Zusammenfassung „Verteilte Systeme 1" SS1995

Michael Jaeger (michael.jaeger@in-flux.de)

29. Oktober 2001

Inhaltsverzeichnis

1 Einleitung

Verteiltes System (Distributed System) besteht aus mehreren **autonomen Subsystemen**, die bewusst und **koordiniert kooperieren**, um eine **gemeinsame Aufgabe** zu erfüllen.

Verteilte Verarbeitung (Distributed Processing) ist die Verarbeitung in einem verteilten System.

Verteilte Anwendung (Distributed Application) ist eien Anwendung in einem verteilten System.

Desweiteren gibt es noch Begriffe wie **Cooperative Processing**, **Parallel Processing**, **Network Computing**,...

Tanenbaum definiert ein verteiltes System in [Tanenbaum1995] folgendermassen:

> „Ein verteiltes System ist eine Sammlung voneinander unabhängiger Computer, die dem Benutzer des Systems den Eindruck vermitteln, es handle sich um einen einzigen Computer."

Dabei wird eine Unterscheidung zwischen **Multiprozessor Computern** (Prozessoren besitzen einen gemeinsamen Speicher) und **Multicomputern** (Prozessoren habe keinen gemeinsamen Speicher) getroffen. Bei der Kopplung der Computer wird wiederumg zwischen **eng und lose gekoppelt** unterschieden, was die Verzögerungen, die beim Senden von Nachrichten entstehen beschreibt.

1.1 Historische Entwicklung

Eine historische Übersicht über die Entwicklung im Bereich der verteilten Systeme:

...-1960 Erste Gehversuche
Röhrenrechner, Stapelbetrieb, erste Programmiersprachen,...

1960-1970 Time Sharing
IBM/360 und /370-Rechnerfamilien, Terminal-Netze, Reservierungssysteme, UNIX, problemorientierte Programmiersprachen,...

1970-1985 Dezentralisierte Kommunikation
Minis, Mikorprozessoren, PCs, Workstations, LANs, SNA, OSI, Erforschung verteilter Systeme, ...

1985-2000 Kooperation in verteilten Systemen
Leistungsexplosion bei Hardware und Kommunikation, Produkte, Dateiserver, Remote Procedure Call, REifungsprozess, OSF DCE, OMG CORBA, ODP,...

2000-... Integration?
Komponententechnologie, Peer-to-Peer, ad-Hoc Networking, Gruppenkommunikation, ...

Meilensteine im Bereich der verteilten Systeme stellen folgende Systeme dar:

1977 Xerox DFS

1979 Cambridge DCS (→Processor Pool Architecture)

1980 Locus (IBM, „advanced distributed operating system that emulated UNIX")

1980 Newcastle Connection (Vorgänger von Sun NFS)

1981 Grapevine (Xerox, „distributed, replicated application-oriented database service")

1982 Cedar (Xerox, „research environment for development od office and personal systems")

1982 Accent (CMU, „distributed OS based on message passing, precursor to Mach")

1983 Argus (MIT, „research project to develop an integrated programming language and system based on the language CLU")

1984 Amoeba (Vrije Universität, „research project on the use of **capabilities** in distributed systems")

1985 UNIX BSD 4.2 + Sun NFS (Sun, distributed filesystem)

1986 Mach (CMU, „OS-kernel for distributed systems, successor to Accent")

1988 Chorus (Chorus, „OS-kernel for distributed an real-time systems")

Middleware

Eine **Verteilungsplattform** (**Midleware**) ist eine Software, die die Realisierung verteilter Anwendungen unterstützt. Varianten sind z.B. Netzwerkbetriebssysteme, verteiltes Betriebssystem, Message Queues, Datenbankzugriff übers Netz, Object Broker,... Meist wird hier zwischen **Kern** und **Dienst** unterschieden.

Vor-/Nachteile verteilter Systeme

Vorteile: **Ressource Sharing** (gemeinsame Betriebsmittel: Datenverbund, Funktionsverbund, Lastverbund, Leistungsverbund, Verfügbarkeitsverbund)
Offenheit (Kompatibilität, Modularität, Auswahlmöglichkeiten, Herstellerunabhängigkeit)
Flexibilität (Anpassung an Geschäftsprozesse, Rekonfiguration einfacher, inkrementelles Wachstum→Skalierbarkeit)
Ergonomie (GUI)
Ökonomie (geringere Kosten für Hard- und Software-Beschaffung)

Nachteile **Komplexität** (Programmierung, Sicherheit, Management, Vernetzung)
Kosten (erhöhte Kosten für Netzwerk-Infrastruktur und Verwaltung der Heterogenität, Personalaufwand nicht geringer)
Verfügbarkeit (Nachholbedarf bei klassischen Anwendungen)
Trainingsaufwand (Umgang mit heterogenen Komponenten in komplexer, vernetzter Umgebung)

Entstehungsprozess

Der **Entstehungsprozess** verteiler Systeme gliedert sich in drei Schritte:

Downsizing Umstellung von zentraler DV mit Großrechnern auf verteilte DV mit Workstations und PCs.

Rightsizing ökonomisch und funktionell sinnvolle Verteiltung der DV-Funktionen auf Großrechner, Midrange Computer und Workstations.

Zusammenführung organisatorischer Einheiten Firmenzusammenschlüsse, Firmenkooperationen, Reorganisation, ...

2 Verteilte Systeme

2.1 Eigenschaften verteilter Systeme

Separation

Die **Separation der Aspekte** bedingt

- **mehr Kommunikation** (Schichten: Anwendung, Verteilungsplattform, Betriebssystem mit Transportsystem)
- **echte Parallelität der Verarbeitung**
- **unabhängig Systemuhren in den beteiligten Systemen**
- **Unsicherheit über den globalen Zustand**
- **Möglichkeit des Teilausfalls des Gesamtsystems**
- **inkrementelles Wachstum**

Heterogenität

Die **Heterogenität** in verteilten Systemen bezieht sich auf

- **Betriebssysteme**
- **Hardware-Architekturen**
- **Kommunikationsarchitekturen**
- **Programmiersprachen**
- **Software-Schnittstellen**
- **Zugangsschutz**
- **Betreibergrundsätze**
- **Informationsarten**

Als **Transformationsverlust** bezeichnet man den Verlust an Leistung, der entsteht, wenn Daten tranformiert werden müssen. Der Einsatz einer Verteilungsplattform soll für einen Ausgleich sorgen, so dass sich die Anwendungsprogrammierer darüber keine Gedanken mehr machen müssen. Gleichzeitig stellt sie **standardisierte Schnittstellen** zur Verfügung.

Ein **offenes System** ist nach **POSIX**

> „Ein System, dass in ausreichendem Maße offene Spezifikationen für Schnittstellen und dazugehörige Formate implementiert, damit entsprechende Anwendungssoftware
> • auf eine Vielzahl verschiedener Systeme (mit Anpassungen) **portiert** werden kann,
> • mit anderen Anwendungen lokal und entfernt **interoperabel** ist und
> • mit Benutzern in einer Art interagiert, die das Wechseln der **Benutzer** zwischen Systemen erleichtert."

Die offene Zugänglichkeit von Schnittstellendefinitionen in verteilten Systemen soll also

- **Portabilität** und die
- **Interoperabilität** (Ressource Sharing)
- **Kompatibilität** (Kooperation)
- **Herstellerunabhängigkeit** (Auswahlmöglichkeit)
- **Wettbewerb** (fairer Wettbewerb zwischen Anbietern)
- **Kohärenz** (einheitliche Erscheinung)
- **Erweiterbarkeit** (Anpassungsmöglichkeiten)

garantieren. Nachteile offener Systeme sind:

- **Zusätzliche Komplexität** durch Heterogenität der Komponenten (Beschaffung, Training, Sicherheit, Management, Wartung, Installation)
- **offizielle Standardisierung zu langsam** und arbeitet oft am Anwender vorbei
- **Verunsicherung wegen Vielfalt von Standards** bei Anwendern und Herstellern

parenz

Das Verbergen der komplexen Mechanismen in verteilten Systemen soll durch **Transparenz** erreicht werden. Sie steigert die Produktivität und ist sowohl für die Programmierung als auch für die Benutzung wichtig, weniger für die operative Kontrolle. Ganz allgemein spricht man von **Verteilungstransparenz**.

Im Speziellen spricht man von verschiedene **Arten der Transparenz**:

- Ortstransparenz (**Location Transparency**)
- Zugriffstransparenz (**Access Transparency**)
- Migrationstransparenz (**Migration Transparency**)
- Replikationstransparenz (**Replication Transparency**)
- Fehlertransparenz (**Failure Transparency**)
- Föderationstransparenz (**Federation Transparency**)
- Gruppentransparenz (**Group Transparency**)
- Ressourcentransparenz (**Ressource Transparency**)
- Transaktionstransparenz (**Transaction Transparency**)
- Nebenläufigkeit, Performance, Wachstum

Ein Beispiel ist der **Remote File Access**.

Die **Transparenzfunktionen** können in verschiedenen Ebenen verankert werden:

1. **Betriebssystem** (als Netzbetriebssystem oder verteiltes BS)
2. **Programmiersprache** (als verteilte Programmiersprache)
3. **Sub-Systeme** (verteilte Datenbanken, Remote Database Access)
4. **Anwendung** (individuelle Mechanismen)

Autonomie

Unter der **Autonomie** wird der Grad der Abhängigkeit von Anderen bzgl. **Entscheidungen**, **Systemkonfiguration**, **Kontrolle** über Einbringung eigener Ressourcen in den Verbund, indivduelle vs. globale **Interessen** (z.B. bei Lastverteilung) verstanden. In einem verteilten Betriebssystem gibt es im Allgemeinen wenig Autonomie bei den Knoten, ein Netzwerkbetriebssytem dagegen läßt potenziell mehr Autonomie. Der Aspekt der Autonomie muss bei Sicherheit und Management beachtet werden.

Flexibilität

Die **Flexibilität** in verteilten Systeme ist durch die Möglichkeit der **Rekonfiguration** (die fast den Normalfall darstellt) gekennzeichnt. Es gibt als keine „festverdrahteten" Konstellationen sondern **Erweiterbarkeit** und **Offenheit**. Um dies zu erreichen ist die **Transparenzfunktion** eine Basis der Verteilungsplattform. **Flexibilität** ist für viele kommerzielle Unternehmen die **wichtigste Eigenschaft** verteilter Systeme.

2.2 Modelle verteilter Systeme

Client/Server-Modell

Das Standard-Modell verteilter Systeme war (ist) das sog. **Client/Server-Modell**. Dabei schickt ein Client eine Anfrage an einen Server, der diese bearbeitet und eine Antwort an den Client zurückschickt. Der Server erbringt also einen **Service**, auf den über die **Service-Schnittstelle** zugegriffen werden kann. Im Allgemeinen wird diese in Form einer $n : m$-Beziehung erbracht. Das C/S-Modell wird im Allgemeinen nicht nur mit einer Software- sondern auch mit einer Hardware-Konfiguration assoziiert.

Die Rollen im C/S-Modell können aber auch wechseln, so kann ein Server auch gleichzeitig Client eines anderen Servers sein und Server können **hierarchisch konfiguriert** werden. Unter einem **Client/Server-System**, einer **Client/Server-Architektur** und einer **Client/Server-Anwendung** verstehen wir allgemein ein verteiltes System/Architektur /Anwendung, die auf dem C/S-Modell aufbaut.

Kooperationsmodelle

Wir unterscheiden dabei die **logische und physische Zuordnung**.

Eine weitere Unterscheidung von C/S-Systemen betrifft die Aufteilungen in **Stufen** oder **Tiers**. So wird ein dreistufiges C/S-Konfigurationsmodell in **Präsentationsschicht**, **Anwendungsschicht** und **Datenschicht** aufgeteilt (typische **3-Tier-Konfiguration**).

Eine anderes Modell ist das **Producer/Consumer** Modell, in dem Produzenten Informationen produzieren, die von Clients konsumiert werden. Die Interaktion erfolgt hier **asynchron** und wird typischerweise in der **Steuer- und Leittechnik** (Sensoren für Produzenten) eingesetzt.

Bei **Peer-to-Peer**-Modellen ist jeder Netzwerkknoten gleichzeitig Server und Client, besitzt also GUI, Anwendung und Daten.

Die **Gruppenarbeit** in verteilten Systemen, also das Arbeiten an einem gemeinsamen Objekt, ist ein weiters Kooperationsmodell. Die Mitgliedschaft in einer Gruppe kann sich dynamisch ändern, was zu komplexen Problemstellungen (**Sicherheit der atomaren Kommunikation**, **Konsistenz der Daten** etc.) führt. Sie stellt as Basismodell für **Computer Supported Cooperative Work** (CSCW) dar.

2.2.1 Open Distributed Processing (ODP)

ODP Referenzmodell

Das **Open Distributed Processing** (ODP) Modell wurde von der ISO und der ITU-T standardisiert. Ziel war ein **Referenzmodell** als generisches Rahmenwerk für die Standardisierung und Standards für **Komponenten und Beschreibungshilfsmittel für typische Dienste** (z.B. Trader) und FDTs zu entwickeln. Dabei besteht es aus den drei Komponenten **Transparenz**, **Funktionen** und **Viewpoints**.

Ziele des ODP Referenzmodells waren:

- **Portabilität** von Anwendungen über heterogene Plattformen

- **Zusammenarbeit** zwischen ODP-Systemen (Austausch von Daten, Benutzung von Diensten im verteilten System)
- **Verteilungstransparenz** für den Programmierer und den Anwender

Die **Struktur** des Referenzmodells besteht im Groben aus 4 Punkten:

1. **Umfang und Überblick** (Ziele, Reichweite, grundlegende Begriffe, Inhaltsübersicht)
2. **Deskriptives Modell** (normativ) (Beschreibungskonzepte zur Modellierung beliebiger verteilter Systeme)
3. **Präskriptives Modell** (normativ) (generische Architektur für ODP Systeme)
4. **Architekturelle Semantik** (Formalisierung der Archiekturkonzepte)

Ein wichtiges Element des ODP Referenzmodells sind die **Perspektiven** oder **Viewpoints**. Sie untergliedern sich in

- **Enterprise** (Unternehmenssicht): Zweck, Aufgaben, Umfang, Kontext, Restriktionen, Richtlinien, Verantwortlichkeiten etc.
- **Information** (Informationssicht): Informationseinheiten, Informationsstrukturen, Informationsflüsse (Semantik und Verarbeitung)
- **Computation** (Modulsicht): Aufteilung in kooperierende Software-Grundeinheiten und deren Beziehung untereinander (Schnittstellen und Aufrufstrukturen)
- **Engineering** (Infrastruktursicht): logische Strukturen, Lösungsverfahren und Methoden
- **Technology** (Technologiesicht): reale Hard- und Software-Komponenten

Die einzelnen Viewpoints sind keine Schichten, sondern liefern ein **Viewpoint Model** (perspektivisches Modell) für ein System. So kann ein System z.B. aus Sicht der Organisation (Enterprise), der Modularisierung (Computation), der Datenbank-Orgranisation (Information), der Kommunikation/Dienste (Engineering) und der Vernetzung (Technology) betrachtet werden. Das ODP Referenzmodell basiert dabei auf einer **objektorientierten Modellierung**:

- ein verteiltes System ist eine Menge kooperierender Objekte
- Objekte interagieren über Nachrichten und Schnittstellen
- es werden verschiedene Interaktionsformen unterstützt.

