Inhaltsverzeichnis

Inhaltsverzeichnis **1**
Verzeichnis der Abbildungen **3**
Verzeichnis der Tabellen **5**
1 Einleitung **6**
2 Grundlagen der Gruppenkommunikation **10**
2.1 Formen der Gruppenkommunikation 10
2.2 Spezielle Aspekte der Gruppenkommunikation 13
2.2.1 Zuverlässigkeit *13*
2.2.2 Flusskontrolle *14*
2.2.3 Gruppenverwaltung *16*
2.3 Reichweite von Multicast-Gruppen 17
2.3.1 TTL-Bereiche *18*
2.3.2 Administrative Bereiche *19*
2.4 Unterstützung im Kommunikationssystem 20
2.4.1 Schichten des Kommunikationssystems *21*
2.4.1.1 Sicherungsschicht 21
2.4.1.2 Vermittlungsschicht 21
2.4.1.3 Transportschicht 24
2.4.1.4 Anwendungsschicht 24
2.4.2 Multicast Backbone *24*
2.4.2.1 MBone-Anwendungen 25
2.5 Die Client/Server-Architektur 26
2.5.1 Fileserver *28*
2.5.2 Datenbankserver *29*
2.5.3 Transaktionsserver *29*
2.5.4 Objektserver *29*
2.5.5 Webserver *30*
2.6 Netzwerkprogrammierung 31
2.6.1 Sockets *31*
2.6.2 Windows-Sockets in MFC *34*
2.6.2.1 Benachrichtigungsfunktionen 36
2.6.2.2 Byte-Reihenfolge 38
2.6.3 WinInet *39*
3 Konzeption der Kommunikationsdienste **41**
3.1 Konzept des Gesamtsystems 41
3.2 Konzeption der Dienste 45
3.2.1 Konzeption synchroner Kommunikationsdienste *47*
3.2.1.1 Audio-/Videokommunikation-Dienst 47
3.2.1.2 Chat-Dienst 48

3.2.2 Konzeption asynchroner Kommunikationsdienste *50*
3.2.2.1 FTP-Dienst 50
3.2.2.2 Mail-Dienst 51
3.2.2.3 Whiteboard 52
3.3 Konzeption der Gruppenverwaltung 54
4 Implementierung der Kommunikationsdienste **56**
4.1 Kommunikation zwischen einem Dienst und dem zentralen Servermodul 60
4.2 Chat 61
4.3 FTP 64
4.4 Whiteboard 68
4.5 Audio- / Videokommunikation 78
5 Ergebnisse der Arbeit **80**
6 Zusammenfassung und Ausblick **88**
Abkürzungsverzeichnis **91**
Literatur- und Quellenverzeichnis **93**
Anhang A: Das TCP/IP Referenzmodell **97**
.1 Die Netzwerkschicht 97
.2 Die Internetschicht 97
.3 Die Transportschicht 98
.4 Die Verarbeitungsschicht 98
Anhang B: Die Internet Protokoll Familie **99**
.1 Das Internet Protokoll (IP) 99
.2 Das TCP-Protokoll 100
.3 Das UDP-Protokoll 102
Anhang C: Quelltexte **104**

Verzeichnis der Abbildungen

Abbildung 2.1 Schematische Darstellung der Unicast-Kommunikation 10
Abbildung 2.2 Schematische Darstellung der Multicast-Kommunikation 11
Abbildung 2.3 Schematische Darstellung der Concast-Kommunikation 12
Abbildung 2.4 Schematische Darstellung der Multipeer-Kommunikation 12
Abbildung 2.5 Schematische Darstellung der Flusskontrolle mit negativen Quittungen ... 15
Abbildung 2.6 Schematische Darstellung des ISO/OSI–Referenzmodells, des Internet-Modells und der TCP/IP- Protokollfamilie 21
Abbildung 2.7 Darstellung des IP-Adressformats 22
Abbildung 2.8 Darstellung spezieller IP-Adressen im Netz 23
Abbildung 2.9 Zwei MBone-Domänen, die über einen Tunnel miteinander verbunden sind. .. 25
Abbildung 2.10 Schematische Darstellung der Client/Server-Architektur 28
Abbildung 2.11 Schematische Darstellung des WinSock-Models im Schichtenmodell 32
Abbildung 3.1 Schematische Darstellung des Aufbaus des gesamten Distance-Learning-Systems 44
Abbildung 3.2 Schematische Darstellung der Teilkomponentenkommunikation am Beispiel eines Chat-Dienstes im Zusammenspiel mit dem zentralen Service 46
Abbildung 3.3 Relationales Modell der Gruppenverwaltung aus der Sicht des zentralen Service 55
Abbildung 4.1 Übertragung von Daten über CSocket mit der Verwendung von CArchive ... 59
Abbildung 4.2 Darstellung eines Fensters mit der Aufforderung, eine Chat-Gruppe auszuwählen 62
Abbildung 4.3 Grafisches Interface zum Chat-Server mit der Protokollierung einzelner Aktivitäten 64
Abbildung 4.4 Kommunikation im grafischen Interface des Chat-Dienstes 64
Abbildung 4.5 Dialogfenster mit Eingaben zum Verbinden 66
Abbildung 4.6 Dialogfenster mit einer Warnungsmeldung bei einer falschen Adresseneingabe 66
Abbildung 4.7 Grafisches Benutzerinterface für FTP-Dienst 67
Abbildung 4.8 Dialogfenster zum Erstellen eines neuen Verzeichnisses und mit der Warnungsmeldung bei einem Fehlerauftritt. 68
Abbildung 4.9 WB- Protokollstapel 68
Abbildung 4.10 Schematische Darstellung des WB-Protokoll-Pakets 69
Abbildung 4.11Whiteboard (WB) 77
Abbildung 4.12 Dialogfenster zur Auswahl des Kommunikationspartners 78
Abbildung 4.13 Dialogfenster für die Einladung zur Teilnahme an der Videokommunikation 79

Abbildung A.0.1 TCP/IP-Referenzmodell 97
Abbildung B.0.1 Schematische Darstellung eines Headers des IP-Protokolls 100
Abbildung B.0.2 Schematische Darstellung eines TCP-Headers 101
Abbildung B.0.3 Schematische Darstellung eines UDP-Headers 103

Verzeichnis der Tabellen

TABELLE 2.1 LISTE DER SCHWELLWERTE FÜR DAS TTL-SCOPING.....19
TABELLE 2.2 LISTE DER DEFINIERTEN ADMINISTRATIVEN BEREICHE 19
TABELLE 2.3 LISTE DER ADRESSBEREICHE IN DEN KLASSEN A-E 22
TABELLE 2.4 BEISPIELE FÜR PERMANENTE GRUPPEN 23
TABELLE 2.5 ÜBERSICHT ÜBER DIE REIHENFOLGE DER OPERATIONEN FÜR DAS EINRICHTEN EINER KOMMUNIKATION ZWISCHEN EINEM SERVER UND EINEM CLIENT 36
TABELLE 2.6 DIE BYTE-REIHENFOLGE BIG ENDIAN UND LITTLE ENDIAN.....38
TABELLE 2.7 LISTE VON WINDOWS-SOCKETS-FUNKTIONEN FÜR DIE UMSTELLUNG DER BYTE-REIHENFOLGE.....38
TABELLE 3.1 DATENDURCHSATZ EINES CHAT-SERVERS OHNE OVERHEAD 50
TABELLE 4.1 ÜBERSICHT ÜBER DIE ATTRIBUTE DER KLASSE *CDATAMSG* 61
TABELLE 4.2 ÜBERSICHT ÜBER DIE ATTRIBUTE DER KLASSE CCTRLMSG.....61
TABELLE 4.3 ÜBERSICHT ÜBER DIE KONTROLLNACHRICHTENTYPEN 62
TABELLE 4.4 LISTE VON SCHRITTEN, DIE IN EINER FTP-CLIENT-ANWENDUNG DURCHZUFÜHREN SIND .65
TABELLE 4.5 LISTE VON TYPEN DES PK_DRW_MSG-PAKETS.....71
TABELLE 4.6 LISTE DER KLASSEN, DIE GEOMETRISCHE OBJEKTE DARSTELLEN 72
TABELLE 5.1 ABSCHÄTZUNGEN DES NUTZDATENDURCHSATZES BEIM CHAT-SERVER (MIT OVERHEAD)..81
TABELLE 5.2 CPU-BELASTUNG VERSCHIEDENER CLIENTS-RECHNER BEI VERWENDUNG VERSCHIEDENER DIENSTEN.....86

1 Einleitung

Internet – Multimedia – Informations-Superhighway etc. sind Schlagwörter, die in der letzten Zeit zunehmend an Bedeutung gewinnen. Mit diesen Schlagwörtern verbindet man die Innovation und die immensen technischen Fortschritte im Bereich der rechnergestützten Kommunikation. Die Kommunikation, die auf dem Austausch bzw. auf der Übertragung von Informationen innerhalb einer Gruppe von Menschen [IRM00] basiert, wird als Gruppenkommunikation bezeichnet. Dieser Form der Kommunikation wird für die Zukunft eine enorm wichtige Rolle eingeräumt [WIT99].

Im Bereich der Aus- und Weiterbildung nimmt das ortsunabhängige Lernen mit stetig sinkender Halbwertszeit des erworbenen Wissens in der Gesellschaft eine immer wichtiger werdende Rolle ein [WIT99].

Unter dem Begriff *Distance Learning* werden die verschiedenen Methoden zur Unterstützung des ortsunabhängigen Lernens zusammengefasst. Unter *Tele-Teaching* versteht man dabei meist die Möglichkeit, von einem Ort aus an einer Veranstaltung zeitgleich an einem anderen Ort teilzunehmen.

Auch hier ist die Gruppenkommunikation zwischen dem Lehrenden und den Lernenden erforderlich. Es müssen Fragen gestellt und Lösungen diskutiert werden. Darüber hinaus muss es ermöglicht werden, dass ein Lernender sich im WWW mit Lernmaterialien versorgen, dort seine Aufgaben abliefern und darauf Rückmeldungen vom Lehrenden bekommen kann.

Weiterhin sind Szenarien denkbar, in denen das *verteilte Arbeiten im Team* unterstützt werden muss (z.B. Projektgruppen). Das gemeinsame computergestützte Arbeiten wird in der Literatur allgemein als *CSCW* (Computer Supported Cooperative Work) oder *Groupware* bezeichnet [STE98]. Die Gruppenkommunikation stellt in diesem Sinne auch hinsichtlich der Anwendungen einen aktuellen Forschungsbereich dar.

Im Zuge der Entwicklungen zu Internet2, der nächsten Generation des Internets, spielen Anwendungen zum Distance Learning eine wesentliche Rolle. Hier sollen vor allem auch deren Anforderungen an das Kommunikationssystem erfüllt werden. International laufen derzeit eine ganze Reihe Forschungs- und Entwicklungsarbeiten in diesem Bereich.

Die bisherigen Systeme genügen nicht den Anforderungen und schöpfen neue Möglichkeiten der breitbandigen Netzwerke und Kommunikation nicht aus. Sie sind kompliziert zu bedienen und erfordern zum Teil teuere Spezialsysteme.

Im Rahmen des Pilotprojekts „OVID" wird am Lehrstuhl „Rechnernetze und Verteilte Systeme" ein neues System für künftige, hochqualitative Kommunikationsformen geschaffen, das sich in großer Breite und bei einfacher Handhabung in weiten Bereichen der Aus- und Weiterbildung in Schulen, Hoch- und Fachschulen, Wirtschaft und Verwaltung einsetzen lässt.

Die integrierte Breitbandkommunikation erlaubt erweiterte Formen der Arbeits- und Nutzungsorganisation, wie multimediale Kommunikation und rechnergestütztes, kooperatives Arbeiten [KER98]. Dabei wird der Zugriff auf gemeinsame, in elektronischer Form dargestellte Dokumente, Unterstützt.

Bei der Durchführung von virtuellen Lehrveranstaltungen oder beim Zugriff auf die abgespeicherten Lehrmaterialien soll auch eine Möglichkeit der Kommunikation zwischen den Benutzern des Systems zur Verfügung stehen.

Die Ziele der vorliegenden Arbeit sind, für ein Distance-Learning-Tool zur Erstellung und Nutzung multimedialer Quellen

- die Nutzbarkeit verschiedener synchroner und asynchroner Kommunikationsdienste zu testen,
- die Dienste mit weitergehender Funktionalität zu entwerfen und zu implementieren,
- die Dienste in ein Gesamtsystem zu integrieren, insbesondere Whiteboard, Chat-Dienst, Mail-Dienst, FTP-Dienst.

Bei Konzeption und Entwicklung von neuartigen Softwareprodukten spielt die Benutzerakzeptanz eine entscheidende Rolle. Die im Rahmen der vorliegenden Arbeit geplanten Dienste sind so konzipiert, dass sie von Benutzern, die weniger mit der Technik Erfahrungen haben, problemlos genutzt werden können. Deshalb ist das Benutzerinterface einfach und Einsteiger-freundlich gestaltet. Auch deshalb wurde Windows als Entwicklungsplattform ausgewählt, die zusammen mit dem Visual C++ Compiler von Microsoft die Entwicklung von einfach und intuitiv zu bedienenden Programmen ermöglicht.

Im Laufe der Arbeit wurde die Effizienz der Nutzung von Unicast und Multicast als Transportmodi für die Gruppenkommunikation untersucht. Für den Chat-Dienst wurde der Unicast-Transportmodus ausgewählt; beim Whiteboard wurde Multicast eingesetzt.

Für einige Dienste wurden eigene Protokolle entwickelt, die auf der Applikationsebene implementiert wurden. Der Chat-Dienst benutzt die Serialisierung von Objekten, was die Übertragung von Objekten über das Netz ermöglicht. Dasselbe Modell wird auch für die Kommunikation mit dem Sitzungsmanagement, das als Server-Modul auf einem entfernten Rechner eingerichtet ist, eingesetzt. Für das Whiteboard wurde ein eigenes, empfängerorientiertes Protokoll entwickelt. Dabei wurden zusätzliche Transportprotokollmechanismen ausgearbeitet, die eine zuverlässige Multicast-Kommunikation ermöglichen. Wie bei den meisten Transportschicht-orientierten Konzepten werden auch beim Whiteboard Übertragungswiederholungen vom Sender eingesetzt, um auftretende Paketverluste auszugleichen.

Im weiteren wurden auch die verschiedenen Gruppenkommunikationsszenarien untersucht und dabei entstehende verwaltungsbedingte Probleme gelöst. Alle Kommunikationsgruppen werden zentral verwaltet, wobei eine Gruppe beim Chat über Empfängerlisten und beim Whiteboard über spezielle Gruppenadressen adressiert wird. Für alle anderen Dienste wird die Gruppenverwaltung vom zentralen Sitzungsmanagement übernommen.

Um die bessere und nahtlose Integration vom FTP-Dienst in das Gesamtsystem zu ermöglichen, wurde ein FTP-Client neu entwickelt und implementiert. Dabei wurde die von Microsoft bereitgestellte Schnittstelle WinInet verwendet. Als E-Mail-Dienst kommt ein beliebiges E-Mail-System, das auf dem Client-Rechner installiert ist, zum Einsatz. Die Entscheidung über das Mail-System, das verwendet werden soll, wird dem Anwender überlassen.

Der Audio-/Videokommunikations-Dienst wurde damit bereitgestellt, dass in die OVID-Distance-Learning-Umgebung das vorhandene Online-Audio-/Video-Kommunikationssystem für PAL-Qualität integriert wurde. Um den Aufruf des Audio-/Videokommunikations-Dienstes aus der OVID-Distance-Learning-Umgebung zu ermöglichen, wurde das vorliegende Programm modifiziert.

Diese Arbeit ist in 6 Kapitel gegliedert. Kapitel 2 führt kurz in die benötigten Grundlagen ein. Dort werden die Probleme bei der Realisierung rechnergestützter Gruppenkommunikation und technische Lösungsalternativen diskutiert. Weiter werden die Architektur, auf der Dienste basieren, und die Programmierungsmodelle, die bei der

Konzeption und Implementierung von Diensten verwendet werden, vorgestellt. Kapitel 3 stellt die Konzeption des Gesamtsystems vor und befasst sich mit der Konzipierung einzelner Dienste. Im Kapitell 4 werden die Details der Implementierung diskutiert. Kapitell 5 stellt die Ergebnisse vor. Abschließend gibt Kapitell 6 einen Ausblick auf möglichen Erweiterungen und Problemlösungen.

Die Verfügbarkeit leistungsfähiger Arbeitsplatzrechner und die Einführung breitbandiger Netzwerke beschleunigen den Trend zu verteilter Informationsverarbeitung sowie die Entwicklung verteilter, kooperativer Systeme. Verteilte Informationsverarbeitung ermöglicht Anwendern den Austausch von Informationen und die gemeinsame Nutzung entfernter Ressourcen und Dienste. Kooperative Anwendungen gestatten einer Gruppe von Anwendern die gemeinsame Nutzung von Anwendungssystemen, wobei jede Eingabe eines Anwenders unmittelbar an allen anderen Arbeitsplatzrechnern sichtbar wird [STE99]. Die im Rahmen dieser Arbeit entwickelten Kommunikationsdienste unterstützen einerseits die verteilte Informationsverarbeitung und stellen andererseits die kooperativen Anwendungen dar.

2 Grundlagen der Gruppenkommunikation

Bevor das eigentliche Konzept der Arbeit beschrieben werden kann, ist es notwendig die erforderlichen Grundlagen zu beschreiben, zu welchen die Konzepte der Gruppenkommunikation, die Client/Server-Architektur sowie die Konzepte der Netzwerkprogrammierung gehören.

2.1 Formen der Gruppenkommunikation

Im Rahmen der Gruppenkommunikation können verschiedene Kommunikationsformen in Abhängigkeit von der Anzahl der eingeschlossenen Sender und Empfänger unterschieden werden:

- Unicast (1 : 1)
- Multicast (1 : n)
- Concast (m : 1)
- Multipeer (m : n).

Unicast-Kommunikation entspricht der traditionellen Punkt-zu-Punkt-Kommunikation (1:1-Kommunikation) [WIT99]. Es existieren genau ein Sender und ein Empfänger. Daraus folgt, dass der Datenaustausch unidirektional erfolgt (siehe Abbildung 2.1). Die Daten fließen nur in einer Richtung. Um einen bidirektionalen Datenaustausch zu ermöglichen, sind zwei Unicast-Kommunikationsbeziehungen erforderlich.

Abbildung 2.1 Schematische Darstellung der Unicast-Kommunikation

Unicast ist auf die Kommunikation zwischen zwei Kommunikationspartnern beschränkt. Um eine Gruppenkommunikation zu unterstützen, müssten zwischen zwei Gruppenteilnehmern zwei Unicast-Kommunikationsbeziehungen aufgebaut werden. Dies resultiert bei einer Gruppengröße von ***n*** in ***n!/2*** Kommunikationsbeziehungen. Ein solcher Aufwand ist für große Gruppen nicht annehmbar.

Im Rahmen einer *Multicast-Kommunikation* sendet eine Datenquelle Daten an mehrere Empfänger (siehe Abbildung 2.2). Es handelt sich also um eine Erweiterung der Unicast-Kommunikation. Diese wird als 1:n-Kommunikation bezeichnet [WIT99]. Die Empfänger können Daten nur empfangen und selbst keine Daten senden. In diesem Sinne ist die Kommunikation also unidirektional.

Soll eine bidirektionale Gruppenkommunikation innerhalb einer Gruppe mit ***n*** Teilnehmern realisiert werden, so sind ***n*** Multicast-Kommunikationsbeziehungen erforderlich, eine pro Teilnehmer.

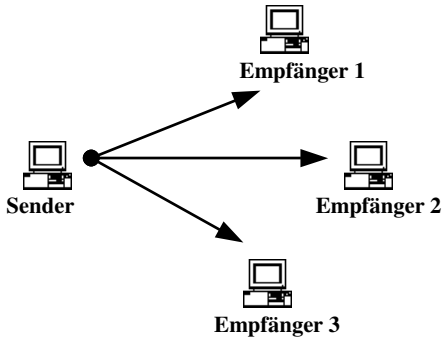

Abbildung 2.2 Schematische Darstellung der Multicast-Kommunikation

Bei einer *Concast-Kommunikation* sind mehrere Sender in der Lage, Daten an einen einzigen Empfänger zu senden (siehe Abbildung 2.3). Es handelt sich also um eine m:1-Kommunikation [WIT99]. Auch hier ist der Datenverkehr unidirektional von den Sendern hin zum Empfänger.

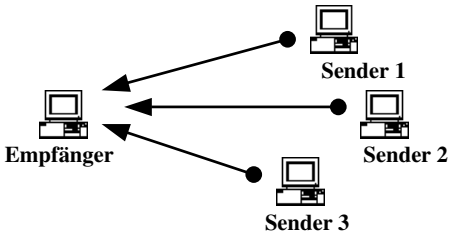

Abbildung 2.3 Schematische Darstellung der Concast-Kommunikation

Eine *Multipeer–Kommunikation* liegt dann vor, wenn mehrere Sender an mehrere Empfänger Daten schicken können. Das entspricht einer m:n-Kommunikationsbeziehung, die häufig als Mehrpunkt-Kommunikation bezeichnet wird [WIT99]. Bei der Kommunikation dieser Form werden keine Einschränkungen hinsichtlich der Anzahl von Sendern und Empfängern gemacht, was einerseits die vielfältigste Form darstellt, andererseits schwierig zu realisieren ist (siehe Abbildung 2.4).

Multipeer-Kommunikation lässt sich allerdings durch gleichzeitiges Betreiben mehrerer Multicast-Kommunikationsbeziehungen emulieren. Für jeden Sender wird eine Multicast-Kommunikation zu allen anderen Gruppenmitgliedern aufgebaut. Dies ist eine häufig gewählte technische Umsetzung.

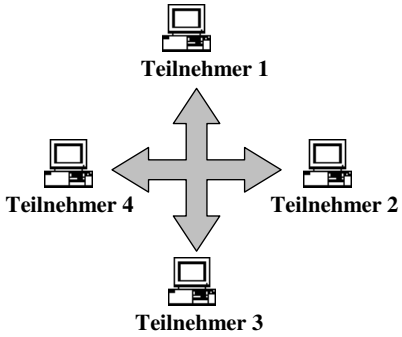

Abbildung 2.4 Schematische Darstellung der Multipeer-Kommunikation

Über die oben aufgeführten Kommunikationsformen hinaus werden heutzutage noch zwei weitere Varianten eingesetzt:

- *Anycast* und
- *Broadcast*.

Bei *Anycast-Kommunikation* findet der Datenaustausch per Unicast statt, wobei der Empfänger aus einer Gruppe heraus selektiert wird [c't00]. Eine typische Anwendung von Anycast kann in der Lokalisierung von Diensten in verteilten Systemen gesehen werden. Die Auswahl des Empfängers aus der Gruppe wird vom Kommunikationssystem getroffen. Der Benutzer hat darauf keinen Einfluss und kennt in der Regel die Gruppe der Empfänger nicht.

Die *Broadcast-Kommunikation* ist mit Multicast-Kommunikation vergleichbar, da hier ebenfalls genau ein Sender exsistiert. Die Gruppe der Empfänger ist allerdings nicht begrenzt, die Daten werden an alle versendet [c't00].

2.2 Spezielle Aspekte der Gruppenkommunikation

Da bei einer Gruppenkommunikation mehr als zwei Teilnehmer kommunizieren, müssen einige in der traditionellen Punkt-zu-Punkt-Kommunikationen bekannten Dienste und Funktionen erweitert werden. Hierzu gehört:

- Zuverlässigkeit,
- Flusskontrolle,
- Gruppenverwaltung.

2.2.1 Zuverlässigkeit

Nach der Definition liegt *ein zuverlässiger Dienst* nur dann vor, wenn alle Daten fehlerfrei, in der richtigen Reihenfolge und ohne Duplikate beim Empfänger ausgeliefert werden. Z.B. für das Whiteboard ist es teilweise erforderlich, dass alle Änderungen am Dokument durch einzelne Gruppenmitglieder bei allen Teilnehmern ausgeliefert werden müssen.

Ein zuverlässiger Dienst für die Punkt-zu-Punkt-Kommunikation wird typischerweise von der Transportschicht bereitgestellt. Aus dem Bereich der Internet-Protokolle ist hier TCP (Transmission Control Protocol) zu nennen.

Um einen zuverlässigen Gruppendienst realisieren zu können, sind Fehlerkontroll- und -behebungsverfahren erforderlich. Eine direkte Umsetzung der im Unicast eingesetzten Mechanismen auf Gruppenkommunikation ist nicht auf einfache Art und Weise möglich.

Jeder Empfänger würde Quittungen an den Sender senden, was im Falle einer großen Gruppe zu Leistungsengpässen führen kann (*Quittungs-Implosion*). Der Sender muss einen gewissen Zeitraum warten und Quittungen sammeln.

Die Zusammensetzung der Gruppe muss bekannt sein. Dynamische Änderungen der Gruppenzusammensetzung müssen den Sendern direkt bekannt gegeben werden.

Die in der Gruppe registrierten Teilnehmer müssen ständig verfügbar sein, da ein nicht reagierender Teilnehmer zum Abbruch der Kommunikation führen kann.

Bei einem *unzuverlässigen Gruppendienst* werden analog zur unzuverlässigen Punkt-zu-Punkt-Kommunikation keine Zusagen über die Auslieferung der Daten getroffen. Mechanismen zur Fehlerkontrolle und -korrektur können somit entfallen. Es kann also durchaus möglich sein, dass kein einziges Mitglied der Gruppe die Daten wirklich enthält.

Das UDP (Universal Datagramm Protokoll), das im Internet eingesetzt wird, stellt z.B. einen unzuverlässigen Dienst dar.

2.2.2 Flusskontrolle

Zur Regulierung des Datenflusses zwischen zwei Kommunikationspartnern sind Protokollmechanismen zur Fluss- und Staukontrolle erforderlich. Oftmals basieren Verfahren für die Flusskontrolle auf positiven Quittungen (**ACK**s, Acknowledgments).

Die Flusskontrolle ist meist direkt mit der Fehlerkontrolle und –behebung gekoppelt. Dies führt beim Sender im Falle einer großen Gruppe zu einer hohen Belastung, da er Quittungen von allen Empfängern erhalten kann und verarbeiten muss. Deshalb wird versucht, die Belastung vom Sender zum Empfänger zu verlagern. Solche Verfahren

werden als empfängerbasierte Verfahren bezeichnet, im Gegensatz zu den ursprünglichen senderbasierten Verfahren. Empfängerbasierte Verfahren führen dazu, dass keine positiven, sondern negative Quittungen (NACKs) gesendet werden, die einen Bescheid über nicht erhaltene Daten geben (siehe Abbildung 2.5) []. Durchgeführte Untersuchungen [WIT99] haben gezeigt, dass empfängerorientierte Verfahren zu einem höheren Durchsatz als senderorientierte Verfahren führen.

