proc TkBroOpen { fname} {
  global curDir can0 fil

  if {[file isdirectory $fname]} {
    cd $fname
    set curDir [pwd]
    TkBroRefresh
  } {
    set thisfile [open $fname r]
     [SymbolicName contentsList] delete 0 end
    [SymbolicName fileName] configure -text $fname
    for { set i 1 } { $i<50 } { incr i } {
      [SymbolicName contentsList] insert end [ gets $thisfile ]
    }
    update
    close $thisfile
  }
}


# Procedure: TkBroRefresh
proc TkBroRefresh {} {

global curDir can0 fil

set list [exec ls -a]
.frame1.frame2.frame2.frame.listbox1  delete 0 end
[SymbolicName contentsList]            delete 0 end
[SymbolicName dirName] configure -text $curDir
[SymbolicName fileName] configure -text {}
foreach i $list {
  [SymbolicName dirList] insert end $i
}
update
}

#########################################
proc FillListBox { fname } {
global curDir can0
global fil
[SymbolicName contentsList]            delete 0 end
   if {[file isdirectory $fname]} {
       set thisfile [open ${fname}.txt r]
       while {[gets $thisfile Zeile] >= 0} {
         [SymbolicName contentsList] insert end  $Zeile
```

```
         }
        close $thisfile
    } {
      TkBroOpen { $fname}
     }
}

##########################################################################
### Bindings: was passiert -abh"angig vom Modus - wenn in log. Sicht
### Ikone angeklickt wird
##########################################################################

bind $fil.frame2.frame.listbox1 <Button-1> {
   switch $Viewmodus {
      Default         { eval ListMemberVLAN $[%W get [%W nearest %y]] }
      VLAN->Sub-VLAN    -
      ALL->Vendor-VLAN  { eval ListMemberSubVLAN $[%W get [%W nearest %y]] }
      Vendor-VLAN-Mgr   { eval  ListMemberHersVLAN $[%W get [%W nearest %y]] }
      VLAN-Domain-View   { eval ListMemberDom $[%W get [%W nearest %y]] }
    }

 }
#######
### was passiert bei Doppelklick
########

bind $fil.frame2.frame.listbox1 <Double-1> {
     switch $Viewmodus {
        Default         { ShowWindow.top0 "[%W get [%W nearest %y]]" }
        VLAN->Sub-VLAN    -
        ALL->Vendor-VLAN     { TextBox "Elementmanager for     \
                           [%W get [%W nearest %y]] not implemented" }
        Vendor-VLAN-Mgr     { TextBox "Manager for     \
                           [%W get [%W nearest %y]] not implemented" }
        VLAN-Domain-View  -
    }

 }

###################################################################
### VLAN ID Sicht    (ShowWindow.top0)
###################################################################
global top0
set top0 ""
global ca
set ca ""

proc ShowWindow.top0 { {vlanid VLAN5 } } {
global top0
set top0 foo$vlanid
global ca
```

```
set ca .$top0.frame3.frame5
toplevel .$top0

# Window manager configurations
global tk_version
wm positionfrom .$top0 ""
wm sizefrom .$top0 ""
wm maxsize .$top0 1000 1000
wm minsize .$top0 10 10
wm title .$top0 "$vlanid"

# build widget .top0.frame0
frame .$top0.frame0  -borderwidth {2}  -relief {raised}

menubutton .$top0.frame0.menubutton1  \
      -menu .$top0.frame0.menubutton1.m \
      -padx {4} \
      -pady {3} \
      -text {File} \
      -underline {0}

# build widget .frame4.menubutton1.m
   menu .$top0.frame0.menubutton1.m  \
   -postcommand {} \
   -tearoff {0}

.$top0.frame0.menubutton1.m add command \
    -command { if {[ YesNoBox "Really Quit?"] == 1} {
                      destroy .$top0
                   }
              } \
    -label {Exit} \
    -underline {0}

# build widget .top0.frame0.menubutton2
  menubutton .$top0.frame0.menubutton2  \
        -menu .$top0.frame0.menubutton2.m  -padx {4}  -pady {3}  \
        -text {Help}  -underline {0}

# build widget .top0.frame0.menubutton2.m
  menu .$top0.frame0.menubutton2.m

# pack master .top0.frame0
  pack configure .$top0.frame0.menubutton1  -side left
  pack configure .$top0.frame0.menubutton2  -side right

tk_menuBar .$top0.frame0 .$top0.frame0.menubutton1 .$top0.frame0.menubutton2

#######################################################
### Button-leiste
#######################################################
```

```
# build widget .$top0.frame1
  frame .$top0.frame1  -borderwidth {2} -height {30} \
                       -relief {raised} -width {3}

# build widget .$top0.frame1.button1
  button .$top0.frame1.button1 -command {} -padx {9} \
      -pady {3} -text "button1" -font  *times-bold-r-normal*10*

# build widget .$top0.frame1.button2
  button .$top0.frame1.button2 -padx {9} -pady {3} -text {button2}

# build widget .$top0.frame1.button3
  button .$top0.frame1.button3 -padx {9} -pady {3} -text {button3}

# build widget .$top0.frame1.button4
  button .$top0.frame1.button4 -padx {9} -pady {3} -text {button4}

# build widget .$top0.frame1.button5
  button .$top0.frame1.button5 -padx {9} -pady {3} -text {button5}

# build widget .$top0.frame1.button6
  button .$top0.frame1.button6 -padx {9} -pady {3} -text {button6}

# build widget .$top0.frame1.button7
  button .$top0.frame1.button7 -padx {9} -pady {3} -text {button7}

#########################################################################
###  pack Button Leiste
#########################################################################

    pack configure .$top0.frame1.button1  -anchor nw -side left
    pack configure .$top0.frame1.button2  -anchor nw -side left
    pack configure .$top0.frame1.button3  -anchor nw -side left
    pack configure .$top0.frame1.button4  -anchor nw -side left
    pack configure .$top0.frame1.button5  -anchor nw -side left
    pack configure .$top0.frame1.button6  -anchor nw -side left
    pack configure .$top0.frame1.button7 -anchor nw -side left