Weiterhin werden die in Abschnitt 2.1 auf Seite 7 vorgestellten Arten der Transparenz unterstützt.

3 Architekturen

Ähnlich wie ein lokales Betriebssystem bietet auch eine Verteilungsplattform **Dienste** an. Dabei wird zwischen **Kern- und Erweiterungsdiensten** unterschieden (siehe Abbildung 1).

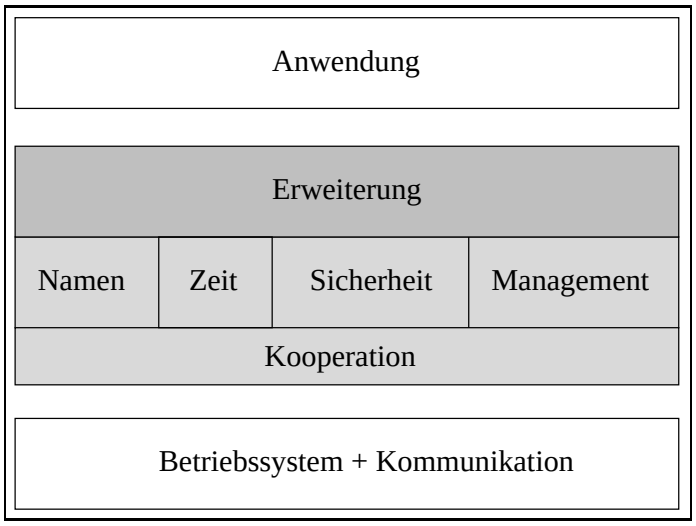


Abbildung 1: Verteilungsplattform

DCE

Eine wichtige Architektur stellt das **Distributed Computing Environment** (DCE) der **Open Software Foundation** (OSF) dar. Sie beschreibt ein verteiltes Dateisystem, Zeit, Namen, Sicherheit, Management, RPC und Datenrepräsentation und die Verwaltung von Threads und Datentransport auf Betriebssystemebene.

DACNOS

Ein anderes Framework stellt das **Distributed Academic Computing Network Operating System** (DACNOS) dar. Es definiert System Services, Remote Service Calls, Kernel Service Calls, ein Presentation System und den Global Transport.

OMA

Die dritte wichtige Architektur ist die **Object Management Architecture** (OMA) der **Object Management Group** (OMG). Sie besteht aus einem zentralen Object Request Broker, der Application Objects, Object Services und Common Facilities miteinander verbindet.

3.1 Kommunikation (OSI Referenzmodell)

Als Basis-Referenzmodell für die Verbindung offener Systeme wurde 1984 das **Open Systems Interconnections** (OSI) Referenzmodell definiert. Es unterteilt die Kommunikation in verschiedene **Schichten**, die über definierte Schnittstellen miteinander kommuniziern und so austauschbar sind. Eine solche Schicht

wird auch **Service Data Unit** (**SDU**) genannt und kommuniziert mit anderen SDUs über **Protocol Data Units** (PDU). Dabei findet beim Durchlaufen der Schichten eine Verschachtelung der PDUs statt, bei der noch **Protocol Control Information** (**PCI**) hinzugefügt wird.

In dem OSI-Modell werden verschieden **Dienstarten und Dienstprimitiven** unterschieden:

- Bei einem **bestätigendem Dienst** erfolgt die Kommunikation über die Kette `D.Request` (C), `D.Indication` (S), `D.Response` (C), `D.Confirm` (S).
- Bei einem **unbestätigendem Dienst** besteht der Austausch nur aus `D.Request` (C), `D.Indication` (S).
- **Vom Anbieter initiierte Dienste** werden durch `D.Indication` (S) angeboten.

Das 7-Schichten-Modell

Im OSI-Referenzmodell wurde eine Aufteilung in 7 Schichten beschlossen: **Anwendung**, **Darstellung**, **Sitzungssteuerung**, **Transport**, **Vermittlung**, **Sicherung** und **Bitübertragung**. Jede Schicht erhält Daten von der darüberliegenden und versieht sie mit einem Kopf und in der Sicherungsschicht auch mit einem Ende.

Schicht 1

Der **Physical Layer** oder die **Bitübertragungsschicht** stellt die unterste Schicht dar. Sie definiert die ungesicherte Übertragung des Bitstroms und signalisiert Störungen. Weiterhin ist in ihr das Setzen von Dienstgüte-Parametern und die Flußsteuerung möglich.

Standards dieser Schicht beschreiben Steckerformen, Belegungen von Leitungen (Daten- und Signalleitungen) und Modulationsverfahren.

Beispiele. V.24 (RS232 C) mit getrennten Daten- und Signalleitungen und der seriellen Übertragung der Bits. Es gibt vier Zustände: Bereit, Verbindungsaufbau, Datentransfer, Verbindungsabbau.
X.21 und IEEE 802.

Schicht 2

Der **Data Link Layer** oder die **Sicherungsschicht** verbessert ungesicherte Systemverbindungen zu gesicherten Systemverbindungen. Zu ihren Aufgaben gehört die **Rahmenbildung**, **Sicherung der Übertragung** und die **Flusssteuerung**.

Es werden 3 Dienstarten zur Verfügung gestellt: **unbestätigte verbindungslose**, **bestätigte verbindungslose** und **verbindungsorientierte**. Bei der ersten werden Rahmen ohne einen Verbindungsaufbau einfach übertragen, was nur bei

Systemen mit niedriger Fehlerquote und der Datenwiederherstellung in höheren Schichten möglich ist. Bei der zweiten schickt der Empfänger immer eine Bestätigung an den Sender, so dass dieser Nachrichten wiederholen kann, wenn innerhalb eines bestimmten Zeitraums keine Bestätigung angekommen ist. Allerdings wird auch hier die Reihenfolge nicht garantiert. Bei der letzten Dienstart wird vor jeder Übertragung eine Verbindung aufgebaut, wonach dieÜbertragung und danach einVerbindungsabbau stattfinden kann.

Die Rahmen werden erstellt, indem die Daten in einen Rahmen verpackt werden, mit einem Header aus Signal-Bits und Prüfsumme versehen werden und zuletzt nach dem Einfügen eines Anfangs- und Endflags mit Nullen gefüllt wird.

Im Falle eines bestätigten Dienstes laufen Timer beim Verschicken von Rahmen, so dass sie erneut geschickt werden, wenn keine Bestätigung empfangen wurde. Damit Rahmen nicht dupliziert werden, werden sie eindeutig nummeriert. Auf diese Art können Wiederholungen erkannt werden.

Die Flusssteuerung wurde dadurch realisiert, dass der Empfänger den Sender auf eventuelle Engpässe aufmerksam macht. Auch hier gibt es verschiedene Protokolle.

Beispiele. **High-Level Data Link Control** (HDLC) mit folgendem Rahmenaufbau: | `Flag` | `Address` | `Control` | `Daten` | `FCS` | `Flag` |. Bei der Angabe `Address` wird zwischen `Command` und `Response` unterschieden. Für `Control`s stehen drei Arten von Blöcken „`Information`", „`Supervisory`" und „`Unnumbered`" zur Verfügung. Die Frame Check Sequence (`FCS`) wird mit dem Polynom $x^{16} + x^{12} + x^5 + 1$ berechnet. Als Protokoll kommt das **Sliding Window** zum Einsatz, für das eine 3-Bit-Sequenznummer reserviert ist – es sind also 7 Fenster möglich. Acknowledgments werden huckepack in `I`-Frames verschickt, die Nachrichten `RR`/`RNR`/`REJ`/`SREJ` werden in `S`-Frames verschickt. Die Kommunikationssteuerung wird durch `U`-Frames gesteuert (Nachrichten `SABM`, `DISC`/`DM`, `UA`, `CMDR`/`FRMR`).
Jeder Rahmen beginnt mit einem speziellen Bitmuster (01111110). Sieht die Sitzungssteuerungsschicht eine Folge 5 aufeinander folgender Einsen, so stopft sie eine Null in den abzusendenden Bitstrom. Diese Null wird wiederum von der Sitzungsschicht des Empfängers entfernt. Dadurch können Datenrahmen eine beliebige Zahl an Bits enthalten und Zeichen können mit einer beliebigen Länge von Bits pro Zeichen kodiert werden.
Zur Flusssteuerung wird das **Sliding Window Verfahren** eingesetzt, bei der eine Folgenummer für die Auslieferung aller Pakete garantiert wird [Tanenbaum1995, S. 221-222]. Bestätigungen werden **Huckepack** verschickt, bis die Vermittlungsschicht das nächste Paket übermittelt. Dadurch wird die Bandbreite besser genutzt und es wird eine kleinere Anzahl an Rahmen benötigt. Probleme ergeben sich mit der Timer-Steuerung.
Ein weiteres Beispiel ist das **Ethernet**.

Schicht 3

Der **Network Layer** oder die **Vermittlungsschicht** verknüpft gesicherte Datenverbindungen zu einer Endsystemverbindung und steuert und überwacht die Verbindung. Während bisher lediglich die Kommunikation zweier Computer, die direkt über ein Medium miteinander verknüpft sind besprochen wurde, geht es nun um die Kommunikation zwischen Computern, die nicht direkt über ein Medium gekoppelt sind. Demnach gehören folgende Funktionen zur Ebene 3:

- **Routing** und **Betriebsmittelzuordnung**
- **Addressierung** der Netzknoten
- **Multiplexen** mehrerer Endsystemverbindungen auf eine Ebene-3-Verbindung
- **Fehlererkennung** und -behebung
- vorrangiger **Datentransfer**
- **Flußsteuerung** und **Überlastungsschutz**
- **Transparenz** der heterogenen Subnetze (**Fragementieren**)

Auf dieser Schicht unterscheidet man zwischen **verbindungsorientierten** (**virtual circuit)** und **verbindungslosen** (**Datagram**) **Diensten.**

Ein verbindungsorientierter Dienst stellt dabei folgende Anforderungen:

- Aufbau, Datentransfer und Abbau der Verbindung durch das Protokoll
- Netz stellt Reihenfolge der Pakete sicher
- Ende-zu-Ende-Flußsteuerung
- netzorientiert
- die Komplexität wird in das Netz verlagert (bekanntes Modell), was den Anforderungen vieler Anwendungen entspricht und für lange Verbindungen effizient ist.

Verbindungslose Dienste verlangen dagegen:

- unabhängige Pakete mit vollständigen Adressen
- Folge- und Duplikatkontrolle beim Empfänger
- keine Flußsteuerung durch das Netz
- endstellenorientiert

- bietet Anwendungen mehr Flexibilität je nach Anforderung und ist effizienter für kurze Verbindungen.

In Broadcast-Netzen ist diese Schicht nur sehr dünn gehalten bzw. gar nicht vorhanden.

Beispiele. Das Protokoll **X.25** ist das Schicht 3-Protokoll zu X.21 (Schicht 1) und HDLC/LAPB (Schicht 2). Es ermöglicht virtuelle Verbindungen (**virtual circuits**), einen geregelten Verbindungsauf- und abbau, Datentransfer, Flußsteuerung, Diagnose etc.
Weiter Beispiele sind IP und SPX.

Schicht 4

Der **Transport Layer** oder die **Transportschicht** erweitert Endsystemverbindungen zu Anwenderverbindungen. Sie stellt das Bindeglied zwischen Anwendungen und Netzen dar, indem Anwendungen auf einem Rechner eindeutig über die Adresse des Rechners und die TSAP-Nummer (**Service Access Point**, **SAP**) adressiert werden. Dabei wird auch **Quality of Service** bzgl.

- Verzögerung des Verbindungsaufbaus
- Durchsatz
- Transitverzögerung
- Fehlerrate
- Schutzvorkehrungen

möglich. Netzwerke werden demnach anhand ihrer Fehlerrate klassifiziert:

- **Typ A** sind qualitativ hochwertige Netzwerke mit geringer Fehlerrate und Fehlerkorrektur durch die Ebenen 1 bis 3.
- **Typ B**-Netzwerke mittlerer Qualität bieten eine akzeptable Fehlerrate mit wenig Fehlerkorrektur.
- **Typ C**-Netze bieten eine nicht mehr vertretbare Fehlerrate, keine Fehlerkorrektur lediglich in der Transportschicht.

Ebenso werden verschiedene **Transportklassen** unterschieden:

- **TP Class 0**, die Einfachklasse (**Simple Class**) dient der Segmentierung und Zusamensetzung der Daten (Typ A)

- **TP Class 1**, die Fehlerbehebungsklasse (**Basic Error Recovery Class**) behebt signalisierte Fehler, ermöglicht jedoch kein Multiplexen (Typ B)
- **TP Class 2**, die Multiplexklasse (**Multplexing Class**) entspricht Class 0 plus Multiplexing (Typ A)
- **TP Class 3**, die Fehlerbehebungs- und Multiplexklasse (**Error Recorvery and Multiplexing Class**) entspricht Class 1+2 (Typ A)
- **TP Class 4**, die Fehlererkennungs- und Behebungsklasse (**Error Detection and Recovery Class**) entspricht Class 3 + Fehlererkennung (Typ C).

TCP/IP

Das **Transmission Control Protocol / Internet Protocol** (**TCP/IP**) subsummiert eine Reihe verschiedener Protokolle und Dienste auf verschiedene Ebenen und heisst daher offiziell **Internet Protocol Suite**. Entstanden ist es aus dem ARPA-Netz Anfang der 70er Jahre, das seit 1984 in zwei Netze geteilt wurde, dem ARPANET und dem MILNET.

Das Protokoll stellt einen semi-offiziellen Internet-Standard dar, hinter dem eine komplette „Standardisierungsorganisation" steht (z.B. IETF). TCP/IP ist nicht herstellergebunden und wird sowohl für LANs als auch für WANs verwendet. Es ist eng verbunden mit UNIX, jedoch auf fast allen BS-Plattformen verfügbar.

Ein Internet besteht logisch gesehen aus einer Menge von Hosts, die erreichbar sind. Physikalisch existieren Rechner (Hosts), Subnetze und IP Router (Gateways), die die Subnetze miteinander verbinden und eine einheitliche logische Sicht ermöglichen.

Die Einordnung der Internet Protokollschichten in das OSI-Modell ist nicht einfach möglich, da beide eigenständige Protokoll-Modelle darstellen. Eine Übersicht ist in Abbildung 2 zu finden.

Vom Ethernet kommend legt ein Datenpaket folgenden Weg zurück:

Von		Nach	Abhängig von
Ethernet	→	IP | ARP	Type Field
IP	→	UDP | TCP	Protocol Field
UDP	→	Anwendung	
TCP	→	Anwendung	Port Field

Damit ein Datenpaket von einem Rechner in einem Netz einen Rechner in einem zweiten Netz erreicht, wird ein sog. **Router** benötigt. Darunter wir ein Rechner mit zwei Anschlüssen verstanden, der zwei Ethernet-Adressen und zwei IP-Adressen besitzt – in jedem Netz eine – und anhand der Ziel-Adresse entscheidet, in welches Netz ein ankommendes Paket weitergeleitet wird.