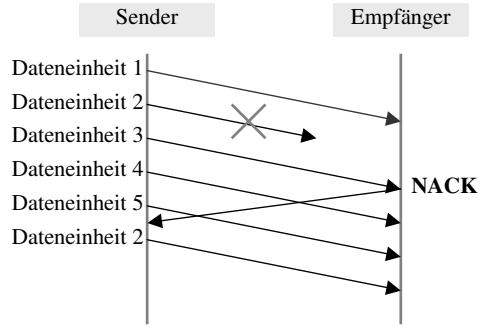

Abbildung 2.5 Schematische Darstellung der Flusskontrolle mit negativen Quittungen

Die *Reaktion des Senders* nach dem Erhalt von Quittungen kann sofort erfolgen oder auch verzögert sein.

Es existieren fensterbasierte und ratenbasierte Verfahren zur Flusskonrtolle, wobei die ratenbasierte Flusskontrolle für die Gruppenkommunikation vorteilhaft ist, da sie die Verarbeitung vieler Kontroll-Dateneinheiten reduziert [GLB97]. Die ratenbasierte Flusskontrolle regelt, wie viele Daten in welchem Zeitraum vom Sender gesendet werden dürfen. Hierzu sind zwei Parameter zu bestimmen : Burst-Grösse (das Datenvolumen, das in einem Zeitraum *T* gesendet werden darf) und die Burst-Rate (der minimale zeitliche Abstand t zweier unmittelbar aufeinanderfolgender Burts) . Die beiden Burts-Parameter müssen sich innerhalb der Gruppe am schwächsten Empfänger orientieren und sind beim Beitreten bzw. Verlassen von Gruppenmitgliedern zu adaptieren.

2.2.3 Gruppenverwaltung

Für die Gruppenkommunikation wird ein neues Adressierungsschema benötigt, da nicht mehr ein einzelner Empfänger (Unicast-Kommunikation) oder alle Empfänger (Broadcast-Kommunikation), sondern eine explizite Gruppe von Empfängern angesprochen werden soll.

Es existieren folgende Alternativen:

- Adressierung über Empfängerlisten oder
- Adressierung über spezielle Gruppenadressen.

Bei der Verwendung von Empfängerlisten hat der Sender Zugriff auf eine Liste der *Unicast-Adressen* aller Empfänger in der Gruppe. Er kann damit alle Gruppenmitglieder identifizieren und ihnen die jeweiligen Daten senden. Die Empfängerliste muss bei jedem Beitritt bzw. Verlassen eines Mitglieds der Gruppe aktualisiert werden. Die Dateneinheiten werden dann per Unicast-Kommunikation an die einzelnen Empfänger verteilt.

Bei der Verwendung spezieller Gruppenadressen wird die gesamte Gruppe durch eine einzige Adresse identifiziert. Der Sender sendet dann die Daten an diese Gruppenadresse. Die Empfänger einer Gruppe können damit nicht individuell identifiziert werden. Der Vorteil liegt darin, dass das mehrfache Aussenden beim Sender entfällt. Der Sender verschickt die mit der Gruppenadresse versehene Dateneinheit genau einmal. Die Empfänger müssen in der Lage sein zu erkennen, dass Daten mit bestimmten Gruppenadressen für sie relevant sind und damit empfangen werden müssen. Die netzinternen Zwischensystemen müssen auf der Basis dieser Gruppenadressen die Entscheidungen für das Weiterleiten von Daten treffen. Sie müssen hierzu gegebenenfalls die Dateneinheiten kopieren und an mehrere Ausgangslinks weiterleiten können.

Bei der Vergabe einer Gruppenadresse kann zwischen

- zentraler und
- dezentraler

Vergabe unterschieden werden.

Die zentrale Adressvergabe wird durch eine autorisierte Institution durchgeführt. Vor der Etablierung einer Gruppe muss dort eine Adresse beantragt werden. Diese Adresse behält

ihre Gültigkeit auch über die Dauer einer aktiven Kommunikationsbeziehung dieser Gruppe hinaus. Sie wird durch explizites Abmelden bei der autorisierten Institution ungültig. Die zentrale Adressvergabe hat bei der langfristigen Vergabe von Gruppenadressen Vorteile, ist aber für eine kurzlebige Gruppe nachteilig.

Bei der dezentralen Adressvergabe wird bei der Etablierung einer Gruppe lokal eine Gruppenadresse ausgewählt. Hierbei muss sichergestellt werden, dass die dynamisch vergebene Adresse eindeutig ist, also nicht bereits von einer anderen Gruppe benutzt wird. Im Internet wird deshalb ein entsprechendes Kontrollpaket versendet. Damit soll sichergestellt werden, dass die Gruppenadresse nicht anderswo verwendet wird. Da solche Kontrollpakete im Netz verloren gehen können, ist damit keine absolute Garantie für Eindeutigkeit der Adresse möglich.

Alternativ können auch Server für die Multicast-Adressen im Netz etabliert werden, von denen dann jeweils eine Adresse abgerufen werden kann. Die Gruppenadresse wird, im Gegensatz zur zentralen Vergabe, beim Beenden der Kommunikation wieder freigegeben und kann dann von einer anderen Gruppe belegt werden.

Die Gruppenverwaltung ist für die Verwaltung der Mitglieder einer Gruppe verantwortlich. Diese Verwaltung kann zentral oder verteilt realisiert sein. Eine verteilte Gruppenverwaltung hat Vorteile hinsichtlich der Skalierbarkeit. Nachteile sind allerdings in der Bereitstellung der konsistenten Zustandsinformation der Gruppe zu sehen. Weiterhin ist zu klären, inwiefern sich die Gruppenzusammensetzung dynamisch ändern darf und wer der Gruppe beitreten bzw. die Gruppe verlassen darf.

2.3 Reichweite von Multicast-Gruppen

Die Reichweite einer IP-Multicast-Dateneinheit ist nicht unbeschränkt. Unter dem Begriff Reichweite wird der Bereich verstanden, in dem die Dateneinheit weitergeleitet wird. Hat die Dateneinheit die Grenze dieses Bereichs erreicht, wird sie nicht mehr weitergeleitet und verworfen. Dafür sprechen viele Gründe, z.B.:

- Begrenzung von gefluteten Netzbereichen
- Mehrfachnutzung von Multicast-Adressen
- Privatheit

Wird ein Routingprotokoll verwendet, bei dem das Netz geflutet wird, sorgt eine eingeschränkte Reichweite dafür, dass dadurch nicht das gesamte Netz unnötig belastet wird.

Adressen sind generell eine knappe Ressource, dies gilt auch für Multicast-Adressen. Mit Hilfe einer eingeschränkten Reichweite können Multicast-Adressen mehrfach verwendet werden, solange sich die Bereiche der Gruppen nicht überschneiden. Damit kann der Adressraum effizienter genutzt werden.

Weiterhin können Reichweitenbegrenzungen auch dazu dienen, einen gewissen Grad an Privatheit zu gewährleisten, indem Stationen, die sich außerhalb des Bereichs befinden, der Multicast-Gruppe nicht beitreten können.

Es gibt zwei Verfahren zur Reichweitenbegrenzung:

- ✓ Reichweitenbegrenzung anhand des TTL-Wertes und
- ✓ Administrative Reichweitenbegrenzung.

2.3.1 TTL-Bereiche

Am häufigsten wird das Verfahren eingesetzt, bei dem das TTL-Feld im Kopf der IP-Dateneinheit zur Reichweitenbegrenzung verwendet wird. Dieses Verfahren wird auch als TTL-Scoping bezeichnet [SIN97].

Das TTL-Feld gibt die maximale Lebensdauer einer IP-Dateneinheit an. Jedes Zwischensystem, welches eine Dateneinheit weiterleitet, dekrementiert dazu den Wert des Feldes. Hat das Feld den Wert Null erreicht, wird die Dateneinheit verworfen und somit nicht weitergeleitet.

Zur Reichweitenbegrenzung von Multicast-Dateneinheiten ist dieser TTL-Mechanismus allerdings nicht ausreichend [TAN98]. Die Anzahl der zu passierenden Zwischensysteme ist im allgemeinen für zwei Empfänger verschieden. So dass für einen bestimmten TTL-Wert ein Empfänger nicht erreicht wird, während der Wert für einen anderen zu groß ist und die Dateneinheit die Grenze überschreiten kann.

Um dieses Problem zu lösen, wird beim IP-Multicast der TTL-Wert zusätzlich mit einem Schwellwert verglichen. Ist der TTL-Wert nach der Dekrementierung kleiner als der

Schwellwert, wird die Dateneinheit verworfen, auch wenn der TTL-Wert größer als Null ist. Die Schwellwerte sind in der Tabelle 2.1 aufgeführt.

Schwellwert	Reichweite
0	Begrenzung auf einen Knoten
1	Begrenzung auf ein Subnetz
32	Begrenzung auf eine Domäne
48	Begrenzung auf ein Land (in vielen Ländern Europas)
64	Begrenzung auf eine Region
128	Begrenzung auf einen Kontinent
255	Unbegrenzt

Tabelle 2.1 Liste der Schwellwerte für das TTL-Scoping

2.3.2 Administrative Bereiche

Bei diesem Verfahren wird eine Reichweite eines Multicasts anhand der Multicast-Adresse festgemacht. Anders gesagt, die Multicast-Adresse wird nach der gewünschten Reichweite gewählt. Die bisher definierten administrativen Bereiche sind in der Tabelle 2.2 aufgeführt.

Adressbereich	Reichweite
239.255.0.0 – 239.255.255.255	lokaler Bereich
239.253.0.0 – 239.254.255.255	erweiterter lokaler Bereich
239.192.0.0 – 239.195.255.255	organisatorischer Bereich
239.0.0.0 – 239.191.255.255	erweiterter organisatorischer Bereich

Tabelle 2.2 Liste der definierten administrativen Bereiche

Dieses Konzept wird auf Netzebene dadurch umgesetzt, dass die M-Router (spezielle *Multicast-Router*) an den Bereichgrenzen IP-Dateneinheiten nur innerhalb des jeweiligen Bereichs weiterleiten bzw. generieren [WIT99]. M-Router werden weiter noch ausführlich beschrieben (siehe Kapitel 2.4.2). Hierzu müssen die M-Router entsprechen konfiguriert werden, da sie nicht selbst entscheiden können, ob sie an einer Bereichsgrenze angesiedelt sind.

2.4 Unterstützung im Kommunikationssystem

Eine Unterstützung der Gruppenkommunikation kann in verschiedenen Bereichen eines Kommunikationssystems angesiedelt sein. Kommunikationssysteme werden in der Regel durch hierarchische Schichten beschrieben, entweder gemäss dem *ISO/OSI-Referenzmodell* oder gemäss dem *Internet-Modell*, die die abstrakte Darstellung, d.h. unabhängig von der konkret eingesetzten Hard- und Software, bieten. Die beiden Modelle mit der TCP/IP-Protokollfamilie werden in der Abbildung 2.6 dargestellt [MSDN].

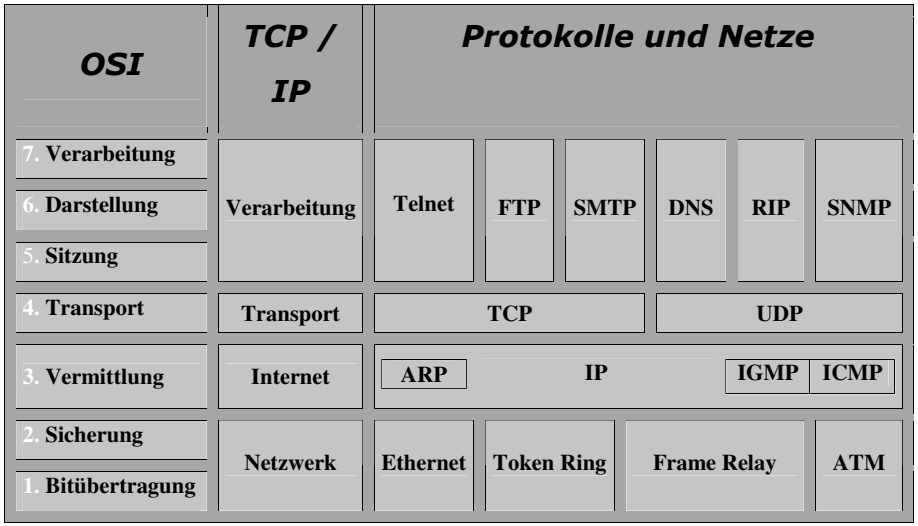

Abbildung 2.6 Schematische Darstellung des ISO/OSI–Referenzmodells, des Internet-Modells und der TCP/IP- Protokollfamilie

Aus Sicht der Datenübertragung ist die Unterstützung in den Schichten 2 bis 4 des ISO/OSI-Referenzmodells bzw. entsprechenden Schichten des Internetmodells erforderlich. Diese Schichten haben die Aufgabe den Datentransfer zu regulieren und den Anwendungen einen entsprechenden Kommunikationsdienst bereitzustellen. Im einzelnen sind dies die folgenden Schichten:

- *Sicherungsschicht*
- *Vermittlungsschicht*
- *Transportschicht*

2.4.1 Schichten des Kommunikationssystems

2.4.1.1 Sicherungsschicht

Die Sicherungsschicht bietet die Möglichkeit die Datenrahmen (*Data Frames*) frei von unerkannten Übertragungsfehlern von einem Kommunikationspartner zum anderen zu transferieren. Die Sicherungsschicht im ISO/OSI-Referenzmodell umfasst insbesondere die MAC-Teilschicht (Media Access Control), die für die Regelung des Medienzugangs verantwortlich ist. Im Internet-Modell ist sie Bestandteil der Komponente Netzwerk.

2.4.1.2 Vermittlungsschicht

Die Vermittlungsschicht stellt eine zentrale Komponente in Kommunikationssystemen dar, deren Hauptaufgaben die netzweite eindeutige Adressierung und das zielgerichtete Weiterleiten (Routing).

Für Internet-Adressen existieren verschiedene Adressen, die sich an der Größe des Netzes orientieren (siehe Abbildung 2.7), die Adressen der Klasse A, B und C.

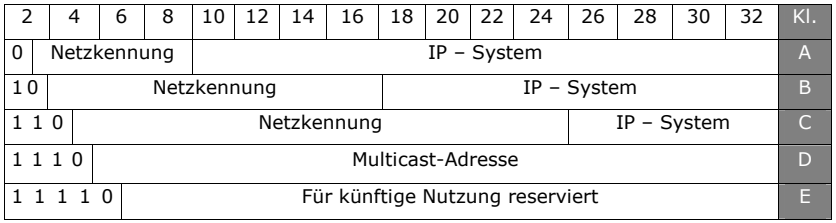

Abbildung 2.7 Darstellung des IP-Adressformats

Sie unterscheiden sich jeweils in der Länge der beiden Adresskomponenten, die einer Netzkennung bzw. einem IP-System zugeordnet werden. IP-Adressen der Klasse D sind Multicast-Adressen. Sie sind intern nicht weiter strukturiert. Die für zukünftige Verwendung reservierte Adressklasse E hat momentan keine praktische Bedeutung. Beschrieben werden die IP-Adressen meistens durch vier Dezimalzahlen, die byteweise durch einen Punkt getrennt werden.

Die Adressbereiche der Klassen A, B, C, D und E sind in der Tabelle 2.3 dargestellt.

Klasse	Host-Adressbereich
A	1.0.0.0 bis 127.255.255.255
B	128.0.0.0 bis 191.255.255.255
C	192.0.0.0 bis 223.255.255.255
D	224.0.0.0 bis 239.255.255.255
E	240.0.0.0 bis 247.255.255.255

Tabelle 2.3 Liste der Adressbereiche in den Klassen A-E

Weiter sind folgende spezielle IP-Adressen für eine besondere Verwendung reserviert:

- Die Adresse 0.0.0.0 bezeichnet den lokalen Host.
- IP-Adressen mit der Netzwerknummer 0 bezieht sich auf das lokale Netz. Die Hosts können darin adressiert werden, ohne dass die eigentliche Netzwerknummer bekannt ist.
- Die Adresse 255.255.255.255 bildet die sogenannte Broadcast-Adresse. Sie wird verwendet, um alle Hosts im lokalen Netz zu erreichen.
- Die IP-Adressen mit der nur aus Einsen bestehenden Hostnummer erlauben es, die Broadcast-Pakete an das entfernte Netz zu senden.

- Datenpakete, die mit der Zieladresse 127.xx.yy.zz versehen werden, werden nicht auf das Übertragungsleitung ausgegeben. Diese auch als Loopback genannte Adresse wird für Schleifentests verwendet.
- Die IP-Adressen der Klasse D sind Multicast-Adressen.

00000000000000000000000000000000		Dieser Host
0 0 0 . . . 0 0	Host	Host in diesem Netz
11111111111111111111111111111111		Broadcast im lokalen Netz
Netz	1 1 1 1 . . . 1 1 1 1	Broadcast in einem entfernten Netz
127	(beliebig)	Schleife (Loopback)

Abbildung 2.8 Darstellung spezieller IP-Adressen im Netz

Die wesentlichen Entwürfe, die für IP–Multicast eingeführt wurden, sind Multicast–Internet–Adressen, Kommunikationsgruppen sowie Multicast–Routing. Die Multicast–Adressen sind erforderlich, damit die Daten nur jeweils einmal in das Internet gesendet werden müssen. Es handelt sich dabei um Klasse–D–Adressen. Kommunikationsgruppen im Internet sind offene, dynamische Gruppen.

Die Zugehörigkeit kann sich im Verlauf einer Kommunikationsbeziehung ändern. Die Gruppen werden durch eine Multicast-IP-Adresse identifiziert. Es können sowohl temporäre als auch permanente Gruppen unterstützt werden. In der Tabelle 2.4 werden Beispiele für permanente Gruppen aufgelistet.

Klasse-D – Adresse	Permanente Gruppe
224.0.0.0	Reserviert
224.0.0.1	Alle Systeme eines Subnetzes
224.0.0.2	Alle Router eines Subnetzes
224.0.0.3	nicht zugewiesen
224.0.0.4	Alle DVMRP-Router
224.0.0.9	Router mit Routing Information Protocol (RIP) Version 2
224.0.1.1	Network Time Protocol (NTP)
224.0.1.9	Multicast Transport Protocol (MTP)

Tabelle 2.4 Beispiele für permanente Gruppen

Internet-Adressen sind derzeit 32 Bit lang; in der neuen Version von IP-IPv6 werden Adressen einer Länge von 128 Bit verwendet. Diese Version ist heute allerdings noch nicht im regulären Einsatz.

2.4.1.3 Transportschicht

Aufgabe der Transportschicht ist es, den Datenaustausch zwischen den Kommunikationspartnern zu unterstützen. Es kann hier typischerweise zwischen zuverlässigen und unzuverlässigen Transportdiensten unterschieden werden.

Heutige Transportprotokolle, wie etwa TCP, basieren auf der reinen Punkt-zu-Punkt-Kommunikation. Zur Fehlerkontrolle werden dabei Quittungen vom Empfänger an den Sender geschickt, um den ordnungsgemäßen Empfang der Daten zu signalisieren. Das einfache Erweitern dieses Mechanismus auf eine Gruppe würde bei großen Gruppen zur Quittungs-Implosion führen. Der weitere Faktor, der auch berücksichtigt wird, ist die räumliche Verteilung der Gruppenmitglieder. Quittungen vom entfernten Standort benötigen mehr Zeit für die Datenübertragung, was die Effizienz der Kommunikation innerhalb der Gruppe sehr beeinträchtigen kann. (Der Sender verbringt viel Zeit damit, dass auf Quittungen vom Empfänger wartet. Folglich reduziert sich der Datendurchsatz).

2.4.1.4 Anwendungsschicht

Multicast-Unterstützung wird bei vielen neuen Anwendungen teilweise von den Anwendungen selbst übernommen. Auf diese Art und Weise können die im Internet etablierten Transportprotokolle benutzt werden. Dabei wird im wesentlichen auf UDP aufgesetzt. Die erforderliche Multicast-Unterstützung wird dann in der Anwendung selbst realisiert.

In diesen Anwendungen wird dann eine zuverlässige Multicast-Kommunikation auf der Basis von UDP, einem unzuverlässigen Protokoll der Transportschicht, realisiert.

2.4.2 Multicast Backbone

Die Einführung eines neuen Dienstes wie dem der Gruppenkommunikation (Gruppenadressen und ein Gruppenmanagement) erfordert, dass sowohl Endsysteme als

auch Zwischensysteme mit der neuen Technologie ausgerüstet werden. Sie sollen in der Lage sein, IP-Multicast-Adressen zu verstehen, das Gruppenmanagement-Protokoll IGMP und ein Multicast-Routing durchzuführen. Bei solcher Größe des Internets ist die Umstellung aller Systeme schwierig durchzuführen, da dafür neue Hardware eingesetzt und neue Software installiert werden müssen. Als Übergangslösung wurde ein virtuelles Multicast-Netzwerk als Overlay-Netzwerk im Internet definiert.

Da nicht alle Router Multicast-fähig sind, werden Tunnel zwischen den Multicast-Routern (M-Router) aufgebaut (siehe Abbildung 2.9). Innerhalb eines Tunnels werden die Daten wie bei Unicast-IP-Routing transportiert.

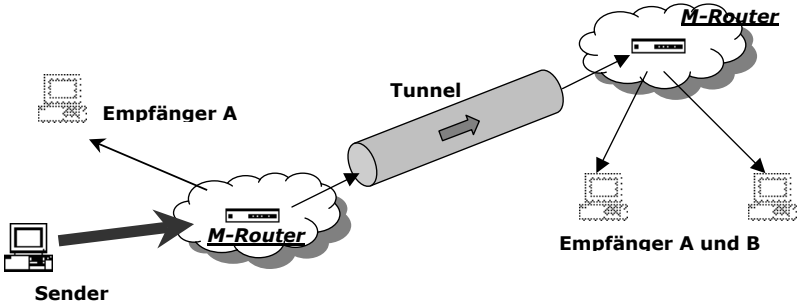

Abbildung 2.9 Zwei MBone-Domänen, die über einen Tunnel miteinander verbunden sind.

2.4.2.1 MBone-Anwendungen

Mit der Gründung des MBone steht eine funktionsfähige Multicast-Plattform zur Verfügung. Heute existieren eine Vielzahl von Anwendungen und Systemen, die auf dem IP-Multicast basieren. Einigen davon sind weiter vorgestellt.

Video Conference (VIC) ist eine Videoanwendung, wobei nur die Übertragung und Darstellung dieses eines Videostroms unterstützt wird. Audiodaten können damit nicht übermitteln können.
VIC unterstützt eine ganze Reihe von Videokompressionsverfahren (H.261, nv, celb, nvdct, bvc, MJPEG). Da diese Verfahren unterschiedliche Eigenschaften bzgl. Der erreichbaren Kompressionsrate aufweisen, ist VIC für den Einsatz in sehr

unterschiedlichen Umgebung geeignet, je nach zur Verfügung stehender Bandbreite und Fehlerrate.

Das *Visual Audio Tool* (VAT) ist quasi das Pendant zu VIC. VAT erlaubt die Durchführung von Audiokonferenzen zwischen mehreren Teilnehmern. Für jeden Teilnehmer wird dessen Name oder andere von ihm gewählte Zeichenkette angezeigt. Damit kann jeder Teilnehmer leicht identifiziert werden. Der aktive Teilnehmer wird mit Hilfe unterschiedlicher Farbe angezeigt, was ersichtlicht, wer der aktuelle Sprecher ist.

Das *Robust Audio Tool* (RAT) bietet eine dem VAT ähnliche Funktionalität. Die Anwendungen VAT und RAT sind zueinander kompatibel; sie können in derselben Sitzung nebeneinander eingesetzt werden. Im Unterschied zu VAT ist bei RAT eine Fehlerkorrektur vorgesehen. Gegenwärtig wird jedoch nur eine sehr einfache Methode zur Behebung von Übertragungsfehlern unterstützt.

Mit der Anwendung *Session Directory* (SDR) können die Informationen zu laufenden oder zukünftigen Sitzungen angezeigt oder initiiert werden. Für diesen Zweck wird im MBone eine Multicast-Gruppe reserviert, in der periodisch Sendungen im MBone bekanntgegeben werden.

2.5 Die Client/Server-Architektur

Dem Distance-Learning-System liegt das Client-Server-Modell zugrunde. Weiter werden die Grundlagen der Client-Server-Architektur beschrieben.

Die Client/Server-Technologie hat in der Anwendung der Rechentechnik zum Paradigmenwechsel geführt [ORF97]. Großen Mainframe-Applikationen wurden durch Anwendungen ersetzt, die miteinander über kürzere oder längere Verbindungsstrecken kommunizierten. Der Client (typischerweise ein PC) stellt dabei eine grafische Benutzeroberfläche zur Verfügung, während der Server Zugriffe auf die verteilten Ressourcen (in der Regel verschiedene Arten der Datenbanken) bietet (siehe Abbildung 2.10). Die neueste Entwicklung in diesem Bereich heißt verteilte Objekte und Internet.

Ein Client-Server-Modell hat folgende Charakteristika:

Dienste:

Client/Server bezeichnet vor allem eine Beziehung zwischen Prozessen, die auf verschiedenen Rechnern laufen. Der Serverprozess ist ein Dienstanbieter; der Client nimmt Dienste in Anspruch. Im wesentlichen bietet das Client-Server-Modell eine klare Trennung von Funktionen, basierend auf dem Dienst-Konzept.

Gemeinsame Nutzung von Ressourcen:

Ein Server kann gleichzeitig mehrere Clients bedienen, und den Zugriff auf gemeinsam genutzte Ressourcen regeln.

Asymmetrisches Protokoll:

Es besteht eine n:1-Beziehung zwischen Clients und Server („viele Clients – ein Server"). Clients initiieren immer den Dialog, indem sie einen Dienst anfordern. Server warten passiv auf eingehende Client-Anfragen.

Transparenz des Standortes:

Der Server ist ein Prozess, der sich auf demselben Rechner wie der Client oder auf einem anderen Computer im Netzwerk befinden kann. Client/Server-Software verbirgt den Standort des Servers vor den Clients, indem die Anfragen bei Bedarf umgeleitet werden. Ein Programm kann ein Server, ein Client oder beides zugleich sein.

Austauschbarkeit:

Die ideale Client/Server-Software ist unabhängig von Hardware- und Betriebssystemenplattformen.

Nachrichten basierter Datenaustausch:

Clients und Server sind lose gekoppelte Systeme, die mittels Nachrichten miteinander kommunizieren. Über Nachrichten werden die Dienstanforderungen von Clients an den Server und Antworten von Server an die Clients übermittelt.