##########################################################
## end  Button Leiste
#########################################################

# build widget .$top0.frame2
  frame .$top0.frame2 -borderwidth {2} -height {30}  \
                      -relief {raised} -width {30}

# build widget .$top0.frame2.frame0
  frame .$top0.frame2.frame0  -borderwidth {2} -height {30}  \
                              -relief {raised} -width {30}
```

```
# build widget .$top0.frame2.frame1
  frame .$top0.frame2.frame1 -borderwidth {2} -height {30}  \
                           -relief {raised} -width {30}

# build widget .top0.frame3
  frame .$top0.frame3  -borderwidth {2}  -height {30} \
                     -relief {raised} -width {30}

# built widget .top0.frame3.frame4
  frame .$top0.frame3.frame4  -borderwidth {2}  -relief {raised}

# built widget .top0.frame3.frame5
  frame .$top0.frame3.frame5  -borderwidth {2}  -relief {raised}

########################################################
### Group Parameter
########################################################

frame  .$top0.frame2.frame0.frame0  -borderwidth 2 -relief groove

label  .$top0.frame2.frame0.frame0.label0 -text "VLAN ID:     " -justify left

switch $vlanid {
        VLAN1 { set iD 1 }
        VLAN2 { set iD 2 }
        VLAN3 { set iD 3 }
        VLAN4 { set iD 4 }
        VLAN5 { set iD 5 }
    }

label  .$top0.frame2.frame0.frame0.label1 -text "$iD" -justify left -bg white

frame  .$top0.frame2.frame0.frame1 -borderwidth 2 -relief groove

label  .$top0.frame2.frame0.frame1.label2 -text "VLANname: " -justify left

entry  .$top0.frame2.frame0.frame1.entry0 -relief groove \
                              -textvariable vlanname -bd 2 -bg white

frame  .$top0.frame2.frame0.frame2 -borderwidth 2 -relief groove
frame  .$top0.frame2.frame0.frame2.frame0
frame  .$top0.frame2.frame0.frame2.frame1
frame  .$top0.frame2.frame0.frame2.frame2

pack configure .$top0.frame2.frame0.frame2.frame0 \
               .$top0.frame2.frame0.frame2.frame1 \
               .$top0.frame2.frame0.frame2.frame2 \
                  -expand 1 \
                  -fill both \
                  -side left \
                  -anchor nw
```

```
label  .$top0.frame2.frame0.frame2.frame0.label3 \
       -text "Description:"

pack configure  .$top0.frame2.frame0.frame2.frame0.label3 -side left -anchor nw

text  .$top0.frame2.frame0.frame2.frame1.text0 \
     -wrap none -height 3 -width 30 \
     -xscrollcommand { .$top0.frame2.frame0.frame2.frame1.scrollbar0 set} \
     -yscrollcommand { .$top0.frame2.frame0.frame2.frame1.scrollbar1 set}


scrollbar .$top0.frame2.frame0.frame2.frame1.scrollbar0 \
    -command { .$top0.frame2.frame0.frame2.frame1.text0 xview} \
    -orient {horizontal} -relief {raised}

scrollbar .$top0.frame2.frame0.frame2.frame1.scrollbar1 \
    -command { .$top0.frame2.frame0.frame2.frame1.text0 yview} \
    -relief {raised}

pack configure .$top0.frame2.frame0.frame2.frame1.scrollbar0  \
                 -side bottom -fill x

pack configure .$top0.frame2.frame0.frame2.frame1.scrollbar1 \
                  -side right -fill y

pack configure .$top0.frame2.frame0.frame2.frame1.text0 \
                  -side top -fill both

checkbutton  .$top0.frame2.frame0.frame2.frame2.check0 \
    -text "write\nprotected"

button  .$top0.frame2.frame0.frame2.frame2.button3 \
    -padx {9} -pady {3} -text "..." -font  *times-bold-r-normal*20*

pack configure .$top0.frame2.frame0.frame2.frame2.check0 \
                  .$top0.frame2.frame0.frame2.frame2.button3 \
                  -side top -expand 1


frame  .$top0.frame2.frame0.frame3 -borderwidth 2 -relief groove

pack configure .$top0.frame2.frame0.frame3 -side bottom -expand 1 -fill both

button  .$top0.frame2.frame0.frame3.button0 -text "Layer-"

pack configure  .$top0.frame2.frame0.frame3.button0 \
        -expand 1 -side left -fill y -anchor w

#### Linke Seite
frame .$top0.frame2.frame1.frame0 -borderwidth 2 -relief groove
```

```
frame .$top0.frame2.frame1.frame0.frame0
frame .$top0.frame2.frame1.frame0.frame1
frame .$top0.frame2.frame1.frame0.frame2

pack configure .$top0.frame2.frame1.frame0.frame0 \
               .$top0.frame2.frame1.frame0.frame1 \
               .$top0.frame2.frame1.frame0.frame2 \
               -expand 1 -side left -fill both

label  .$top0.frame2.frame1.frame0.frame0.label0 \
       -text "Contact Person:"

pack configure  .$top0.frame2.frame1.frame0.frame0.label0 \
               -expand 1 -side top -anchor nw

text  .$top0.frame2.frame1.frame0.frame1.text0 \
      -wrap none -height 1 -width 30 \
      -xscrollcommand { .$top0.frame2.frame1.frame0.frame1.scrollbar0 set} \
      -yscrollcommand { .$top0.frame2.frame1.frame0.frame1.scrollbar1 set}


scrollbar .$top0.frame2.frame1.frame0.frame1.scrollbar0 \
    -command { .$top0.frame2.frame1.frame0.frame1.text0 xview} \
    -orient {horizontal} -relief {raised}

scrollbar .$top0.frame2.frame1.frame0.frame1.scrollbar1 \
    -command { .$top0.frame2.frame1.frame0.frame1.text0 yview} \
    -relief {raised}

pack configure .$top0.frame2.frame1.frame0.frame1.scrollbar0  \
                 -side bottom  -fill x

pack configure .$top0.frame2.frame1.frame0.frame1.scrollbar1 \
                  -side right -fill y

pack configure .$top0.frame2.frame1.frame0.frame1.text0 \
                  -side top -fill both -expand 1  -anchor nw
button  .$top0.frame2.frame1.frame0.frame2.button0 \
    -padx {9} -pady {3} -text "..." -font  *times-bold-r-normal*20*

button  .$top0.frame2.frame1.frame0.frame2.button1 -text "w"

pack configure  .$top0.frame2.frame1.frame0.frame2.button0 \
                .$top0.frame2.frame1.frame0.frame2.button1 \
                  -side bottom -expand 1 -anchor s

frame .$top0.frame2.frame1.frame1 -borderwidth 2 -relief groove

frame .$top0.frame2.frame1.frame2 -borderwidth 2 -relief groove

pack configure  .$top0.frame2.frame1.frame0 \
```

```
                .$top0.frame2.frame1.frame1 \
                .$top0.frame2.frame1.frame2 \
                -expand 1 -side top -fill both