Diese Unterscheidung wird anhand von 5 **Klassen** getroffen, die von der NIC festgelegt wurden:

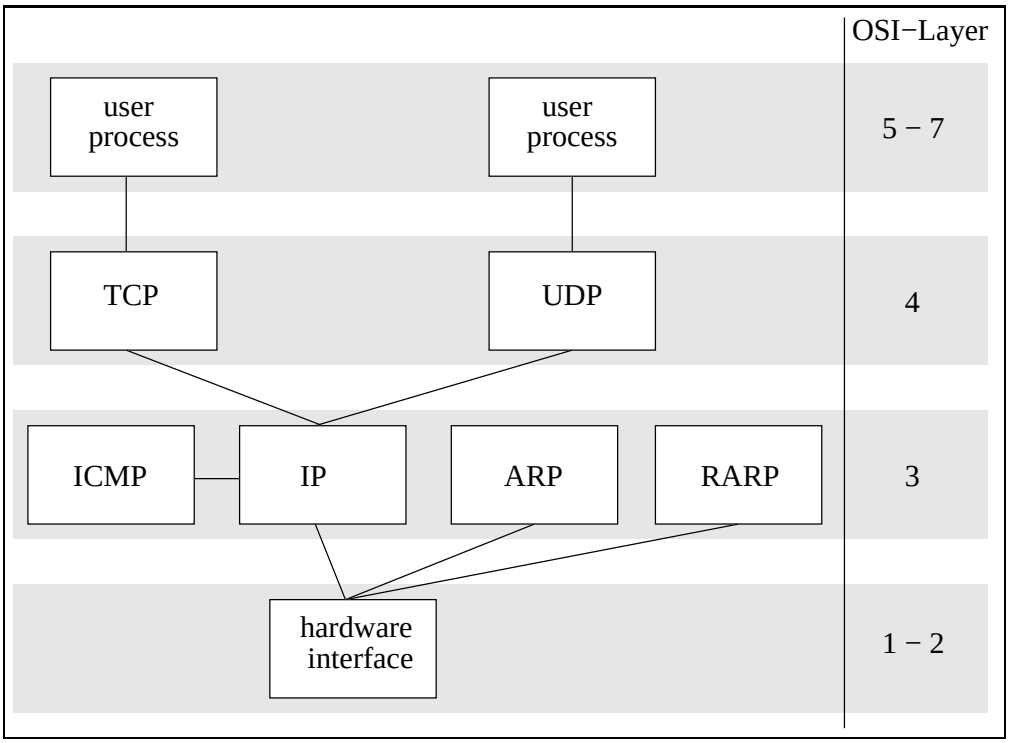


Abbildung 2: Einordnung von TCP/IP in das OSI-Referenzmodell

Klasse	Präfix	Größe der Netz-ID	Größe der Rechner ID
A	0	7 Bit	24 Bit
B	10	14 Bit	16 Bit
C	110	21 Bit	8 Bit
D	1110	28 Bit Multicast	
E	11110	27 Bit reserviert	

ARP

Die Zuordnung einer Ethernet-Adresse zu einer gegebenen IP-Adresse erledigt das **Address Resolution Protocol** (**ARP**). Dazu sucht der Host zunächst seinen ARP-Cache nach der gewünschten IP-Adresse ab. Ist sie dort nicht vorhanden, so sendet er ein Broadcast-ARP-Paket und wartet auf eine Antwort, die dann in den ARP-Cache eingetragen wird. Jeder Rechner im Netz überprüft, ob die Adresse, die in einem Broadcast-ARP-Paket nachgefragt wird ihm gehört und antwortet, wenn das der Fall ist. Gehört die Zieladresse zu einem anderen Subnetz, so antwortet entweder der Router (→Proxy-ARP) oder der Client erkennt das selbst und schickt das Paket sofort an den Router.

Routing

Beim **Routing** unterscheidet man zwischen dem **direkten Routing**, bei dem Netze über einen Router verbunden sind und dem **indirekten Routing**, bei dem ein Datenpaket u.U. mehrere Router passieren muss. Bei letzterem enthalten die

Routing-Tabellen nur Netzadressen und IP-Datagramme werden solange weitergereicht, bis sie nicht mehr zugestellt werden können oder die TTL abgelaufen ist. Das Routing wird **hop-by-hop** durchgeführt und wird durch Name-Server unterstützt. Updates von Routing-Tabelen können manuell oder durch Routingprotokolle (RIP, OSPF, BGP) erfolgen.

Ports

Die Adressierung einer bestimmten Anwendung auf einem Host erfolgt mit Hilfe der sog. **Portnummer**. Hier hat sich das Modell der **Socket-Abstraktion** (IP-Adresse, Portnummer) etabliert, deren API an das UNIX I/O-Modell, das dateiorientiert arbeitet, angelegt ist (entsprechend gibt es die Operationen `socket()`, `bind()`, `connect()`, `write()`, `read()`, `close()`, `listen()`, `accept()`,...).

Schicht 5

Der **Session Layer** oder die **Kommunikationsschicht** stellt Ausdrucksmittel zur Verfügung, die zur geordneten Durchführung einer Kommunikationsbeziehung nötig sind. Sie strukturiert den Dialog von Anwendungen, was durch Bestätigung von **Haupt-** und **Nebensynchronisationspunkten** erfolgt. Eine **Aktivität** stellt dabei eine übergreifende logische Einheit dar. Das Rücksetzen einer Kommunikation wird durch den **Resynchronize-Dienst** erledigt – die Semantik der **SyncPoints** wiederum ist Sache der Anwendungen.

Token

Die Steuerung der Dialogfunktionen erfolgt durch Benutzung von **Tokens**. Man unterscheidet **Activity**, **Data** und **Release Token**, wobei der Wechsel des Tokens durch Dienste wie **S-Token-Please** und **S-Token-Give** erfolgt. Der **Session Layer Standard** definiert Teilmengen mit unterschiedlichem Dienstumfang und Anwendungen handeln beim Aufbau der Session Connection den Umfang und die Dienstgüteparameter aus (z.B. Prioritäten, Fehlerrate, Transitverzug etc.).

In der Praxis ist der Session Layer sehr umstritten und hat kaum Bedeutung.

Schicht 6

Der **Presentation Layer** oder die **Darstellungsschicht** stellt Ausdrucksmittel zur Verfügung, um Daten beschreiben zu können. Dabei werden Regeln festgelegt, wie Daten zu übertragen sind. Achtung: Verbindung ist nicht gleich Kommunikation.

ASN.1

Der ISO-Standard **Abstract Syntax Notation.1** (**ASN.1**) soll eine einheitliche Datenrepräsentation ermöglichen, so dass Anwendungen Informationen austauschen können, ohne vorher dafür konfiguriert zu werden. Der Ablauf ist dabei folgender:

1. Die Anwendungen handeln eine abstrakte Syntax der Daten aus.
2. Es wird eine Mitteilung an die Presentation Entity gemacht.

3. Die Presentation Entities handeln eine Transfersyntax aus und
4. Konvertieren die Daten gemäß der Transfersyntax.
5. Die Daten werden übertragen und
6. nach lokaler Syntax konvertiert.

Die Definition einer Syntax in ASN.1 gliedert sich in **Module**, **Modulkopf** und den Import/Export. Es werden drei verschiedene Arten von Deklarationen unterschieden: **Typen**, **Werte** und **Makros**. Jede Art hat ihre eigene Namenskonvention (ein Typ beginnt mit einem Großbuchstaben, ein Wert mit einem Kleinbuchstaben und ein Makro besteht nur aus Großbuchstaben) und es gibt die Basisdatentypen **Boolean**, **Integer**, **Enumerated**, **Real**, **String**,... Einfache Typen werden durch Zuweisung definiert:

$$\texttt{Muliple-Contexts} ::= \texttt{Boolean}$$
$$\texttt{simple Multiple-Contexts} ::= \texttt{FALSE}$$

Als zentrales Element gilt der ASN.1 **Object Identifier** (oder einfach ASN.1-Typ). Er bezeichnet ein registriertes Objekt und ist eine Folge nicht-negativer Integer-Werte, die optional mit einem Namen versehen sind. Sie werden zentral vergeben und hierarchisch angeordnet.

Tags

ASN.1 **Tags** sollen das Problem der eindeutigen Kennzeichnung bei der Übertragung lösen – nicht der Name sondern der Tag wird geschickt (z.B. `optional`). Es werden vier Klassen von Tags unterschieden: `UNIVERSAL`, `APPLICATION`, `PRIVATE` und kontextspezifisch (wie `optional`). Die Tags werden mit übertragen, es sei denn ein Element ist als `IMPLICIT` markiert.

Der darstellungsspezifische Teil von ASN.1 wir als **Basic Encoding Rules** (BER) bezeichnet. ASN.1 unterstützt das C-Typsystem ohne Funktionszeiger, definiert eine kanonische Zwischenform und benutzt Typ-Tags. Die Stubs können wahlweise interpretiert oder kompiliert sein. Die Beliebtheit von ASN.1/BER liegt unter anderem daran, dass dass es von dem Internet-Standard SNMP benutzt wird. Es wird das Big-Endian-Format verwendet und Gleitkomma-Zahlen werden in der 2er-Komplement-Darstellung kodiert.

TLV

Datenelemente werden mit Tripeln der Form `<tag, length, value>` dargestellt, wobei `tag` normalerweise ein 8-Bit-Feld ist, auch wenn die ASN.1-Definition auch mehr zulässt, und als eindeutiger Typkennzeichner für den übertragenen Wert dient. Zusammengesetzte Datentypen können durch Verschachtelung gebildet werden. Ist die Länge von `value` ≤ 127 Byte, so wird die Länge in einem einzigen Byte spezifiziert. Die dedizierte Angabe einer Länge weist darauf hin, dass oft keine „natürlichen" Bytegrenzen bei der Länge verwendet werden, was das unmarshalling komplizierter als nötig macht.

Kritik

Die TLV-Kodierung ist sehr mächtig und die BER entsprechend vielfältig. Die Kodierung erweist sich dementsprechend als sehr aufwendig im Ressourcenverbrauch. Oft ist eine Umwandlung auch einfach unnötig, da viele Anwendungen die Daten kennen, so dass die Angabe von Tag und Länge überflüssig ist.

Ausserdem orientiert sich die Länge wie oben angesprochen nicht an der Rechnerhardware, sondern an der konkreten Anzahl der Bits, was ineffizient und aufwendig ist.

Die Aussage, dass Mächtigkeit vor Schnelligkeit geht, muss an diesem Punkt relaitivert werden, wurde doch die Konkurrenz von OSI und TCP/IP von letzterem gewonnen.

Schicht 7

Der **Application Layer** oder die **Anwendungsschicht** bietet anwendungsnahe Unterstützung für die Kommunikation. Kommunizierende Anwendungsinstanzen bilden dabei verteilte Anwendungen. Die Struktur aus Anwendungssicht besteht dabei aus **Application Processes**, die **Application Entities** beinhalten, die wiederum aus **Application Service Elements** (ASE) bestehen, die über ein **Application Protocol** miteinander kommunizieren.

Beispiele für **Association Control Service Elements** (ACSE) sind:

- Verbindung zweier Anwendungsinstanzen (**Application Entities**)
- fast jede OSI-Anwendung benutzt ACSE
- **Application Entity Title=Application Process Title + Application Entity Qualifier**
- die Adressierung erfolgt über die **Presentation Address** (PSAP Adresse), wobei ein Verzeichnis die AE Title auf eine PSAP-Adresse abbildet
- ACSE-Dienste sind:
  - A-Associate (Aufbau)
  - A-Release (Abbau – normal)
  - A-Abort (Abbau – abnormal durch User)
  - A-P-Abort (Abbau – abnormal durch Provider)

Beispiele für **Common Application Service Elements** (**CASE**) sind:

- **Remote Operation Service Element** (ROSE):
  - Aufruf von Operationen über das Netz
  - legt allgemeine Strukturen und Abläufe für Operationsaufrufe fest
  - ermöglicht auch RPC-artige Kommunikation
- **Reliable Transfer Service Element** (RTSE)

  - gesicherte Übertragung großer Datenmengen (store and forward)
  - im Zusammenhang mit **Message Handling Standards** entwickelt
  - Benutzt ACSE

- **Commitment, Concurrency and Recovery** (CCR)
  - Grunddienste zur Erhaltung der Datenkonsistenz
  - überwacht atomare Ausführung von Operationen
  - setzt two-phase-commit Protokoll ein

Application Context

Der **Application Context** wird von zwei **Application Entities** ausgehandelt und legt fest, welche ASEs zu einer Anwendung gehören. Sie werden durch **Object Identifier** gekennzeichnet – Application Service Elements wiederum handeln einen **Presentation Context** aus (der ebenfalls **Object Idenitifier** genannt wird).

Beispiele für Anwendungsspezifische ASEs (**Specific Application Service Element**, **SASE**) sind:

- Message Handling (MHS) (X.400)
- Directory Service (X.500)
- File Transfer, Access and Management (FTAM)
- Virtual Terminal (VT)
- Job Transfer and Manipulation (JTM)
- Remote Database Access (RDA)
- Manufacturing Messaging Specification (MMS)

3.2 Namen und Verzeichnisse

Im Allgemeinen hält ein Namensverzeichnis die Felder „Name", „Adresse", „Weg" und weitere Attribute vor. Namen identifizieren dabei Objekte im verteilten System (z.B. Benutzer, Gruppen, Rechner etc.). Dabei ergeben sich Probleme bzgl. der **Eindeutigkeit**, **Generierung**, **Struktur der Namen / Namensraum**, **Abbildung auf andere Namen / Adressen**, **Transparenz**.

Verzeichnisdienst

Beispiele für Namensdienste sind Variablennamen in höheren Programmiersprachen (Name→Adresse) und das Telefonbuch (Name→Telefonnummer).

Der **Verzeichnisdienst** (**Directory Service**) und der **Namensdienst** (**Name Service**) bildet Namen auf Adressen ab. Zu seinen Funktionen gehören:

- Lesen eines Eintrags bei vorgegebenem Namen (**White Pages**)
- Lesen von Einträgen auf Basis bestimmter Attributwerte (**Yellow Pages**)
- Verwaltung von Querverweisen (**Alias**)
- Verwaltung von Gruppen von Namen (**Verteilerliste**)

Fragen ergeben sich damit bezüglich

- Umfang des Namensraums
- Zuverlässigkeit und Verfügbarkeit des Verzeichnisdienstes (Replikation)
- Performance des Verzeichnisdienstes (Caching)
- heterogene lokale Namensräume
- Synonyme (Aliasse)
- Zugriffsschutz
- Flexibilität (Namen für Objekte unterschiedlicher Art)

Der Verzeichnisdienst gehört zu den Kerndiensten einer Verteilungsplattform mit ganz besonderen Anforderungen an Verfügbarkeit und Performance.

Das **Name Binding** bindet Namen an Objekte und interpretiert sie in einem Namenskontext. Dabei werden Namen auf andere Namen (meist mehrstufig) abgebildet.

Syntax

Die **Syntax** der Namen wird unterteilt in **unstrukturiert** (primitiv), **strukturiert** (DNS) und **attributiert** (X.500). Mit dem **Namensraum** bezeichnet man alle möglichen Namen bei gegebener Namenssyntax – die **Struktur** kann dabei **flach**, **hierarchisch** oder **wegorientiert** sein mit folgenden Vor- und Nachteilen:

flach + einfache Namen; leichte, einstufige Interpretation.
– nicht beliebig skalierbar; Eindeutigkeit der Namen durch zentrale Vergabe-Instanz.

hierarchisch + Delegation der Vergabe an dezentrale Instanzen; Effizienz durch relative Namen.
– komplexe, lange Namen möglich; mehrstufige Interpretation.

wegorientert + vereinfacht die Wegwahl (source routing).
– keine Transparenz; schwierige Rekonfiguration; viel Wissen über Endkoten.