Verkapselung von Diensten:

Der Server ist der „Spezialist". Eine Nachricht von einem Client übermittelt Anforderung eines Dienstes. Der Server entscheidet selbst, wie die Aufgabe erledigt werden soll. Server können ohne Einfluss auf Clients modifiziert werden, solange das Nachrichtenformat nicht verändert wurde.

Skalierbarkeit :

Client/Server Systeme können vertikal und horizontal skaliert werden. Vertikal skalierbar bedeutet, dass sich die Performance nicht oder nur geringfügig ändert, wenn der Server auf eine langsamere oder schnellere Maschine verlagert wird. Horizontal skalierbar bedeutet, dass die Performance kaum beeinflusst wird, wenn Clients hinzugefügt oder entfernt werden.

Integrität:

Der Quellcode des Servers und seine Daten werden zentral verwaltet. Das bedeutet niedrige Wartungskosten und garantiert die Integrität der gemeinsamen Daten.

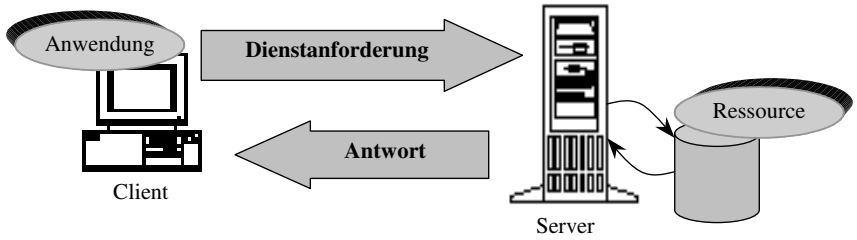

Abbildung 2.10 Schematische Darstellung der Client/Server-Architektur

Es existieren mehrere Arten von Servern. Weiter sind einige wenige aufgelistet, die am häufigsten gebraucht werden und die für das Gesamtsystem relevant sind:

2.5.1 Fileserver

Beim *Fileserver* sendet ein Client die Dateianforderungen an den Server. Es ist eine sehr primitive Form des Dienstes, die eine Menge vom Nachrichtenaustausch über das Netzwerk benötigt, um die gewünschten Dateien zu finden. Fileserver werden für die Nutzung von gemeinsamen Archiven gebraucht, z.B. Dokumenten, Bildern usw., die von mehreren Benutzern bearbeitet werden sollen[ORF99].

2.5.2 Datenbankserver

Die *Datenbankserver* bearbeiten SQL-Anfragen, die der Client als Nachrichten an den Datenbankserver sendet. Die Ergebnisse jeder SQL-Anfrage werden über das Netzwerk zurückgeschickt [ORF99]. Der Code, der die SQL-Anfrage bearbeitet, und die Daten befinden sich auf demselben Rechner. Der Server benutzt seine eigene Rechenleistung, um die benötigten Daten zu finden, und liefert nur die gefilterten Ergebnisse an den Client zurück, was die Netzbelastung niedrig hält und die effizientere Nutzung der verteilten Rechenkapazität erlaubt. Der Applikationscode befindet sich auf dem Clientrechner.

2.5.3 Transaktionsserver

Bei einem *Transaktionsserver* ruft der Client die *Remote-Prozeduren* auf, die sich mit der SQL- Datenbank auf dem Server befinden [ORF99]. Diese Prozeduren führen dann eine bestimmte Anzahl von SQL- Anfragen aus. Der Netzwerktransfer besteht aus einer einzigen Anfrage/Antwort-Nachricht, im Unterschied zum Datenbank-Server, wo jede einzelne SQL- Anfrage mit einer Nachricht beantwortet wird. Diese SQL- Befehle werden entweder vollständig erfolgreich ausgeführt, oder, im Falle eines Fehlers, als ganzes zurückgewiesen. Solche Gruppen von SQL- Befehlen werden Transaktionen genannt.

Für eine Client/Server-Anwendung müssen Komponenten sowohl für den Client als auch für den Server bereitgestellt werden. Die Client-Komponente enthält in der Regel eine grafische Benutzeroberfläche. Die Server-Komponente besteht aus SQL- Transaktionen auf einer Datenbank. Solche Anwendungen werden *OLTP-Anwendungen* (*Online Transaction Processing*) genannt. Sie sind typischerweise unternehmenskritisch und erfordern eine strenge Kontrolle über die Sicherheit und die Integrität der Datenbank.

2.5.4 Objektserver

Bei einem *Objektserver* werden die Client/Server-Anwendungen als ein Satz von Kommunikationsobjekten konzipiert [ORF99]. Die Clientobjekte kommunizieren mit den Serverobjekten über einen *Object Request Broker* (*ORB)*. Der Client ruft die Methoden des entfernten Objekts auf. Der ORB lokalisiert eine Instanz der Objektserverklasse, ruft

die gewünschte Methode auf und gibt das Ergebnis an das Clientobjekt zurück. Das Serverobjekt muss dabei die parallele Verarbeitung und gemeinsame Nutzung unterstützen. Heute wird das alles von den ORBs und der neuen Generation von CORBA Application Servern erledigt. Es gibt mehrere kommerzielle ORBs, die den CORBA-Standard (*CORBA - Common Object Request Broker Architecture*) der Object Management Group erfüllen, z.B. Orbix von Iona, VisiBroker von Inprise usw.
CORBA ist auch die fundamentale Technologie für das *Enterprise JavaBeans Komponenten Model (EJB)*.

Allerdings ist CORBA nicht der einzige Einsatz, der auf diese Technologie aufsetzt. Microsoft hat seinen eigenen ORB, der *Distributed Component Object Model* oder kurz DCOM genannt wird. DCOM ist die grundlegende Technologie im *ActiveX Komponenten Modell*. Der *Microsoft Transaction Server (MTS)* ist ein Applikations-Server für die ActiveX- Komponente.

2.5.5 Webserver

Das World Wide Web ist die erste wirklich weltweite Client/Server-Applikation [ORF99]. Dieses neue Client/Server-Modell besteht aus kleinen, portablen Clients, die mit den übergroßen *Web-Application-Servern* kommunizieren. In ihrer einfachsten Form liefern die Webserver Dokumente zurück, die von Clients angefordert werden. Die Clients kommunizieren mit dem Server über ein Protokoll namens *HTTP*. Dieses Protokoll definiert einen einfachen Satz von Befehlen; die Parameter werden als Zeichenketten übergeben.

Das Web-Client/Server-Modell befindet sich zur Zeit in der Entwicklung. Die Web-Technologie verschmilzt zunehmend mit der Technologie der verteilten Objekte zu einer mehr und mehr interaktiv orientierten Form der Client/Server Architektur. Diese neue Form wird *„Object-Web*" genannt. Java-Applets und die Browser mit der CORBA-Unterstützung sind die ersten Manifestationen dieses neuen Objekt-Webs. Bei Microsoft wird dieses Modell durch die ActiveX- Komponenten repräsentiert, die über COM+ oder HTTP kommunizieren.

Funktionell sind die Webserver ähnlich zu den Objektservern. In naher Zukunft wird es aber unmöglich sein, zwischen ihnen zu differenzieren.

Zum Beispiel ist schon jetzt ein MTS Distributed Object Server in der Microsoft Welt gleichzeitig auch ein Web Application Server und benutzt den HTTP-Server von Microsoft als Front-End.

2.6 Netzwerkprogrammierung

Es gibt viele Techniken, um eine Netzwerkanwendung zu entwerfen. Im folgenden Kapitel werden zwei solche Technologien, die in dieser Arbeit auch genutzt wurden, vorgestellt:

- Windows–Sockets-Schnittstelle
- WinInet-Schnittstelle

2.6.1 Sockets

Die Sockets wurden im Jahre 1981 von Unix BSD 4.2 (*Berkeley Software Distribution*) vorgestellt und dienten als eine allgemeine Schnittstelle, die eine Kommunikation zwischen Unix-Rechnern über das Netzwerk ermöglichen sollte. Heute werden die Sockets von praktisch allen Betriebsystemen unterstützt. Das *Windows-Sockets-API*, auch als *WinSock* genannt, ist eine herstellerübergreifende Spezifikation, die den Einsatz von TCP/IP unter Windows-Betriebsystemen standardisiert (siehe Abbildung 2.11). Das WinSock-API basiert auf der Sockets-Schnittstelle von Berkeley.

Im BSD-Unix Betriebsystem sind die Sockets ein Bestandteil des Kernels. Die Windows-Systeme stellen die Sockets in Form von Bibliotheken zur Verfügung. Man kann sagen, dass die Sockets heutzutage den portablen *de-facto-Standard* für die Hersteller von Netzwerkapplikationen über TCP/IP- Netzwerke bilden.

Ein Socket ist ein Kommunikationsendpunkt, d. h. ein Objekt, durch das eine Windows-Sockets-Anwendung Datenpakete über das Netzwerk sendet oder empfängt. Ein Socket hat einen Typ und wird einem in Ausführung befindlichen Prozess zugeordnet und kann zudem einen Namen haben. Derzeit tauschen Sockets Daten nur mit anderen Sockets in derselben "Kommunikationsdomäne" aus, in der die Internet Protocol Suite verwendet wird.

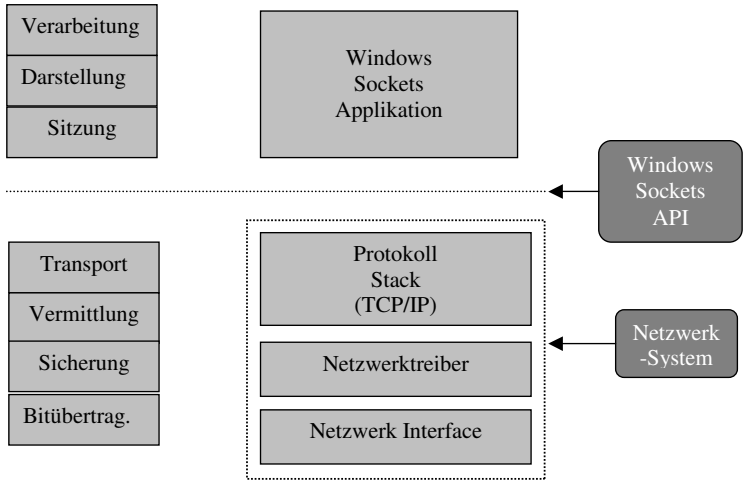

Abbildung 2.11 Schematische Darstellung des WinSock-Models im Schichtenmodell

Jedes Socket-Objekt wird einer Adresse zugeordnet. Die Socket-Adresse besteht aus zwei Teilen: einer Internet-Adresse und einer Portnummer. Wie eine IP-Adresse aufgebaut ist, wurde bereits ausführlich besprochen.

Ein *Port* kennzeichnet einen eindeutigen Vorgang, für den ein Dienst bereitgestellt werden kann. Im vorliegenden Kontext wird ein Port mit einer Anwendung verknüpft, die Windows-Sockets unterstützt. Es wird mit einer 16-bit langen Integerzahl repräsentiert. Die Idee dabei ist, jede Windows-Sockets-Anwendung eindeutig zu identifizieren, so dass mehr als eine Windows-Sockets-Anwendung auf einem Computer zur selben Zeit ausgeführt werden kann.
Die Portnummern unter 256 nennt man allgemein bekannte Ports (*well-known Ports*). Sie sind für Standarddienste reserviert, wie z. B. FTP (Port 21). Man sollte es vermeiden, diese Portnummern für eigene Anwendungen zu verwenden, es sei denn, man stellt diese Art von Dienst zur Verfügung, oder benutzt sie. Die Liste allgemein bekannter Ports ist in der RFC 1700 enthalten.

Um es der Windows-Sockets-DLL zu überlassen, einen passenden Port auszuwählen, wird 0 als Portnummer übergeben. Die Portnummer wird dann vom Betriebsystem aus dem Bereich größer als dezimal 1.024 gewählt.

Ein Socket kann für mehrere Verbindungen gleichzeitig benutzt werden. Anders ausgedrückt, zwei oder mehr Verbindungen enden auf dem gleichen Socket. Verbindungen werden nach der Zugriffsnummer (Socket-Handle) an beiden Enden identifiziert.

Sockets sind in mindestens drei Kommunikationszusammenhängen äußerst nützlich:

- Client-Server-Modellen
- Peer-to-Peer-Szenarien (z. B. Chat)
- Aufruf von Remote-Prozeduren (RPC), bei denen eine Nachricht als Funktionsaufruf interpretiert wird

Es gibt zwei Socket-Typen:

- *Stream-Sockets*
  Sie stellen einen Datenfluss ohne Satzgrenzen her - ein Stream aus Bytes. Die Streams werden garantiert abgeliefert, sind in der richtigen Reihenfolge und enthalten keine Duplikate.
- *Datagramm-Sockets*
  Sie unterstützen einen satzorientierten Datenfluss, der nicht garantiert abgeliefert wird, eventuell nicht mehr in der Reihenfolge der ursprünglichen Sendung ist und Duplikate enthalten kann.

"Richtige Reihenfolge" bedeutet, dass die Pakete in der Reihenfolge empfangen werden, in der sie gesendet wurden. "Keine Duplikate" bedeutet, dass Sie ein bestimmtes Paket nur einmal erhalten. Beide Socket-Typen sind bidirektional: Sie sind Datenflüsse, die in beide Richtungen gleichzeitig übertragen werden können (Vollduplex).

Stream-Sockets stellen einen Datenfluss ohne Satzgrenzen bereit - ein Stream von Bytes, der bidirektional sein kann (die Anwendung ist vollduplex: sie kann über den Socket senden und empfangen). Bei Streams kann man sich darauf verlassen, dass die Daten in der richtigen Reihenfolge und ohne Duplikate abgeliefert werden. Der Empfang von Stream-Nachrichten ist garantiert, und Streams eignen sich vor allem für die Behandlung großer Datenmengen.

Die Netzwerk-Transportschicht kann die Daten in Pakete angemessener Größe aufteilen oder zusammenstellen. Die Klasse *CSocket* übernimmt das Packen und Entpacken.

Streams basieren auf expliziten Verbindungen: Socket A fordert eine Verbindung mit Socket B an, der die Verbindungsanforderung akzeptiert oder zurückweist.
Stream-Sockets eignen sich z. B. für Implementierungen wie dem File Transfer Protocol (FTP), das die Übertragung von ASCII- oder Binärdateien beliebiger Größe ermöglicht.

Datagramm-Sockets unterstützen einen bidirektionalen Datenfluss, für den es keine Garantie gibt, dass er in der richtigen Reihenfolge oder ohne Duplikate ankommt. Datagramme sind nicht absolut verlässlich; es kann sein, dass sie nicht ankommen. Die Ankunft der Daten kann in falscher Reihenfolge oder mit Vervielfältigungen geschehen; die Datensatzgrenzen werden jedoch eingehalten, solange die Sätze kleiner als die beim Empfänger intern festgelegte maximale Größe sind. Für die Reihenfolge und Verlässlichkeit ist der Entwickler selbst verantwortlich.

Datagramme sind "verbindungslos", d. h. es wird keine explizite Verbindung hergestellt; Sie senden eine Datagramm-Nachricht an einen bestimmten Socket und können Nachrichten von einem bestimmten Socket empfangen. Datagramm-Sockets sind auch dazu fähig die Nachrichten an eine große Anzahl von Netzwerkadressen zu verbreiten. Das typische Einsatzfeld der Datagramme sind kurze Nachrichten, Kontrollinformationen.
Für satzorientierte Daten sind Datagramm-Sockets besser geeignet als Stream-Sockets.

2.6.2 Windows-Sockets in MFC

MFC bietet zwei Modelle für das Erstellen von Netzwerk-Kommunikationsprogrammen mit Windows-Sockets, die in zwei MFC-Klassen verkörpert sind [MSDN]. In diesem Abschnitt werden die beiden Modelle sowie die MFC-Unterstützung von Sockets beschrieben.

Die zwei Programmiermodelle für Windows-Sockets in MFC werden durch die folgenden Klassen unterstützt:

- ***CAsyncSocket***

  Diese Klasse kapselt das Windows-Sockets-API ein. CAsyncSocket ist dann zu verwenden, wenn die direkte Programmierung des Sockets-API und die Rückruffunktionen zur Bekanntgabe von Netzwerk-Ereignissen benutzt werden sollen.

CAsyncSocket bietet die folgenden Abstraktionen:

- das Verpacken der Sockets in eine objektorientierte Form zur Verwendung in C++
- gewisse Socket-bezogene Windows-Nachrichten werden in Rückrufe umgewandelt.

- ***CSocket***

Diese Klasse ist von CAsyncSocket abgeleitet und stellt eine höhere Abstraktionsebene für das Arbeiten mit Sockets über ein MFC-Objekt *CArchive* zur Verfügung. CSocket erledigt viele Aspekte der Kommunikation, die man sonst selbst übernehmen müsste. Am wichtigsten ist jedoch, dass CSocket das Blockieren (mit Hintergrundverarbeitung von Windows-Nachrichten) ermöglicht, was wesentlich für die synchrone Operation von CArchive ist.

Die Tabelle 2.5 illustriert die Reihenfolge der Operationen für das Einrichten einer Kommunikation zwischen einem Client und einem Server.

Die Klasse CAsyncSocket kapselt das Windows-Sockets-API auf einer sehr niedrigen Ebene ein. Ausgehend von dieser Annahme bietet dieser Beitrag nur grundlegende Anweisungen. Die Verwendung von CAsyncSocket ist sinnvoll, wenn man in einer MFC-Anwendung die einfache Behandlung mehrerer Netzwerkprotokolle durch Windows-Sockets nutzen möchte, ohne Flexibilität einzubüßen.

Die Klasse CSocket stellt die Socket-Unterstützung auf einer höheren Abstraktionsebene als die Klasse CAsyncSocket bereit. CSocket verwendet eine Version des MFC-Serialisierungs-Protokolls, um Daten eines Socket-Objekts über ein CArchive-Objekt von MFC in beide Richtungen zu transferieren (siehe Tabelle 2.5). CSocket bietet die Blockierung und gewährt den Zugang zu CArchive, das viele Aspekte der Kommunikation verwaltet, die man andernfalls selbst über das Basis-API oder die Klasse CAsyncSocket erledigen müsste.

Server	Client
`// Socket erstellen` `CSocket sockSrvr;`	`// Socket erstellen` `CSocket sockClient;`
`// SOCKET erstellen` `sockSrvr.Create(nPort);`	`// SOCKET erstellen` `sockClient.Create( );`
`// Abhörvorgang beginnen` `sockSrvr.Listen( );`	
	`// um Verbindung bemühen` `sockClient.Connect(strAddr, nPort);`

`// neues, leeres Socket erstellen`
`CSocket sockRecv;`
 	
`// Verbindung bestätigen`
`sockSrvr.Accept( sockRecv );`	

`// Datenobjekt erstellen` `CSocketFile file(&sockRecv);`	`// Datenobjekt erstellen` `CSocketFile file(&sockClient);`

`// Archiv erstellen`
`CArchive arIn(&file, CArchive::load);`
– oder –
`CArchive arOut(&file, CArchive::store);`
– oder beide –
	`// Archiv erstellen`
	`CArchive arIn(&file, CArchive::load);`
	– oder –
	`CArchive arOut(&file, CArchive::store);`
	– oder beide –

`//Daten mit Hilfe von Archiv übergeben:`
`arIn >> dwValue;`
– oder –
`arOut << dwValue;`
	`//Daten mit Hilfe von Archiv übergeben:`
	`arIn >> dwValue;`
	– oder –
	`arOut << dwValue;`

Tabelle 2.5 Übersicht über die Reihenfolge der Operationen für das Einrichten einer Kommunikation zwischen einem Server und einem Client

2.6.2.1 Benachrichtigungsfunktionen

Die Benachrichtigungsfunktionen in den Socket-Klassen sind Rückruffunktionen, die vom Gerüst aufgerufen werden, um das Socket-Objekt über wichtige Netzwerk-Ereignisse zu unterrichten.

Die Benachrichtigungsfunktionen sind:

- ***OnReceive***: teilt dem Socket mit, dass im Puffer Daten vorhanden sind, die durch Aufrufen von Receive zu lesen sind.
- ***OnSend***: teilt dem Socket mit, dass nun durch Aufrufen von Send Daten gesendet werden können.
- ***OnAccept***: teilt dem hörenden Socket mit, dass es ausstehende Verbindungsanfragen durch Aufrufen von Accept akzeptieren kann.
- ***OnConnect***: teilt dem verbindenden Socket mit, dass der Verbindungsversuch abgeschlossen wurde: vielleicht erfolgreich, vielleicht fehlerhaft.
- ***OnClose***: teilt dem Socket mit, dass das mit ihm verbundene Socket geschlossen wurde.
- ***OnOutOfBandData***: Teilt dem empfangenden Socket mit, dass das sendende Socket "bandexterne Daten" zum Versenden bereit hält.

Bandexterne Daten sind ein logisch unabhängiger Kanal, der einem miteinander verbundenen Socket-Paar zugeordnet wird. Dieser Kanal wird in der Regel dafür verwendet, "dringende" Daten zu senden. MFC unterstützt bandexterne Daten.

Die Benachrichtigungsfunktionen werden aufgerufen, wenn das Socket über ein Ereignis, das von Interesse benachrichtigt wird, wie z. B. die Existenz von Daten, die gelesen werden können. MFC ruft die Benachrichtigungsfunktionen auf, um das Verhalten des Sockets zu dem Zeitpunkt anzupassen, wenn Benachrichtigungen eintreffen.

Während einer Operation wie dem Empfangen oder Senden von Daten befindet sich ein CSocket-Objekt in einem synchronen Zustand. In diesem Zustand werden alle Benachrichtigungen, die andere Sockets betreffen, in eine Warteschlange eingereiht, während das aktuelle Socket auf eine gewünschte Benachrichtigung wartet. (Während eines Aufrufs von *Receive* wartet das Socket z. B. auf eine Benachrichtigung zum Lesen.)

Wenn das Socket die synchrone Operation beendet hat und wieder asynchron geworden ist, können andere Sockets damit beginnen, die Benachrichtigungen aus der Warteschlange zu empfangen.

Die Art, wie Benachrichtigungen behandelt werden, ist ein MFC-Implementierungsdetail.

2.6.2.2 Byte-Reihenfolge

Unterschiedliche Rechnerarchitekturen speichern Daten teilweise in unterschiedlicher Byte-Reihenfolge. Intel-basierte Computer speichern z. B. die Daten in umgekehrter Reihenfolge, bei der das niederwertige Byte einer Zahl an der ersten Stelle erscheint. Die Byte-Reihenfolge von Intel, "Little Endian" genannt, ist auch die Umkehrung der Netzwerkstandardreihenfolge "Big Endian". Die Tabelle 2.6 erläutert diese Begriffe.

Byte-Reihenfolge	Bedeutung
Big Endian	Das höherwertige Byte ist am linken Ende eines Wortes.
Little Endian	Das höherwertige Byte ist am rechten Ende eines Wortes.

Tabelle 2.6 Die Byte-Reihenfolge Big Endian und Little Endian

Wenn *CAsyncSocket* verwendet wird, ist es die Entwicklersaufgabe, alle erforderlichen Umstellungen der Byte-Reihenfolge selbst vorzunehmen. Windows-Socket macht das Byte-Reihenfolgemodell "Big Endian" zum Standard und stellt Funktionen bereit, mit denen diese Reihenfolge in andere umgewandelt werden können. In der Tabelle 2.7 werden diese Umstellungsfunktionen aufgeführt. Die Klasse *CArchive*, die zusammen mit CSocket verwenden wird, setzt jedoch die umgekehrte Reihenfolge ("Little Endian") ein.

Funktion	Zweck
Ntohs	Umstellen einer 16-Bit-Menge von der Netzwerk-Byte-Reihenfolge in die Host Byte-Reihenfolge (Von Big Endian zu Little Endian)
Ntohl	Umstellen einer 32-Bit-Menge von der Netzwerk-Byte-Reihenfolge in die Host Byte-Reihenfolge (Von Big Endian zu Little Endian)
Htons	Umstellen einer 16-Bit-Menge von der Host Byte-Reihenfolge in die Netzwerk-Byte-Reihenfolge (Von Little Endian zu Big Endian)
Htonl	Umstellen einer 32-Bit-Menge von der Host Byte-Reihenfolge in die Netzwerk-Byte-Reihenfolge (Von Little Endian zu Big Endian)

Tabelle 2.7 Liste von Windows-Sockets-Funktionen für die Umstellung der Byte-Reihenfolge

2.6.3 WinInet

Microsoft stellt eine Reihe von Technologien bereit, mit der die Anwendungen für die Internet-Unterstützung entwickelt werden können. WinInet (Internet-Erweiterungen für Win32) ist eine davon und stellt eine Schnittstelle zum einfachen Wiederauffinden und Speichern von Dateien über das Internet bereit.

WinInet-Klassen kapseln die Netzwerkprotokolle ein; sie bieten einen einfachen Funktionssatz, mit dem eine Client-Anwendung auf dem Internet zum Herunterladen von Dateien unter Verwendung von HTTP, FTP und Gopher geschrieben werden kann, wobei die vertraute Win32 API-Schnittstelle verwendet wird. WinInet bietet die Möglichkeit die Verzeichnisse auf der lokalen Festplatte, aber auch auf Datenträgern in der ganzen Welt durchzusuchen. Außerdem können Daten unterschiedlichen Typs transparent gesammelt und dem Benutzer über eine integrierte Schnittstelle präsentiert werden.

Diese Klassen abstrahieren die Internet-Protokolle für HTTP, FTP und Gopher in eine Anwendungsprogramm-Schnittstelle auf höherer Ebene. So bieten sie einen schnellen und direkten Weg, um Anwendungen Internet-fähig zu machen. Der Entwickler braucht sich bei vielen Routineoperationen um die Details des verwendeten Protokolls nicht zu kümmern.

WinInet bietet folgende Verwendungsmöglichkeiten:

- Herunterladen von HTML-Seiten.
- Senden von FTP-Anforderungen zum Hoch- oder Herunterladen von Dateien oder zum Holen von Verzeichnislisten.
- Verwenden des Gopher-Menüsystems zum Zugriff auf Ressourcen im Internet.

MFC stellt drei Klassen von Internet-Dateien bereit:

1. ***CInternetFile***,
2. ***CHttpFile*** und
3. ***CGopherFile***.

Mit Hilfe dieser Klassen können Internet-Daten sofort in gewohnter Weise aufgegriffen und bearbeitet werden. Außerdem können lokale und Internet-Dateien mit diesen Klassen konsistent und transparent behandelt werden.