##################################################
## Listbox
##################################################
global li
set li .$top0.frame3.frame4

  # built widget $li.frame
      frame $li.frame
# build widget $li.frame.scrollbar3
  scrollbar $li.frame.scrollbar3 \
    -command {$li.frame.listbox1 xview} -orient {horizontal} -relief {raised}

  # build widget $li.frame.scrollbar2
  scrollbar $li.frame.scrollbar2 \
    -command {$li.frame.listbox1 yview} -relief {raised}

  # build widget $li.frame.listbox1
  listbox $li.frame.listbox1 \
    -exportselection {true} \
    -relief {raised} \
    -xscrollcommand {$li.frame.scrollbar3 set} \
    -yscrollcommand {$li.frame.scrollbar2 set}

# pack widget $li.frame
  pack append $li.frame \
    $li.frame.scrollbar2   {right frame center filly} \
    $li.frame.listbox1   {top frame center expand fill} \
    $li.frame.scrollbar3   {bottom frame center fillx}

  pack configure $li.frame -expand 1 -fill both


##################################################
##  Canvas
#################################################

  # build widget $ca.frame0
  frame $ca.frame0 -relief {raised}

  # build widget $ca.frame0.scrollbar3
  scrollbar $ca.frame0.scrollbar3 \
    -command { $ca.frame0.canvas2 xview } \
    -orient {horizontal} \
    -relief {raised}

  # build widget .frame0.scrollbar1
  scrollbar $ca.frame0.scrollbar1 \
```

```
    -command { $ca.frame0.canvas2 yview } -relief {raised}


# build widget .frame0.canvas2
 canvas $ca.frame0.canvas2 \
    -confine {true} \
    -height {400} \
    -relief {raised} \
    -bd {2}  \
    -scrollregion {0c 0c 30c 30c} \
    -width {500} \
    -xscrollcommand {$ca.frame0.scrollbar3 set} \
    -yscrollcommand {$ca.frame0.scrollbar1 set} \
    -background white


 label $ca.frame0.label0 \
       -textvariable labelca \
       -justify left     \
       -bd 2 \
       -relief {groove}

 pack $ca.frame0.label0 -side top


# pack widget .frame0
  pack append $ca.frame0 \
    $ca.frame0.scrollbar1   {right frame center filly} \
    $ca.frame0.canvas2   {top frame center expand fill} \
    $ca.frame0.scrollbar3   {top frame center fillx}

pack configure $ca.frame0   -expand 1 -fill both

###########################################################
## end Canvas
###########################################################

# pack master .top0
  pack configure .$top0.frame0  -fill x
  pack configure .$top0.frame1  -expand 1  -fill both
  pack configure .$top0.frame2.frame0.frame0.label0 \
                 .$top0.frame2.frame0.frame0.label1 -side left
  pack configure  .$top0.frame2.frame0.frame1.label2 \
                  .$top0.frame2.frame0.frame1.entry0  -side left -fill x
  pack configure  .$top0.frame2.frame0.frame0  -expand 1 -fill both -side top
  pack configure  .$top0.frame2.frame0.frame1 -expand 1 -fill both -side top
  pack configure  .$top0.frame2.frame0.frame2 -expand 1 -side top -fill x

##########################################################################

  pack configure .$top0.frame2.frame0 -expand 1 -fill both -side left
```

```
  pack configure .$top0.frame2.frame1 -expand 1 -fill both -side right
  pack configure .$top0.frame2  -expand 1  -fill both
  pack configure .$top0.frame3.frame4 -expand 1 -fill both -side left
  pack configure .$top0.frame3.frame5 -expand 1 -fill both -side right
  pack configure .$top0.frame3  -expand 1  -fill both

tk_menuBar .$top0.frame0 .$top0.frame0.menubutton1 \
           .$top0.frame0.menubutton2


######################################################################
### Fill Listbox + Canvas
#####################################################################

global ALLKomp ALLLine
foreach i $ALLKomp {
     global build$i
}
foreach i $ALLLine {
  global build$i
}
global VLAN1 VLAN2 VLAN3 VLAN4
switch $vlanid {
    VLAN1   {   eval $ca.frame0.canvas2   $buildrechner1
                eval $ca.frame0.canvas2   $buildrechner2
                eval $ca.frame0.canvas2   $buildrechner6
                eval $ca.frame0.canvas2   $buildsegment1
                eval $ca.frame0.canvas2   $buildline1
                eval $ca.frame0.canvas2   $buildline2
                eval $ca.frame0.canvas2   $buildline6
                eval $ca.frame0.canvas2   $buildswitch1
                eval $ca.frame0.canvas2   $buildswitch2
                eval $ca.frame0.canvas2   $buildbackbone1
                eval $ca.frame0.canvas2   $buildsegline1

            foreach i $VLAN1 {
                 $li.frame.listbox1  insert end $i
                  }
               }
    VLAN2    {   eval $ca.frame0.canvas2   $buildrechner3
                eval $ca.frame0.canvas2   $buildrechner9
                eval $ca.frame0.canvas2   $buildrechner12
                eval $ca.frame0.canvas2   $buildline3
                eval $ca.frame0.canvas2   $buildline9
                eval $ca.frame0.canvas2   $buildline12
                eval $ca.frame0.canvas2   $buildlaneicon
                eval $ca.frame0.canvas2   $buildswitch1
                eval $ca.frame0.canvas2   $buildswitch2
                eval $ca.frame0.canvas2   $buildswitch3
                eval $ca.frame0.canvas2   $buildbackbone1
                eval $ca.frame0.canvas2   $buildlane1
```