3.2.1 DCE Verzeichnis

Bei der Entwicklung des DCE Verzeichnis standen folgende Anforderungen im Vordergrund:

- **Verfügbarkeit** (single point of failure? Replikation? Konsistenz? Ausfall bei Wartung?)
- **Sicherheit** (Zugriffsschutz: Authorisierung, Authentisierung?)
- **Performance** (Antwortzeit?)
- **Erweiterbarkeit** (Skalierbarkeit?)
- **Managebarkeit** (interne Struktur? Aufteilung in kooperierende Directory Server?)
- **Integrierbarkeit** (Zugriff auf Einträge in anderen Verzeichnissen?)
- **Transport-Unabhängigkeit** (Zugriff auf unterschiedliche Netze? Transportprotokolle?)

Organisation

Man entschied sich für eine Organisation in sog. **Namenszellen**, die jeweils über einen eigenen **Cell Directory Service** (CDS) verfügen. Ein zell-lokaler Name beginnt dann bei der **cell-root** „/. :". Als **Global Direcory Service** (GDS) wird X.500 verwendet. Dabei kommuniziert der GDS mit sog. **Global Directory Agents** (GDA), die sich wieder mit dem CDS austauschen. An Stelle von X.500 ist auch DNS als Namesverzeichnis vorgesehen.

Ein globaler Name im GDS auf Basis von X.500 besteht aus zwei Teilen: dem Namen der Zelle und dem internen Namen, z.B.

```
/.../c=us/o=uno/ou=abc/principals/hugo
```

Die Zellen selbst bilden organisatorische Einheiten, wie z.B. Gruppen, Stockwerke oder Abteilungen.

3.2.2 X.500

Ein allgemeiner Verzeichnisdienst, der von der ISO standardisiert wurde ist **X.500**. Er stellt seine Dienste OSI Anwendungen zu OSI Management-Aufgaben zur Verfügung und speichert Informationen zu beliebigen Objekten (z.B. Adresse von Benutzern und PSAP-Adressen von Anwendungen). Beim Entwurf wurden folgende Annahmen gemacht:

- Abfragen finden viel häufiger statt als Änderungen
- eine kurzzeitige Inkonsistenz der Informationsbasen ist akzeptabel

- der Namensraum soll hierarchisch gegliedert sein.

pieller Aufbau

Alle Informationen werden in der **Directory Information Base** (**DIB**) abgelegt. Möchte ein Benutzer eine Information aus der DIB, so befragt er über einen **Directory User Agent** (**DUA**) das Verzeichnis, das er in der DIB sucht.

Organisation

Im organisatorischen Modell befinden sich DUA und **Directory Service Agent** (**DSA**) in einer **Directory Management Domain** (**DMD**), wobei es durchaus mehrere DSAs in einer DMD geben kann.

funktionales Modell

Im funktionalen Modell greift der DUA über das **Directory Access Protocol** (**DAP**) auf das Verzeichnis zu, das aus DSAs besteht, die miteinander über das DAP kommunizieren.

Informationsmodell

Die Informationen, die in einem X.500-Verzeichnis (DIB) abgelegt sind, werden folgendermaßen modelliert:

- sie speichern Informationen über Objekte
- Objekte sind Instanzen von Objektklassen
- jedes Objekt enthält eine Angabe zu seiner Klasse und Oberklasse
- Einträge bestehen aus einer Menge von Attributen
- jedes Attribut ist ein Tupel `(Attributtyp,Attributwert(e))`
- Objektklassen legen den Attributtyp und die Syntax fest (in ASN.1 definiert)
- Attribut-Wert-Zuweisungen (**Attribute Value Assertion**, AVA) beschreibt Annahmen über den Wert eines Attributs – kann richtig, falsch oder undefiniert sein.
- Attributwerte können als „herausgehobener Wert" (**distinguished value**) markiert werden (z.B. zur Namensbildung)

DIT

Der **Directoy Information Tree** (DIT) setzt sich aus Verzeichnis-Einträgen zusammen, die Informationen zu einem Objekt enthalten. Die Einträge sind dabei in einem Baum angeordnet (DIT), wobei die Knoten die Einträge sind und Objekte Zwischen- und Endknoten sein können. So kann eine Organisation z.B. nach Land, Organisation, Organisationseinheit, Person modelliert werden. Die Wurzel des Baums ist immer ein leerer Eintrag.

Bei den Namen der Einträge unterscheidet man zwischen dem **Distinguished Name** (**DN**), der eine Folge von richtigen AVAs für die herausgehobenen Werte eines Eintrags beginnend bei der Wurzel des DIT darstellt, und dem **Relative Distinguished Name** (**RDN**), der als Teilfolge des DNs also als relativer Name angegeben wird.

Dienste

In X.500 stehen drei verschieden Dienste zur Verfügung:

- **Abfragedienst**: dient dem Abfragen des Verzeichnisses
  lesen: von Attributwerten eines Eintrags
  vergleichen: eines gegebenen Wertes mit Attributwerten
  auflisten: des Unterbaums eines geg. Eintrags
  suchen: mit geg. Suchkriterien
  abbrechen: einer gestellten Anfrage.
- **Änderungsdienst**: dient dem Ändern des Verzeichnisses
  hinzufügen: eines Eintrags zum DIT
  entfernen: eines Eintrags
  ändern: der Attribute und Werte eines Eintrags
- **Dienst-Steuerung**:
  Dienstparameter setzen: max. Dauer, Ausdehnung der Suche, Filter, etc.
  Sicherheitsparameter setzen: für die Zugriffskontrolle

Protokolle

Zur Kommunikation in X.500 stehen folgende Protokolle zur Verfügung:

- **Directory Access Protocol** (DAP): ACSE+ROSE+drei spezifische ASEs (readASE, searchASE, modifyASE)
- **Directory System Protocol** (DSP): ACSE+ROSE+drei spezifische ASEs (s.o.)
- Abbildung der Dienstprimitiven auf ACSE und ROSE

3.2.3 DNS

Das **Domain Name System** wurde als Namensdienst für das Internet eingeführt. Als Anforderungen wurden gestellt:

- sprechende, benutzerfreundliche Namen statt IP-Adressen
- Benennung beliebiger Objekttypen
- dezentrale Namensauflösung mit guter Skalierbarkeit
- hohe Performance und Verfügbarkeit
- dezentrale Instanzen zur Namensvergabe mit hierarchischem Namensraum

Der DNS definiert dabei die Syntax der Namen, ein verteiltes System zur Abbildung der Namen auf IP-Adressen und ein hierarchische Domänen-Konzept. Die oberste Knoten-Ebene (die **Top-Level Domains**, TLD) sind global vom InterNIC festgelegt und in organisatorische und geographische TLDs eingeteilt.

Eine Domäne kann mehrere Nameserver besitzen, und verschieden Name Server können miteinander kooperieren. Die Replikation von Name Servern führt

zu Steigerung der Zuverlässigkeit, jeder Nameserver muss für den Bootstrap allerdings mindestens einen Vorgängerserver kennen. Anfragen von Clients gehen immer an eine Well-Known-Port-Adresse mit `(Name, Objekttyp,..., partiell/komplett)`. Das Caching von Namensinformationen im NS wird durch eine TTL für Einträge organisiert. Trotzdem können veraltete Informationen mitgeteilt werden. Die Einträge im Cache eines Name Server bestehen aus Name, IP-Adresse und dem Namen des Servers, von dem die Information stammt.

3.3 Dienstvermittlung

Wie bereits angesprochen, ist die Vermittlung von Diensten ein ganz zentrales Element bei verteilten Systemen. Das Verzeichnis liefert nur eine elementare Abbildung von Dienstwunsch auf Dienstangebot, so dass für eine komfortablere Dienstauswahl ein Vermittler (**Trader**, **Broker**) benötigt wird. Ein von der ISO standardisierter Trader-Standard existiert im Rahmen von ODP, das als Referenzmodell entsprechende Grundbegriffe wie z.B. Dienst, Diensterbringer, Dienstnutzer, Schnittstelle, Import, Export etc. definiert.

Export

Unter dem **Export** verstehen wir die Veröffentlichung von Diensttyp und Dienstattribut (dem Dienstangebot). Der Trader speichert das Dienstangebot in seiner Datenbasis und kann es nach eigenen Strategien Nachfragen zuordnen.

Import

Der **Import** bezeichnet die Anfrage nach einem gewünschten Diensttyp und Dienstattributen. Sie kann z.B. durch logische Ausdrücke über Attribute formuliert sein und, falls sie erfolgreich ist, eine Schnittstellen-Referenz zurückliefern. Als nächster Schritt findet dann das Binden des Klienten an den Diensterbringer statt.

Trading-Kontext

Alle exportierten Dienste werden einem **Trading-Kontext** zugeordnet, der das Dienstangebot strukturiert und die Verwaltung (z.B. der Zugriffsrechte und der Abrechnungsrichtlinien) übernimmt. Die Kontexte können dabei hierarchisch angeordnet sein.

Trader-Föderation

Arbeiten unabhängige Trader zusammen, so spricht man von einer **Trader-Föderation**, bei der ein sog. Förderationsvertrag festlegt, welche Trading-Kontexte für andere Trader zugreifbar sind.

Verwaltung von Diensttypen

Damit ein Dienstangebot strukturiert werden kann, wird es durch eine Angabe des **Diensttyps** (plus Attribute) gekennzeichnet. Die Verwaltung dieser Diensttypen wird als **Type Management** bezeichnet, das konzeptionell unabhängig vom Trader zu sehen ist. Meist wird der Typraum hierarchisch über Subtyp-Relationen strukturiert. Ein Diensttyp kann sowohl **syntaktisch** als auch **deklarativ** mit Methoden der KI beschrieben werden.

3.3.1 Object Request Broker

Die **Object Management Architecture** (**OMA**) ist ein Standard der **Object Management Group** (**OMG**). Der **Object Request Broker** (**ORB**) stellt ein zentrales Element in dieser Architektur dar, da er das Bindeglied für die Interaktion von Objekten in heterogenen verteilen Umgebungen darstellt.

Für den ORB hat ein Objekt einen **Zustand** (Attribute), **Verhalten** (Methoden) und **Identität** (Name), wobei sich der Zustand durch Aufrufe von Methoden ändern kann. Zu seinen Aufgaben gehört:

- Finden einer geeigneten Objekt-Implementation für den vom Client getätigten Aufruf
- Vorbereitung der Objekt-Implementation auf die Ausführung des Auftrags
- Übertragung des Aufrufs mit Parametern

Dabei sieht der Client nur die Schnittstelle des Server-Objekts, nicht jedoch seine Lokation, die Programmiersprache, den Hardware-Typ etc. Er benötigt lediglich eine **Object Reference** auf ein Server-Objekt, bevor er den Aufruf absetzen kann.

CORBA

In der der **Common Object Request Broker Architecture** (CORBA) greift der Client über dynamische Schnittstelle, statische IDL Stubs oder die ORB Schnittstelle auf den ORB Kern zu. Dabei findet die Kommunikation des Clients mit der Objekt Implementation über die ORB Schnittstelle statt. Die Objekt Implementation selbst kommuniziert mit dem ORB Kern über den Object Adapter bzw. das IDL Skeleton. Der ORB Kern selbst baut auf dem Schnittstellen-Verzeichnis und dem Implementierungs-Verzeichnis auf (siehe auch Abbildung 3).

Schnittstellenbeschreibung

Damit für Clients ein Zugriff auf heterogene Objekte möglich sein kann, müssen diese ihre Schnittstelle in der sog. **Interface Definition Language** (IDL) beschreiben. Aus dieser können dann Stubs und Skeletons erzeugt werden, über die der Client auf die Objektimplementation zugreifen bzw. der ORB einen Aufruf an die Objektimplementation weiterleiten kann. Die in der IDL definierte Schnittstellenbeschreibung wird in dem Schnittstellen-Verzeichnis abgelegt und so recherchierbar gemacht.

Komponenten

Die wichtigsten Komponenten von CORBA sind:

- **Interface Definition Language** (IDL), in der Operationen und ihre Parameter festgelegt werden. IDL-Konstrukte werden dann auf Elemente der Programmiersprache mittels **Language Binding** abgebildet. Es wird sogar die Vererbung zwischen Schnittstellendefinitionen unterstützt.

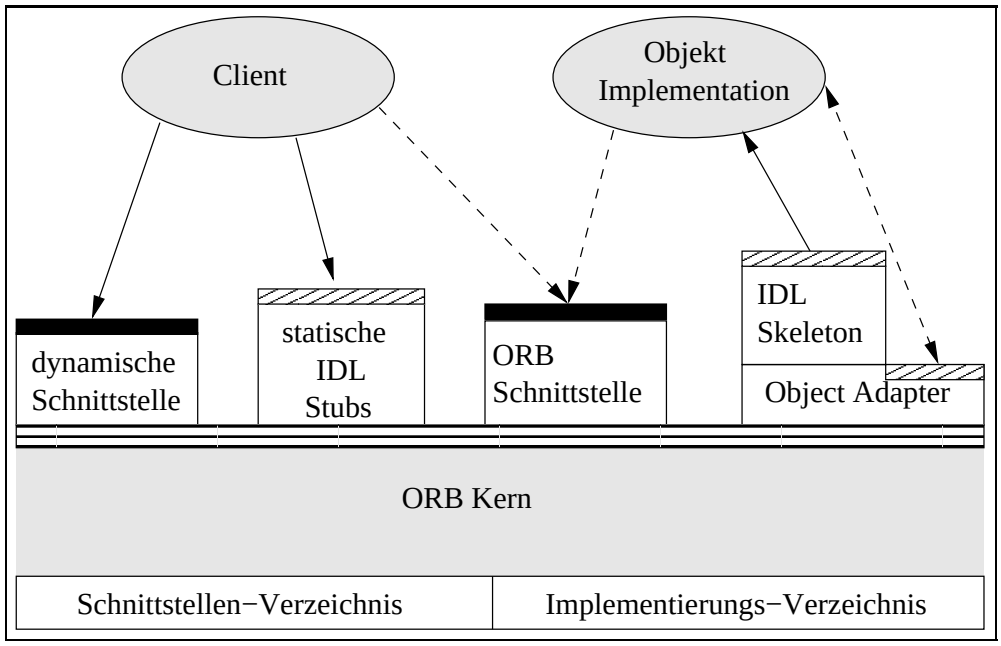


Abbildung 3: CORBA-Architektur

- **Objektreferenz** sind „opake" Zeiger auf Objekte, für die mehrere Darstellungsformen möglich sind. Ihre Gültigkeit umfasst die Lebenszeit des Clients.
- **ORB Schnittstelle** ist eine standardisierte Schnittstelle für zusätzliche Dienste des ORBs wie z.B. die Behandlung von Objektreferenzen.
- **Dynamische Schnittstelle** erlaubt das Zusammenstellen von Aufrufen zur Laufzeit. Anstatt eine Stub-Routine aufzurufen, werden Objekt, Operation und Parameter explizit benannt und der Aufruf abgeschickt. Im Client-Programm sind dann die Typen der Parameter mit Typ-Codes gekennzeichnet.
- **Objekt-Adapter** betten Server-Objekte in den ORB ein. Standard-Dienste sind Objekt-Aktivierung, Passivierung, Sicherheit, Generierung von Objektreferenzen, Registrierung von Implementationen etc. Die Anforderungen können je nach Anwendungsumfeld variieren – CORBA schlägt dazu den **Basic Object Adapter** (BOA) vor.