MFC-WinInet-Klassen bieten die folgenden Vorteile:

- Gepufferte E/A
- Typsichere Zugriffsnummern für die Daten
- Standardparameter für viele Funktionen
- Ausnahmebehandlung für häufiger vorkommende Internet-Fehler
- Automatisches Aufräumen geöffneter Zugriffsnummern und Verbindungen

Basis jeder Internet Client-Anwendung ist die Internet-Sitzung. MFC implementiert Internet-Sitzungen als Objekte der Klasse *CInternetSession*.

Für die Kommunikation mit einem Server wird ein *CInternetConnection*-Objekt und eine *CinternetSession* benötigt. Eine *CInternetConnection* wird mit Hilfe eines CInternetSession::Get***Connection-Aufrufs erstellet. Danach kann eine Datei auf dem Server zum Lesen oder Schreiben geöffnet werden.

3 Konzeption der Kommunikationsdienste

An der Konzipierung und Implementierung des Distance-Learning-Tools **OVID** sind außerdem noch zwei weitere Arbeiten beteiligt, deren Aufgaben sind:

- die Konzipierung und der Entwurf des Gesamtsystems [ROS00],
- die Entwicklung des Sitzungsmanagements für das System[KUR00].

Der Abschnitt „Konzept des Gesamtsystems" beschreibt die Konzeption des gesamten Systems nach Roskin [ROS00]. Weiter wird die Konzeption der Dienste im einzelnen ausführlich behandelt und danach das Sitzungsmanagement kurz erläutert.

3.1 Konzept des Gesamtsystems

Es wurde ein System konzipiert und entwickelt, das es allen registrierten Benutzern erlaubt, am Prozess des Distance-Learning teilzunehmen. Es wird dem Benutzer eine Möglichkeit gegeben, auf die Vorlesungsinhalte, die bereits vorbereitet wurden, zuzugreifen. Bei Bedarf können die Kommunikationsdienste aus dem System aufgerufen werden. Für die Kurznachrichten ist ein Kurznachrichtendienst vorgesehen.

Die Architektur des Systems weist folgende Eigenschaften auf:

- Verwendbarkeit in den Netzwerken mit einer hohen Bandbreite und/oder auch in den schmalbandigen Netzwerken
- ein für alle angemeldeten Benutzer offenes System
- transparenter Standort
- Einfachheit in der Bedingung und Wartung
- modulare Struktur, was die Weiterentwicklung und die Erweiterung vereinfachen soll

Die Entwicklung wurde unter Windows NT durchgeführt. Das fertige System soll sowohl unter Windows NT als auch unter Windows 98 lauffähig sein.

Der Entwurf basiert auf einer Client/Server-Architektur, bei der zwei Komponenten eingesetzt werden:

1. Die erste Komponente (Server) übernimmt die Benutzerverwaltung und das Sitzungsmanagement. Sie wird vom Systemadministrator gewartet.
2. Die zweite Komponente (Client) stellt dem Benutzer die verfügbaren Dienste bereit. Sie wird vom Anwender bedient.

Die erste Komponente übernimmt auch die Aufgaben der Verwaltung von Benutzer-Accounts in einer Datenbank, der Rechtevergabe zum Aufruf von zusätzlichen Diensten und des Sitzungsmanagements. Unter Sitzungsmanagement wird dabei die Kontrolle von allen Aktivitäten des Benutzers ab dem Zeitpunkt der Anmeldung am System bis zum Verlassen des Systems gemeint. Es wird von den zusätzlichen Diensten die angeforderte Information über die Benutzer des Systems für interne Verwaltungszwecke zur Verfügung gestellt.

Der Anwender enthält eine grafische Benutzeroberfläche, aus der die Nutzung aller verfügbaren Dienste ermöglicht wird. Folgende Dienste sind bereitgestellt:

1. Wiedergabe von abgespeicherten und digital aufbereiteten Vorlesungsinhalten
2. Chat-Dienst
3. Whiteboard-Dienst
4. Videokommunikation mit ausgewählten Partnern
5. E-Mail
6. FTP
7. Kurzmitteilungen-Austausch-Dienst

Die Wiedergabe von Vorlesungen spielt eine zentrale Rolle, die anderen Dienste stellen Zusatzkomponenten dar, d.h. der Hauptdienst wird als eine Komponente zur Wiedergabe von multimedialen Inhalten konzipiert. Die Zusatzdienste können dann ebenfalls zur Unterstützung des Lernprozesses aufgerufen werden.

Die Wiedergabe von Vorlesungen geschieht folgendermaßen:

- Die Vorlesungsdaten (Folien, Videos) werden als HTML-Seiten aufbereitet.
- Die Daten werden vom Web-Server verwaltet.
- Die Wiedergabe erfolgt in einem Web-Browser (Internet Explorer ist erforderlich).

Die Konzipierung der Zusatzdienste sieht vor, dass, wenn ein Dienst eine Client/Server-Architektur besitzt, wie z.B. der Chat-Dienst, der entsprechende Dienst-Service-Anbieter (hier: Chat-Server) unabhängig vom zentralen Service läuft. Dabei ist der zentrale Service in der Lage den Ablauf der Dienstausführung mit Hilfe von Nachrichten an die Anwendungen, die diese Zusatzdienste darstellen, zu steuern. Dafür bauen diese Anwendungen, falls erforderlich, einen Kommunikationskanal mit dem zentralen Service-Anbieter auf.

Hier nochmals die Anforderungen, die an einen Dienst gestellt werden:

1. Ein Dienst entnimmt die Aufrufparameter aus der Kommandozeile. Alle benötigten Parameter werden in der Kommandozeile an den Dienst übergeben.
2. Sobald aktiv, nimmt ein Dienst die Kommunikation mit dem zentralen Service auf, um eventuelle Steuernachrichten zu verschicken oder zu verarbeiten

Diese zwei Punkte erlauben einen modularen Aufbau des Systems, denn die Zusatzdienste können dabei als unabhängige Anwendungen entwickelt werden, die dann lediglich um die Funktionalität zur Kommunikation mit dem zentralen Service erweitert werden sollen. Auch die bereits geschriebenen Anwendungen können so in das System aufgenommen werden.

Einige der Dienste, die angeboten werden, sind frei verfügbar, andere wurden erst im Laufe dieser Arbeit implementiert. Sie wurden deshalb als auch getrennt vom System lauffähige Komponenten konzipiert und entworfen, die aber bei der Integration in das Gesamtsystem vom Sitzungsmanagement kontrolliert werden können.

Für die Speicherung der Benutzerdaten wurde eine Benutzerdatenbank implementiert und die Benutzerverwaltung realisiert. Es wurde ein Model der zentralen Benutzerverwaltung gewählt, bei der alle autorisierten Logins/Accounts von einem Systemadministrator in die Benutzerdatenbank eingetragen werden. Den Benutzern werden dann ihre Zugangsdaten mitgeteilt (Login und Passwort). Ein Benutzer kann sich unter Verwendung dieser Daten in das System einloggen, und wird anhand seines Logins, das nur einmal in der Datenbank vorkommt, verwaltet. Anwenderspezifische Angaben, die auch in der Benutzerdatenbank abgespeichert werden, können vom Anwender selbst jederzeit verändert werden.

Die Abbildung 3.1 zeigt den Aufbau des Systems.

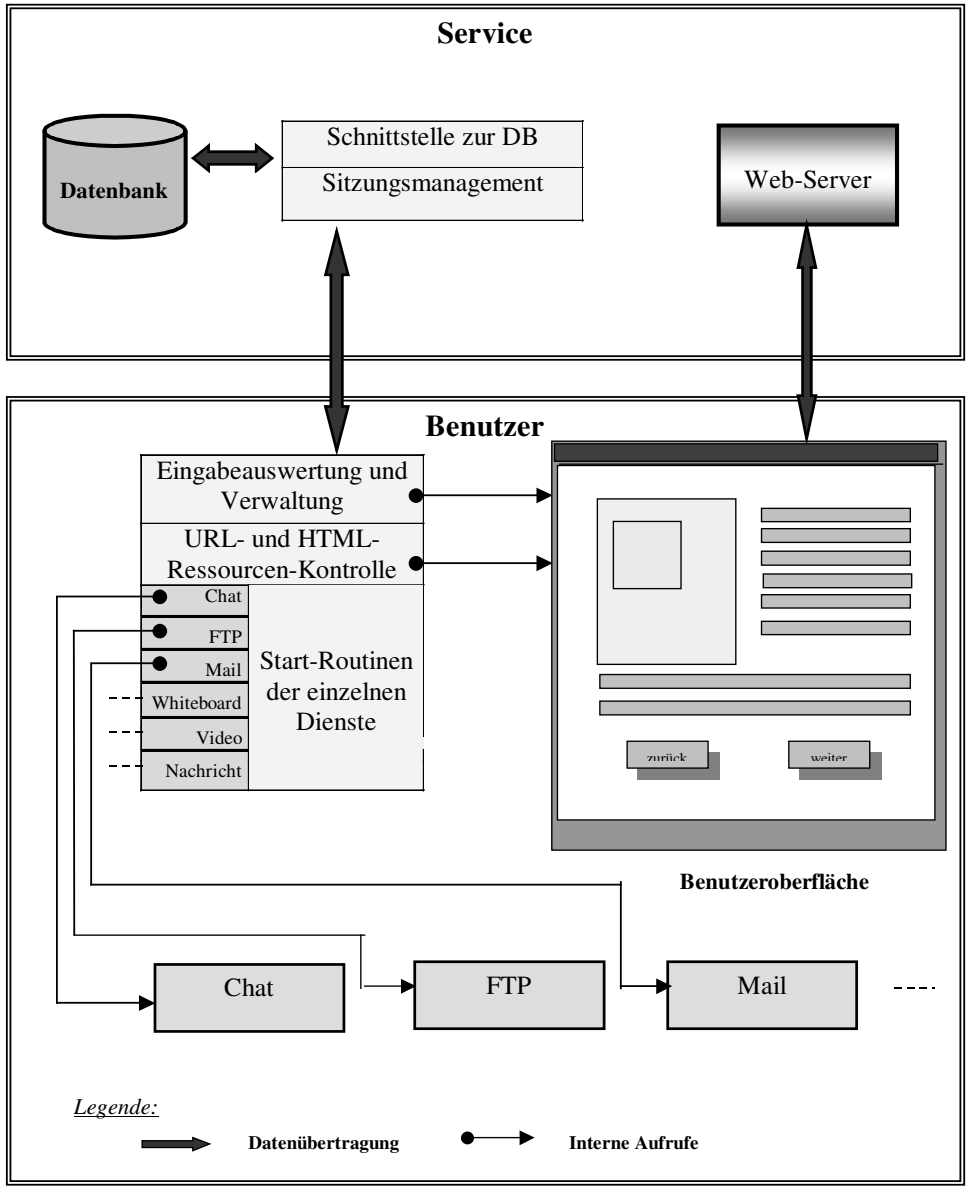

Abbildung 3.1 Schematische Darstellung des Aufbaus des gesamten Distance-Learning-Systems

3.2 Konzeption der Dienste

Im Rahmen dieser Arbeit wurden Kommunikationsdienste konzipiert und implementiert. Außer dem Kurzmitteilungen-Dienst wurden diese als vom System unabhängige Dienste konzipiert, die aber bei der Integration in das Gesamtsystem vom Sitzungsmanagement verwaltet werden können.

Um die Kontrolle der Abläufe im Dienst zu gewährleisten, bauen die Anwendungen zusätzlich zu den Steuer- und Datenverbindungen zu den eigenen Services auch eine weitere Verbindung zum zentralen Service auf. Das geschieht gleich nach dem Aufruf des Dienstes aus der zentralen Anwendung. Der zentrale Service trägt dann einen Vermerk in die Tabelle des Sitzungsmanagements ein, dass ein entsprechender Dienst gestartet wurde.

Wenn ein Anwender seine Dienst-Sitzung (z.B. Chat-Sitzung) beendet hat, wird eine Nachricht an den zentralen Service geschickt, die ihm mitteilt, dass ein entsprechender Dienst beendet wurde. Der zentrale Service löscht die entsprechenden Einträge aus der Tabelle des Sitzungsmanagements und teilt es dann seinerseits der zentralen Anwendung mit.

Wenn der Anwender die Hauptanwendung verlässt, ohne vorher seine Dienst-Sitzung ordnungsgemäß zu schließen, wird beim Beenden der Sitzung eine entsprechende Nachricht an den zentralen Service generiert. Der zentrale Service frischt die Sitzungsmanagementtabelle auf und erzeugt dann seinerseits die Abbruch-Nachricht für die Dienst-Applikation, die zum automatischen Abbruch der Dienst-Sitzung führt [ROS00].

Auch wenn die Kommunikation zwischen den Teilnehmern einer Dienst-Sitzung unerwünscht ist, kann die bestehende Sitzung zentral unterbunden werden.

Somit lässt sich, dank der zusätzlichen Verbindung vom Dienst zum zentralen Service, eine vollständige Synchronisation zwischen den Teilkomponenten des Gesamtsystems erreichen [KUR00].

In Abbildung 3.2 werden die Kommunikation und die Synchronisation zwischen den Teilkomponenten am Beispiel eines Chat-Dienstes schematisch dargestellt.

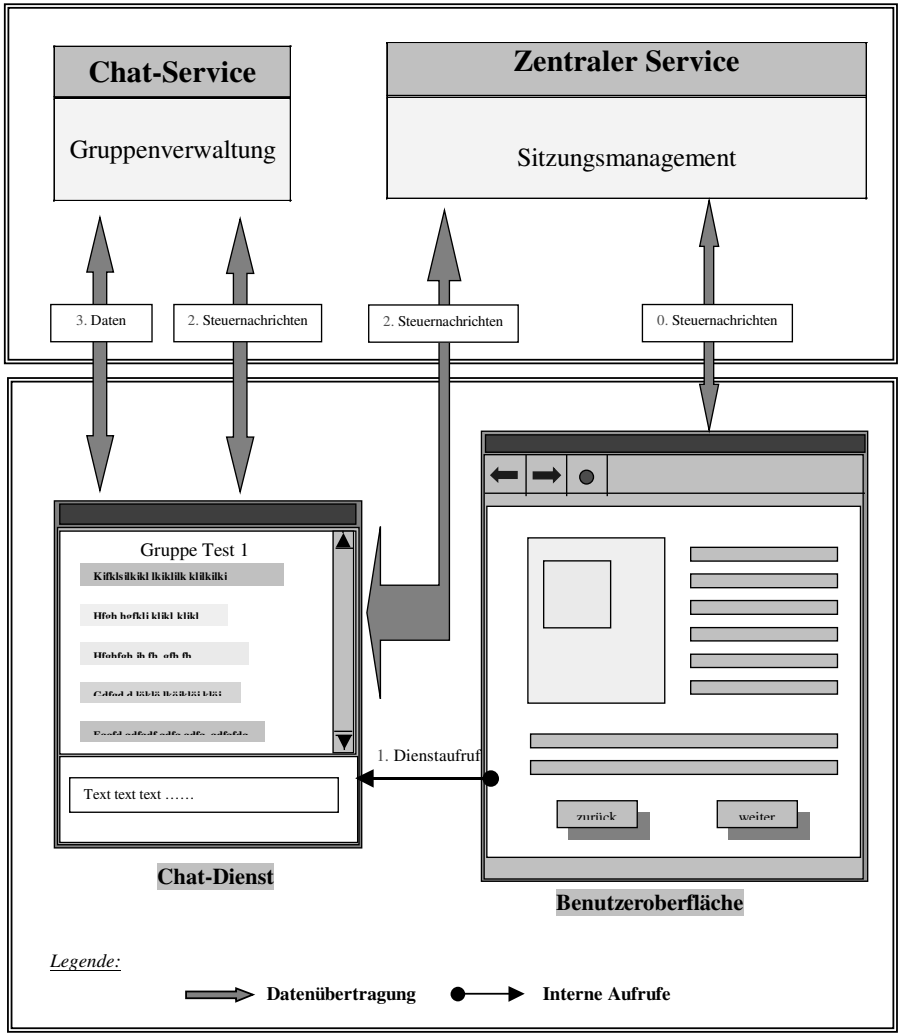

Abbildung 3.2 Schematische Darstellung der Teilkomponentenkommunikation am Beispiel eines Chat-Dienstes im Zusammenspiel mit dem zentralen Service

Hier ist nochmals der Ablauf eines Dienstaufrufes:

- Nach dem Start von der Hauptapplikation werden alle verfügbare Ressourcen des Computers, auf den die ausgeführt wurde, festgestellt und dem Service mitgeteilt. Der zentrale Service wertet diese aus und benachrichtigt dann die Hauptapplikation, welche Dienste bereitgestellt werden können (in der Abhängigkeit von verfügbaren Ressourcen) (siehe Abbildung 3.2 -- 0. Steuernachrichten).

- Danach kann der Anwender den gewünschten Dienst (aus der verfügbaren) aufrufen (siehe Abbildung 3.2 -- 1. Dienstaufruf).

- Der Dienst baut eine Steuerverbindung zum zentralen Service und, wenn ein Dienst-Service vorhanden ist (z.B. Chat-Service), eine Steuerverbindung auch zum Dienst-Service auf (siehe Abbildung 3.2 -- 2. Steuernachrichten). Über die wird dann der Ablauf der Kommunikation gesteuert. Die Informationen werden getauscht (z.B. Gruppennameanforderung) und empfangen (z.B. Abbruchbefehl).

- Wenn alles gut verlaufen ist (die benötigten Ressourcen sind vorhanden, die Dienstausführung wurde erlaubt u.s.w.), wird dann die Datenverbindung aufgebaut (siehe Abbildung 3.2 -- 3. Daten) und der Dienst gestartet.

3.2.1 Konzeption synchroner Kommunikationsdienste

3.2.1.1 Audio-/Videokommunikation-Dienst

Eine bequeme Art der Kommunikation stellt die Audio-/Videokommunikation dar, denn sie kommt der gewohnten zwischenmenschlichen Kommunikation am nähesten. Für die Bereitstellung des Audio-/Videokommunikations-Dienstes wird das vorhandene Online-Audio-/Video-Kommunikationssystem für PAL-Qualität [NGU98] in die OVID-Distance-Learning-Umgebung integriert.

Das Audio-/Video-Kommunikationssystem wird in der Distance-Learning-Umgebung für die Bereitstellung der Punkt-zu-Punkt Verbindungen zwischen zwei Benutzrn verwendet. Dabei kann der Initiator der Videokommunikation einen *Request* an die gewünschte Person schicken. Wenn der *Request* akzeptiert wird, wird automatisch eine Videoverbindung aufgebaut.

3.2.1.2 Chat-Dienst

Für die Kommunikation zwischen den Teilnehmern wird in das System ein Chat-Dienst integriert. Er stellt eine textbasierte Konferenzkomponente dar, wie bereits beschrieben wurde.

Die allgemeinen Anforderungen an einen komfortablen Chat-Dienst sind:

- mehrere Chat-Gruppen werden verwaltet
- ein Benutzer erstellt nach Wunsch eine eigene Gruppe oder kann einer bestehenden Gruppe beitreten, die aus der Liste aller bestehender Gruppen ausgewählt werden kann
- die von verschiedenen Nutzern eingegebenen Beiträge werden mit verschiedenen Farben dargestellt, um die Lesbarkeit und die Differenzierung zwischen den Beiträgen einzelner Benutzer zu erleichtern. Die Farben werden vom System automatisch vergeben.

Ein Chat-Dienst ist auf die Client/Server-Architektur aufgebaut. Das heißt, auf den Rechnern von Anwendern kommen die Anwendungen (Chat-Clients) zum Einsatz, mit denen Texte eingegeben und Beiträge anderer Benutzer dargestellt werden. Alle diese Dienst-Applikationen bauen die Verbindungen mit dem Chat-Service (Chat-Server) auf, an den sie die neu eingegebenen Daten übermitteln. Der Chat-Service erledigt dann die Verteilung dieser Daten an andere Teilnehmer der Gesprächsrunde. Außerdem enthält der Service auch sein eigenes Sitzungsmanagement, das die Verwaltung von sich mit dem Service in Verbindung befindenden Diensten und Gesprächsgruppen übernimmt.

Die Verwaltung von Gruppen erfolgt dynamisch, d.h. die Gruppenzusammensetzung ändert sich mit der Zeit. Für jede Gruppe wird eine entsprechende Empfängerliste geführt. Die Empfängerliste enthält vor allem die IP-Adressen aller Empfänger in der Gruppe. Die Gruppenmitglieder werden über diese Empfängerliste identifiziert.

Der Service kann Steuernachrichten an die Dienst-Applikationen verschicken oder/und von dieser empfangen. Steuernachrichten können folgende sein:

- Anforderung der Liste aller Gruppen
- Beitritt in eine Gruppe
- Austritt aus einer Gruppe
- Aufforderung den Dienst zu verlassen

Bei einem Chat-Dienst wird reine alphanumerische Information übertragen. Hier folgt die Abschätzung des möglichen Datendurchsatzes (siehe Tabelle 3.1).

Ein Benutzer kann normalerweise etwa 5 Zeichen pro Sekunde eingeben. Nehmen wir weiter an, das der ANSI- Zeichensatz eingesetzt wird.
Bei einer Benutzerzahl von 100 Personen ergibt sich folgende Rechnung:

Wird Unicast zur Datenverteilung verwendet, werden die Daten vom Server auf alle 100 Benutzer verteilt. Somit ergibt sich eine Belastung von etwa 4 KBit/s für einen Benutzer, oder, falls alle gleichzeitig schreiben, max. 400 KBit/s.

Die Ergebnisse der Testläufe haben gezeigt, dass, wenn sich mehr als 20 Benutzer in einer Chat-Gruppe befinden, die Kommunikation zwischen ihnen unübersichtlich wird.

Deshalb hier eine veränderte Rechnung mit 100 Benutzern und 20 Benutzer/Gruppe:
Ein Benutzer verursacht in einer Gruppe einen Datenstrom von 5*20*8=0,8 KBit/s

Bei 20 Benutzern pro Gruppe und 5 Gruppen ergibt sich ein Datenstrom von Max. 80 KBit/s

Da der Datendurchsatz auch bei einer hohen Anzahl von Benutzern niedrig bleibt, wurde die für den Chat-Dienst Unicast-Kommunikation gewählt. Aus demselben Grund kann der Chat-Service sich auf einem Rechner mit dem zentralen Service befinden. Dabei bleibt der Chat-Service vom zentralen Service unabhängig. Die Kommunikation zwischen den beiden Services ist nicht vorgesehen, die Synchronisation erfolgt indirekt über den Chat-Dienst. Der Chat-Service muss lediglich zum Zeitpunkt des Starts des Systems auch gestartet werden und läuft dann parallel weiter.

Angaben	100 Benutzer, 1 Gruppe	100 Benutzer, 5 Gruppen
Schreibgeschwindigkeit eines Benutzers	5 Zeichen/s	
Anzahl der Benutzer in einer Gruppe	100	20
Belastung des Netzwerks durch einen Benutzer	4 KBit/s	0,8 KBit/s
Datendurchsatz pro Gruppe, max.	400 KBit/s	16 KBit/s
Gesamte Belastung des Netzwerks, max.	400 KBit/s	80 KBit/s

Tabelle 3.1 Datendurchsatz eines Chat-Servers ohne Overhead

3.2.2 Konzeption asynchroner Kommunikationsdienste

3.2.2.1 FTP-Dienst

Für die komfortable Arbeit mit einem Distance-Learning-System sollen die Teilnehmer auch eine Möglichkeit haben, untereinander Dateien auszutauschen und vom entfernten Rechner Daten herunterladen. Die dafür eingesetzte Kommunikationskomponente benötigt keine zusätzliche Anbindung an den zentralen Service. Es soll lediglich eine Möglichkeit zum Aufruf des FTP-Dienstes aus der zentralen Client-Applikation gegeben sein.

Für die Zwecke der besseren und nahtlosen Integration in das Gesamtsystem wurde eine FTP-Applikation neu entwickelt.

FTP ist ein Internet-Dienst, der speziell dazu dient, sich auf einem bestimmten Server-Rechner im Internet einzuwählen und von dort Dateien auf den eigenen Rechner zu

übertragen (Download) oder eigene Dateien an den Server-Rechner zu übertragen (Upload).

Die allgemeinen Anforderungen an einen komfortablen FTP-Dienst sind:

- Die Möglichkeit, auf dem entfernten Rechner Operationen durchzuführen wie:
  - Verzeichnisinhalte anzeigen;
  - Verzeichnisse wechseln, anlegen, umbenennen, löschen;
  - Dateien umbenennen oder löschen.
- Die Möglichkeit, auf den "normalen" FTP-Service oder auf den anonymen FTP-Service zuzugreifen.
- Die Möglichkeit, den Namen des entfernten Rechners aus der Kommandozeile abzulesen.
- Ergonomische grafische Benutzeroberfläche.

Für den FTP-Dienst ist es sinnvoll, einen entsprechenden FTP-Service auf einem der für das Distance-Learning-System verwendeten Server-Rechner zu installieren. Dieser Service wird dann vom Administrator des Systems verwaltet, und die Inhalte werden von dem Administrator regelmäßig aktualisiert und/oder überprüft.

3.2.2.2 Mail-Dienst

E-Mail (elektronische Post) ist wohl der am meisten genutzte Internet-Dienst. E-Mail erlaubt die persönliche Übermittlung von Nachrichten und Dateien von einem Sender an einen Empfänger.

Für den E-Mail-Austausch werden zwei Arten von Services benötigt, die nicht unbedingt im System selbst installiert sein müssen. Das eine ist ein sogenannter POP3-Service, der es den E-Mail-Clients ermöglicht, die E-Mails abzuholen. Abgeschickt werden die E-Mails über einen SMTP-Service. Im Distance-Learning-System können auch die Services, die für den E-Mail-Austausch bereits eingerichtet sind, weiterverwendet werden.

Als E-Mail-Dienst kann Microsoft Outlook Express oder ein anderes E-Mail-System, das auf dem Benutzer-Rechner installiert ist, verwendet werden. Der E-Mail-Dienst soll im allgemeinen dann benutzt werden, wenn der Teilnehmer nicht über den Dienst zum Verschicken der Kurzmitteilungen erreicht werden kann, d.h. wenn dieser Teilnehmer

nicht mit dem System arbeitet. Er kann z.B. genutzt werden, um eine solche Person zur Zusammenarbeit einzuladen.

Das E-Mail-System wird aufgerufen, indem man der Hauptapplikation einen URL übergibt.

3.2.2.3 Whiteboard

Wenn mehrere in das System eingeloggte Benutzer an einem gemeinsamen Projekt arbeiten, kann dadurch die gemeinsame Bearbeitung eines Dokumentes notwendig werden. Die Programme, die mehreren Personen eine gemeinsame und gleichzeitige Bearbeitung eines grafischen Dokumentes erlauben, sind allgemein unter dem Begriff *„Whiteboard"* bekannt.