```
                eval $ca.frame0.canvas2   $buildlane2

           foreach i $VLAN2 {
                $li.frame.listbox1  insert end $i
                 }
              }
    VLAN3     {   eval $ca.frame0.canvas2   $buildrechner13
                  eval $ca.frame0.canvas2   $buildsegment2
                  eval $ca.frame0.canvas2   $buildrechner7
                  eval $ca.frame0.canvas2   $buildrechner4
                  eval $ca.frame0.canvas2   $buildline13
                  eval $ca.frame0.canvas2   $buildline7
                  eval $ca.frame0.canvas2   $buildline4
                  eval $ca.frame0.canvas2   $buildsegline2
                  eval $ca.frame0.canvas2   $buildswitch1
                  eval $ca.frame0.canvas2   $buildswitch2
                  eval $ca.frame0.canvas2   $buildswitch3
                  eval $ca.frame0.canvas2   $buildlaneicon
                  eval $ca.frame0.canvas2   $buildlane1
                  eval $ca.frame0.canvas2   $buildlane2
                  eval $ca.frame0.canvas2   $buildbackbone1

               foreach i $VLAN3 {
                   $li.frame.listbox1  insert end $i
                }
             }
    VLAN4      {  eval $ca.frame0.canvas2   $buildrechner5
                  eval $ca.frame0.canvas2   $buildrechner8
                  eval $ca.frame0.canvas2   $buildrechner10
                  eval $ca.frame0.canvas2   $buildsegment3
                  eval $ca.frame0.canvas2   $buildline5
                  eval $ca.frame0.canvas2   $buildline8
                  eval $ca.frame0.canvas2   $buildline10
                  eval $ca.frame0.canvas2   $buildsegline3
                  eval $ca.frame0.canvas2   $buildlaneicon
                  eval $ca.frame0.canvas2   $buildlane1
                  eval $ca.frame0.canvas2   $buildswitch1
                  eval $ca.frame0.canvas2   $buildswitch2
                  eval $ca.frame0.canvas2   $buildbackbone1

                foreach i $VLAN4 {
                    $li.frame.listbox1  insert end $i
                }
              }
    }

#######################################################################
###  Bindings
#######################################################################

foreach i  $ALLKomp  {
```

```
              $ca.frame0.canvas2 bind $i <Enter> {
                $ca.frame0.canvas2 itemconfigure current -foreground blue
                  set j [$ca.frame0.canvas2 gettags current]
                 set labelca [lindex $j 0]
                  }
              }
foreach i  $ALLLine  {
              $ca.frame0.canvas2 bind $i <Enter> {
                $ca.frame0.canvas2 itemconfigure current -fill blue
                 set j [$ca.frame0.canvas2 gettags current]
                 set labelca [lindex $j 0]
                 }
              }

foreach i  $ALLKomp {
           $ca.frame0.canvas2 bind "$i" <Leave> {
              $ca.frame0.canvas2 itemconfigure current  -foreground black
               }
            }

foreach i  $ALLLine  {
              $ca.frame0.canvas2 bind $i <Leave> {
                         $ca.frame0.canvas2 itemconfigure current -fill black
                         }
             }
}

###########################################################################
###
###    Destroy VLAN_ID
###
###########################################################################
proc DestroyWindow.$top0 {} {# xf ignore me 7
  if {"[info procs XFEdit]" != ""} {
    if {"[info commands .$top0]" != ""} {
      global xfShowWindow.top0
      set xfShowWindow.top0 0
      XFEditSetPath .
      after 2 "XFSaveAsProc .top0; XFEditSetShowWindows"
    }
  } {
    catch "destroy .$top0"
    update
  }
}

####################################################################
### YesNoBox
####################################################################
####################################################################
# Procedure: YesNoBox
```

```
# Description: show yesno box
# Arguments: {yesNoBoxMessage} - the text to display
#            {yesNoBoxGeometry} - the geometry for the window
# Returns: none
# Sideeffects: none
##################################################################
global yesNoBox
set yesNoBox(activeBackground) ""
set yesNoBox(activeForeground) ""
set yesNoBox(anchor) n
set yesNoBox(background) ""
set yesNoBox(font) "*times-bold-r-normal*24*"
set yesNoBox(foreground) ""
set yesNoBox(justify) center
set yesNoBox(afterYes) 0
set yesNoBox(afterNo) 0
set yesNoBox(button) 0



proc YesNoBox {{yesNoBoxMessage {Yes/no message}} \
                            {yesNoBoxGeometry 350x150}} {
#
# global yesNoBox(activeBackground) - active background color
# global yesNoBox(activeForeground) - active foreground color
# global yesNoBox(anchor) - anchor for message box
# global yesNoBox(background) - background color
# global yesNoBox(font) - message font
# global yesNoBox(foreground) - foreground color
# global yesNoBox(justify) - justify for message box
# global yesNoBox(afterNo) - destroy yes-no box after n seconds.
#                            The no button is activated
# global yesNoBox(afterYes) - destroy yes-no box after n seconds.
#                             The yes button is activated

  global yesNoBox

  set tmpButtonOpt ""
  set tmpFrameOpt ""
  set tmpMessageOpt ""
  if {"$yesNoBox(activeBackground)" != ""} {
    append tmpButtonOpt "-activebackground \"$yesNoBox(activeBackground)\" "
  }
  if {"$yesNoBox(activeForeground)" != ""} {
    append tmpButtonOpt "-activeforeground \"$yesNoBox(activeForeground)\" "
  }
  if {"$yesNoBox(background)" != ""} {
    append tmpButtonOpt "-background \"$yesNoBox(background)\" "
    append tmpFrameOpt "-background \"$yesNoBox(background)\" "
    append tmpMessageOpt "-background \"$yesNoBox(background)\" "
  }
```