Sprachunabhängigkeit

Die **Sprachunabhängigkeit** in CORBA wird durch ein Language-Binding mittels der IDL erreicht. Damit wird die Definition einer Schnittstelle in den Quellcode der jeweiligen Programmiersprache übersetzt, was auch für Attribute gilt. Diese können über **Getter-** und **Setter-Methoden** manipuliert werden. Basisdatentypen werden ebenfalls über das Language Binding abgebildet.

DII

Dynamische Aufrufe in CORBA werden durch das **Dynamic Invocation Interface** (DII) ermöglicht. In diesem werden die Parameter der Schnittstelle beschrieben und der Aufruf angelegt.

Basic Object Adapter

Zu den **Aufgaben des BOA** gehören Aktivierung und Registrierung der Object Implementation, Aktivierung einzelner Objekte, Aufrufe von Methoden über das Skeleton und die Bereitstellung von **BOA Diensten**. Diese sorgen z.B. für die Generierung einer Objektreferenz und die Authentisierung.

Object Services

Die **Object-Services** stellen einen integralen Bestandteil der CORBA dar, da sie die Kooperation von Objekten unterstützen und für die Infrastruktur erforderlich sind. Durch die OMG ist definiert, welche Dienste gebraucht werden und wie die Architektur des Diensterbringers sowie die erforderlichen Schnittstellen dieser Dienste aussehen. Es gibt folgende Dienstarten:

- **Lebensphasen** (Kreieren, Löschen, Migration etc.)
- **Persistenz** (dauerhafte Abspeicherung des Zustands)
- **Ereignisse** (Spezifikation und Übermittlung von Ereignissen)
- **Beziehungen** (Definition und Verwaltung von Objektbeziehungen)
- **Namen** (Bennenung und Auffinden in heterogenen Umgebungen)
- **Sicherheit** (Zugriffsschutz)
- **Transaktionen** (Gruppierung von Aufrufen zu Transaktionen)

Lebensphasen-Dienst

Die **Lifecycle Services** stellen folgende Operationen zur Verfügung:

- **Create Objekt** (kann Client Ort und Implementation bestimmen? Wer führt create-Auftrag aus? Wie wird das Objekt initialisiert?)
- **Copy Object, Move Object** (Was geschieht mit dem Code des Objektes? Wie wirkt sich die Operation auf andere mit dem Objekt verbundene Objekte aus? s.o.?)
- **Delete Object**

Die **Object Factory** ist primär für die Generierung von Objekten zuständig, ist aber ein ganz gewöhnliches CORBA-Objekt. Für die Generierung eines neuen Objektes kann dabei eine Initialisierung in Form von Name-Wert-Paaren gegeben werden. Filter können dabei zur Steuerung übergeben werden (z.B. OperatingSystem!=WindowsNT) und Preferences geben gewünschte Eigenschaften als String an (z.B. ComputerType=RS6000Mod590").

Persistenz-Dienst

Der **Persistent Data Service** ermöglicht die persistente Speicherung des dynamischen Zustands eines Objekts, das dann **Persistent Object** genannt wird.

Dabei erhält es einen **Persistent Identifier**, der den Ort der Speicherung im **Data Store** beschreibt. Es kommuniziert mit dem **Persistent Object Manager**, der dem Objekt Persistenz-Operationen anbietet – dafür existiert genau ein POM pro PO. Der **Persistent Data Service** bietet den Zugang zum Datenspeicher, dem **Data Store**. Die Kommunikation des PO mit dem PDS erfolgt über ein festgelegtes **protocol**.

Das Interface eines PO umfasst das Attribut `pid` und die Operationen `connect()`, `disconnect()`, `store()`, `restore()` und `delete()`.

Ereignis-Dienst

Der **Event Service** entkoppelt Client und Server. Normalerweise sind CORBA-vermittelte Aufrufe nämlich synchron, d.h. es besteht eine direkte zeitliche Kopplung zwischen Client und Server. Das ist in verteilten Systemen nicht immer sinnvoll, weshalb der CORBA-Dienst eine entkoppelte Interaktion zwischen Ereignis-Erzeuger und Ereignis-Verbraucher ermöglicht. Als Modell kommt dabei das **Push- und Pull-Modell** in Frage.

Die Schnittstelle des Ereignis-Dienstes umfasst dabei die Operationen `push()`, `disconnect_push_consumer()`, `disconnect_push_supplier()` und das gleiche mit `pull`.

Für die Entkopplung der Interaktion stellt der Event Service den **Event Channel** (Ereignis-Kanal) zur Verfügung. Dieser lässt sowohl das Push-, als auch das Pull-Verarbeitungsmodell zu und ermöglicht auch gemischte Modelle.

Common Facilities

Allgemeine Dienste, sogenannte **Common Facilities**, sind

- **Presentation Management** (Präsentation von Objekten, z.B. zum Drucken)
- **Help** (Mechanismen und Richtlinien, um ein Hilfe-System zu gestalten)
- **Information Storage and Retrieval** (persistente Speicherung und Wiederfinden von Informationen)
- **Collection Facility** (Basisklassen zur Manipulation von Gruppierungen von Objekten)
- **Management Tools Facility** (Interoperabilität von Management-Werkzeugen)
- **Rule Management Facility** (Wissenserwerb, Pflege und Ausführung in regelbasierten Objektsystemen)
- **Message Facility** (E-Mail für Objekte)
- **Time Facility** (Verschiedene Darstellung der Uhrzeit und Übersetzungen)
- **Imagery Facility** (Interoperabilität zwischen „Bild-Objekten" und bildverarbeitenden Anwendungen)

3.4 Sicherheit

Ressourcen in einem System müssen gegen mutwillige und unbeabsichtigte Manipulation von Subjekten geschützt werden. Das umfasst

- unberechtigten Zugriff auf Informationen
- unberechtigte Veränderung von Informationen
- unberechtigter Gebrauch von „Geräten"
- Belästigung, Störung
- Vandalismus

Als **Angriffsformen** unterscheiden wir **Mithören**, **Maskerade**, **Nachrichtenverfälschung**, **Nachrichtenwiederholung**. Als **Mechanismen** der Sicherheit stehen **Kryptographie**, **Authentisierungsprotokolle** und **Zugriffskontrollverfahren** zu Verfügung. Aufgabenbereiche der Sicherheit umfassen **Authorisierung**, **Authentisierung**, **Vertraulichkeit**, **Integrität** und **Nachweisbarkeit**.

In verteilten Systemen steigt die Komplexität der Sicherheit durch das Vorhandensein **mehrer Angriffspunkte** (z.B. Kommunikationskanäle, Arbeitsplatzrechner). Ausserdem ist es in heterogenen Systemen nur schwer möglich, einen gemeinsamen Nenner für heterogene Sicherheitsvorkehrungen bei einzelnen Systemen zu finden, um so eine zentrale Steuerung zu ermöglichen. Das alles bedeutet einen erhöhten Aufwand für kryptographische Protokolle.

Kryptographie

Im Allgemeinen unterscheidet man zwei grundsätzliche Verfahren in der Kryptographie: Die **symmetrischen Systeme**, die mit **Secret Keys** arbeiten, und die **asymmetrischen Systeme**, die mit **Private und Public Keys** arbeiten.

(a)symmetrische Systeme

Symmetrische Systeme ermöglichen eine effiziente Implementierung in Hardware, habe jedoch große Probleme, was die Schlüsselverwaltung und -verteilung angeht. Dieser Nachteil ist bei asymmetrischen Verfahren nicht mehr so schwerwiegend, jedoch leidet hier die Performance und die Sicherheit ist nicht „bewiesen".

DES

Der **Data Encryption Standard** (DES) wurde in der 70ern von IBM erfunden und später von dem ANSI standardisiert. Das Prinzip basiert auf blockweisem Chiffrieren von 64-Bit Datenblöcken mit einem 56-Bit Schlüssel in insgesamt 19 Schritten. Die Dechriffierung erfolgt in umgekehrter Reihenfolge mit demselben Schlüssel. Heute gilt das DES-Verfahren als nicht mehr sicher und wird höchstens noch im Triple-DES verwendet.

Die Funktionsweise beruht auf sog. „S-Boxen" (die Funktion $f()$), die aus XOR, Duplikation von Bits, Permutation und Reduktion besteht. Dabei wird bei jedem der Schritte 2-17 ein anderer Schlüssel verwendet.

RSA

Der bekannteste asymmetrische Algorithmus ist **RSA**. Seine Sicherheit basiert auf der Verwendung einer **One-Way-Function**, die auf der Schwierigkeit beruht, Zahlen in ihre Primfaktoren zu zerlegen. Der Empfänger generiert in dem Verfahren einen **Public Key** und einen **Private Key**. Ersteren macht er öffentlich bekannt, denn Nachrichten, die mit diesem verschlüsselt werden können nur mit Letzterem wieder entschlüsselt werden. Der Ablauf ist wie folgt:

1. Wählen zweier großer Primzahlen p und q.
2. Berechnung von $r = p \cdot q$ und $n = (p-1)(q-1)$.
3. Wahl des öffentlichen Schlüssels e mit $\mathrm{ggT}(e,n) = 1$ und des geheimen Schlüssels d mit $e \cdot d \equiv 1 \mod n$.
4. $E(t) \equiv t^e \mod n$, $D(t') \equiv t'^d \equiv (t^e)^d \equiv t^{ed} \equiv t^{k \cdot n+1} \equiv (t^n)^k t \equiv t \mod n$.

3.4.1 Authorisierung

Die **Zugriffskontrolle** (Autorisierung) kann durch Abspeicherung von Zugriffsrechten in einer Matrix (**Zugangsmatrix**) erfolgen, in der in den Spalten die Objekte und in den Zeilen die Subjekte stehen. Der Inhalt eines Matrix-Elements ist a_{ij}, das Zugriffsrecht von Subjekt s_i auf das Objekt o_j.

Dünn besetzte Matrizen können entstehen, wenn es z.B. viele Objekte und nur wenig Subjekte gibt. In diesem Fall muss eine efiiziente Speicherung der Matrix gefunden werden. In verteilten Systemen stellt sich überdies die Frage nach der Verwaltung der Matrix. Wird sie zentral gehalten oder repliziert?

ACL

Eine andere Möglichkeit ist die Speicherung der Zugriffsrechte in einer **Access Control List** (ACL) zur objektbezogenen Zugriffskontrolle. Für jedes Objekt wird eine Matrix vorgehalten, in deren Zeilen die Subjekte mit ihren Rechten vorgehalten werden.

Capabilities

Eine andere Implementationsform sind **Capabilities** (Befugnisscheine). Diesmal wird die Zugriffskontrolle subjektbezogen verwaltet. Für jedes Subjekt gibt es also eine Matrix, in der für jedes Objekt (Spalte) die Rechte dieses Subjekts verwaltet werden.

Vergleich

Ein Vergleich von Zugriffslisten ist in Tabelle 1 angeführt.

Authorisierung in DCE

In DCE werden ACLs eingesetzt, in denen Namen von **Principals** zusammen mit Operationen, die von ihnen ausgeführt werden gespeichert sind. Ein Principal ist dabei ein einzelner Benutzer oder eine Gruppe. Die Überprüfung des Zugriffs erfolgt durch den **ACL Manager**, der die Zugriffsliste für eine Ressource verwaltet. Ein Client ruft also einen Service zusammen mit seiner Authentisierungsinformation auf, so dass der Dienst die Berechtigung für diesen Aufruf durch den ACL Manager prüfen lassen kann.

Tabelle 1: Vergleich ACL-Capabilities

Zugriffslisten	Capabilities
+ Kontrolle beim Objekt	+ Recht direkt zu erkennen
+ leicht zu entziehen	+ einfache Delegation
+ genauer Überblick, wer Zugriff hat	+ guter Überbilck, was der Benutzer alles darf
− Delegation der Zugriffsrechte	− Kein Überblick, wer Zugriff hat
− Benutzer muss authentisiert werden	− Entzug der Zugriffsrechte problematisch
− kein Überblick, was der Benutzer alles darf	− Schutz gegen unberechtigtes Kopieren erforderlich

Ein Dienstprogrammierer generiert einen spezifischen ACL Manager für diesen Dienst (die DCE Software enthält zu diesem Zweck vorgefertigte Schablonen). Für Standard Dienste wie den **Cell Directory Service** und den **Distributed File Service** ist bereits eingebaut.

Die Authorisierung eines Clients arbeitet eng mit der Authentisierung des Client zusammen. Dieser erwirbt sich ein **Privilege Attribute Certificate** (PAC) und präsentiert dieses beim Server. Das PAC enthält u.A. den Namen des Clients und seinen Gruppennamen, so dass der ACL Manager auf Basis des PACs über den Zugriff entscheiden kann.

3.4.2 Authentisierung

Die Authentisierung in DCE basiert auf dem 1978 von Needham-Schroeder erfundenen Authentisierungsdienst: Zwei Kooperationspartner werden mit einem geheimen Konversationsschlüssel versorgt. Dabei unterscheidet man zwischen der Version mit Public Key- und der mit Secret Key-Verschlüsselung.

Secret Key Verschlüsselung

Damit A mit B geheim kommunizieren kann, benötigen beide einen gemeinsamen geheimen Schlüssel. Ausserdem besitzt jeder Teilnehmer einen eigenen **geheimen Schlüssel**, über den eine geheime Kommunikation ermöglicht wird.

1. $A \rightarrow S : A, B, N_A$
   Dazu sendet A eine Nachricht an B, in der A einen solchen Schlüssel anfordert.
2. $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$
   Daraufhin antwortet S mit einem nur für A lesbaren Schlüssel für AB und einem Ticket für B zusammen mit einem Prüfwert.
3. $A \rightarrow B : \{K_{AB}, A\}_{K_B}$
   Jetzt schickt A eine Nachricht an B, die das Ticket für die Kommunikation enthält.

4. $B \rightarrow A : \{N_B\}_{K_{AB}}$
   Nach dem Entschlüsseln des Tickets schickt B den Prüfwert verschlüsselt mit dem gemeinsamen Schlüssel an A zurück und bestätigt so die Korrektheit des Tickets.

5. $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$
   Zu guter letzt beweist A B den Erhalt der Nachricht, indem es den um eins erniedrigten Prüfwert, verschlüsselt mit dem gemeinsamen Schlüssel an B zurückschickt.

Diese Prozedur setzt natürlich voraus, dass alle Clients ihre geheimen Schlüssel irgendwann einmal dem Server übergeben haben – wird der Server Korrumpiert, ist die Sicherheit des gesamten Systems verloren.

Public Key Verschlüsselung

Bei der **Public Key Version** werden die unkritischen öffentlichen Schlüssel auf dem zentralen Server abgelegt – eine korrumpierung des Servers fällt also nicht derart ins Gewicht wie bei der der ersten Variante:

1. $A \rightarrow B : A, B$
   A schickt an S eine eine Anforderung nach dem öffentlichen Schlüssel von B.

2. $S \rightarrow A : \{PK_B, B\}_{SK_S}$
   Daraufhin antwortet S mit dem gewünschten Schlüssel, signiert von S.

3. $A \rightarrow B : \{N_A, A\}_{PK_B}$
   Jetzt schickt A eine Anfrage an B, die einen Prüfwert enthält und nur von B lesbar ist.

4. $B \rightarrow S : B, A$:
   Daraufhin erfragt B den öffentlichen Schlüssel von A durch eine Anfrage an S.

5. $S \rightarrow B : \{PK_A, A\}_{SK_S}$
   Der Server antwortet mit dem öffentlichen Schlüssel von A, die Nachricht ist wieder signiert.