Ein Whiteboard-Dienst bietet die Funktionalität eines einfachen Bildbearbeitungsprogramms an, und die neuen Elemente werden zeitgleich mit ihrer Eingabe oder mit minimalen Verzögerungen zwischen allen Teilnehmern einer Sitzung verteilt.

Die Verwendung von IP-Multicast ist eine attraktive Möglichkeit dafür, da mit einer einzigen Multicast-Adresse alle Mitglieder einer Gruppe adressiert werden können. Zur Zeit existiert kein Multicast-Protokoll, das als Standardprotokoll bezeichnet werden kann. Mehrere Whiteboard-Konzepte und Multicast-Protokolle (die werden weiter im Kapitel 5 detailliert besprochen) wurden auf ihre Tauglichkeit zum Einsatz mit dem Distance-Learning-System untersucht. Es wurde festgestellt, dass kein Protokoll allen Anforderungen genügen kann.

Im Rahmen dieser Arbeit wurde für das Whiteboard ein eigenes Protokoll entwickelt, bei dem die Eigenschaften des objektorientierten Modells ausgenutzt wurden. Das Protokoll setzt auf UDP/IP auf.

Allgemeine Anforderungen zum Whiteboard-Dienst:

- *Zuverlässigkeit*
  Es muss ein Mechanismus existieren, der die korrekte Auslieferung aller gesendeten Dateneinheiten bei allen Empfängern ohne Duplikate garantieren kann (die richtige Reihenfolge ist nicht unbedingt notwendig).

- *Fehlererkennung*
  die Mechanismen zur Fehlererkennung, die bei der Unicast-Kommunikation verwendet werden, sind nur wenig geeignet; es muss ein Verfahren geben, das das Problem der Fehlererkennung und Fehlerbehebung löst.

- *minimale Verzögerung*
  die Zeit zur Bearbeitung von Dateneinheiten bzw. Fehlerkorrektur muss minimiert werden.

- *Gruppenverwaltung*
  große Gruppen müssen dynamisch und effizient verwaltet werden; die Zusammensetzung der Gruppen ist dem Sender nicht bekannt.

Für Whiteboard spezifische Anforderungen sind:

- Die Einbindung von RTF-Dateien wird unterstützt. Die Einbindung anderer Dokumente wird vorgesehen (z.B. PS-Dateien oder Word-Dokumenten).
- Alle Werkzeuge werden zentral bereitgestellt.
- Es existieren geometrische Basisfiguren, wie die Ellipse, der Rechteck, die Linie u.s.w.; die Dicke einer Linie kann gewählt werden.
- Die Eingabe in einer Textform ist möglich.
- Die Farbe für eine Figur oder einen Text kann jederzeit gewählt oder geändert werden
- Übersichtliche, leicht zu bedienen Benutzeroberfläche.

Der Whiteboard-Dienst wurde so konzipiert, dass nicht die Dokumente selbst übertragen werden, sondern nur die entsprechenden Änderungen an Dokumenten. In dieser Arbeit wird deshalb vorgesehen, dass je nach Bedarf auch Bitmaps gesendet und empfangen werden können. Ein Bitmap repräsentiert ein Bild beliebiger Art und Komplexität durch ein zweidimensionales Array von Farbwerten.

Die Zwischenablage von Windows wird in die Whiteboard-Applikation integriert. Damit kann ein Teil des Dokuments markiert, kopiert, ausgeschnitten und an anderer Stelle wieder eingefügt werden. Dieser Teil des Dokuments wird dann als ein Bitmap über das Netz übertragen.

3.3 Konzeption der Gruppenverwaltung

Chat

Beim Chat-Dienst wurde für die Gruppenverwaltung folgendes Schema gewählt:

- zentrale Gruppenverwaltung
  Alle Chat-Gruppen werden zentral gehalten und verwaltet. Dies erlaubt zu jedem Zeitpunkt eine konsistente Zustandsinformation der gesamten Gruppe zu erhalten.
  Bei jedem Beitritt bzw. Verlassen wird die Gruppenliste überprüft und aktualisiert.

- Adressierung über Empfängerlisten
  Für jede Gruppe führt der entsprechende Service eine Liste von Empfängern. In dieser Liste werden alle Unicast-Adressen jeweiliger Empfänger gehalten. Alle Gruppenmitglieder können damit vom entsprechenden Service identifiziert werden.
  Bei jedem Beitritt bzw. Verlassen wird die Empfängerliste überprüft und aktualisiert.

Whiteboard

Beim Whiteboard-Dienst wurde eine andere Alternative umgesetzt:

- zentrale Gruppenverwaltung
  Alle Whiteboard-Gruppen werden zentral gehalten und verwaltet. Es wird eine Liste von Gruppen geführt. Der Unterschied zum Chat-Dienst liegt darin, dass für diese Aufgabe kein Service extra eingerichtet werden soll. Dies übernimmt der zentrale Service.
  Bei jedem Beitritt bzw. Verlassen wird die Gruppenliste überprüft und aktualisiert.

- Adressierung über Gruppenadresse
  Jede Gruppe wird mit einer eigenen Gruppenadresse (IP-Multicast-Adresse) versehen. Hierbei übernimmt der zentrale Service auch die Aufgabe der Adressenvergabe. Weiterhin wird sichergestellt, dass die vergebene Gruppenadresse eindeutig ist und nicht anderswo verwendet wird.

Ein Benutzer kann mehreren Gruppen angehören, eine Gruppe kann mehrere Benutzer enthalten. Ein Dienst kann zu einer bestimmten Zeit von mehreren Gruppen benutzt werden. Die Nutzer einer Gruppe können aber nur zu einer bestimmten Zeit einen einzigen Dienst in Anspruch nehmen (siehe Abbildung 3.3). Dabei wird durch [0..*] die Kardinalität der Beziehungen bezeichnet. Zum Beispiel bedeutet [0..*] beim Nutzer, dass ein Nutzer mindestens 0 und maximal unendlich vielen Gruppen beitreten kann.

Das ganze Modell basiert auf einem Relationalen Modell der Gruppenverwaltung. Es existieren zahlreiche Gruppen, die mit ihrem Namen und entsprechender Dienstart, zu der sie gehören, eindeutig identifiziert werden können. Es existieren eine Liste von Gruppenobjekten und eine Liste von Benutzerobjekten.

Die Abbildung 3.3 zeigt das Gruppenverwaltungsmodell aus der Sicht des zentralen Service [KUR00].

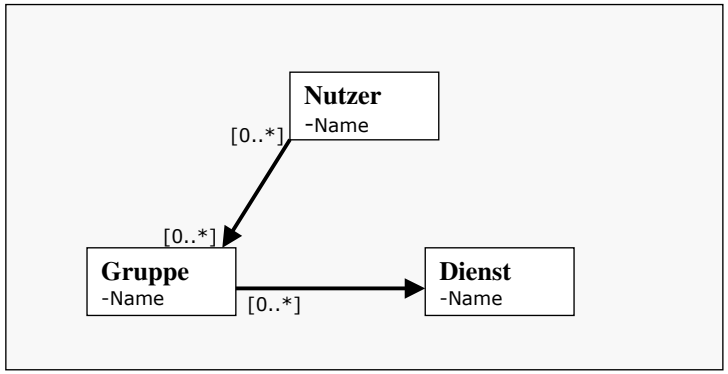

Abbildung 3.3 Relationales Modell der Gruppenverwaltung aus der Sicht des zentralen Service

4 Implementierung der Kommunikationsdienste

Das gesamte Distance-Learning-System wurde für die Windows-Plattform, wegen deren Verbreitung und deren Akzeptanzgrad bei den Anwendern, entwickelt. Deshalb erfolgte die Implementierung der Dienste für die Windows-Plattformen (Windows 9x, Windows NT und Windows 2000) und unter Verwendung der entsprechenden Entwicklungsinstrumente.

Als Programmiersprachen standen Java und C++ zur Auswahl. Obwohl die Programmierungssprache Java mehrere Vorteile gegenüber der C++ hat (u.a. Hardware- und Betriebssystem-Unabhängigkeit), waren die Stabilität der Arbeit des Systems und vor allem die Ausführungszeit von Funktionen entscheidend. Die Performance von C++ - Programmen ist höher als bei Programmen, die mit Java geschrieben werden: Ein C++ - Compiler liefert einen optimierten und auf die CPU zugeschnittenen Binärcode, die Java-Programme werden aber erst während der Laufzeit im Interpreter-Modus in den Maschinencode übersetzt und ausgeführt, um die Plattformunabhängigkeit zu erreichen. Daher wurde C++ als Programmierungssprache gewählt.

Als Entwicklungsumgebung wurde der Microsoft Visual C++ aus dem Microsoft Visual Studio 6.0 verwendet. Da die Entwicklungsumgebung vom selben Hersteller wie das Betriebsystem kommt, ist Visual C++ für Entwicklungsarbeiten unter Windows besonders geeignet und bietet Zugriff auf alle Systemressourcen.

Die Entwicklung erfolgte durchgehend objektorientiert. Dabei wurden MFC-Klassen (Microsoft Foundation Classes) verwendet. Der Begriff MFC ist untrennbar mit Visual C++ verbunden. Dabei stellt die MFC-Bibliothek vorrangig eine Kapselung der Win32 API dar. Fast alle technischen Merkmale von Windows 9x bzw. NT und Windows 2000 sind in irgendeiner Form durch die MFC abgedeckt.

Die Zusatzdienste wurden so implementiert, dass, wenn ein Dienst eine Client/Server-Architektur besitzt (z.B. der Chat-Dienst), der entsprechende Server (hier Chat-Server) unabhängig vom zentralen Servermodul läuft. Dabei verschickt der zentrale Server Steuernachrichten an die Zusatzdienste. Diese bauen dafür, falls erforderlich, eine Steuerverbindung mit dem zentralen Servermodul auf.

Bis auf den Kurzmitteilungen–Austausch-Dienst, sind alle anderen Dienste selbständige Programme. Der Benutzer kann beim Aufruf eines Dienstes nicht unterscheiden, ob es sich dabei um ein autonomes Programm oder um ein neu geöffnetes Fenster des Hauptprogramms handelt. Dementsprechend müssen diese Dienste (Chat, Whiteboard usw.) mit dem Hauptprogramm synchronisiert werden.

Die Synchronisation zwischen den Diensten, wie z.B. FTP und Mail, und dem Hauptmodul ist einfach zu gestalten: Beim Starten eines solchen Dienstes wird ein neuer Prozess erzeugt, über den das Hauptmodul die Kontrolle behält. Deshalb können diese Prozesse, wenn erforderlich, von dem Hauptprozess abgeschaltet werden, falls die Sitzung beendet wurde oder die Verbindung zwischen dem Hauptmodul des Clients und dem zentralen Servermodul getrennt wurde. Eine weitergehende Kontrolle über FTP- und Mail-Dienste wie z.B. eine Kommunikation zwischen ihnen und dem Hauptmodul ist nicht gegeben und auch nicht erforderlich.

Für die Kommunikation zwischen dem Client (z.B. Chat-Dienst) und dem Server (zentraler Servermodul oder Chat-Server) wurden Stream-Sockets verwendet, und zwar die von MFC bereitgestellte Klasse *CSocket*.

Vorteile dieses Ansatzes:

- Die abgeschickten Daten werden in der richtigen Reihenfolge und ohne Duplikate abgeliefert.
- Das Packen und Entpacken wird von der Klasse CSocket übernommen.
- CSocket verwendet das MFC-Serialisierungs-Protokoll, das die Übertragung von Daten über ein CArchive-Objekt von MFC ermöglicht.
- Die Kommunikation zwischen Unicode- und ANSI-Socket-Anwendungen ist möglich.
- ist effektiv für das angewendete Client/Server-Modell.

Nachteile dieses Ansatzes:

- Die Verbindung wird im Laufe der ganzen Sitzung festgehalten.
- Er ist nicht für sehr große Gruppen geeignet (siehe Kapitel 5).
- Für die synchrone Operation von CArchive wird die Verarbeitung von Windows-Nachrichten blockiert.

Im weiteren wird erläutert, wie ein *CSocket*-Objekt, ein *CSocketFile*-Objekt und ein *CArchive*-Objekt miteinander kombiniert werden, um das Senden und Empfangen von Objekten über ein Windows-Socket zu ermöglichen:

> Alle Klassen, die von der Klasse *CObject* abgeleitet wurden, sind serialisierbar. Der Terminus Serialisierung bezeichnet den Prozess, über den sich die Objekte selbst auf ein Medium mit sequentieller Organisation (z.B. eine Datei auf der Festplatte) schreiben bzw. von dort lesen. Die MFC-Funktion *Serialize* hat einen Parameter vom Typ *CArchive*. Diese Klasse stellt ein Archiv dar, in das Daten geschrieben und aus dem Daten gelesen werden können.
>
> Der wesentliche Unterschied liegt darin, dass bei Sockets das Archiv nicht an ein Standard-*CFile*-Objekt gehängt wird (in der Regel einer Plattendatei zugeordnet), sondern an ein *CSocketFile*-Objekt. Anstatt die Verbindung zu einer Plattendatei herzustellen, verbindet sich das *CSocketFile*-Objekt mit einem *CSocket*-Objekt.
>
> Ein *CArchive*-Objekt verwaltet einen Puffer. Wenn der Puffer eines speichernden (sendenden) Archivs voll ist, dann schreibt ein zugeordnetes *CFile*-Objekt den Pufferinhalt heraus. Das Leeren des Puffers eines Archivs, das mit einem Socket verbunden ist, ist gleichbedeutend mit dem Senden einer Nachricht. Wenn der Puffer eines ladenden (empfangenden) Archivs voll ist, dann unterbricht das *CFile*-Objekt das Lesen, bis der Puffer wieder zur Verfügung steht.
>
> Das *CSocketFile*-Objekt muss nur Byte-Folgen von einem oder in ein *CSocket*-Objekt lesen oder schreiben. Dafür ruft es die Member-Funktionen seines *CSocket*-Objekts auf, um Daten zu senden oder zu empfangen.

Die Abbildung 4.1 veranschaulicht das alles.

Das Ziel ist die Abstraktion der Socketverwaltung. Man erstellt nur das Socket, die Datei und das Archiv und beginnt mit dem Senden oder Empfangen von Daten, indem man diese in das Archiv einfügt oder es daraus extrahiert. *CArchive*, *CSocketFile* und *CSocket* verwalten die Details im Hintergrund.

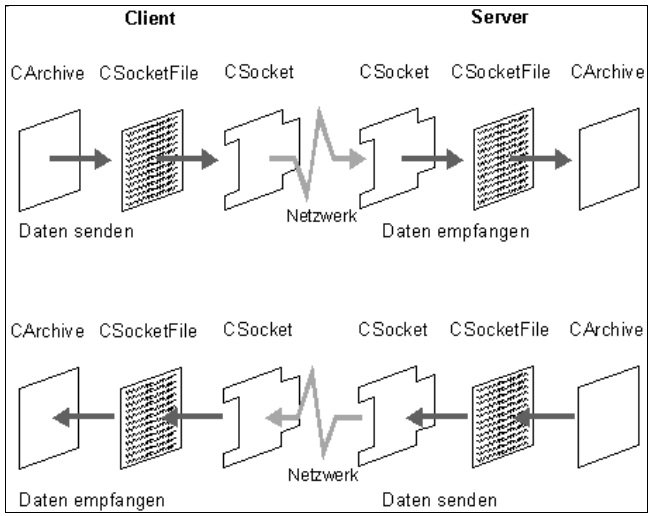

Abbildung 4.1 Übertragung von Daten über CSocket mit der Verwendung von CArchive

Ein *CSocket*-Objekt kann sich in zwei Zuständen befinden:

- *asynchron* (der Normalfall) und
- *synchron*.

Im asynchronen Zustand kann ein Socket asynchrone Benachrichtigungen vom Anwendungsgerüst erhalten. Beim Empfangen oder Senden von Daten geht der Socket in den synchronen Zustand über, bei dem er keine weiteren asynchronen Benachrichtigungen erhält. Der Socket bleibt solange in diesem Zustand, bis die synchrone Operation beendet worden ist, was die Serialisierung ermöglicht und zu keinen Rekursionen (z.B. erhält CSocket eine *OnReceive*-Benachrichtigung, während OnReceive bearbeitet wird) führt. Durch das Wechseln der Zustände vermeidet CSocket die Rekursion, da keine weiteren Benachrichtigungen empfangen werden können (also, keine Benachrichtigungen innerhalb von Benachrichtigungen).

Diese Eigenschaft wird beim Chat-Dienst und bei der Verbindung mit dem zentralen Server ausgenutzt.

4.1 Kommunikation zwischen einem Dienst und dem zentralen Servermodul

Die Zusatzdienste sind so implementiert, dass sie, falls erforderlich, einen Kommunikationskanal mit dem zentralen Server aufbauen, um die einheitliche Sitzungsverwaltung zu gewährleisten. Der zentrale Server verwaltet die Gruppen in den einzelnen Diensten, und ist in der Lage Steuernachrichten an die Dienste zu verschicken oder/und von den Diensten zu empfangen.

Beim Starten des Dienstes baut die Applikation die Steuerverbindung mit dem zentralen Server auf und sendet an ihn eine Anfrage in Form einer Instanz der Klasse *CCtrlMsg*. Die Klasse *CCtrlMsg* kapselt alle Steuernachrichten ein. Der Server akzeptiert diese Nachricht als eine Aufforderung zum Beitritt der ausgewählten Gruppe, fügt einen neuen User in die Gruppenliste des entsprechenden Dienstes und aktualisiert seine Verwaltungstabellen. Danach wird der Dienst gestartet.

Beim Beenden des Dienstes sendet die Applikation zuerst an den Server die Nachricht, eine Instanz der Klasse *CCtrlMsg*, mit der Information, dass der Nutzer einen entsprechenden Dienst beendet hat. Der Server aktualisiert seine Verwaltungstabellen und die Applikation wird dann beendet.

Falls erforderlich, kann die Verwendung eines Dienstes vom Server gesperrt, oder, falls ein Dienst schon in Ausführung ist, abgebrochen werden. Zu diesem Zweck sendet der Server an den Client (oder an eine Reihe von Clients) die Nachricht, auch in Form einer Instanz der Klasse *CCtrlMsg*, wo die Aufforderung, den Dienst zu verlassen, kodiert ist.

Alle benötigten Parameter, wie die IP-Adresse und der Port des zentralen Servers oder der Name des Benutzers, werden der Dienstapplikation in der Kommandozeile übergeben:

z.B. *ChatClient ALEX localhost 700 750*

ChatClient - Dienst
ALEX - Benutzer
localhost 700 - IP-Adresse und der Port des zentralen Servers

4.2 Chat

In diesem Abschnitt wird kurz der Aufbau des Chat-Dienstes vorgestellt. Dabei werden die zwei wichtigsten Klassen mit ihren Member-Funktionen vorgestellt: *CDataMsg* und *CCtrlMsg*.

Die Klasse *CDataMsg* stellt die Benutzernachricht dar. Sie enthält die in der Tabelle 4.1 bezeichnete Attribute.

Attribut	Beschreibung
m_Name	der Nutzer, der diese Nachricht erzeugt hat
m_Farbe	die Farbe, mit der den Text der Nachricht dargestellt werden soll
M_Data	der Text der Nachricht

Tabelle 4.1 Übersicht über die Attribute der Klasse *CDataMsg*

Die Klasse *CCtrlMsg* stellt die Kontrollnachricht dar. Sie enthält die in der Tabelle 4.2 bezeichnete Attribute.

Attribut	Beschreibung
m_Name	der Nutzer, an den diese Nachricht adressiert ist; oder der diese Nachricht erstellt hat
m_Ftyp	Bezeichnet den Typ der Kontrollnachricht
m_Str	Inhalt der Nachricht (Kontrollinformation)

Tabelle 4.2 Übersicht über die Attribute der Klasse CCtrlMsg

Es sind Kontrollnachrichtentypen definiert, die in der Tabelle 4.3 aufgelistet sind.

Typ	Beschreibung
F_LISTE	die Anforderung der Liste mit den Namen aller im Moment existierenden Gruppen
F_GRUPPE	die Nachricht enthält den Namen ausgewählten Gruppe
F_MSG	Anforderung von einer Instanz der Klasse CDataMsg
F_USERINIT	Aufforderung einen neuen Benutzer einzurichten
F_ABORT	Verwendung des Dienstes abbrechen

Tabelle 4.3 Übersicht über die Kontrollnachrichtentypen

Beim Starten des Dienstes „Chat" wird an den Chat-Server zuerst eine Anforderung zur Übermittlung der Liste mit den Namen aller Gruppen, also eine Steuernachricht vom Typ „F_LISTE", geschickt. Diese Liste wird anhand der vom Sitzungsmanagement geführten Tabellen, da der Chat-Server ein eigenes Sitzungsmanagement hat, erstellt und mit ihr wird ein Objekt der Klasse *CStringList* instanziert. Wenn dieses Objekt vom Client empfangen wird, wird dem Benutzer eine Liste aller möglichen Gruppen präsentiert (siehe Abbildung 4.2). Er wählt dann eine Gruppe aus dieser Liste aus und sendet dem Server eine Steuernachricht vom Typ „F_GRUPPE", in der im Attribut `m_Str` der Name der ausgewählten Gruppe gespeichert ist.

Abbildung 4.2 Darstellung eines Fensters mit der Aufforderung, eine Chat-Gruppe auszuwählen

Dann sendet der Client dem Server eine Steuernachricht vom Typ „F_MSG", also eine Anforderung zur Übermittlung einer Instanz der Klasse *CDataMsg*.

Der Server richtet zuerst einen neuen temporären Nutzer ein, sucht in der gewünschten Gruppe nach einer Farbe, die zu dieser Zeit von keinem Nutzer dieser Gruppe benutzt wird. Dann erzeugt er ein Objekt der Klasse *CDataMsg*, instanziert es (die Farbe und der Name werden zugewiesen) und schickt dieses Objekt dem Client. Der Client verwendet dann dieses vom Server instanzierte Objekt zum Austausch von Nachrichten.

Sofern alles erledigt wurde, kommuniziert der Client noch mit dem zentralen Server und teilt ihm mit, dass der entsprechende Client einen entsprechenden Dienst gestartet hat. Danach wird die Applikation gestartet.

Bevor die Applikation gestartet wird, sendet der Client dem Server die Steuernachricht vom Typ „F_USERINIT". Diese besagt, dass der Dienst an der Client-Seite erfolgreich instanziert wurde und der Server jetzt einen temporär eingerichteten Nutzer in die Tabelle des Sitzungsmanagements endgültig einfügen kann. Dies geschieht, um die Konsistenz der Gruppenzusammensetzung zu gewährleisten.

Alle Clients dieser Gruppe werden dann durch eine Nachricht informiert, dass sie ab diesem Zeitpunkt ein neues Gruppenmitglied haben (siehe Abbildung 4.4).

Beim Verlassen der Gruppe sendet der Client dem Server die Steuernachricht vom Typ „F_ABORT". Der Server löscht diesen Client aus der Gruppentabelle und teilt den Clients dieser Gruppe mit, dass der entsprechende Nutzer aus ihrer Gruppe ausgetreten ist (siehe Abbildung 4.4).

Bei der Texteingabe wird das Objekt der Klasse *CDataMsg*, das vom Chat-Server erzeugt und initialisiert wurde, ausschließlich nur mit der eingegebenen Nachricht gefüllt (*CDataMsg::m_Data*). Der Client hat keinen Zugriff auf andere Objektattribute; diese sind in der Klasse *CDataMsg* verkapselt. Dieses gefüllte Objekt wird dann an den Server gesendet und von ihm an alle Gruppenmitglieder verteilt. Das ermöglicht eine klare Trennung zwischen den Clients und eine bessere zentralisierte Verwaltung von Chat-Gruppen.

Alle Vorgänge werden beim Chat-Server protokolliert. Wenn der Benutzer beim Beitritt eine neue Gruppe eingegeben hat, wird sie automatisch an der Server-Seite eingerichtet. Falls ein Mitglied die Gruppe verlässt, wird überprüft, ob noch weitere Mitglieder dieser Gruppe existieren, oder ob sie leer ist. Wenn es der Fall ist, wird sie automatisch gelöscht (siehe Abbildung 4.3).

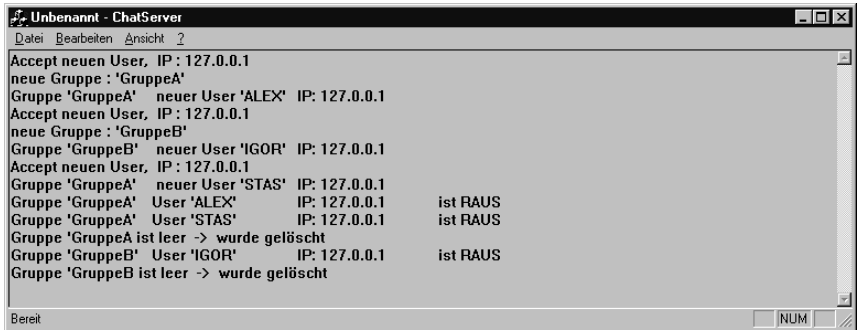

Abbildung 4.3 Grafisches Interface zum Chat-Server mit der Protokollierung einzelner Aktivitäten

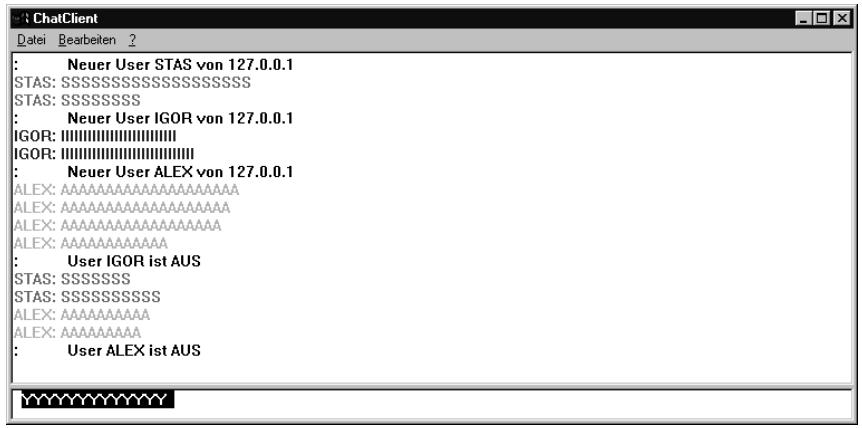

Abbildung 4.4 Kommunikation im grafischen Interface des Chat-Dienstes

4.3 FTP

Bei der FTP-Implementierung wurden die Internet-Erweiterungen für Win32 von Microsoft (MFC-WinInet-Schnittstelle) verwendet. Die Tabelle 4.4 zeigt, welche Schritte in einer FTP-Client-Anwendung durchzuführen sind.