```
if {"$yesNoBox(font)" != ""} {
  append tmpButtonOpt "-font \"$yesNoBox(font)\" "
  append tmpMessageOpt "-font \"$yesNoBox(font)\" "
}
if {"$yesNoBox(foreground)" != ""} {
  append tmpButtonOpt "-foreground \"$yesNoBox(foreground)\" "
  append tmpMessageOpt "-foreground \"$yesNoBox(foreground)\" "
}

# start build of toplevel
if {"[info commands XFDestroy]" != ""} {
  catch {XFDestroy .yesNoBox}
} {
  catch {destroy .yesNoBox}
}
toplevel .yesNoBox \
  -borderwidth 0
catch ".yesNoBox config $tmpFrameOpt"
if {[catch "wm geometry .yesNoBox $yesNoBoxGeometry"]} {
  wm geometry .yesNoBox 350x150
}
wm title .yesNoBox {Alert box}
wm maxsize .yesNoBox 1000 1000
wm minsize .yesNoBox 100 100
# end build of toplevel

message .yesNoBox.message1 \
  -anchor "$yesNoBox(anchor)" \
  -justify "$yesNoBox(justify)" \
  -relief raised \
  -text "$yesNoBoxMessage"
catch ".yesNoBox.message1 config $tmpMessageOpt"

set xfTmpWidth \
  [string range $yesNoBoxGeometry 0 [expr [string first x $yesNoBoxGeometry]-1]]
if {"$xfTmpWidth" != ""} {
  # set message size
  catch ".yesNoBox.message1 configure \
   -width [expr $xfTmpWidth-10]"
} {
  .yesNoBox.message1 configure \
    -aspect 1500
}

frame .yesNoBox.frame1 \
  -borderwidth 0 \
  -relief raised
catch ".yesNoBox.frame1 config $tmpFrameOpt"

button .yesNoBox.frame1.button0 \
  -text "Yes" \
```

```
    -command "
      global yesNoBox
      set yesNoBox(button) 1
      if {\"\[info commands XFDestroy\]\" != \"\"} {
        catch {XFDestroy .yesNoBox}
      } {
        catch {destroy .yesNoBox}
      }"
  catch ".yesNoBox.frame1.button0 config $tmpButtonOpt"
  button .yesNoBox.frame1.button1 \
    -text "No" \
    -command "
      global yesNoBox
      set yesNoBox(button) 0
      if {\"\[info commands XFDestroy\]\" != \"\"} {
        catch {XFDestroy .yesNoBox}
      } {
        catch {destroy .yesNoBox}
      }"
  catch ".yesNoBox.frame1.button1 config $tmpButtonOpt"

  pack append .yesNoBox.frame1 \
              .yesNoBox.frame1.button0 {left fillx expand} \
              .yesNoBox.frame1.button1 {left fillx expand}

  # packing
  pack append .yesNoBox \
              .yesNoBox.frame1 {bottom fill} \
              .yesNoBox.message1 {top fill expand}

  if {$yesNoBox(afterYes) != 0} {
    after [expr $yesNoBox(afterYes)*1000] \
      "catch \".yesNoBox.frame1.button0 invoke\""
  }
  if {$yesNoBox(afterNo) != 0} {
    after [expr $yesNoBox(afterNo)*1000] \
      "catch \".yesNoBox.frame1.button1 invoke\""
  }

  # wait for the box to be destroyed
  update idletask
  grab .yesNoBox
  tkwait window .yesNoBox

  return $yesNoBox(button)
}

############################################################################
###
###   TextBox
###
```

```
############################################################################
global textBox
set textBox(activeBackground) ""
set textBox(activeForeground) ""
set textBox(background) ""
set textBox(font) ""
set textBox(foreground) ""
set textBox(scrollActiveForeground) ""
set textBox(scrollBackground) ""
set textBox(scrollForeground) ""
set textBox(scrollSide) right
set textBox(state) disabled
set textBox(toplevelName) .textBox
set textBox(button) 0
set textBox(contents) ""

proc TextBox {{textBoxMessage {Text message}} {textBoxCommand ""} \
           {textBoxGeometry 350x150} {textBoxTitle "Text box"} args} {
##########
# Procedure: TextBox
# Description: show text box
# Arguments: {textBoxMessage} - the text to display
#            {textBoxCommand} - the command to call after ok
#            {textBoxGeometry} - the geometry for the window
#            {textBoxTitle} - the title for the window
#            {args} - labels of buttons
# Returns: The number of the selected button, or nothing
# Sideeffects: none
# Notes: there exist also functions called:
#          TextBoxFile - to open and read a file automatically
#          TextBoxFd - to read from an already opened filedescriptor
##########
#
# global textBox(activeBackground) - active background color
# global textBox(activeForeground) - active foreground color
# global textBox(background) - background color
# global textBox(font) - text font
# global textBox(foreground) - foreground color
# global textBox(scrollActiveForeground) - scrollbar active background color
# global textBox(scrollBackground) - scrollbar background color
# global textBox(scrollForeground) - scrollbar foreground color
# global textBox(scrollSide) - side where scrollbar is located

  global textBox

  # show text box
  if {[llength $args] > 0} {
    eval TextBoxInternal "\{$textBoxMessage\}" "\{$textBoxCommand\}" \
                     "\{$textBoxGeometry\}" "\{$textBoxTitle\}" $args
  } {
    TextBoxInternal $textBoxMessage $textBoxCommand $textBoxGeometry $textBoxTitle
```