6. $B \rightarrow A : \{N_A, N_B\}_{PK_A}$
   Nun schickt B an A eine nur für A lesbare Nachricht, die zwei Prüfwerte enthält, wobei einer neu ist und der andere dem entspricht, der im dritten Schritt von A übertragen wurde.

7. $A \rightarrow B : \{N_B\}_{PK_B}$
   Im letzten Schritt sendet A den neuen Prüfwert an B verschlüsselt zurück und beweist damit, dass er die Nachricht aus dem vorherigen Schritt entschlüsseln konnte.

Zur genauen Arbeitsweise von Kerberos siehe [JaegerMiddle]

3.4.3 Digitale Signatur

Die **Anforderungen** an eine digitale Unterschrift umfassen:

- Ein Dokument ist vom Unterzeichner zu verantworten und
- Ein Dokument ist nicht geändert worden.

Mit der **digitalen Unterschrift** soll der Empfänger also ein Dokument auf **Authentizität** und **Integrität** überprüfen können. Dabei ist die Anforderung an die Unterschrift, dass sie im Vergleich zu dem Dokument sehr klein ist – anstelle einer kompletten Kopie des zu unterzeichnenden Dokuments M wird also eine **charakteristische Zusammenfassung** mit Hilfe einer **Hashfunktion** $D(M)$ ermittelt.

Signatur mit Public Key

Bei der Verwendung asymmetrischer Verschlüsselungsverfahren kann eine Signatur $D(M)$ mit dem geheimen Schlüssel des Absenders verschlüsselt werden. Damit kann $D(M)$ nur mit dem öffentlichen Schlüssel des Absenders entschlüsselt werden und eine Änderung des Hashwertes ist somit unmöglich, sofern man nicht im Besitz des geheimen Schlüssels ist.

Digitale Unterschrift nach Needham Schoeder

Die Realisierung einer digitalen Unterschrift nach Needham Schroeder erfolgt in 5 Schritten:

1. $A \rightarrow S : A, \{D(M)\}_{K_A}$
   A schickt das Original mit der Signatur an den Server.
2. $S \rightarrow A : \{A, D(M), t\}_{K_S}$
   S entschlüsselt und vergleicht die Unterschrift. Dann zertifiziert er die Echtheit, indem er A's Namen an den Text hängt und alles mit seinem geheimen Schlüssel verschlüsselt.
3. $A \rightarrow B : M, \{A, D(M), t\}_{K_S}$
   A schickt das Original und das Zertifikat an B.
4. $B \rightarrow S : B, \{A, D(M), t\}_{K_S}$
   B schickt eine Kopie des Zertifikats an S.
5. $S \rightarrow B : \{A, D(M), t\} K_B$
   S entschlüsselt das Zertifikat und schickt das Ergebnis verschlüsselt mit dem geheimen Schlüssel von B an B zurück.

3.5 Die IEEE 802.x-Standards

Die IEEE 802.x-Standards definieren Standards in den Schichten 1 (Physical) und 2 (Link), insbesondere die Schichten des **Überragungsmediums**, der **Signalisierung**, des **MAC** und der **LLC**.

Desweiteren wird ein auf diesen Schichten aufbauendes Management beschrieben. In den Schichten **Medium**, **Signalling**, **MAC** gibt es die Standards **CSMA/CD** (Bus), **Token** (Bus), **Token** (Ring), **MAN** (DQDB), **IVD** (Sprache und Daten) und **Wireless LANs**. Auf der MAC-Ebene ist das **MAC Bridging** definiert und der **Logical Link** definiert die LLC.

ISO Link Layer	L2	SAP-Kommunikation (LLC)
	L1	Zugriff, Rahmenbildung (MAC)
ISO Physical L.	P2	Signalisierung
	P1	Medienabhängig (PMD)

3.5.1 Media Access Control (MAC)

Die MAC definiert Zugriffsverfahren und Rahmenbildung für das Übertragungsmedium. Diese Rahmen werden an die Signalisierungsschicht weitergegeben, so dass eine Kommunikation mehrerer Stationen über nur ein einziges Kabel ermöglicht wird.

Die Hauptaufgabe ist dabei – neben der eigentlichen Zugriffssteuerung – die Paketierung der Daten in Rahmen. Ein solcher **Rahmen** nach Ethernet 2.0 besteht aus:

PBL	Ziel	Quelle	Typfeld (Schicht 3)	PDU=Wasserköpfe+Daten	Prüfsumme
8	6	6	2	46...1500	4

Die **Zugriffsverfahren** und der **Zeitmultiplexbetrieb** soll eine möglichst hohe Kanalausnutzung bringen. Dabei wird zwischen Verfahren unterschieden, bei denen eine Kollision möglich (ALOHA, CSMA/CD) oder unmöglich (Polling, Token) ist. **CSMA/CD** ist sehr einfach (in Hardware) zu realisierung und benötigt keine Verwaltung, wohingegeben das **Token**-Verfahren einen Zuriff garantiert und bei starkem Verkehr gut skaliert — dafür benötigt es eine relativ aufwendige Verwaltung.

Bei CSMA/CD lauscht der Client auf dem Kabel, bis nichts gesendet wird und sendet dann seine Daten. Senden zwei Stationen gleichzeitig, so wird die Kollision erkannt und beide Teilnehmer warten eine zufällig lange Zeit ab, um die Nachricht erneut zu senden. Beim **Slot-Time-Verfahren** wird versucht eine Taktung des Zugriffs auf das Kabel zu realisieren, indem jeder Client ein ganzaligen Vielfachen des Zeitraums W wartet, bis er auf das Kabel zugreift.

Im Vergleich bieten Token und CSMA/CD folgende Vorteile:

Token	CSMA/CS
gute Eignung für Ring-Topologie	gute Eignung für Bus, Baum und Stern
berechenbare Zugriffsverzögerung	einfaches Einfügen von Stationen
gut bei hoher Last	einfaches Abtrennen von Stationen
erlaubt Prioritäten	einfache Algorithmen
ebf. für Bus, Baum, Stern verwendbar	gut bei niedriger Netzlast
keine Mindestrahmenlänge	keine zentrale Managementfunktion
keine Präambel	kein Token (Verlustrisiko)
Rückmeldung im Rahmen	Zugriffsverhalten unabh. von Stationsanzahl
	benötigt keinen logischen Ring

3.5.2 Logical Link Control

Die LLC ist für die Bereitstellung von Kommunikationsdiensten („**Link Service Primitives**") in Schicht 3 verantwortlich. Dabei können dank Multiplexing mehrere Schicht-3-Prozesse simultan mit einer LLC-Schicht zusammenarbeiten. Es stehen drei Typen der Kommunikation zu Verfügung:

1. Verbindungslose Kommunikation (ohne Quittung)
   Broadcast, Multicast, Punkt-zu-Punkt
2. Verbindungsortientierte Kommunikation
   Punkt-zu-Punkt, zuverlässig, weniger effektiv
3. Verbindungslose Kommunikation (mit Quittung)
   Punkt-zu-Punkt, selten

4 Netz- und Systemmanagement

Mit der Einführung von Computernetzwerken stellt sich die Frage nach der Verwaltung des Netzes. Dabei können Probleme aus **Größe und Heterogenität der Netze**, bereits **vorhandene Managementsysteme** und das **unterschiedliche Management in allen Anwendungsbereichen** entstehen. Gefordert ist also ein **homogenes integriertes Management heterogener Netze**.

4.1 OSI-Netzmanagement

Die Definition des **OSI-Netzmanagement** lautet:

> „The facilities to control, coordinate, and monitor the resources which allow communication to take place in an OSI environment." [ISO 7498-4]

Dabei werden 4 Teilmodelle unterschieden:

- Informationsmodell
- Organisationsmodell
- Kommunikationsmodell
- Funktionsmodell

4.1.1 Informationsmodell

Der Ansatz ist nach [ISO 10165] eine **objektorientierte Beschreibung von Managementinformationen unter Verwendung normierter Spezifikationsmittel**.

Die **Managed Objects** (**MO**) repräsentieren eine abstrakte Sicht auf logische und physikalische Ressourcen und sind durch **Attribute** (Eigenschaften/Status), **Notifikation** (Meldung von Ereignissen), **Operationen** (auf Attributen und dem MO als Ganzem) und **Verhalten** (Wirkung der Operationen) charakterisiert. Die Verwaltung der MOs übernimmt die **Management Information Base** (**MIB**). Die Zusammenfassung gemeinsamer Charakteristiken von MOs wird **MO-Klasse** genannt. Dafür steht eine **Template-Metasprache** zur Verfügung.

4.1.2 OSI-Organisationsmodell

Die Architektur des OSI-Organisationsmodells besteht aus der **Manager-/Agenten-Rolle**, dem **Datenaustausch zwischen unterschiedlichen Managementsystemen über normierte Dienste/Protokolle** und dem **Domänenkonzept**.

Dabei wird zwischen verschiedenen **Managementkategorien** unterschieden:

- **Systemmanagement** ermöglicht ein schichtübergreifendes Management eines Systems bzw. kooperierender Systeme
- **Schichtmanagement** nimmt Managementaktivitäten mit Bezug auf die n-te Schicht war
- **Protokollmanagement** stellt Mechanismen bereit, um eine Protokollinstanz innerhalb einer Schicht zu überwachen.

Für den Austausch von Managementinformationen im Systemmanagement gibt es die Standards **CMIS** (**Common Management Information Service**) und **CMIP** (**Common Management Information Protocol**). Dabei definiert CMIS eine Menge von Dienstprimitivien, Parameter, Semantik aber nicht die Implementierung der Dienste. Es werden Dienstgruppen zur Verbindungsverwaltung (Auf- und Abbau) und Festlegung des Anwendungskontexts (gemeinsames Managementwissen) und Managementoptionen und Notifikationen unterschieden.

4.1.3 OSI-Funktionsmodell

Die Managementaktivitäten werden in folgende **Funktionsbereiche** unterteilt:

- **Fehlermanagement** (Erkennung, Lokalisierung und Behebung von Störungen)
- **Leistungsmanagement** (Verbesserung des Leistungsverhalte von Ressourcen/MOs)
- **Konfigurationsmanagement** (Datenbereitstellung, Ändern von Attributen, Verwaltung der MOs etc.)
- **Abrechnungsmanagement** (Gebührenerhebung für die Nutzung von Ressourcen)
- **Sicherheitsmanagement** (Authentisierung, Zugangsüberwachung, Schlüsselverwaltung etc.)

Außerdem werden allgemeine **Systemmanagement Funktionen** (**SMF**) zur Verfügung gestellt, die aus Benutzeranforderungen, Definition von MOs / Attributen / Aktionen / Notifikationen, Beziehungen zu anderen SMFs etc. besteht (z.B. Object Management Function, State Management Function, Log Control etc.).

4.2 Internet-Management

Ziel des Internets war der Informationsaustausch zwischen Forschungseinrichtungen. Die Implementierung sollte einfach und effektiv sein, so dass man sich für ein Architekturmodell entschied, dass aus **Managern** und **Agenten** besteht, die auf den zu überwachenden Rechnern laufen. Als Protokoll ist das **Simple Network Management Protocol** (**SNMP**) definiert, das einfache Werte zurückliefert.

4.3 Vergleich

Ein Vergleich der beiden Systeme soll in der folgenden Tabelle angestellt werden:

Modell	ISO	Internet
Organisationsmodell	Verteiltes, kooperierendes Management über Manager und Agenten. Jedes OSI-System kann jede Rolle einnehmen.	Es wird zwischen aktiven (**Management station**) und passiven (**Managed nodes**, enthalten Agenten) Rollen unterschieden.
Informationsmodell	Objektorientierter Ansatz (Klassen-, Enthaltensein-, Registrierungshierarchie).	Managementinformationen werden durch abstrakte Datentypen beschrieben.
Kommunikationsmodell	Präferenz bei CMIP auf Anwendungsschicht.	Einfaches Managementprotokoll SNMP, das jeweils nur auf ein nichtstrukturiertes, adressiertes Objekt zugreift.
Funktionsmodell	5 Managementfuktionsbereiche und viele Systemmanagementfunktionen.	Existiert nicht.

4.4 CCITT-Management

„...TMN (Telecommunications Management Network) to support the management requirements of administrations to plan, provision, install, maintain, operate, and administer telecommunications networks and services"

5 Kooperation

Kooperation besteht aus **Kommunikation** (Transport der Daten über das Netz), **Synchronisation** (zeitliche Abstimmung der Aktionen) und **Koordination** (Steuerung und Überwachung der Komponenten).

5.1 Nachrichten-basierte Kommunikation

Message Passing

Im Allgemeinen besteht eine Nachricht aus den Feldern **Quelle**, **Ziel**, **Typ** und **Daten**. Nachrichten können auch lokal zwischen Prozessen verschickt werden (**Interprozess-Kommunikation**). Dabei kann ein **blockierendes Send** blockieren, bis die Nachricht

- beim Sender abgeschickt wurde.
- vom Kommunikationssystem empfangen wurde (Ack).
- vom Server-Prozess entgegengenommen wurde.

- die Nachricht vom Server-Prozess verarbeitet wurde.

Genauso unterscheidet man beim **blockierenden Receive** zwischen

- dem Abtesten ohne zu blockieren,
- dem bedingten Empfangen und
- dem Blockieren bis zum Time-Out.

Nachrichten können nach dem Empfang entweder in einem **Puffer** zwischengespeichert werden oder direkt zur Verarbeitung an den Ziel-Prozess übergeben werden (**Rendezvous**). Bei der **Erzeuger-Verbraucher-Interaktion** macht es Sinn Puffer einzusetzen, damit beide Akteure möglichst gut ausgelastet sind, genauso können Erzeuger und Verbraucher aber auch auf einen „gemeinsamen Speicher" zugreifen – dann besteht aber eine inhärente **Deadlockgefahr**.

5.2 RPC

Unter dem **Remote Procedure Call** (**RPC**) wird der Aufruf einer nicht-lokalen Prozedur verstanden. Die Idee dahinter war es, die Netzkomplexität hinter dem bekannten, bewährten und bequemen Programmiermodell zu verstecken. Seine Konzeption passt ideal in das Client/Server-Modell und ähnelt stark dem lokalen Prozeduraufruf.

Prozedur

Unter einer **Prozedur** verstehen wir eine **abgeschlossene Menge von Instruktionen mit eindeutigem Namen**. Sie besteht aus **Prozedurkopf** und **Prozedurrumpf**, der Aufruf geschieht **synchron** und die Abarbeitung ist sequentiell. Ausserdem ist sie **gedächtnislos** und **kennt den Aufrufer nicht**. Sie ist **an das aufrufende Programm gebunden** (Linker) und ist im Prinzip eine Sprunganweisung mit Namen und Parameter. Der Aufrufer und die aufgerufene Prozedur laufen i.A. im gleichen Adressraum (der Datenaustausch erfolgt über globale Variablen) – eine Prozedur kann jedoch **unerwünschte Seiteneffekte** haben.

Die **Parameterübergabe** kann **by value**, **by reference** oder **by name** erfolgen – so enthält die Prozedurdefinition die Parameter Name, Typ und Übergabeart. Der Parameter muss dann (typkonform) übergeben werden.

RPC

Die **Bestandteile eines RPC** sind also:

- **Schnittstellen-Beschreibung:** beschreibt Prozedurnamen und Parameter der aufzurufenden Prozedur.
- **Aufrufer (Client):** nimmt den Dienst in Anspruch.
- **Aufgerufener (Server):** bietet Dienst an.