Ziel	*Aktionen*
Beginnen einer FTP-Sitzung.	Erstellen eines **`CInternetSession`**-Objekts
Herstellen einer Verbindung mit einem FTP-Server	Verwenden von **`CInternetSession::GetFtpConnection`**
Wechseln zu einem neuen FTP-Verzeichnis auf dem Server	Verwenden von **`CFtpConnection::SetCurrentDirectory`**
Überprüfen, ob das Verzeichnis auf dem FTP-Server richtig eingestellt ist.	Verwenden von **`CFtpConnection::GetCurrentDirectory`** oder **`CFtpConnection::GetCurrentDirectoryAsURL`**
Suchen der ersten Datei im FTP-Verzeichnis	Aufrufen von **`CFtpFileFind::FindFile`**
Suchen der nächsten Datei im FTP-Verzeichnis	Aufrufen von **`CFtpFileFind::FindNextFile`**
Öffnen der Datei zum Lesen oder Schreiben.	Verwenden von **`CFtpConnection::OpenFile`**
Lesen aus Datei oder Schreiben in die Datei	Verwenden von **`CInternetFile::Read`** oder **`CInternetFile::Write`**.
Speichern der Datei auf dem FTP-Server	Verwenden von „high-level"-Funktion **`CInternetFile::PutFile`**
Kopieren der Datei vom FTP-Server	Verwenden von „high-level"-Funktion **`CInternetFile::GetFile`**
Löschen der Datei	Verwenden von **`CFtpConnection::Remove`**,
Behandeln von Ausnahmen	Verwenden der Klasse **`CinternetException`**
Beenden der FTP-Sitzung.	Freigeben des **`CInternetSession`**-Objekts

Tabelle 4.4 Liste von Schritten, die in einer FTP-Client-Anwendung durchzuführen sind

Beim Starten des Dienstes „FTP" wird ein dialogbasiertes Fenster geöffnet. Auf der rechten Seite wird der Inhalt des lokalen Verzeichnisses, das gerade aktuell ist, dargestellt. Nach Wunsch kann ein neues lokales Verzeichnis eingestellt werden.

Beim Verbinden mit einem FTP-Server wird der Benutzer aufgefordert, den FTP-Server und, falls nötig ist, sein Login und Passwort einzugeben (siehe Abbildung 4.5). Wenn Login und Passwort nicht ausgefüllt werden, wird der Nutzer als „anonym" beim FTP-Server angemeldet.

Abbildung 4.5 Dialogfenster mit Eingaben zum Verbinden

Die eingegebene FTP-Adresse wird danach in eine URL umgewandelt und mittels der globalen Funktion *::AfxParseURL()* geprüft. Bei einer falschen Adresseneingabe wird eine Warnungsmeldung ausgegeben (siehe Abbildung 4.6).

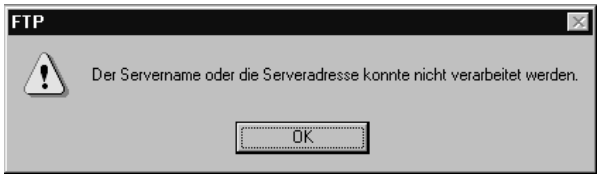

Abbildung 4.6 Dialogfenster mit einer Warnungsmeldung bei einer falschen Adresseneingabe

Andernfalls wird eine Verbindung zum entsprechenden FTP-Server aufgebaut und der Inhalt des FTP-Ordners angezeigt (siehe Abbildung 4.7).

Die folgenden Aktionen zum Manipulieren von Dateien und Verzeichnissen wurde bereitgestellt (siehe Abbildung 4.7):

- ☞ Kopieren von Dateien
  Es kann spezifiziert werden, um welchen Typ von Dateien es handelt. Die Dateien können als Binärdateien oder als Textdateien transferiert werden.

- ☞ Löschen von Dateien und Verzeichnissen

- ☞ Umbenennen von Dateien und Verzeichnissen

- ☞ Erstellen eines neuen Verzeichnisses
  Der Benutzer wird aufgefordert, den Namen eines neuen Verzeichnisses einzugeben. Die Benutzerrechte werden automatisch geprüft und bei der fehlgeschlagenen Funktion *::CreateDirectory()* wird eine Warnung ausgegeben. Andernfalls wird ein neues Verzeichnis erstellt (siehe Abbildung 4.8).

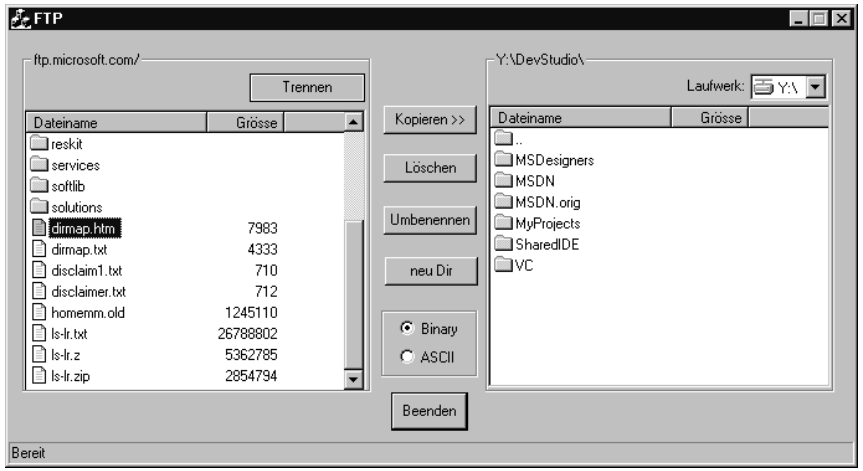

Abbildung 4.7 Grafisches Benutzerinterface für FTP-Dienst

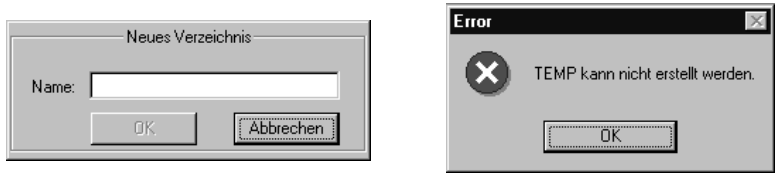

Abbildung 4.8 Dialogfenster zum Erstellen eines neuen Verzeichnisses und mit der Warnungsmeldung bei einem Fehlerauftritt.

4.4 Whiteboard

Das Whiteboard stellt eine gemeinsame Arbeitsfläche dar, auf der Grafiken und Texte erstellt werden können. Als Zeichenelemente werden im einzelnen Linien, Rechtecke, Ellipsen und Texte unterstützt. Die Zeichenelemente können dabei auch in unterschiedlichen Farben dargestellt werden.

Wie schon im Kapitell 3.2.2.3 erwähnt wurde, wurde im Rahmen dieser Arbeit für den Whiteboard-Dienst ein eigenes Protokoll entwickelt, bei dem die Eigenschaften des objektorientierten Modells ausgenutzt wurden. Das Whiteboard-Protokoll, im weiteren als WB-Protokoll bezeichnet, setzt auf UDP/IP auf. Die Abbildung 4.9 zeigt, wo das WB-Protokoll angesiedelt ist.

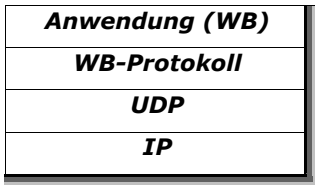

Abbildung 4.9 WB- Protokollstapel

Das WB-Protokoll stellt ein empfängerorientiertes Protokoll dar, bei dem die Empfänger für die Fehlererkennung selbst verantwortlich sind.

Fehlerhaft übertragene Daten werden mit Hilfe der Objekt-Initialisierung erkannt, da jede Dateneinheit, die empfangen wurde, genau ein graphisches Objekt enthält. Wenn das Objekt nicht instanziert werden konnte, enthalten die empfangenen Daten einen Fehler.

Verlorene Daten werden anhand der fehlenden Sequenznummern erkannt. Jeder Sender hat dafür einen eigenen Zähler. Wenn die Reihenfolge von empfangenen Datenpaketen, die ein bestimmter Sender verschickt hat, nicht stimmt, bedeutet das, dass eine entsprechende Anzahl von Datenpaketen verloren gegangen ist.

Erst wenn die nachfolgenden Dateneinheiten empfangen werden, wird vom System angenommen, dass alles wieder in Ordnung ist.

Hat ein Gruppenmitglied den Verlust einer oder mehrere Dateneinheiten festgestellt, so sendet die Station eine Anfrage an die Gruppe und fordert damit die erneute Übertragung der betreffenden Dateneinheiten. Die Übertragungswiederholung nimmt der Sender vor, der die betreffenden Dateneinheiten gesendet hat.

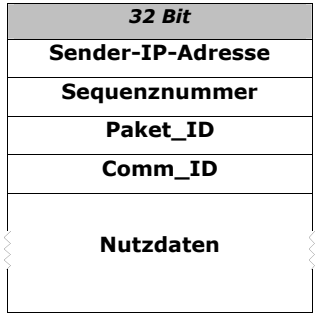

Abbildung 4.10 Schematische Darstellung des WB-Protokoll-Pakets

Jede Dateneinheit (Paket) wird durch die IP-Adresse des Senders und die Sequenznummer, die nur für diesen Sender aktuell ist (nicht global), identifiziert. Weiter enthält ein Datenpaket Paket_ID, Comm_ID und Nutzdaten (siehe Abbildung 4.10).

Die Dateneinheitsgröße ist nicht festgelegt. Die Größe hängt davon ab, was für ein Paket gesendet wird. Die Pakettypen werden weiter erläutert.

Sender-IP-Adresse

ist 32 Bit lang, dient zur Identifikation der Quelle (zusammen mit der Sequenznummer kann der Empfänger den Datenverlust feststellen).

Sequenznummer

Jede Dateneinheit ist mit einer Sequenznummer versehen (zusammen mit der Sender-IP-Adresse kann der Empfänger den Datenverlust feststellen), sie bestimmt die Sendereihenfolge des entsprechenden Senders.

Paket_ID

In dieser Arbeit wurden vier Typen von Datenpaketen eingeführt. Die Definitionen sind offen geführt, sodass bei Bedarf weitere Datentypen definiert werden können. Die definierten Typen sind:

PK_DRW_MSG - diese Dateneinheit ist ein Paket zum Zeichnen. Dieses Paket enthält die Information über ein geometrisches Objekt, das abgebildet bzw. ausgegeben werden soll.

PK_ANFR_MSG - Wenn der Empfänger den Datenverlust festgestellt hat, sendet er ein Paket dieses Typs. Das Paket enthält die Sequenznummern von fehlenden Datenpaketen.

PK_RUCK_MSG - Wenn der Sender das PK_ANFR_MSG-Paket empfangen hat, sendet er die fehlenden Pakete und markiert sie als die Wiederholung von gesendeten Dateneinheiten.

PK_ACK_MSG - Bestätigungs-Paket; dieses Feld wurde optional eingeführt.

Comm_ID

Das PK_DRW_MSG-Paket wurde weiter typisiert. Da dieser Typ von Paketen anteilmäßig am meisten auftritt, wurde das Feld Comm_ID in den Paket-Header eingefügt und nicht in den Nutzdaten verborgen (siehe Abbildung 4.10). Dieses Feld kann dann auch bei anderen Pakettypen verwendet werden.

Es wurden zehn Typen vom PK_DRW_MSG-Paket definiert. Diese sind in der Tabelle 4.5 aufgelistet.

Typ vom PK_DRW_MSG-Paket	Bedeutung
DRW_NULL	Dieses Paket zeichnet **nichts** und wurde für die bessere Fehlerkorrektur bzw. Kontrolle eingeführt.
DRW_RECT	Enthält ein Rechteck
DRW_ROUND	Enthält ein gerundetes Rechteck
DRW_LINE	Enthält eine Linie
DRW_OVAL	Enthält eine Ellipse
DRW_STRK	Darstellt eine Figur, die von freier Hand gezeichnet wurde
DRW_TXT	Enthält eine Textstring
DRW_DEL	Mit diesem Paket werden gekennzeichnete Figuren gelöscht
DRW_COPY	Dieser Pakettyp wurde zur Übertragung von Bitmaps eingeführt
DRW_NEW_PG	Dieses Paket besagt, das eine neue Zeichnungsseite erzeugt werden soll

Tabelle 4.5 Liste von Typen des PK_DRW_MSG-Pakets

Nutzdaten

Die Daten werden in Abhängigkeit vom gesendeten Pakettyp interpretiert, z.B. bei einem PK_DRW_MSG-Paket handelt es sich um eine Figur oder Texteingabe, bei einem PK_ANFR_MSG-Paket stellen die Nutzdaten die Sequenznummern der fehlenden Dateneinheiten dar.

Der Whiteboard-Applikation wurde so implementiert, dass nur die entsprechenden Änderungen an Dokumenten übertragen werden.

Um ein Zeichenelement zu implementieren, wird die Klasse *CFigure* definiert. Diese Klasse kapselt ein Zeichenelement ein und stellt eine Basisklasse für alle weiteren Klassen dar. Ein Objekt der Klasse *CFigure* enthält:

- Die Position des umschließenden Rechtecks
- Die Farbe des Zeichenelements

Von der Klasse *CFigure* werden dann alle Klassen wie CLinie, CCircle u.s.w abgeleitet, die im einzelnen ein geometrisches Objekt oder einen Textstring repräsentieren. In der Tabelle 4.6 werden alle Klassen, die geometrische Objekte darstellen, aufgeführt:

Klasse	Beschreibung
CFigure	Repräsentiert ein Zeichenelement; beschreibt das Rechteck in dem das Zeichen erstellt wird und die Farbe des Zeichens
CLinie	Repräsentiert eine Linie; enthält alle *CFigure*-Eigenschaften
CCircle	Repräsentiert eine Ellipse; enthält alle *CFigure*-Eigenschaften
CRechteck	Repräsentiert ein Rechteck; enthält alle *CFigure*-Eigenschaften
CStroke	Repräsentiert frei-Hand-Zeichnen; enthält alle *CFigure*-Eigenschaften
CTxt	Repräsentiert eine Textstring; beschreibt das Format und die Schriftart der Texteingabe; enthält alle *CFigure*-Eigenschaften

Tabelle 4.6 Liste der Klassen, die geometrische Objekte darstellen

Weiterhin wird die Klasse CWBPaket definiert, die ein Datenpaket, das versendet oder empfangen wird, einkapselt. Die Hauptaufgabe dieser Klasse ist folgende:

- ein Zeichen-Objekt entgegenzunehmen und in eine Bytefolge umzuwandeln, die weiter über das Netz gesendet wird.
- die Bytefolge, die über das Netz empfangen wurde, zu überprüfen und in ein Zeichen-Objekt umzuwandeln.

Als eine Basisklasse für ein Socket wurde CAsyncSocket von MFC gewählt. Es wurden zwei Sockets definiert: einer für das Senden und ein zweiter für das Empfangen von Daten. Dies erleichtert die Handhabung von Daten und ermöglicht eine bessere Fehlererkennung, da bei solchen Implementierungen die Sockets (Empfangender und Sendender) nicht einander blockieren.

Als Multicast-Empfänger sind folgende Schritte notwendig:

- ☑ Öffnen eines UDP-Sockets
- ☑ Binden des Sockets an die Portnummer, an die der Sender senden wird
- ☑ Beitritt zur Multicast-Gruppe
- ☑ Daten empfangen
- ☑ Schließen des Sockets am Ende der Kommunikation

Als Multicast-Empfänger sind folgende Schritte notwendig:

- ☑ Öffnen eines UDP-Sockets
- ☑ Setzen des TTL-Wertes
- ☑ Senden der Daten an die vorgegebene Gruppenadresse mit der entsprechenden Portnummer
- ☑ Schließen des Sockets am Ende der Kommunikation

Die folgenden Programmfragmente illustrieren, wie die Daten per IP-Multicast gesendet und empfangen werden können.

```
#define GROUP_ADDR      224.100.100.100   /* Multicast-Adresse der Gruppe */
#define GROUP_PORT      8888              /* Portnummer                   */
#define DATA_LEN        1024              /* Große der Datenpuffers       */

int   s,                /* Socketdiskreptor                             */
      ttl,              /* TTL-Wert                                     */
      retVal,           /* Ergebnisse                                   */
      len,              /* Anzahl der empfangenen Bytes                 */
      radrLen;          /* Größe der Adresse                            */


struct sockaddr_in      adr, radr;  /* Strukturen für Sende- und        */
                                    /* Empfangsadresse                  */
struct ip_mreq          mreq;       /* Strukturen für den Beitritt      */
                                    /* zur Gruppe                       */
char        data[DATA_LEN + 1];     /* Datenpuffer                      */
```


Das Öffnen eines Sockets muss sowohl vom Sender als auch vom Empfänger durchgeführt werden. Der Socket bildet den Kommunikationsendpunkt.

```
s = socket (AF_INET, SOCK_DGRAM, 0);/* Erzeugen einen Datagramm-Socket */

if ( s < 0 ){           /* Wenn INVALID_SOCKET zurückgeliefert wurde,   */
                        /* Fehlermeldung ausgeben                       */
      fprintf(stderr, "Fehler beim Öffnen des Sockets\n");
      exit (1);         /* und Programm beenden                         */
}
```

Das Binden des Sockets ist eigentlich nur für einen Multicast-Empfänger zwingend notwendig. Da UDP eine Multipeer-Kommunikation unterstützt, kann über denselben Socket sowohl empfangen als auch gesendet werden.

Beim Binden wird nicht die Multicast-Gruppenadresse angegeben, sondern die Konstante *INADDR_ANY*. Dies bewirkt, dass der Empfänger Multicast-Dateneinheiten von jeder Netzwerk-Schnittstelle der Rechners akzeptiert.

```
adr.sin_family = AF_INET;                   /* Protokollfamilie           */
adr.sin_addr.s_addr = htonl (INADDR_ANY); /* Binden an beliebige Adresse */
adr.sin_port = htonl (GROUP_PORT);          /* Portnummer                 */

                        /* Socket an die lokale Adresse binden            */
retVal = bind (s, (struct sockaddr*) &adr,
                  sizeof(struct sockaddr_in));

if ( s < 0 ){           /* Wenn Fehler aufgetreten ist                    */
                        /* Fehlermeldung ausgeben                         */
      fprintf(stderr, "Fehler beim Binden des Sockets\n");
      exit (1);         /* und Programm beenden                           */
}
```

Außerdem muss eine Portnummer angegeben werden, die dem Socket zugeordnet werden soll. Diese Portnummer gibt der Sender als Zielport beim Versenden von Daten an. Beim Empfänger kann anhand der Portnummer der Socket identifiziert werden, an den die Daten ausgeliefert werden müssen.

Zum Beitritt in die Multicast-Gruppe wird die Struktur *mreq* verwendet. Die Multicast-Adresse wird in die Struktur *mreq* eingetragen. Außerdem muss die lokale IP-Adresse der Netzwerk-Schnittstelle angegeben werden.

Der Aufruf von *setsockopt* bewirkt, dass das System ein IGMP-Paket generiert, um der Multicast-Gruppe beizutreten.

```
/*    verbinden mit der Gruppe,                                    */
/*    von welcher die Datagramme empfangen werden sollen           */
mreq.imr_multiaddr.s_addr = inet_addr (GROUP_ADRESSE);
mreq.imr_interface.s_addr = htonl(INADDR_ANY);

/*    der Gruppe beitretten                                        */
retVal = setsockopt (s, IPPROTO_IP, IP_ADD_MEMBERSHIP,
                     &mreq, sizeof(mreq));

if ( retVal < 0 ){      /* Wenn Fehler aufgetreten ist           */
                        /* Fehlermeldung ausgeben                */
      fprintf(stderr, "Fehler beim Beitritt zur Multicast-Gruppe\n");
      exit (1);         /* und Programm beenden                  */
}
```

Das Setzen des TTL-Wertes beim Sender ist notwendig, da der Standardwert für den TTL-Wert für IP-Multicast-Pakete auf Eins gesetzt ist.

```
ttl = 48;   /*    Time-to-Live-Wert setzen                        */
retVal = setsockopt (s, IPPROTO_IP, IP_MULTICAST_TTL,
                     (char*) &ttl, sizeof(ttl));

if ( retVal < 0 ){      /* Wenn Fehler aufgetreten ist           */
                        /* Fehlermeldung ausgeben                */
      fprintf(stderr, "Fehler beim Beitritt zur Multicast-Gruppe\n");
      exit (1);         /* und Programm beenden                  */
}
```


Das Senden von Daten erfolgt mittels des *sendto*-Aufrufs, bei dem die IP-Multicast-Adresse und die Nummer des Zielports angegeben werden müssen. Bei der Portnummer handelt es sich um diejenige, an die die Sockets der Empfänger gebunden worden sind.

```
adr.sin_family      = AF_INET;             /* Protokollfamilie           */
adr.sin_addr.s_addr = htonl (INADDR_ANY); /* Binden an beliebige Adresse */
adr.sin_port        = htonl (GROUP_PORT);  /* Portnummer                 */

/*    Dtagramme an die Gruppe senden                                    */
retVal = sendto (s, data, dataLen, 0,
                 (struct sockaddr*) &adr, sizeof(adr));

if ( retVal < 0 ){      /* Wenn Fehler aufgetreten ist                 */
                        /* Fehlermeldung ausgeben                      */
      fprintf(stderr, "Fehler beim Senden\n");
      exit (1);         /* und Programm beenden                        */
}
else{                   /* andernfalls Anzahl der gesendeten Bytes     */
                        /* ausgeben                                    */
      printf("%d Bytes gesendet\n", retVal);
}
```

Das Empfangen erfolgt durch einen Aufruf von *recvfrom*.

```
/*    Datagramme von der Gruppe empfangen                               */
len = recvfrom (s, data, DATA_LEN, 0,
                (struct sockaddr*) &radr, sizeof(radrLen));

if ( len < 0 ){         /* Wenn Fehler aufgetreten ist                 */
                        /* Fehlermeldung ausgeben                      */
      fprintf(stderr, "Fehler beim Empfangen\n");
      exit (1);         /* und Programm beenden                        */
}
else{                   /* andernfalls Anzahl der empfangenen Bytes    */
                        /* ausgeben                                    */
      printf("%d Bytes empfangen\n", len);
}
```

Das Schließen des Sockets nach Beendigung der Kommunikation erfolgt durch den Aufruf close(s). Vorher kann durch einen Aufruf von setsockopt mit der Option IP_DROP_MEMBERSHIP die Multicast-Gruppe explizit verlassen werden. Ein solcher Aufruf bewirkt, dass ein IGMP Host-Membership-Leave-Paket generiert wird, mit dem eine Multicast-Gruppe explizit verlassen wird.

Das Empfangen von Daten wird über Netzwerk-Ereignissen implementiert. Die Benachrichtigungsfunktionen *OnReceive* wird automatisch ausgelöst, wenn Daten im Puffer vorhanden sind.

Alle benötigten Parameter, wie die IP-Multicast-Adresse der Gruppe und der Port, werden der Dienstapplikation in der Kommandozeile übergeben:

```
WB ALEX 139.13.12.11 700 234.5.6.7 4567
```

ALEX	Benutzer
139.1.2.3 700	IP-Adresse und der Port des zentralen Servers
234.5.6.7 4567	Multicast-Adresse der Gruppe und der Port

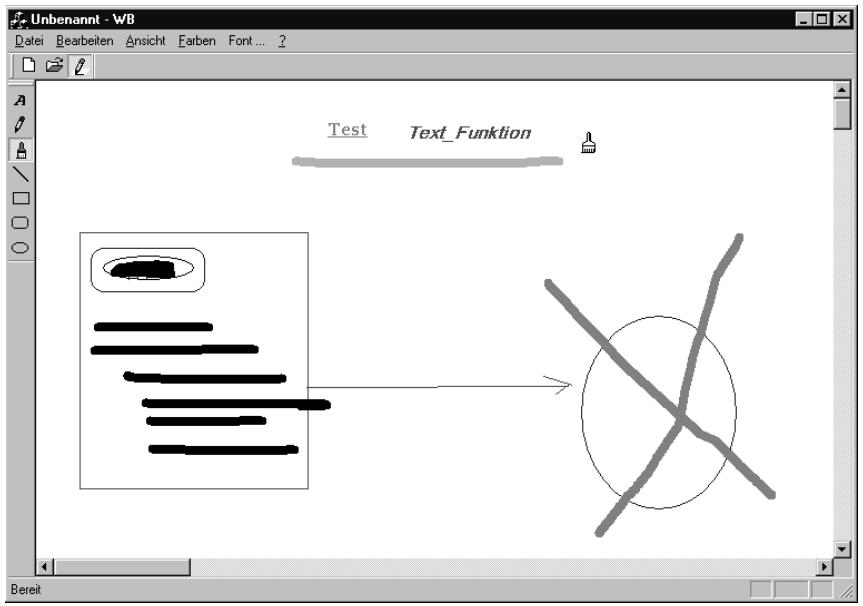

Abbildung 4.11Whiteboard (WB)

4.5 Audio- / Videokommunikation

Für die Videokommunikation zwischen den Benutzern des Systems wird ein existierendes Online-Audio-/Video-Kommunikationssystem verwendet [NGU98].

Für den Einsatz in der Distance-Learning-Umgebung wurde das vorliegende Programm modifiziert. Als Eingabeparameter werden die IP-Adresse der Gegenstelle in der Kommandozeile übergeben, mit der eine Audio-/Videoverbindung hergestellt werden soll. Diese Eingabe erfolgt im Dialog-Modus. Deshalb ist es notwendig, das vorhandene Programm zur Online-Audio-/Video-Kommunikation so umzuändern, dass es die Eingabe der IP-Adresse aus der Kommandozeile auslesen und auswerten kann.

Des weiteren übernimmt das Hauptmodul die Verwaltung der IP-Adressen. Dem Benutzer wird lediglich eine Liste mit den möglichen Kommunikationspartnern angeboten, aus der er einen aussuchen kann.

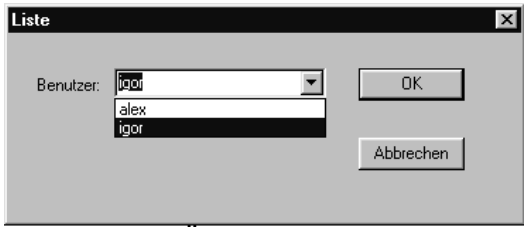

Abbildung 4.12 Dialogfenster zur Auswahl des Kommunikationspartners

Anschließend wird die Audio-/Videoverbindung automatisch aufgebaut. Der Prozess der Synchronisation zwischen Initiator der Audio-/Videokommunikation und Gegenstelle verläuft wie folgt:

- Es wird ein Objekt der Klasse *CVideoRq* erstellt, und über den Server an den ausgewählten Benutzer geschickt. Dabei fügt der Server in dieses Objekt noch die IP-Adresse des Initiators ein.
- Falls die Gegenstelle den *Request* akzeptiert, wird auch von ihr ein *CVideoRq-Objekt* an den Initiator verschickt, wobei der Server in diesem Fall die IP-Adresse der Gegenstelle einfügt.