```
  }

  if {[llength $args] > 0} {
    # wait for the box to be destroyed
    update idletask
    grab $textBox(toplevelName)
    tkwait window $textBox(toplevelName)

    return $textBox(button)
  }
}

proc TextBoxFd {{textBoxInFile ""} {textBoxCommand ""} \
              {textBoxGeometry 350x150} {textBoxTitle "Text box"} args} {
###########################################################################
# Procedure: TextBoxFd
# Description: show text box containing a filedescriptor
# Arguments: {textBoxInFile} - a filedescriptor to read. The descriptor
#                              is closed after reading
#            {textBoxCommand} - the command to call after ok
#            {textBoxGeometry} - the geometry for the window
#            {textBoxTitle} - the title for the window
#            {args} - labels of buttons
# Returns: The number of the selected button, ot nothing
# Sideeffects: none
# Notes: there exist also functions called:
#          TextBox - to display a passed string
#          TextBoxFile - to open and read a file automatically
####################################################################

  global textBox

  # check file existance
  if {"$textBoxInFile" == ""} {
    puts stderr "No filedescriptor specified"
    return
  }

  set textBoxMessage [read $textBoxInFile]
  close $textBoxInFile

  # show text box
  if {[llength $args] > 0} {
    eval TextBoxInternal "\{$textBoxMessage\}" "\{$textBoxCommand\}" \
            "\{$textBoxGeometry\}" "\{$textBoxTitle\}" $args
  } {
    TextBoxInternal $textBoxMessage $textBoxCommand $textBoxGeometry $textBoxTitle
  }

  if {[llength $args] > 0} {
    # wait for the box to be destroyed
```

```
    update idletask
    grab $textBox(toplevelName)
    tkwait window $textBox(toplevelName)

    return $textBox(button)
  }
}

proc TextBoxFile {{textBoxFile ""} {textBoxCommand ""} \
  {textBoxGeometry 350x150} {textBoxTitle "Text box"} args} {
######################################################################
# Procedure: TextBoxFile
# Description: show text box containing a file
# Arguments: {textBoxFile} - filename to read
#            {textBoxCommand} - the command to call after ok
#            {textBoxGeometry} - the geometry for the window
#            {textBoxTitle} - the title for the window
#            {args} - labels of buttons
# Returns: The number of the selected button, ot nothing
# Sideeffects: none
# Notes: there exist also functions called:
#          TextBox - to display a passed string
#          TextBoxFd - to read from an already opened filedescriptor
#######################################################################


global textBox

  # check file existance
  if {"$textBoxFile" == ""} {
    puts stderr "No filename specified"
    return
  }

  if {[catch "open $textBoxFile r" textBoxInFile]} {
    puts stderr "$textBoxInFile"
    return
  }

  set textBoxMessage [read $textBoxInFile]
  close $textBoxInFile

  # show text box
  if {[llength $args] > 0} {
    eval TextBoxInternal "\{$textBoxMessage\}" "\{$textBoxCommand\}" \
           "\{$textBoxGeometry\}" "\{$textBoxTitle\}" $args
  } {
    TextBoxInternal $textBoxMessage $textBoxCommand \
          $textBoxGeometry    $textBoxTitle
  }
```

```
  if {[llength $args] > 0} {
    # wait for the box to be destroyed
    update idletask
    grab $textBox(toplevelName)
    tkwait window $textBox(toplevelName)

    return $textBox(button)
  }
}

##########################################################################
# Procedure: TextBoxInternal
# Description: show text box internal
# Arguments: textBoxMessage - the text to display
#            textBoxCommand - the command to call after ok
#            textBoxGeometry - the geometry for the window
#            textBoxTitle - the title for the window
#            args - labels of buttons
# Returns: none
# Sideeffects: none
############################################################################
proc TextBoxInternal {textBoxMessage textBoxCommand textBoxGeometry \
                      textBoxTitle args} {
  global textBox

  set tmpButtonOpt ""
  set tmpFrameOpt ""
  set tmpMessageOpt ""
  set tmpScrollOpt ""
  if {"$textBox(activeBackground)" != ""} {
    append tmpButtonOpt "-activebackground \"$textBox(activeBackground)\" "
  }
  if {"$textBox(activeForeground)" != ""} {
    append tmpButtonOpt "-activeforeground \"$textBox(activeForeground)\" "
  }
  if {"$textBox(background)" != ""} {
    append tmpButtonOpt "-background \"$textBox(background)\" "
    append tmpFrameOpt "-background \"$textBox(background)\" "
    append tmpMessageOpt "-background \"$textBox(background)\" "
  }
  if {"$textBox(font)" != ""} {
    append tmpButtonOpt "-font \"$textBox(font)\" "
    append tmpMessageOpt "-font \"$textBox(font)\" "
  }
  if {"$textBox(foreground)" != ""} {
    append tmpButtonOpt "-foreground \"$textBox(foreground)\" "
    append tmpMessageOpt "-foreground \"$textBox(foreground)\" "
  }
  if {"$textBox(scrollActiveForeground)" != ""} {
    append tmpScrollOpt "-activeforeground \"$textBox(scrollActiveForeground)\" "
  }
```

```
if {"$textBox(scrollBackground)" != ""} {
  append tmpScrollOpt "-background \"$textBox(scrollBackground)\" "
}
if {"$textBox(scrollForeground)" != ""} {
  append tmpScrollOpt "-foreground \"$textBox(scrollForeground)\" "
}

# start build of toplevel
if {"[info commands XFDestroy]" != ""} {
  catch {XFDestroy $textBox(toplevelName)}
} {
  catch {destroy $textBox(toplevelName)}
}
toplevel $textBox(toplevelName) \
  -borderwidth 0
catch "$textBox(toplevelName) config $tmpFrameOpt"
if {[catch "wm geometry $textBox(toplevelName) $textBoxGeometry"]} {
  wm geometry $textBox(toplevelName) 350x150
}
wm title $textBox(toplevelName) $textBoxTitle
wm maxsize $textBox(toplevelName) 1000 1000
wm minsize $textBox(toplevelName) 100 100
# end build of toplevel

frame $textBox(toplevelName).frame0 \
  -borderwidth 0 \
  -relief raised
catch "$textBox(toplevelName).frame0 config $tmpFrameOpt"

text $textBox(toplevelName).frame0.text1 \
  -relief raised \
  -wrap none \
  -borderwidth 2 \
  -yscrollcommand "$textBox(toplevelName).frame0.vscroll set"
catch "$textBox(toplevelName).frame0.text1 config $tmpMessageOpt"

scrollbar $textBox(toplevelName).frame0.vscroll \
  -relief raised \
  -command "$textBox(toplevelName).frame0.text1 yview"
catch "$textBox(toplevelName).frame0.vscroll config $tmpScrollOpt"