- **Stubs:** Platzhalter-Prozeduren.
- **Laufzeitsystem:** Bindeglied zum Betriebssystem und Kommunikation.
- **Infrastruktur:** unterstützende Dienste, z.B. Verzeichnis.
- **Kommunikationsnetz:** transportiert Aufruf und Antwort.

Client Stub

Damit der Client einen Server-Dienst nutzen kann, benötigt er einen sog. **Client Stub**, der als Platzhalter im Anwendungsprogramm dient. Er wandelt Dienstanfragen in Datenpakete um und kümmert sich um die Kodierung der aufgerufenen Prozedur und das Verpacken (**Marshalling**) der Parameter. Gegebenenfalls ruft er noch andere Infrastrukturdienste auf (wie z.B. Verzeichnis, Vermittler etc.). Danach übergibt er das Paket an das Kommunikationssystem via Routinen des Laufzeitsystems und wartet auf eine Antwort. Die Ergebnisse werden dann ausgepackt und an die aufrufende Prozedur zurückgegeben.

Server Stub

Seitens des Servers bildet der **Server Stub** die Schnittstelle, über die Dienste durch Clients benutzt werden können. Er wartet auf Datenpakete und nimmt sie in Empfang. Dann kümmert er sich um die Dekodierung der Prozedurnamen und der Parameter (**Unmarshalling**), ruft die Prozedur auf und steckt die Ergebnisse in ein Antwort-Paket. Zuletzt schickt er das Datenpaket an den Client zurück und führt evtl. noch Verwaltungsarbeiten durch.

IDL

Damit Client und Server Stub automatisch erzeugt werden können, muss die Schnittstelle einer Prozedur festgelegt werden. Dafür gibt es die **Interface Definition Language** (**IDL**), die das Interface eines Service-Objekts durch **Namen der Operationen**, **Namen und Typen der Parameter und Zusatzinformationen** (z.B. eindeutige Kennzeichnung der Definition, Art der Operation, Fehlerverhalten etc.) beschreibt. Die DCE IDL ist lediglich **deklarativ** und stark an die Programmiersprache C angelehnt. Der Server-Programmierer spezifiziert die IDL und ein **automatischer Umsetzer** (**Stub Compiler**) erzeugt daraus automatisch die Stubs.

In der DCE IDL bekommt jede Schnittstellen-Deklaration einen eindeutigen Bezeichner, einen **Unique Universal Identifier** (**UUID**) zugewiesen, der 16 Byte lang ist und vom System erzeugt wird. Die Erzeugung beruht auf der Kodierung von Ort, Zeit und einem Zähler.

Pipes

Zur Übertragung großer Datenmengen gibt es **Pipes**, deren Arbeitsweise ähnlich den UNIX-Pipes ist: Bei einer Pipe $A \longrightarrow B$ blockiert A, wenn die Pipe voll ist und B, wenn sie leer ist.

Anwendungsentwicklung

Die Entwicklung von Anwendungen erfolgt in DCE in 3 Schritten:

1. **Schnittstellenbeschreibung**
   Aus der Schnittstellenbeschreibung erzeugt der IDL-Compiler Client und Server Stubs, die über den Linker in das Server und das Client Programm eingebunden werden. Ausserdem wird eine include-Datei erzeugt, die in beide Programme eingefügt wird.

2. **Server**
   Auf Serverseite müssen die verwendeten **Kommunikationsprotokolle festgelegt** werden und die **Prozedurschnittstellen registriert werden**. Anschliessend werden die **Prozedurschnittstellen zum Verzeichnisdienst exportiert**, der Server-Zustand evtl. initialisiert und auf Aufrufe gewartet.
3. **Client**
   Der Client **bindet den Server** mit Hilfe einer **Bindekennung** (**Binding Handle**) und ruft Prozeduren auf.

Bindung

Das **Binden** des Clients an den Server kann auf 3 verschiedene Arten erfolgen:

1. **automatisch:** das Verzeichnis wählt automatisch einen passenden Server für den Client aus.
2. **implizit:** der Client wählt selbst einen Server aus dem Verzeichnis aus und besorgt sich die Bindekennung.
3. **explizit:** der Client bestimmt den Server mit einer Bindekenung.

Aufrufsematik

Beim RPC werden verschiedene **Aufrufsemantiken** unterschieden:

- **exactly once:** genau einmal
- **at least once:** mindestens einmal
- **at most once:** höchstens einmal
- **maybe:** evtl. mehrfache Ausführung ohne Rückgabewert und ohne Fehlermeldung
- **broadcast:** evtl. mehrfache Ausführung bei allen passenden Servern und Rückgabe des ersten erfolgreichen Aufrufs
- **idempotent:** evtl. mehrfache Ausführung (ohne Wiederholungseffekt), ggf. mit Rückgabewert

RPC Eigenschaften

Die Haupteigenschaften des RPC sind:

- **synchroner (blockierender) Aufruf**
  Die Parallelität in verteilten Systemen wird durch eine erzwungene Sequentialisierung nicht ausgenutzt. Parallele Anfragen an mehrere Server sind also nicht ohne weiteres möglich.
- **Parameter**
  Die getrennten Adressräume lassen keine Referenzen zu. Der **call by reference**-Aufruf wird mittels **call by copy/restore** nachgebildet. Es ist nicht möglich beliebig komplexe verkettete Strukturen zu übergeben.

- **Fehler**
  Die Aufrufsemantik unterscheidet sich vom lokalen Prozeduraufruf und es wird ein getrennter Ausfall von Aufrufer und Aufgerufenem möglich (Waise).
- **Erweiterungen des klassischen RPC-Modells**
  Erweiterungen werden durch Einführung von asynchronen Aufrufen und **Futures**, die spezielle Kennzeichnung von Referenz-Parametern und call by copy/restore, zusätzliche Verfahren für sichere Objektreferenzen (Delegation), ein Pipe-Konstrukt für Massendaten (DCE) und den Broadcast RPC (DCE) erreicht.

Sichere Objektreferenzen

Die Einführung eines **Object Passing Agent** (**OPA**) soll **sichere Objektreferenzen beim RPC** ermöglichen. Durch sie wird eine **rekursive Delegation** möglich und ein zusätzlicher Parametertyp **Object Reference** in der IDL zeichnet diesen aus. Als Prototyp wurde OPA in NCS integriert. Stub Compiler kümmern sich um die entsprechenden OPA-Aufrufe in den Stubs – OPA muss aber ein **vertrauenswürdiger Server** sein, was einen **zusätzlichen Zugriffsschutz** erfordert.

5.3 DACNOS RSC

Das **Distributed Academic Computing Network Operating System** (**DACNOS**) ist Teil der HECTOR Kooperation der Uni Karlsruhe und IBM (1984-1988). Ziel war die Schaffung einer **Verteilungsplattform für heterogene verteilte Systeme**, wobei die **Funktionalität eines lokalen Betriebssystems auf eine verteilte Umgebung projeziert** werden sollte. Einen Überblick über die Architektur von DACNOS gibt Abbildung 4. Wie man sieht soll DACNOS keinesfalls das Betriebssystem ersetzen, sondern ist nur als Erweiterung desselben gedacht. Ziel war die Schaffung von Verteilungstransparenz, so dass Anwendungen ohne Änderungen in verteilten Umgebungen lauffähig sind, sowie ein **integrierender Ansatz**, der sich nicht nur auf die Kollaboration sondern auch auf Sicherheits- und Managementaspekte konzentrierte.

Komponenten

DACNOS besteht aus 5 Komponenten:

- **Remote Service Call** (**RSC**)
  Stellt eine API für die Kooperation und Management mit objektbasierter Schnittstelle zur Verfügung.
- **Kernel Service Call** (**KSC**)
  Ist ein Thread-Paket mit eigener abstrakter Syntaxnotation. Sinn dieser Schicht war es eine größere Portabilität durch einen Abstraktionslayer über dem unterliegenden Betriebssystem zu erreichen.
- **Presentation System** (**PS**)
  Ist für die Datenpräsentation mit eigener abstrakter Syntaxnotation zuständig.

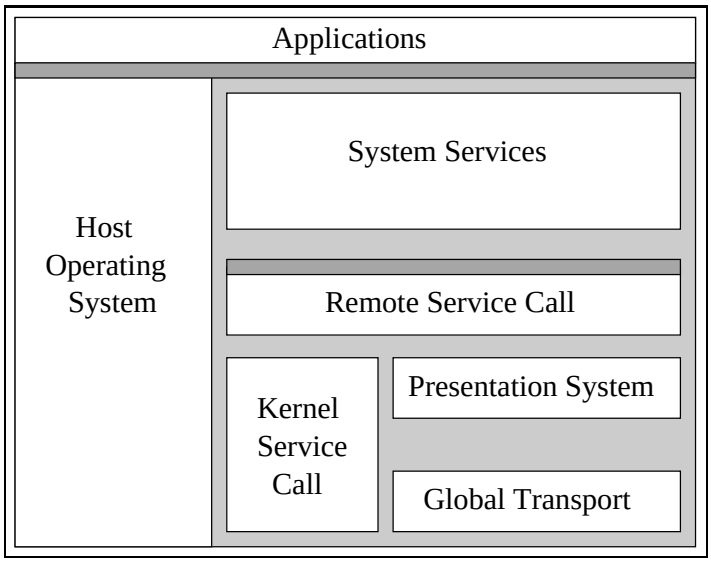


Abbildung 4: Architektur von DACNOS

- **Global Transport System** (**GT**)
  Bietet ein einheitliches Transportsystem in heterogenen Netzen.
- **System-Infrastrukturdienste** (**System Services**)
  Ein Verzeichnis, verteiltes Dateisystem, Sicherheit (Authorisierung und Authentisierung), Abrechnung, Orientierung, Remote Execution etc.

RSC

Der **Remote Service Call** (**RSC**) bietet eine objektbasierte Programmierschnittstelle, setzt jedoch keine OO-Programmierung voraus. Sie stellt die Basis für das Client/Server-Modell dar und bietet mit **Object Sharing** die Grundlage für Kooperation. RSC Objekte präsentieren sich dabei als typische BS-Konstrukte – die Auftragsabwicklung erfolgt dabei **nicht blockierend**. Ereignisse können Objekten zugeordnet und spezifiert werden und zu Ereignislisten zusammengefasst werden.

Beim RSC werden folgende **Objekttypen** unterschieden:

- **PORT** bezeichnet den Zugangspunkt mit definierter Aufruf-Schnittstelle
- **CARRIER** ist ein Auftrag mit definierter Parameterstruktur (Daten und Objekte)
- **WINDOW** ist ein Datenpuffer mit definierter Datenstruktur.
- **LOCK** ist ein Synchronisationsobjekt.
- **NOTICE** ist eine Nachricht (ohne Antwort).

- **EVENTLIST** ist ein Behälter für Ereignisobjekte.
- **PROCESS** ist ein ablaufendes Programm.

Programmierung

Bei der Programmierung mit RSC macht der Client Aufrufe auf das CARRIER-Objekt, während der Server auf dem PORT-Objekt lauscht und über das CARRIER-Objekt antwortet bzw. über ein WINDOW-Objekt Daten einliest.

Object Sharing

Für die gemeinsame Verwendung von Objekten stellt DACNOS den **Shared Object Space** zur Verfügung. Auf ihn kann sowohl von Client- als auch von Server-Seite zugegriffen werden.

ACCOUNT-Objekt

Ein besonderes Objekt ist das **ACCOUNT-Objekt**, das Klienten eindeutig identifiziert. Es ermöglicht so eine Abrechnung von erbrachten Diensten und wird mit einem CARRIER verbunden. Jedes ACCOUNT-Objekt enthält also eine **Account Number** (Kontonummer) für die Abrechnung beim Accouting Server und kann zum Erfassen unterschiedlicher Abrechnungsinformationen (CPU-Zeit, Speicher, I/O etc.) verwendet werden.

Als Operationen auf einem ACCOUNT-Objekt stehen `create`/`delete`, Extrahieren eines ACCOUNT aus einem CARRIER, Anhängen eines ACCOUNT an einen CARRIER und Verändern der ACCOUNT-Daten zur Verfügung. Dabei schreibt der Server Verbrauchsinformationen in das ACCOUNT-Objekt des Clients und schickt am Ende die Daten an den Accounting Server. Dieser sammelt die Daten in einer Datenbank und ermöglicht so eine Rechnungsstellung. Problematisch ist allerdings die Verrechnung heterogener Abrechnungsdaten. Der Anwendungsserver selbst kann mit dem ACCOUNT des Clients weitere Server aufrufen – was natürlich auf Kosten des Clients geht.

6 Software-Test

Zu Beginn eines Tests müssen die **Ziele des Tests** festgelegt werden. Dabei soll die Anzahl der Felder, die mit einer endlichen Menge von Tests gefunden werden können maximiert werden. Als **Randbedingung** sollen Zeit und Kosten minimiert werden.

Testprinzipien

Genauso müssen die **Testprinzipien** festgelegt werden, die aus folgenden Richtlinien bestehen:

- Definition der erwarteten Werte für jeden Testfall.
- Programmierer sollen nicht ihre eigenes Programm testen.
- Testergebnisse müssen sorgfältig geprüft werden.
- Testfälle müssen sowohl für gültige als auch für ungültige Eingaben definiert werden.

- Die Wahrscheinlichkeit für die Existenz weiterer Fehler ist proportional zu der Anzahl bereits gefundener Fehler.
- Vermeidung von Wegwerftestfällen.

6.1 Testmethoden

Bei den Testmethoden gibt es mehrere unterschiedlich Alternativen. So unterscheiden wir:

- **Strukturelles Testen** (**white-box testing**)
  Hier liegt der Quelltext vor und es können interne Strukturen getestet werden.
  Als Methoden stehten hier zur verfügung
  - **human testing:** Code-Inspektion im Team
  - **statement coverage:** Jede Anweisung mind. einmal ausführen
  - **branch coverage:** Jeden Pfad mind. einmal durchlaufen
  - **data-flow testing:** Pfadtesten mit Überprüfung von Datenobjekten (**Datenanomalien**)
- **Funktionales Testen** (**black-box testing**)
  Hier ist der Code unbekannt und es wird lediglich das Ein-/Ausgabeverhalten getestet. Folgende Methoden werden angewandt:
  - **equivalence partitioning:** Einteilung der Eingabedaten in Äquivalenzklassen
  - **boundary-value analysis:** Grenzbereiche möglicher Eingaben untersuchen
  - **error guessing:** Intuitiver Prozess
  - **state testing:** Testen eines Zustandsautomaten

Das Hauptproblem beim Testen ist, dass ein vollständiges Testen aus theoretischen und praktischen Gründen nicht möglich ist. Als Teststrategie sollte also eine Kombination aller Methoden gewählt werden.

Testebenen

Wir unterscheiden im Folgenden 3 verschiedene **Testebenen**:

- **Isolationstests** testen nur einzelne Komponenten eines Systems.
- **Integrationstests** testen das Zusammenspiel der Komponenten (z.B. durch **module testing**, **(non-)incremental**, **bottom-up**, **top-down**, **big-bang**, **sandwich**).

- **Systemtests** testen das vollständige System.

ktorientierte Program-

Beim Test **Objektorientierter Programme** können die **Charakteristiken Objektidentität**, **Kapselung**, **Vererbung** und **Polymorphie** beeinflusst werden. In diesem Fall gibt es folgende Testebenen:

- Testen jeder Operation
- Testen einzelner Objekte (**object unit testing**)
- Testen einer Menge interagierender Objekte **(object integration testing**)
- Testen des gesamten Systems

Dabei ergeben sich eine Reihe offener Probleme wir z.B.