- Somit verfügen am Ende der Synchronisationsphase sowohl der Initiator als auch die Gegenstelle wechselseitig über die jeweiligen IP-Adressen. Anschließend wird die Verbindung gestartet.
- Wird der Request von der Gegenstelle abgelehnt, so wird eine entsprechende Nachricht an den Initiator geschickt.

Die Abbildung 4.12 zeigt das Dialogfenster mit der Liste der Sitzungsteilnehmer, das auf dem Rechner des Initiators geöffnet wird.

In der Abbildung 4.13 ist die Anfrage zu sehen, die auf dem Rechner des Benutzers erscheint, der zur Teilnahme an einer Videokommunikationssitzung eingeladen wird.

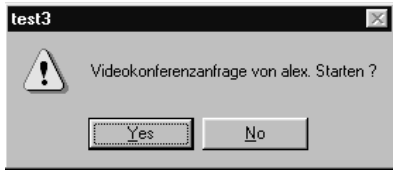

Abbildung 4.13 Dialogfenster für die Einladung zur Teilnahme an der Videokommunikation

5 Ergebnisse der Arbeit

Mit der vorliegenden Arbeit wurden Kommunikationsdienste für das Distance-Learning-Tool OVID konzipiert und implementiert. Im Rahmen dieser Arbeit wurden mehrere Kommunikationskonzepte und Implementierungen auf ihre Tauglichkeit zum Einsatz mit dem Gesamtsystem untersucht. Dabei wurden die qualitativ besten Lösungsalternativen aus den verschiedenen Teilgebieten der Kommunikations-, System- und Netzwerktechnik aufeinander abgestimmt und optimiert.

Für die Implementierung des Chat-Dienstes wurde eine Client/Server-Applikation entwickelt. Es ist vorgesehen, dass der Chat-Server auf demselben Rechner wie das zentrale Servermodul läuft, was aber keine notwendige Bedingung darstellt. Dabei werden vom Server die Ports 701 und 702 belegt. Port 701 dient dem Austausch von Steuernachrichten mit den Chat-Clients. Über den Port 702 werden die Textzeilen, die von den Benutzern eingegeben werden, zwischen allen Benutzern verteilt.

Als Kommunikationstyp wurde für den Chat-Dienst eine Punkt-zu-Punkt-Kommunikation gewählt. Zur Übertragung von Textzeilen wurde ein eigenes Transportprotokoll entwickelt, das auf TCP basiert. TCP stellt einen zuverlässigen Dienst für die Punkt-zu-Punkt-Verbindung dar. Der Vorteil dieser Lösung besteht in folgenden Punkten:

- Fehlererkennung und Fehlerkorrektur werden von TCP übernommen. Der TCP-Dienst stellt einen zuverlässigen verbindungsorientierten Bitstrom bereit.
- Die Möglichkeit, ein benutzerdefiniertes Objekt als ganzes und nicht als Bytestrom über das Netz zu schicken. Diese Möglichkeit stellen die MFC-Klassen von Microsoft (CArchive-Klasse) in Verbindung mit Stream-Sockets bereit, was einen neuen Grad der Abstraktion mit sich bringt.
- Die Reihenfolge von gesendeten Objekten wird erhalten.

Der Nachteil liegt darin, dass über die ganze Zeit der Ausführung des Chat-Dienstes, vom Start bis zum Ende, die virtuelle Verbindung erhalten bleibt. Dies hat aber auf die Leistung des Servers und auf die Netzbelastung, dank der hohen Bandbreite und dem kleinen Datendurchsatz (siehe Tabelle 5.1), der beim Chat-Dienst entsteht, keine ungünstigen oder negativen Auswirkungen, was auch die Testergebnisse gezeigt haben.

Hier folgen die Abschätzungen des Nutzdatendurchsatzes, den ein Chat-Server verursacht.

Dabei wurden folgenden Annahmen gemacht:

- Einen Chat-Dienst verwenden zur Zeit 100 Benutzer
- Es existieren 5 Chat-Gruppen, 20 Benutzer je eine Gruppe
- Ein Benutzer gibt im Durchschnitt etwa 5 Zeichen pro Sekunde

Angaben	100 Benutzer, 5 Gruppen
Durchschnittliche Zeichenanzahl pro Satz	50 Zeichen/Satz
Schreibegeschwindigkeit eines Benutzers	5 Zeichen/s 0,1 Satz/s
Anzahl der Benutzer in einer Gruppe	20
Belastung des Netzwerks durch einen Benutzer (obere Grenze)	1,3 KBit/s
Datendurchsatz pro Gruppe, max.	26 KBit/s
Gesamte Belastung des Netzwerks, max.	130 KBit/s

Tabelle 5.1 Abschätzungen des Nutzdatendurchsatzes beim Chat-Server (mit Overhead)

(der mittlere Fehler liegt dabei bei ca. 0,5%)

Bei einem Benutzer wird, mit der Geschwindigkeit 5 Zeichen/s, 0.1 Satz/s eingegeben und somit eine Netzbelastung von etwa 1,3 KBit/s verursacht. Dies ergibt sich aus der Satzlänge und Overhead, das von dem Header des Pakets und der Objektattributen entsteht.

Dieser Bitstrom wird dann vom Server an 20 Benutzer verteilt, was einen Datendurchsatz von 26 KBit/s pro Gruppe hervorrufen wird.

Mit den 5 eingerichteten Gruppen wird die gesamte Netzbelastung, die der Chat-Server verursacht, mit 130 KBit/s eingeschätzt.

Beim Whiteboard-Dienst wurde als ein Kommunikationsmodell das IP-Multicast verwendet. Damit ist es möglich, alle Mitglieder einer Gruppe mit einer IP-Multicast-Adresse zu adressieren.

Wie schon in Kapitell 3.2.2.3 erwähnt wurde, sind mehrere Whiteboard-Konzepte und Multicast-Protokolle (siehe auch [STE98], [STE99], [WIT99]) auf ihre Tauglichkeit zum Einsatz mit dem Distance-Learning-System untersucht worden. Einige davon werden im Detail diskutiert. Es wurde festgestellt, dass kein Protokoll allen Anforderungen genügen kann:

- Zur Zeit existiert kein Multicast-Protokoll, das als Standardprotokoll bezeichnet werden kann.
- Aufgabenstellungen für jede Anwendung sind sehr unterschiedlich, so dass ein einziges Multicast-Protokoll nicht alle lösen kann.
- Das Multicast-Protokoll befindet sich zur Zeit in der Entwicklungsfase.

Ein weiterer Grund liegt darin, dass bei der Konzipierung des Distance-Learning-Systems davon ausgegangen wurde, dass das System ein einheitliches Sitzungsmanagement für verschiedene Dienste bereitstellen soll.

Bei bisherigen Implementierungen wurden das Sitzungsmanagement und die Gruppenverwaltung nicht zentral realisiert. Also, jeder Client ist selbst dafür verantwortlich, die Gruppe zu verwalten und die Gruppenzusammensetzung bei jeder Änderung zu aktualisieren.

Dafür senden alle Gruppenmitglieder z.B. bei SRM-Protokoll (*Scalable Reliable Multicast*) in regelmäßigen Abständen Statusmeldungen. Diese Statusmeldungen erhalten unter anderem die höchste Sequenznummer, welche der Betreffende von jedem Mitglied empfangen hat. Andere Stationen können diese Information dann mit den eigenen vergleichen.

Die regelmäßigen Statusberichte bedeuten in den zum Vergleich herangezogenen Systemen eine unnötige Netzbelastung. Diese Aufgabe wird hier vom zentralen Sitzungsmanagement übernommen und reduziert deshalb die Netzbelastung.

Das zentrale Sitzungsmanagement übernimmt auch die Gruppenverwaltung und die regelmäßige Aktualisierung der Gruppenzusammensetzung. Beim Starten des Whiteboard-Dienstes wird zuerst eine Steuerverbindung (Unicast-Verbindung) zum zentralen Server aufgebaut. Nach der Auswahl der Gruppe erhält der Client die benötigte

Information über die Gruppe. Alle anderen Gruppenmitglieder werden dann benachrichtigt, dass ihrer Gruppe ein neues Mitglied beigetreten ist. Die Nachricht enthält auch die IP-Adresse des neuen Mitgliedes, was allen Gruppenmitgliedern ermöglicht, die Information über die Zusammensetzung der Gruppe konsistent zu halten.

Beim Beenden des Dienstes wird ähnlich vorgegangen. Alle Gruppenmitglieder werden benachrichtigt, die Zusammensetzung der Gruppe zu aktualisieren und das entsprechende Mitglied zu löschen.

Die Fehlererkennung bei manchen Multicast-Protokollen (z.B. RTP – *Real-Time-Transport Protocol*) basiert auf der Abschätzung der Umlaufzeit einer Nachricht. Damit wird die Übertragungsverzögerung abgeschätzt. Hat eine Station den Verlust von Daten bemerkt, wartet sie eine zufällige Zeitspanne, die von der Entfernung zum ursprünglichen Sender der Daten abhängt. Beim Distance-Learning-System hat sich dieses Verfahren als ein überflüssiger Mechanismus zur Fehlerkorrektur erwiesen. Die Anforderungen zur erneuten Übertragung der Daten werden sofort geschickt.

Ein wesentliches Merkmal eines weiteren Multicast-Protokolls MTP (*Multicast Transport Protocol*) ist die Ordnungserhaltung, die mit Hilfe eines Tokens realisiert wird, an den die Sendeberechtigung gebunden ist. Nur die Station, die im Besitz des Tokens ist, hat die Sendeerlaubnis. Die Kommunikationssteuerung wird von einem sogen. MTP-Master übernommen. Der Vorteil liegt darin, dass außer dem Master keine Station eine Liste der Gruppenmitglieder führen muss. Beim Whiteboard-Dienst führt jede Station ihre eigene Liste der Gruppenmitglieder. Die Aktualisierung der Listen wird vom zentralen Sitzungsmanagement veranlasst und nur dann durchgeführt, wenn ein neues Mitglied der Gruppe beitritt oder ein Gruppenmitglied die Gruppe verlässt. Dies ermöglicht dem zentralen Sitzungsmanagement die einheitliche Verwaltung. Jede Station kann zu jedem Zeitpunkt senden, sobald die graphische Figur gezeichnet ist.

Beim in dieser Arbeit implementierten Whiteboard wird auch der Verlust von Daten anders behandelt als bei den vorgestellten Protokollen. Der Datenverlust wird, wie bei allen Multicast-Protokollen, mit Hilfe der Sequenznummer festgestellt. Tritt dieser Fall ein, so sendet die Station sofort eine negative Quittung an die Gruppe und fordert damit eine erneute Übertragung der betreffenden Datenpakete an. Die erneute Übertragung wird vom ursprünglichen Sender, der diese Datenpakete erzeugt hat, und nicht von beliebigen übernommen.

Die Station wartet nicht, wie es bei vielen Multicast-Protokollen (RTP oder SRM-Protokol) üblich ist, sondern fordert sofort mit negativen Quittungen eine erneute Übertragung der fehlenden Datenpakete an, sobald sie den Datenverlust festgestellt hat.

Die Netzbelastung verursachen nur drei Typen von PK_DRW_MSG-Paketen:

- DRW_STRK - frei-Hand-Zeichnen-Figur
- DRW_TXT – eine Textstring
- DRW_COPY - ein Bitmap

Sie haben eine variable Größe, die von Figur zur Figur verschieden sein kann und von der Pixel- oder Zeichenanzahl, die übertragen werden sollen, abhängt. Alle anderen Pakete haben eine feste Paketgröße von 36 Bytes.

Für die Implementierung des FTP-Dienstes (Client-Applikation) wurden die MFC-WinInet-Klassen (siehe Kapitell 2.6.3) verwendet.

Dies hat folgende Gründe:

- WinInet-Klassen kapseln die WinSock-Funktionalität mit entsprechenden Netzwerkprotokollen ein.
- Das Lesen von Informationen aus HTTP-, FTP- und Gopher Servern ist vom Aufwand genau so wie das Lesen einer Datei auf der Festplatte.
- Ausführung vieler Routineoperationen ohne zusätzliche Arbeit für die Verwendung der Protokolle.
- Das Verwenden von HTTP-, FTP- und Gopher-Protokollen ohne direkte WinSock- oder TCP/IP-Programmierung.

Die Nachteile dieses Ansatzes liegen darin, dass viele Operationen relativ lange dauern (in manchen Fällen kann der Verbindungsaufbau zum entfernten Server bis zu 20 Sekunden dauern). Es hängt, abgesehen von der Geschwindigkeit der Netzwerkverbindung, von der Komplexität der Operation ab. Das Herstellen der Verbindung mit einem Remote-FTP-Server erfordert beispielsweise folgende Schritte:

- Suchen nach den Namen des Servers, um seine Adresse zu ermitteln.
- Herstellen einer Verbindung mit dem Server über diese Adresse.
- Veranlassung einer Konversation mit dem Dateiübertragungsprotokoll.
- Erst danach kann diese Verbindung genutzt werden.

Wie die Untersuchungen gezeigt haben, hängt die Dauer einer Operation jedoch in der Regel von der Geschwindigkeit der Netzwerkverbindung zum Server ab.

Im hier entwickelten Distance-Learning-System ist auch die Möglichkeit des E-Mail-Austausches bereitgestellt. Die Auswahl eines E-Mail-Systems wird dem Benutzer überlassen. Das E-Mail-System wird völlig unabhängig vom Distance-Learning-System eingerichtet. Das wird dann beim Verwenden des Distance-Learning-Tools aus dem Hauptmodul aufgerufen.

Da das Hauptmodul die Web-Browser-Funktionalität enthält, wurden an dieser Stelle Eigenschaften des Web-Browsers ausgenutzt, um den E-Mail-System-Aufruf zu implementieren. Als E-Mail-Dienst kann Microsoft Outlook Express oder ein anderes E-Mail-System, das auf dem Benutzer-Rechner installiert ist, verwendet werden.

Die Nutzung von Audio-/Videokommunikations-Diensten wurde dadurch bereitgestellt, dass eine Schnittstelle für ein vorhandenes Online-Audio-/Video-Kommunikationssystem eingerichtet wurde. Diese Schnittstelle definiert die Aufrufparameter, mit der das Audio-/Video-Kommunikationssystem gestartet wird. Die Auswahl eines Kommunikationspartners wird durch das zentrale Sitzungsmanagement unterstützt.

An dieser Stelle wird nur auf die entsprechenden Arbeiten über das Audio-/Video-Kommunikationssystem verwiesen ([NGU98], [HOR99], [FRA99]), in denen ein solches System entwickelt wurde, das hier als Beispiel miteingebunden worden ist.

Im Rahmen der Entwicklung des Distance-Learning-System wurden zahlreiche Tests durchgeführt. Einige Dienste wurden auch in schmalbandigen Netzwerken getestet, dabei wurden die Rechner miteinander über Null-Modem-Kabel verbunden und die Verbindungsbandbreite wurde nach Wahl von 9600 bis 115200 bps eingestellt. Die Bandbreite von 115200 bps entspricht ungefähr der Bandbreite eines ISDN-Adapters mit zwei gebündelten Kanälen.

Diese Tests haben gezeigt, dass einige Dienste auch bei langsamen Netzwerken verwendet werden können. Dabei wurde festgestellt, dass der zentrale Server durch gleichzeitigen Zugriff von mehreren Benutzern nur gering beeinflusst wird: Die Verzögerungen liegen im nicht messbaren Bereich (weniger als 1 Millisekunde), weil zwischen Clients und zentralem Server nur Steuerdaten, die nur wenige Bytes enthalten, übertragen werden [KUR00]. Die Nutzeranzahl eines Dienstes ist nur durch den Durchsatz des Dienst-Servers begrenzt (siehe Tabelle 5.1).

Wie die Tabelle 5.2 zeigt, ist die Hardwareausstattung eines Client-Rechners nur dann von Bedeutung, wenn der Audio-/Videokommunikations-Dienst in Anspruch genommen wird. Dabei ist noch zu erwähnen, dass dieser Dienst nur dann ausgeführt werden kann, wenn zwei Video-Karten auf dem Client-Rechner vorhanden sind. Diese Aufgabe (Feststellung, ob es der Fall ist) wird vom Ressourcenmanagement übernommen (siehe [KUR00]).

Die Tabelle 5.2 enthält die Testergebnisse:

Dienste	Client 1 (60 MHz)	Client 2 (133 MHz)	Client 3 (450 MHz)
Benutzerinterface des zentralen Benutzermoduls	2%	1%	<1%
Verbindung zum Server hergestellt. Zugriff auf die Lehrinhalte ohne Audio-/Video-Daten	max. 40%	max. 15%	Max. 4%
Der Zugriff auf die Lehrinhalte mit Audio-/Video-Daten (MPEG)	100%, kein fliessender Bildablauf, CPU überlastet	90 bis 100%, fliessender Bildablauf (25 Frames/s)	20 bis 30%, fliessender Bildablauf (25 Frames/s)
Audio-/Videokommunikation zwischen zwei Teilnehmer	nicht möglich	80 bis 90%	20 bis 35%
Chat und Whiteboard	3%	2%	1%

Tabelle 5.2 CPU-Belastung verschiedener Clients-Rechner bei Verwendung verschiedener Diensten

(nach [ROS00]). Die relativen Fehler liegen im Bereich bis 30% des angegebenen Wertes.

Dabei spielt die Hardwareausstattung des Servers (Plattenkapazität, Hauptspeicherkapazität) eine wesentliche Rolle, da bei der Gruppenverwaltung und Abspeicherung der temporären Daten viel Speicherplatz in Anspruch genommen wird. Beim Chat-Dienst beansprucht z.B. ein Tabellenvermerk bis zu 2 KBytes. Beim Whiteboard-Dienst wird viel mehr Speicherplatz benötigt, da alle Figuren (alle Send-Pakete) jedes Benutzers auch aufbewahrt werden sollen. Für simple Figuren wie ein

Rechteck oder eine Linie wird 36 Bytes in Anspruch genommen. Die frei-Hand-Zeichnen-Pakete oder Textstring-Pakete können dagegen mehrere Kilobytes groß sein (siehe Abschnitt 4.4).

6 Zusammenfassung und Ausblick

Mit der vorliegenden Arbeit wurden Kommunikationsdienste für das Distance-Learning-Tool OVID konzipiert und implementiert. Sie unterstützen die verteilte Informationsverarbeitung und stellen die kooperativen Anwendungen dar.
Alle Dienste sind für den Einsatz unter Windows 98/NT/2000 eingerichtet; folgende Dienste wurden implementiert:

- Chat
- Whiteboard
- FTP
- Mail
- Audio-/Video-Kommunikationssystem

Die Dienste besitzen folgende Eigenschaften:

- Unterstützung von Netzwerken mit einer hohen Bandbreite
- Unterstützung der Gruppenkommunikation
- Einheitliches Sitzungsmanagement
- Modulare Struktur, offene Architektur
- Eine leicht zu bedienende Benutzerinteroberfläche

Alle Dienste basieren ausschließlich auf dem Client/Server-Modell. Diesem liegt eine asymmetrische Rollenverteilung zwischen den Kommunikationspartnern zugrunde und ist im wesentlichen nur für Unicast-Kommunikation geeignet. Dies wird durch das einheitliche Sitzungsmanagement ermöglicht.

Die verschiedenen Dienste sind als Module konzipiert und entworfen, die auch getrennt vom System lauffähig sind, was auch ihre anderweitige Verwertung ermöglicht. Einige vorhandene Dienste, z.B. Audio-/Video-Kommunikationssystem, werden komplett mit kleinen Quellcodeänderungen in das Gesamtsystem übernommen. Sie können aber bei der Integration in das Gesamtsystem vom Sitzungsmanagement kontrolliert werden.

Einige Dienste (z.B Chat, Whiteboard) verwenden eigene Protokolle, die speziell für sie entwickelt wurden. Dabei wurden neue Technologien und Programmiermodelle verwendet. Für den Chat-Dienst und die Kommunikation mit dem zentralen Sitzungsmanagement-Server wurde z.B. das CSocket-Programmiermodell mit der Möglichkeit eine Version des MFC-Serialisierungs-Protokolls zu nutzen, verwendet. Das

Whiteboard benutzt ein eigenes Multicast-Protokoll, das ein zuverlässige Multicast-Kommunikation ermöglicht. Bei der Implementierung des FTP-Clients wurden neue Microsoft Win32 Internet-Funktionen (WinInet-Schnittstelle) verwendet. Für alle Dienste wurde eine Schnittstelle zum Sitzungsmanagement geschaffen. Über diese erhält der Sitzungsmanagement-Server dann die volle Kontrolle über den Dienstablauf und die Benutzeraktivitäten.

Dabei wurden verschiedene Gruppenkommunikationsmodelle untersucht. Für den Chat-Dienst wurde die Unicast-Kommunikation ausgewählt. Beim Whiteboard-Dienst wurde Multicast eingesetzt. Alle Gruppen werden zentral vom Sitzungsmanagement-Server verwaltet.

Das Distance-Learning-Tool OVID wird als erstes an der Universität Leipzig eingesetzt. Hier werden dank der hohen Bandbreite, die im Netz der Universität zur Verfügung steht, und dank dem speziell ausgerüsteten Teleteaching-Labor, alle entsprechend vorgesehenen Dienste des Tools zum Einsatz kommen können. Eine hohe Bandbreite wird z.B. bei der Videoübertragung gefordert. Das Einsatzgebiet ist aber nicht auf ein Intranet beschränkt. Andere Dienste, die ohne die hohen Bitraten auskommen, können auch aus dem Internet gestartet und benutzt werden.

Das Gesamtsystem wurde mit implementierten Diensten erfolgreich getestet (siehe Kapitel 5 und auch Arbeiten von [ROS00] und [KUR00]). Es wurde die Fähigkeit des Systems analysiert, mehrere Benutzer und Arbeitsgruppen zu verwalten. Außerdem wurde auch die Prozessorbelastung bei verschiedenen Nutzeranfragen/-aktivitäten gemessen (siehe Kapitel 5). Die Teilkomponenten wurden auf ihre Eignung zum Einsatz mit verschiedener Hardware überprüft. Dabei wurde herausgestellt, dass nur das Audio-/Video-Kommunikationssystem viele Ressourcen in Anspruch nimmt. Alle anderen Dienste können auch auf herkömmlichen PC-Systemen genutzt werden:

- Intel-Pentium PC mit 60 MHz Taktfrequenz - Zugriff auf die Lehrinhalte ohne Audio-/Video-Daten.
- Intel-Pentium PC ab 120 MHz Taktfrequenz - Zugriff auf die Lehrinhalte mit Audio-/Video-Daten.
- Intel-Pentium PC ab 120 MHz Taktfrequenz und das Vorhandensein zweier Videoschnittadapter – die Nutzung des Audio-/Video-Kommunikationssystem.

Es gibt einige Komponenten, die erweitert oder auch implementiert werden müssen. So muss z.B. für den Whiteboard-Dienst ein geeignetes Sitzungsmanagement implementiert werden. Dieses wurde in der Arbeit [KUR00] ausgearbeitet. Für den Whiteboard-Dienst

muss noch die Zwischenablage unter Verwendung von vorgesehenen Schnittstellen implementiert werden. Beim Chat-Dienst wird z.B. die Farbe automatisch vergeben, dabei ist aber die ergonomische Anpassung noch zu realisieren. Es wäre günstig, wenn weitere Dienste, wie z.B. VoIP und Shared Application, ebenfalls eingebunden werden, da die Grundlagen dafür mit der vorliegenden Arbeit gelegt wurden.

Abkürzungsverzeichnis

ACK	Acknowledgment
API	Application Program Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
CGI	Common Gateway Interface
COM	Component Object Model
CSCW	Computer-Supported Collaborative Work
DAO	Data Access Objects
DNS	Domain Name System
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
JVM	Java Virtual Mashine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protokol
MAC	Media Access Control
MBone	Multicast Backbone
MFC	Microsoft Foundation Classes
MIME	Multipurpose Internet Mail Extensions
MPEG	Motion Picture Expert Group
Mrouter	Multicast-Router
MTP	Multicast Transport Protocol
NACK	Negative Acknowledgment
ODBC	Opend Database Connectivity
OLE	Object Linking and Embedding
OLTP	Online Transaction Processing
ORB	Object Request Broker
OSI	Open System Interconnection

PAL	Phase Alternating Line
PDF	Portable Document Format
RAID	Redundant Array of Inexpensive Disks
RFC	Request For Comment
RIP	Routing Information Protocol
RMI	Remote Method Invocation
RTFM	Real-Time Traffic Flow Measurement Protocol
RTP	Real-Time-Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SRM	Scalable Reliable Multicast
TCP	Transmission Control Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol
WAN	Wide Area Network
WWW	World Wide Web

Literatur- und Quellenverzeichnis

[GLB97] Gemmell, Jim, Liebeherr, Jorg, and Bassett, Dave: „In Search of an API for Scalable Reliable Multicast", Microsoft Research Technical Report, MSR-TR-97-17, June 1997.
ftp://ftp.research.microsoft.com/pub/tr/tr-97-17.ps

[GEM97] Gemmell, Jim: „Scalable Reliable Multicast Using Erasure-Correcting Resends", Microsoft Research Technical Report, MSR-TR-97-20, June 1997.
ftp://ftp.research.microsoft.com/pub/tr/tr-97-20.ps

[FLO95] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", Proceedings of ACM SIGCOMM '95, Cambridge, MA, August 1995.
ftp://ftp.ee.lbl.gov/papers/wb.tech.ps.Z

[c't00] von Leitner, Felix: „Bilder für alle. Multicast: Sendeverfahren für Audio und Video im Netz", c't 2000, Heft 12.

[FRA99] Franke, Mario ; „Entwicklung und Implementierung einer Native-ATM-Datentransportschnittstelle für das Videokonferenzsystem ‚Visitphone'", Diplomarbeit, Universität Leipzig, Institut für Informatik 1999.

[HOR99] Horn, Sven ; „Entwurf und Implementierung eines Sitzungsmanagement-Systems für das Multicast-Videokonferenzsystem ‚Visitphone'", Diplomarbeit, Universität Leipzig, Institut für Informatik 1999.