frame $textBox(toplevelName).frame1 \
  -borderwidth 0 \
  -relief raised
catch "$textBox(toplevelName).frame1 config $tmpFrameOpt"

set textBoxCounter 0
set buttonNum [llength $args]

if {$buttonNum > 0} {
  while {$textBoxCounter < $buttonNum} {
```

```
    button $textBox(toplevelName).frame1.button$textBoxCounter \
      -text "[lindex $args $textBoxCounter]" \
      -command "
        global textBox
        set textBox(button) $textBoxCounter
        set textBox(contents) \[$textBox(toplevelName).frame0.text1 \
                                get 1.0 end\]
        if {\"\[info commands XFDestroy\]\" != \"\"} {
          catch {XFDestroy $textBox(toplevelName)}
        } {
          catch {destroy $textBox(toplevelName)}
        }"
    catch "$textBox(toplevelName).frame1.button$textBoxCounter config \
                                              $tmpButtonOpt"

    pack append $textBox(toplevelName).frame1 \
                $textBox(toplevelName).frame1.button$textBoxCounter \
                                                   {left fillx expand}

    incr textBoxCounter
  }
} {
  button $textBox(toplevelName).frame1.button0 \
    -text "OK" \
    -command "
      global textBox
      set textBox(button) 0
      set textBox(contents) \[$textBox(toplevelName).frame0.text1 get 1.0 end\]
      if {\"\[info commands XFDestroy\]\" != \"\"} {
        catch {XFDestroy $textBox(toplevelName)}
      } {
        catch {destroy $textBox(toplevelName)}
      }
      $textBoxCommand"
  catch "$textBox(toplevelName).frame1.button0 config $tmpButtonOpt"

  pack append $textBox(toplevelName).frame1 \
              $textBox(toplevelName).frame1.button0 {left fillx expand}
}

$textBox(toplevelName).frame0.text1 insert end "$textBoxMessage"

$textBox(toplevelName).frame0.text1 config \
  -state $textBox(state)

# packing
pack append $textBox(toplevelName).frame0 \
            $textBox(toplevelName).frame0.vscroll "$textBox(scrollSide) filly" \
            $textBox(toplevelName).frame0.text1 {left fill expand}
pack append $textBox(toplevelName) \
            $textBox(toplevelName).frame1 {bottom fill} \
```