[HTM92] T.Hoshi, Y.Takahashi, K.Mori : „An Integrated Multimedia Desktop Communikation an Collaboration Platform for Broadband ISDN", Proceedings of Multimedia 92

[IET94] IETF. „Minutes of the multipart multimedia session control working group (mmusic)", Proceedings of 29th IETF, 1994

[IRM00] Irmscher, Klaus: Lehrveranstaltung „Rechnernetze", Lehrstuhl Rechnernetze und Verteilte Systeme, Universität Leipzig, Institut für Informatik 2000.

[IRM99] Irmscher, Klaus : „Technische Infrastruktur und Nutzungsszenarien beim Einsatz von TeleTeaching", in TeleTeaching: eine neue Komponente in der universitären Lehre, RAABE, 1999.

[KER98] Kerres: „ Multimediale und telemediale Lernumgebungen", Oldenburg 1998.

[KUR00] Kurkov, Igor: „Sitzungsmanagement und Ressourcenverwaltung für ein Distance-Learning-Tool zur Erstellung und Verwaltung multimedialen Lehr- und Lernmaterials", Diplomarbeit, Universität Leipzig, Institut für Informatik 2000.

[MSDN] Microsoft Developer Network Library 2000, Microsoft Corporation 2000.

[MVC98] „Microsoft Visual C++ 6.0, Programmierhandbuch", Microsoft Press 1998.

[NGU98] G. Nguyen: „Entwicklung eines Videotransportprotokolls über UDP", Diplomarbeit, Universität Leipzig, Institut für Informatik 1998.

[ORF97] Orfali, Robert; Harkey Dan; Edwards, Jeri: „Abenteuer Client/Server", Addison-Wesley 1999.

[ORF99] Orfali, Robert; Harkey Dan; Edwards, Jeri: „Client/Server Survival Guide", Wiley Computer Publishing 1999.

[PRO99] Prosise, Jeff: „Windows-Programmierung mit MFC: Objektorientierte Programmierung für Alle 32-bit Windows Plattformen", 2. Auflage, Microsoft-Press 1999.

[RFC760] Postel, J.: "Internet Protocol," RFC 760, 1980.
http://www.cis.ohio-state.edu./htbin/rfc/rfc0760.html

[RFC761] Postel, J.: „Transmission Control Protocol", RFC 761, 1980.
http://www.cis.ohio-state.edu./htbin/rfc/rfc0761.html

[RFC768] Postel, J.: „User Datagram Protocol", RFC 768, 1980.
http://www.cis.ohio-state.edu./htbin/rfc/rfc0768.html

[RFC791] Information Sciences Institute University of Southern California: „INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION" , RFC 791, 1981.
http://www.cis.ohio-state.edu./htbin/rfc/rfc0791.html

[RFC793] Information Sciences Institute University of Southern California: „TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION", RFC 793, 1981.
http://www.cis.ohio-state.edu./htbin/rfc/rfc0793.html

[RFC1122] Braden, R: „Requirements for Internet Hosts - Communication Layers", RFC 1122, 1989.
http://www.cis.ohio-state.edu./htbin/rfc/rfc1122.html

[ROS00] Roskin, Alexander: „Benutzer- und Ressourcen-/Dokumentenverwaltung für ein Distance-Learning-Tool zur Erstellung und Verwaltung multimedialen Lehr- und Lernmaterials", Diplomarbeit, Universität Leipzig, Institut für Informatik 2000.

[ROS93] Rose, Marshall T.: „Verwaltung von TCP/IP-Netzen" , Prentice Hall 1993.

[SIN97] Sinha, Alok K.: „Netzwerkprogrammierung unter Windows NT 4.0: Systemarchitektur und Methoden der Interprozess-Kommunikation", Addison-Wesley 1997.

[STE98] Steinmetz, Ralf: „Multimedia-Technologie: Einführung und Grundlagen", Springer-Verlag 1998.

[STR92] Stroustrup, Bjarne: „Die C++ Programmiersprache", Addison-Wesley 1992.

[STE99] Steinmetz, Ralf: „Multimedia-Technologie: Grundlagen, Komponenten und Systeme", Springer-Verlag 1999.

[TAN98] Tanenbaum, Andrew S.: „Computernetzwerke", Prentice Hall 1998.

[QUI95] Quinn, Bob; Shute, Dave: „Windows Sockets Network Programming", Addison-Wesley 1995.

[WIT99] Wittmann, Ralf; Zitterbart Martina: „Multicast. Protokolle und Anwendungen", dpunkt-Verlag 1999.

[WSM91] K. Watabe, S.Sakata, K.Maeno : „Distributed Desktop Conferencing System with Multi-user Multimedia Interface", IEEE JSAC, 9(4):531-539, 1991

[YOU97] Young, Michael : „Das Programmierbuch Visual C++ 5", Sybex 1997

Anhang A: Das TCP/IP Referenzmodell

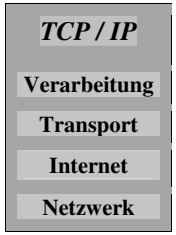

Abbildung A.0.1 TCP/IP-Referenzmodell

Das TCP/IP-Referenzmodell wurde zunächst im APRANET angewendet, aus dem sich später das Internet entwickelt hat. Das Modell beschreibt die Architektur der Kommunikationsprozesse im Internet und ist aus vier voneinander unabhängigen Protokollschichten zusammengesetzt, deren Funktion den Diensten der entsprechenden Schichten des OSI-Referenzmodells ähnelt (siehe Abbildung A.0.1).

.1 Die Netzwerkschicht

Die *Netzwerkschicht* ist für das Versenden und das Empfangen von IP–Pakete verantwortlich. Die Funktionalität dieser Schicht umfasst die Bitübertragungs- und Sicherungsschicht des OSI-Referenzmodells. Das TCP/IP-Referenzmodell sagt nur, dass sich ein Host über ein bestimmtes Protokoll am Netz anschließen muss, um die IP-Pakete darüber versenden zu können.

.2 Die Internetschicht

Die *Internetschicht* stellt einen paketvermittelnden Dienst dar und entspricht in ihrer Funktion der Vermittlungsschicht des OSI-Referenzmodell. Ihre Aufgabe ist es zu ermöglichen , die Pakete von der Quelle zum Ziel zu transportieren. Der Zielhost kann sich dabei in einem anderen Netzwerk befinden. Dafür stellt die Internetschicht

entsprechende Routingfunktionen zur Vermittlung der Pakete in die einzelnen Teilnetze zur Verfügung.
Die Internetschicht definiert ein offizielles Paketformat und Protokoll IP (*Internet Protokoll*).

.3 Die Transportschicht

Die Aufgabe der *Transportschicht* ist analog zur gleichnamigen Schicht im OSI-Referenzmodel. Sie soll den Datentransport von einem Quell- zum Zielhost unabhängig von der verwendeten Netzstruktur ermöglichen. Hier wurde zwei Ende-zu-Ende-Protokolle definiert.
Das *TCP (Transmission Control Protocol)* ist ein zuverlässiges verbindungsorientiertes Protokoll.
Das *UDP (User Datagram Protocol)* ist ein unzuverlässiges verbindungsloses Protokoll.

.4 Die Verarbeitungsschicht

Das TCP/IP-Modell hat keine Sitzungs- und Darstellungsschicht. Die meisten Anwendungen brauchen diese Schichten nicht. Über der Transportschicht befindet sich die *Verarbeitungsschicht*. Sie umfasst alle höheren Protokolle. Zu den ersteren zählen:

- *TELNET* (virtuelle Terminal),
- *FTP* (File Transfer Protocol - Dateitransferprotokoll),
- *SMTP* (Simple Mail Transfer Protocol – E-Mail).

Im Laufe der Jahre wurden viele andere Protokolle hinzugefügt, z.B. *DNS* (Domain Name System), *RIP* (Routing Information Protocol), *SNMP* (Simple Network Management Protocol) und *HTTP* (HyperText Transfer Protocol).

Anhang B: Die Internet Protokoll Familie

Die Internet-Protokoll-Familie bildet eine Sammlung von Kommunikationsprotokollen, die für eine Verwendung im weltweiten Internet entwickelt wurden. Ziel dieser Entwicklung war es, einen Kommunikationsstandard zu schaffen, der es ermöglicht, heterogene Computernetze nahtlos zusammenzuschließen. Auf Grund der guten Standardisierung, sowie der weiten Verbreitung erlangte diese Protokollfamilie eine große Bedeutung auch außerhalb des Internets.

.1 Das Internet Protokoll (IP)

Das *IP-Protokoll* wird in der Internet-Schicht definiert. Seine Aufgabe ist die Bereitstellung einer Möglichkeit, nach bestem Bemühen als Datagramme bezeichnete Datenpakete von der Quelle zum Ziel zu befördern, unabhängig davon, ob sich diese Maschinen im gleichen Netz befinden oder ob andere Netze dazwischenliegen.
Ein IP-Datagramm besteht aus einem Header- und Textteil und kann maximal 65.535 Byte lang sein. Der Header hat einen festen 20 Byte großen Teil und einen optionalen Teil mit variabler Länge. Der Aufbau des Headers eines IP-Datagramms wird in der Abbildung B.0.1 dargestellt.

Bedeutung einzelner Felder des Headers:

- *Version* – Version des Protokolls
- *IHL* – Header- Länge in 32-Bit-Wörtern
- *TOS (Type of service)* – der gewünschte Dienst
- *Total Length* – Länge des kompletten Datagramms bis max. 65.535 Byte
- *Identification* – erforderlich, um die Zugehörigkeit eines neu angekommenes Fragments zu einem Datagramm festzustellen
- *DF (Don`t Fragment)* – Befehl an die Router, wenn ein Datagramm nicht fragmentiert werden darf
- *MF (More Fragments)* – bei allen Fragmenten außer dem letzten gesetzt
- *Fragment Offset* – bezeichnet, an welche Stelle im Datagramm ein Fragment gehört
- *Time to Live* – Zähler, mit dem die Lebensdauer von Paketen begrenzt werden kann
- *Protocol* – bestimmt an welchen Transportprozess ein Datagramm weiterzugeben ist
- *Header Checksum* – Prüfsumme des Headers
- *Source Address* und *Destination Address* – bezeichnen die Netz- und Hostnummer
- *Options* – ist für spätere Versionen gedacht, derzeit nur fünf Optionen definiert sind.

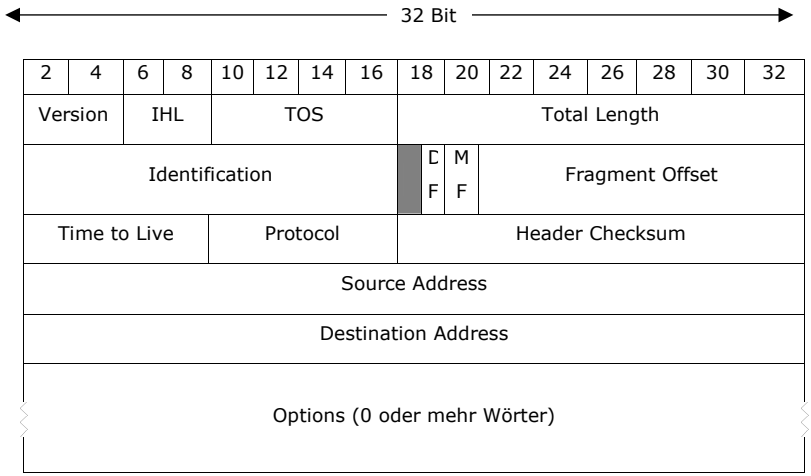

Abbildung B.0.1 Schematische Darstellung eines Headers des IP-Protokolls

.2 Das TCP-Protokoll

Das *TCP* (*Transmission Control Protocol*) ist ein zuverlässiges verbindungsorientiertes Protokoll, durch das ein Bytestrom von einer Maschine fehlerfrei im Internet einer anderen Maschine zugestellt wird.

Eine TCP-Instanz nimmt eingehende Byteströme von lokalen Prozessen entgegen, zerlegt sie in höchstens 64Kbyte einzelne Datenblöcke, die an die Internet-Schicht weitergeleitet werden. Jeder Datenblock wird dann als getrenntes IP-Datagramm über das Netz versendet.

Das TCP enthält auch die Flußteuerung für die Synchronisation zwischen Sender und Empfänger. Kommen die IP-Datagramme mit TCP-Daten beim Empfänger an, werden sie an die TCP-Instanz weitergegeben, die den ursprünglichen Bytestrom wieder zusammensetzt. Im Falle fehlerhafter oder fehlender Datenpakete wird beim Sender die erneute Übertragung beantragt. Werden die Datagramme in einer falschen Reihenfolge empfangen, so werden sie entsprechend sortiert.

Damit das TCP den fehlerfreien Datentransport bereitstellen kann, muss vor der eigentlichen Datenübertragung ein formeller Verbindungsaufbau zwischen Empfänger und Sender erfolgen. TCP-Verbindungen sind stets vollduplexe Punkt-zu-Punkt-Verbindungen, d.h. an der Kommunikation können immer genau zwei Stationen teilnehmen. Multicast- oder Broadcast-Pakete werden nicht unterstützt. Der Datenstrom innerhalb einer Verbindung kann in beide Richtungen fließen.

Um einen TCP-Dienst nutzen zu können, müssen Sender und Emfänger Endpunkte – sogenannte Sockets – erstellen. Die Sockets werden in dieser Arbeit in Kapitel 2.6 behandelt.

Bei der Kommunikation tauschen die sendende und die empfangende TCP-Einheiten Daten in Form von Segmenten aus. Ein Segment besteht aus einem festen 20-Byte-Header (siehe Abbildung B.0.2), einem optionalen Teil und den Datenbytes. Die Grösse von Segmenten wird von der TCP-Software festgelegt.

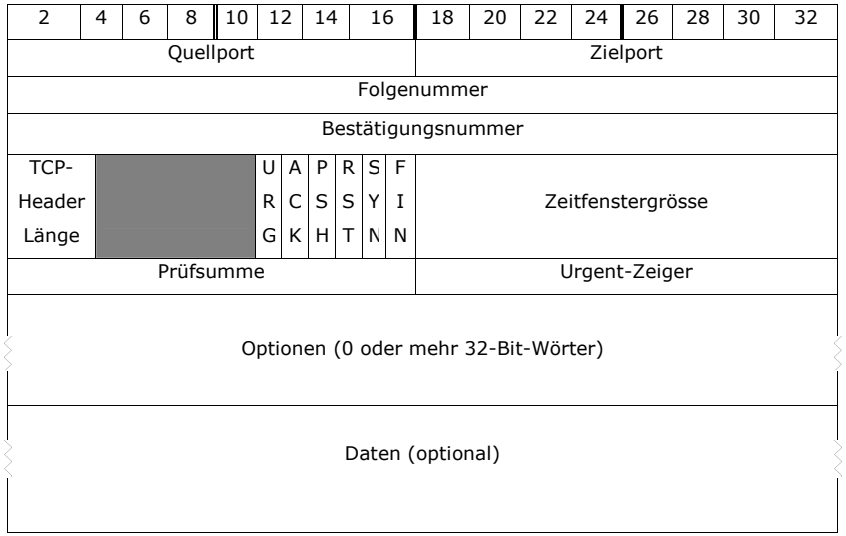

Abbildung B.0.2 Schematische Darstellung eines TCP-Headers

Die Grenzen von Nachrichten werden vom TCP-Dienst nicht beibehalten. Die Blockgröße, mit der die Daten empfangen werden, muss mit der ursprünglichen Blockgröße der gesendeten Daten nicht übereinstimmen. Die TCP-Schicht überträgt somit einen Bytestom, ohne Rücksicht auf Struktur und semantik der darin enthaltenen Daten.

Hier ist die Bedeutung einzelner Felder des Headers:

- *Quellport* und *Zielport* – identifizieren die lokalen Endpunkte der Verbindung
- *Folgenummer* – gibt das als nächstes erwartete Byte an
- *TCP-Header-Länge* – gibt an, wie viele 32-Bit-Wörter im TCP-Header enthalten sind
- 1-Bit-Flags:
  - *URG* – wird auf 1 gesetzt, wenn Urgent Pointer benutzt wird
  - *ACK* – wird auf 1 gesetzt, um anzugeben, dass die Bestätigungsnummer gültig ist
  - *PSH* – bezeichnet PUSH-Daten
  - *RST* – wird benutzt, um eine Verbindung zurückzusetzen, wenn ein Host abgestürzt oder eine ähnliche Störung aufgetreten ist
  - *SYN* – wird benutzt, um Verbindungen aufzubauen
  - *FIN* – bestimmt, dass der Sender keine weiteren Daten mehr zu übertragen hat
- *Zeitfenstergrösse* – bezeichnet, wie viele Bytes ab dem bestätigten Byte gesendet werden können
- *Prüfsumme* – prüft den Header, die Daten und den konzeptionellen Pseudo-Header, um hohe Zuverlässigkeit zu bieten
- *Optionen* – bieten eine Möglichkeit, zusätzliche Funktionen bereitzustellen, die im normalen Header nicht verfügbar sind
- *Daten* – bis zu 65.535-20-20=65.515 Datenbyte, wobei sich die ersten 20 auf den IP-Header und die zweiten 20 auf den TCP-Header beziehen

.3 Das UDP-Protokoll

Das *UDP* (*User Datagram Protocol*) ist ein unzuverlässiges verbindungsloses Protokoll für Anwendungen, die anstelle der Abfolge oder Flußkontrolle von TCP diese Aufgaben selbst bereitstellen. Es bietet eine Möglichkeit für Anwendungen, gekapselte rohe IP-Datagramme zu übertragen, ohne eine Verbindung aufzubauen. Die vollständige und

fehlerfreie Übertragung sowie die richtige Reihenfolge der Segment können nicht garantiert werden.

Anders als bei TCP werden die übertragenen Datenblöcke beim Empfänger in der ursprünglich gesendeten Größe wieder bereitgestellt.
Das Versenden von Multicast- bzw. Broadcast-Datenpakete wird von UDP unterstützt.
Ein UDP-Segment besteht aus einem 8-Byte-Header, gefolgt von den Daten. Der Header ist in der Abbildung B.0.3 dargestellt.

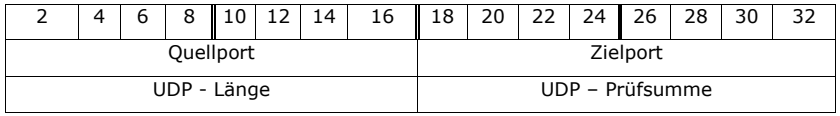

Abbildung B.0.3 Schematische Darstellung eines UDP-Headers

Anhang C: Quelltexte

Wie im Kapitel 3.2 beschrieben wurde, bauen die Anwendungen zusätzlich zu den Steuer- und Datenverbindungen zu den eigenen Services auch eine weitere Verbindung zum zentralen Service auf. Dies geschieht, um die Verwaltung der Dienstabläufe vom Sitzungsmanagement zu gewährleisten. Zwei Klassen: *CCtrlMsg* und *CClientSocket* stellen die Schnittstelle zum Sitzungsmanagement dar.

```
class CClientSocket : public CSocket
{
DECLARE_DYNCREATE(CClientSocket)

// Konstruktoren
public:
      CClientSocket();  // Default
      CClientSocket(enum _SockID sockID);
      virtual ~CClientSocket();

// Attribute
public:
      enum _SockID      m_sockID;         // Typ des Sockets
      CSocketFile*      m_pFile;          // Datenobjekt
      CArchive*         m_pArchiveIn;     // Archive zum Empfangen
      CArchive*         m_pArchiveOut;    // Archive zum Senden
      BOOL              m_Status;         // Status

//Member-Funktionen
public:
// Objektzustand wird zurückgegeben
      BOOL Status();
// Empfangen von Objekten; ein Zeiger auf Basisklasse wird zurückgeliefert
      CObject* EmpfangObj(void);
// Senden von Nachrichten; Zeiger auf Basisklasse wird übergeben
      void SendObj(CObject* pObj);
// überladene Funktionen; eine Referenz auf die Basisklasse wird übergeben
      void EmpfangObj(CObject& Obj);
      void SendObj(CObject& Obj);
 // SocketInitialisierung
      BOOL InitSocket(LPCTSTR lpszAddress, UINT nPort);
};
```

Der Verbindungsaufbau mit dem Server findet mittels der Funktion *InitSocket* statt, indem man den Namen bzw. die Internet-Adresse sowie die Portnummer des Servers als Parameter übergibt.

```
class CCtrlMsg : public CObject
{
      DECLARE_SERIAL(CCtrlMsg)

// Konstruktoren
public:
      CCtrlMsg(){}; // Default
      virtual ~CCtrlMsg();
      CCtrlMsg(CString name, enum _FktnTyp fTyp = F_UNDEF,
                                              CString str = _T(""));
// Attribute
public:
      enum _FktnTyp     m_FTyp;     // Nachrichtentyp
      CString           m_Name;     // Bezugsclient
      CString           m_Str;      // Nachrichtenparameter

//Member-Funktionen
public:
// Überschreiben der Serialize-Funktion der Basisklasse
      void Serialize(CArchive &archive);
//Setzt die Nachrichtenattribute (NachrichtenBody)
      void Set(enum _FktnTyp fTyp = F_UNDEF, CString str = _T(""));
};
```

Mit Hilfe der Klasse *CCtrlMsg* wird eine Kontrollnachricht erstellt und mittels der Funktion *EmpfangObj* bzw. *SendeObj* der Klasse *CClientSocket* wird sie über das Netz transferiert.

Die Chat-Applikation verwendet die Klasse *CClientSocket* auch für die Übertragung von Chat-Nachrichten (Paketen), indem die Objekte der Klasse *CDataMsg* zum Senden bzw. Empfangen an das *CClientSocket*-Objekt übergeben werden. Insgesamt existieren zur Laufzeit drei Objekten der Klasse *CClientSocket*. Jedes Objekt stellt eigene Verbindung zu den entsprechenden Servern dar. Der Unterschied liegt darin, dass die Gruppenverwaltung und Nachrichtenverteilung in einer Applikation (Chat-Server) implementiert wurden, während das Sitzungsmanagement eine eigene Anwendung, die auch unabhängig vom Chat-Server läuft, darstellt.


```
class CChatClientApp : public CWinApp
{
// Konstruktor
public:
      CChatClientApp();

// Attribute
public:
      CClientSocket*    m_pDataSock;      // Datenverbindung
      CClientSocket*    m_pCtrlSock;      // Kontrollverbindung
      CClientSocket*    m_pMngSock;       // Sitzungsmanagementverbindung
      CDataMsg*         m_pSendMsg;       // Nachricht zum Senden
      CString           m_Gruppe;         // Gruppenname
      CString           m_Name;           // Benutzername

//Member-Funktionen
public:
// Empfangen der Kontrollnachrichten vom Chat-Server
      void RecCtrlMsg(void);
// Empfangen der Nachrichten mit Daten vom Chat-Server
      void RecDataMsg(void);
// Empfangen der Nachrichten vom Sitzungsmanagement
      void RecMngMsg(void);
// DienstInitialisierung
      BOOL InitChatClient(void);
};
```

Die Funktion *InitChatClient* erledigt die Initialisierung der Dienst-Applikation: den Verbindungsaufbau zu entsprechenden Servern, den Beitritt der Gruppe, das Erhalten der Kontrollinformationen usw.

Das ***enum*** *_FktnTyp* beschreibt die Nachrichtentypen. Dieses wird wie bei Daten-Nachrichten, als auch bei der Kontrollnachrichten verwendet.

```
enum _FktnTyp{
      F_LISTE,          /* GruppenListe-Anforderung   */
      F_GRUPPE,         /* GruppenName                */
      F_MSG,            /* UserDataPaket-Anforderung  */
      F_USERINIT,       /* UserInitialisierung        */
      F_ABORT,          /* Abbruch                    */
      F_UNDEF           /* Undefinierte Anforderung   */
};
```

Die Klasse CDataMsg kapselt die Daten-Nachrichten ein. Die enthält die Attribute wie den Benutzernamen, der diese Nachricht erzeugt hat, die Farbe, mit der den Text dieser Nachricht dargestellt werden soll.

```
class CDataMsg : public CObject
{
      DECLARE_SERIAL(CDataMsg)

// Attribute
public:
      Cstring     m_Name;     // Nutzername
      COLORREF    m_Farbe;    // Textfarbe
      Cstring     m_Data;     // Text

//Member-Funktionen
public:
      ...
};
```

Die FTP-Anwendung ist eine dialogbasierte Applikation. Die Klasse *CFTPDlg* ist von der Klasse *CDialog* abgeleitet und enthält die Funktionalität eines Dialogfensters.

```
class CFTPDlg : public CDialog
{
public:
      CFTPDlg(CWnd* pParent = NULL);      // Standard-Konstruktor
// Attribute
      static int m_prevSort;              // Sort-Status
      int m_prevSel;                      // Select-Status
      BOOL m_aktiveListe;                 // aktives Fenster
      CImageList m_BildList;              // Liste mit Icons
      CStatusBarCtrl m_StatusBar;         // status bar
      CInternetSession* m_pInetSession;   // Internet-Sitzung
      CFtpConnection*   m_pFtpConnection; // FTP-Verbindung
      BOOL              m_ConnectedFlag;  // Verbindungsstatus

//Funktionen
protected:
// Zeigt die Dateien des aktuellen Verzeichnisses
      void dir();
      BOOL trennen();                     //Verbindung trennen
      BOOL verbinden();                   // Verbindung aufbauen
      BOOL copyDir(CString& dirName);     // Daten Kopieren

      ...
};
```


FTP-Applikation verwendet die WinInet-Unterstützungsklassen. Bei der Dienstausführung wird zuerst die Internet-Sitzung erzeugt:

```
m_pInetSession = new CinternetSession (AfxGetAppName(), 1,
                                          INTERNET_OPEN_TYPE_PRECONFIG);
```

Danach wird die Verbindung zum entfernten Server erstellt:

```
m_pFtpConnection = m_pInetSession -> GetFtpConnection (m_strServerName,
                                                  pstrUserName, pstrPassword);
```

Dabei werden den Servername, das Login und Passwort übergeben.

Die wichtige Klasse in der Whiteboard-Anwendung ist die Klasse CMySock, die eine Multicast-Kommunikation ermöglicht.

```
class CMySock : public CAsyncSocket
{
// Attribute
protected:
      DWORD       m_theIP;          // IP-Adresse in der DWORD-Form
      Int         m_nPort;          // Port
      Cstring     m_grpAddr;        // Gruppenadresse
      CWBDoc*     m_pDoc;           // Zeiger auf Hauptfenster
private:
      BOOL RecSock;                 // Socket zum Senden oder Empfangen?

// Operationen
public:
      BOOL InitSendSock();    // Initialisierung als Senden-Socket
      BOOL InitRecSock();     // Initialisierung als Empfangen-Socket
      int Holen(LPTSTR Buf, int BufLen);        // Empfangen von Daten
      int Senden(LPCTSTR message, int MesLen); // Senden der Daten

      ...
};
```