```
            $textBox(toplevelName).frame0 {top fill expand}
}
```

Anhang B

Abkürzungsverzeichnis

A

AAL	ATM Adaption Layer
ARP	Adress Resolution Protocol
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
AUI	Attachment Unit Interface

B

B-ISDN	Broadband Integrated Services Digital Network
bps	bits per second
BUS	Broadcast and Unknown Server

C

CCITT	Comité Consulatif International Télégraphique et Téléphonique (heute ITU-T)
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CT	Cut Through
CUG	Closed User Group

D

DBMS	Data Base Management System
DGDB	Distributed Queue Dual Bus

E

ELAN	Emuliertes LAN

F

FCS	Frame Check Sequence
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
G	
GUI	Graphical User Interface
H	
HDLC	High Level Data Link Control
HEC	Header Error Check
HW	Hardware
I	
IAB	Internet Activities Board
ID	Identikator (Identifier)
IEEE	Institute of Electrical and Electronics Engineers
IETF	International Engineering Task Force
IP	Internet Protocol
IPX	Internet Packet eXchange
ISDN	Integrated Services Digital Network
ISO	International Standardization Organisation
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector (CCITT)
L	
LAN	Local Area Network
LANE	LAN Emulation
LAT	Local Area Transport (Digital)
LE	LAN Emulation
LE_ARP	LAN Emulation Address Resolution Protocol
LEC	LAN Emulation Client
LECS	LAN Emulation Configuration Server
LED	Light Emmiting Diode
LES	LAN Emulation Server
LLC	Logical Link Control
LUNI	LAN Emulation User Network Interface
M	
MAC	Medium Access Control
MAN	Metropolitan Area Network
MIB	Management Information base
Mbps	Megabits pro Sekunde

MPOA	Multiprotocol over ATM
MWM	Motif Window Manager
N	
NetBIOS	Network Basic Input Output System
O	
OLWM	Open Look Window Manager
OO	Object-oriented, objektorientiert
OOA	Object orientierte Analyse
OOP	Object-oriented Programming
OMT	Object Modeling Technique
OOSE	Object-Oriented Software Engineering
OOUI	Object-oriented User Interface
OSF	Open Systems Foundation
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
P	
PHY	Physical Layer
Q	
QoS	Quality of Services
R	
RAM	Random Access Memory
RARP	Reverse Address Resolution Protocol
RFC	Request for Comments
RIB	Routing Information Protocol
RISC	Reduced Instruction Set Computer
RMON	Remote Monitoring
S	
S&F	Store and Forward
SF	Store and Forward
SMTP	Simple Mail Transfer Protocol
SW	Software
T	
TDM	Time Division Multiplexing
TWM	Tom's Window Manager

U

UI	User Interface
UIMS	User Interface Management System
UVM	Ultrix Window Manager

V

VC	Virtual Circuit
VCC	Virtual Channel Connection
VLAN	Virtual Local Area Network
VNET	Virtual Network
VP	Virtual Path
VPC	Virtual Path Connection
VPN	Virtual Private Network

W

WAN	Wide Area Network

X

X	X Window System

Literaturverzeichnis

[AHK95] Anatol, Erwin Hoffmann und Olaf Knauer. *High Speed Internetworking*. Addison-Wesley, 1995.

[Bor96] Petra Borowka. *Internetworking*. Datacom, 1996.

[cla94] Classical IP and ARP over ATM. Technical report, IETF, Januar 1994.

[CY90] P. Coad und E. Yourdon. *Object-Oriented Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[Dah95a] H.G. Dahn. Digital-Switches als Produktfamilie. *NetWorks*, Seiten 64–66, Mai 1995.

[Dah95b] H.G. Dahn. Was sind virtuelle LANs. *NetWorks*, Seiten 61–63, Mai 1995.

[Des95] D. Deshpande. Virtuelle Datennetze. *DATACOM*, Seiten 40–42, März 1995.

[DKLDSW96] Peter Dr. Kaiser, Johann Lindmayr, Rainer Dr. Sauerwein und Norbert Wienold. VLAN-Management: An Applikation for Integrated Network Management. Research note, Siemens AG/LMU München, April 1996.

[Eea95] Bill Ellington und et. al. LAN Emulation Over ATM Version 1.0. Technical report, The ATM Forum, Januar 1995.

[End94] Albert Endres. *Bedienoberflächen*, WS 1994.

[Epp94a] Klaus Eppele. Migrationspfade zu ATM. *Gateway*, Seiten 44–46, Juni 1994.

[Epp94b] Klaus Eppele. Strukturierung von Ethernets mit Hilfe von Bridges. *Gateway*, Seiten 94–97, Mai 1994.

[For95] Thomas Forster. Einsatz des Konzepts der virtuellen Workgroups von ATM in einer bestehenden Betreiberumgebung. Diplomarbeit, Technische Universität München - Institut für Informatik, August 1995.

[HA93] Heinz-Gerd Hegering und Sebastian Abeck. *Integriertes Netz- und Systemmanagement*. Addison-Wesley, 1. Auflage, 1993.

[Has94a] Harald Hassenmüller. Integration lokaler Netze zu ATM mit LAN Emulation und IP-Encapsulation. *Gateway*, Seiten 130–132, Oktober 1994.

[Has94b] Harald Hassenmüller. Netzdesign der Zukunft, Teil 1: Ohne Hops durchs Netz. *Gateway*, Seiten 96–104, Oktober 1994.

[Has94c] Harald Hassenmüller. Netzdesign der Zukunft, Teil 2: Im virtuellen Netz. *Gateway*, Seiten 98–104, November 1994.

[Has95] Harald Hassenmüller. Im virtuellen Netz. *Gateway*, Seite 48, Juli 1995.

[HNW95] H.-G. Hegering, B. Neumair und R. Wies. Integriertes Management verteilter Systeme - Ein Überblick über den State-of-the-Art -. Technischer bericht, Ludwig-Maximilians-Universität München, Januar 1995.

[ISA] *ISA-Dialogmanager*.

[Jea94] Ivar Jacobson und et. al. *Objekt-Oriented Software Engineering*. Addison-Wesley, 1994.

[Jef94] Ron Jeffries. ATM LAN Emulation: The Inside Story. *Data Communications*, Seiten 95–100, September 1994.

[K̈95] Ulrich Körber. Motif Programmierung: UIL und Mrm kontra Toolkit-Routinen. *iX*, Seiten 56–65, März 1995.

[Kau96] Franz-Joachim Kauffels. *Lokale Netze — Grundlagen, Standards, Perspektiven*. DATACOM Buchverlag, 1996.

[Ker93] Helmut Hrsg. Kerner. *Rechnernetze nach OSI*. Addison-Wesley, 2. Auflage, 1993.

[Kin94] Steven S. King. Switched Virtual Networks. *Data Communications*, Seiten 66–80, September 1994.

[Kre94] Frank Kresse. Switching-Technologien. *N&C Network & Communication*, Seiten 47–54, Dezember 1994.

[Kre95] Frank Kresse. Netzwerkumgestaltung mit Switching-Technik: Fitneßkur für das Netz. *Gateway*, Seiten 28–36, Juli/August 1995.

[Kya96] Othmar Kyas. *ATM-Netzwerke*. Datacom, 1996.

[Lip94] N. Lippis. Virtual LANs: Real Drawbacks. *Data Communications*, Seiten 23–24, Dezember 1994.

[Lip95] N. Lippis. VLANs: Ahead of their Time? *Data Communications*, Seiten 29–30, Dezember 1995.

[mpo93] Multiprotocol Encapsulation over ATM Adaption Layer 5. Technical report, The ATM Forum, Juli 1993.

[N.N95] N.N. Virtual LANs Get Real. *Data Communications*, Seiten 87–100, März 1995.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[PD95] Faye Ly Prakash Desai. Emulated LAN MIB. Research note, ATM Forum Technical Committee, LAN Emulation Sub-Working Group, August 1995.

[Poi95] Anton Pointner. Ausgewählte Techniken und Systeme für das Konfigurationsmanagement. Hauptseminar, Technische Universität München - Institut für Informatik, Juni 1995.

[Red96] Bernd Reder. Management virtueller LANs. *Gateway*, Seite 97, Juli 1996.

[RT] Paul Raines und Jeff Tranter. *Tcl/Tk Reference Guide for Tcl 7.4/Tk 4.0*.

[Sal95] S. Salamone. Virtual LANs Get Real. *BYTE*, Seiten 181–184, Mai 1995.

[Sch95a] Ute Schneider. Benutzerschnittstellen: GUI-Builder im praktischen Einsatz. *iX*, Seiten 50–55, März 1995.

[Sch95b] Stefan Schwarz. OOP-Werkzeuge: Marktübersicht: Objektorientierte Interfacebuilder. *iX*, Seiten 42–49, März 1995.

[Seg95] Peter Segner. *Ein betreibergerechtes View-Konzept für das Netz- und Systemmanagement*. Dissertation, Technische Universität München - Institut für Informatik, 1995.

[Tan89] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 2. Auflage, 1989.

[Vuk96] Stefan Vukelic. Entwurf und Spezifikation eines Objektmodells für das VLAN-Management. Diplomarbeit, Technische Universität München - Institut für Informatik, November 1996.

[Weg95] Hans Wegener. GUI-Programmierung: Wiederverwendbare Komponenten für die Benutzerschnittstelle. *iX*, Seiten 34–40, März 1995.

[Wel95] Brent Welch. *Practical Programming in Tcl and Tk*. Addison-Wesley, 1995.

[You92] Douglas A. Young. *Object-Oriented Programming with C++ and OSF/Motif*. Prentice Hall, Englewood Cliffs, 1992.

[You94] Doug Young. Programmentwicklung mit C++ und Motif. *iX*, Seite 44, Januar 1994.