- Welche Arten von Fehlern treten auf?
- Wie und welche Testfälle sollen spezifiziert werden?
- Wie werden Testfälle ausgewertet?
- Welche Testfälle können wiederverwendet werden?
- Wieviel Testen ist notwendig?
- Wie können Testfälle generiert werden?
- Wie können Fehler vermieden werden (**design for testability**)?

6.2 OSI-Konformitätstests

Zum Testen von **Diensten und Protokollen aus den Schichten des OSI-Basisreferenzmodells** wurde der **OSI-Konformitätstest** definiert. Motivation war die fehlerfreie Interaktion verschiedener Implementationen der gleichen Protokollnorm, die oft wegen fehlerhafter Implementation oder unterschiedlicher Interpretation nicht gegeben war. Mit diesen Tests sollte also die Wahrscheinlichkeit der Interoperabilität erhöht werden.

Als Ansatz wurde die Überprüfung des nach Aussen hin sichtbaren Verhaltens einer Protokollimplementation mit Hilfe normierter Testfälle gewählt, was dem **black-box testing** entspricht. Richtlinien und Verfahren für das Konformitätstesten sind in dem Standard ISO 9646 festgelegt (**Conformance Methodology and Framework**). Dort werden die folgenden „Test-Angriffspunkte" unterschieden, die in Abbildung 5 illustriert sind:

- **IUT** – Implementation Under Test

- **PCO** – Point of Control and Observation
- **ASP** – Abstract Service Primitive
- **PDU** – Protocol Data Unit
- **UT** – Upper Tester
- **LT** – Lower Tester

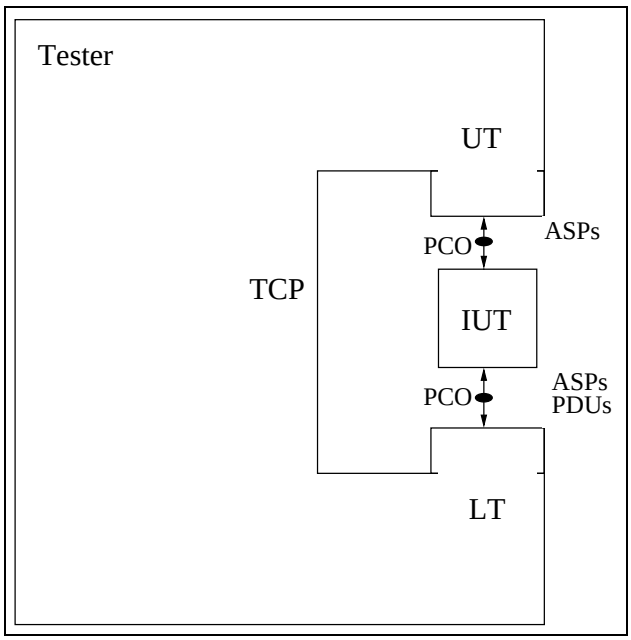


Abbildung 5: OSI-Konformitätstest-Architektur

In der **Testsuite** ist eine Menge alle Testfälle für eine Protokollnorm zusammengefasst. Die eigentliche **Testentwicklung** wird dabei durch die **Konformitätsanforderungen**, die **Definition von Testgruppen** und die **Spezifikation von Testfällen für jede Gruppe** (Überführung der IUT in den benötigten Anfangszustand, Senden der Testereignisse (ASP/PDU), Empfangen von Testereignissen (ASP/PDU), Auswerten der Ergebnisse und Vergabe des Testurteils, Überführung der IUT in einen stabilen Zustand) bestimmt. Die **Tree and Tabular Combined Notation** (**TTCN**) ist eine normierte Testnotation zur Definition von Testfällen.

Mit der **Konformatität** bezeichnen wir die Übereinstimmung mit den Konformitätsanforderungen für eine Protokollnorm. Sie besteht aus eine Auflistung statischer Konformitätsanforderungen (**PICS** – **Protocol Implementation Conformance Statement**) und implementierungspezifischer Information (**PIXIT** – **Protocol Implementation eXtra Information for Testing**).

Die **Phasen eines Konformitätsbewertungsprozesses** sind **Testvorbereitung, Testdurchführung und Erstellung eines Testreports**.

Unter dem **Prevention Testing** verstehen wir die Integration des Testprozesss in den Softwareentwicklungszyklus. Das Vorgehen wird mit **Demonstration & Detection** bezeichnet: es soll sichergestellt werden, dass das Programm das Problem löst und Fehler gefunden werden. Eine Übersicht gibt die folgende Tabelle:

	Focus	Timing		Coverage	Visibility
		Planning	Design		
Prevention Testing	Prevention	Before & after software requirements	After software requirements	Largely known	Publicly documented and reviewed
Industry Practice	Detection & Demonstration	After software design	After software design	Largely unknown	Publicly or privatelay documented or undocumented with little or no review

7 Prüfungsfragen

- **Was ist ein offenes System?**
  Nach dem POSIX-Standard der IEEE ist ein offenes System *„ein System, das in ausreichendem Maße offengelegte Spezifikationen für Schnittstellen und dazugehörige Formate implementiert, damit entsprechend gestaltete Anwendungssoftware*
  - *auf einer Vielzahl verschiedener Systeme portiert werden kann*
  - *mit anderen Anwendungen lokal und entfernt interoperabel ist*
  - *mit Benutzern in einer Art interagiert, die das Wechseln der Benutzer zwischen Systemen erleichtert."*

- **Wie werden verteilte Systeme charakterisiert?**

  „Ein verteiltes System besteht aus autonomen Subsystemen, die koordiniert kooperieren, um eine gemeinsame Aufgabe zu erfüllen." [Geihs1995]

  „Verteiltes System = verteilte Hardware und/oder verteilte Kontrolle und/oder verteilte Daten" [Sloman et al. 1989]

  „Ein verteiltes System ist eines, in dem mehrere autonome Prozessoren und Datenspeicher, die Prozesse bzw. Datenzugriffe unterstützen, so kooperierend zusammenarbeiten, dass ein gemeinsames

Ziel erreicht wird. Die Prozesse koordinieren ihre Aktivitäten und tauschen Informationen über ein Netzwerk aus." [Sloman et al. 1989]

- **Allgemeine Probleme in verteilten Systemen?**
  - Variable Zeitverzögerungen,
  - keine gemeinsame Zeit,
  - kein gemeinsamer Speicher,
  - neue Fehlerklassen,
  - schwierige Verwaltung und Kontrolle,
  - Interoperabilität,
  - komplexere Handhabung,
  - parallele Ausführung,
  - Synchronisierung,
  - Sicherheit,
  - keine Voraussagen über Performance möglich,
  - komplexes Design,
  - Heterogenität,
  - Inkompatibilitäten,
  - Testen und Debuggen schwer.
- **Vorteile eines verteilten Systems?**
  Resource Sharing, Datenzugriff, Zentrale Dienste, Flexibilität, Skalierbarkeit, Effizienz, Verfügbarkeit.
- **Welche Arten von Transparenz gibt es (Beispiele)?**
  - Ort (CORBA: Zugriff auf Objekt über IOR, unabhängig vom Ort)
  - Zugriff (NFS: kein erkennbarer Unterschied zwischen lokalem und entferntem Zugriff)
  - Migration (CORBA: ein Objekt wird „verlegt", ist aber unter seiner alten IOR ansprechbar),
  - Replikation (Mobile Geräte, die resynchronisiert werden),
  - Transakion (Transaktiondienst, der nicht explizit angesprochen wird, sondern implizit da ist),
  - Nebenläufigkeit (Bereitstellung eines virtuellen gemeinsamen Speichers),
  - Fehler (erlaubt es Programmen trotz Ausfall von Hard- oder Software weiterzuarbeiten).

- **Was heisst Heterogenität?**
  Heterogene Computer unterscheiden sich bzgl. Hard- und Software, z.B. durch unterschiedliche Bauelementebasis, Aufrufbreite und Betriebssystem. *„Eine Verteilungsplattform sollte Heterogenität weitgehend maskieren und beschränkt sich nicht auf Datenformat-Konversion sondern bietet auch Portabilität, Verträglichkeit mit den Dateisystemen, Sicherheitsmechanismen und Managementvorkehrungen."* [Geihs1995]

- **Welche Verteilungsplattformen kennen sie?**
  CORBA, DCE, RMI, DCOM, Mobile Agenten. *„Eine Verteilungsplattform sollte unabh. vom Transportsystem und den darunter liegenden Protokollschichten sein"* [Geihs1995]. Die Protokollunabhängigkeit wird dann durch entsprechend standardisierte API's der Transportschicht gewährleistet. Insgesamt werden in [Geihs1995] 5 verschiedene Varianten von Verteilungsplattformen unterschieden:

  1. Message Passing („Assembler Programmierung" in verteilten Systemen)
  2. Netz-Betriebssystem (NFS, Novell Netware)
  3. RPC (nahtlose Integration in die Programmiersprache)
  4. Entfernter Datenbankzugriff
  5. Verteilte Objektsysteme (CORBA)

- **Was ist ein Referenzmodell?**
  Es reduziert die Komplexität des Kommunikationsprozesses durch Unterteilung in eine Reihe miteinander verbundener Teilaufgaben. Diese Teilaufgaben werden in sog. Schichten erledigt, die miteinander über definierte Schnittstellen kommunizieren, um gemeinsam eine reibungslose Übertragung zu gewährleisten. Das Referenzmodell legt die einzelnen Schichten fest und beschreibt, welche Teilaufgaben in welcher Schicht erledigt werden sollen.
  Einfach gesagt beantwortet es die Frage, wie in heterogenen Systeme miteinander kommuniziert werden kann. Es beschreibt die logische Architektur des Rechnernetzes (→Netzwerkarchitektur), nicht die Implementierungsvorschrift.

- **Welche Referenzmodelle kennen sie?**
  Das OSI-Referenzmodell der ISO und das TCP/IP-Modell der IETF.

- **Was ist genau eine Schicht?**
  Eine Schicht im Referenzmodell stellt der darüberliegenden Schicht eine Reihe von Diensten zur Verfügung, u.a. dadurch, dass sie Funktionen darunterliegender Schichten nutzt.

- **Was leistet das OSI Referenzmodell?**
  Das OSI Referenzmodell unterteilt das Kommunikationssystem in 7 Schichten.

- **Erklären sie die 7 Schichten des OSI RM!**

  1. Bitübertragung (pysical layer): In dieser Schicht werden alle physikalisch-technischen Eigenschaften der Übertragungsmedien zwischen den verschiedenen End- bzw. Transitsystemen festgelegt, sowie die Herstellung von ungesicherten Verbindungen zur Übertragung von Bitfolgen über das Netz.
  2. Sicherungsschicht (data link layer): Die Aufgabe dieser Schicht ist es, die Bitübertragungsschicht gegen auf den Übertragungsstrecken auftretende Übertragungsfehler abzusichern. Dies geschieht z.B. durch Berechnung und Vergleich von Prüfsummen.
  3. Vermittlungsschicht (network layer): Hier wir die Adressierung von Zielsystemen über Transitsysteme hinweg sowie die Wegsteuerung der Nachrichten durch das Netz übernommen. Eine weitere Aufgabe ist die Flusskontrolle zwischen End- und Transitsystemen zur Verhinderung von Überlastungen der Übertragungswege.
  4. Transportschicht (transport layer): Mit Hilfe der drei unteren Schichten stellt diese Schicht eine Endsystemverbindung zwischen den Anwendungsinstanzen zur Verfügung. Vom Netz her gesehen ist dies die unterste Ebene mit Ende-zu-Ende Signifikanz. Über die Schichten 1 bis 3 erfolgt lediglich die Kommunikation.
  5. Kommunikationsschicht (session layer): Ab dieser Schicht beginnen die Schichten des Anwendersystems. Hier werden Sprachmittel zur Verfügung gestellt, mit deren Hilfe eine Kommunikationsbeziehung (Sitzung) gesteuert (also aufgebaut, unterhalten und abgebaut) werden kann. Damit soll es auch möglich sein eine Sitzung nach einem ungeplanten Abbruch in einer unteren Schicht wieder aufnehmen zu können, um sie dann geordnet zu beenden.
  6. Datendarstellungsschicht (presentation layer): Diese Schicht bietet den Anwendungsinstanzen die Möglichkeit, Vereinbarungen bzgl. der Datenstrukturen für den Datenaustausch zu treffen.
  7. Anwendungsschicht (application layer): Die anwendungsspezifischen Funktionen einer Kommunikation werden in der obersten Schicht vereinbart. Das heisst, hier wird der inhaltsbezogenen Aspekt berücksichtigt.

- **Was wird benötigt, damit zwei Schichten auf zwei Rechnern miteinander kommunizieren können?**
  Ein Protokoll.

- **Was ist ein Protokoll?**
  Eine Menge von Vereinbahrungen zu Kommunikation zwischen zwei Kommunikationspartnern. Das Protokoll legt die Syntax, Semantik und die Reihenfolge der von den Kommunikationspartnern ausgetauschten Rahmen, Paketen oder Nachrichten fest.

- **Was ist ein Dienst?**
  Eine Menge von Operationen (Primitiven), die eine Schicht der über ihr liegenden zur Verfügung stellt.

- **Welche Arten von Diensten gibt es?**

  1. Verbindungsorientierte Dienste: Eine Verbindung wird aufgebaut, Daten werden übertragen, die Verbindung wird wieder aufgebaut (wie beim Telefonsystem). Es kann keine Fehler oder Duplikate geben, die Reihenfolge bleibt immer korrekt.
     Zuverlässige verbindungsorientierte Dienste fordern Bestätigungen beim Datentransfer an.
     Unzuverlässige verbindungsorientierte Dienste fordern keine Bestätigungen an und sind deshalb ressourcen-schonender.
  2. Verbindungslose Dienste: Ein Pakt bekommt eine Adresse und wird unabhängig von anderen abgeschickt, die Reihenfolge wird nicht garantiert (Datagrammdienst).
     Bestätigte verbindungslose Dienste: z.B. E-Mail
     Unbestätigte verbindungslose Dienste: Internet-Telefonie.

- **Was ist die Instanz einer Schicht?**
  Ein aktives Element (Programm), das das spezifische Protokoll der Schicht implementiert und gleichzeitig Dienste erfüllt. Das kann Software oder Hardware sein.

- **Wie greift eine Schicht auf die Dienste der darunterliegenden Schicht zu?**
  Jede Schicht bietet der darüberliegenden Schicht eine Menge von Diensten an, die durch sog. **Service Access Points** (**SAP**s) bereitgestellt werden. SAP's sind vergleichbar mit Ports beim TCP-Protokoll.

- **Wie funktioniert die Zusammenarbeit zwischen Schichten?**
  Das Zusammenspiel zweier aneinandergrenzender Schichten wird durch den Austausch von Grundelementen (primitiven Dienstoperationen) ermöglicht:

  1. Anfrage (Request): $N+1$ fordert von N einen Dienst an und übergibt die nötigen Parameter.
  2. Benachrichtigung (Indication): N schickt an $N+1$ ein Grundelement, das $N+1$ über die Aktivierung eines angeforderten Dienstes bzw. eine gestartete Aktion informiert.
  3. Antwort: $N+1$ antwortet auf die Benachrichtigung.
  4. Bestätigung (Confirm): N bestätigt $N+1$ eine zuvor aufgerufene Prozedur.

Schicht 1 und 2

- **Welchen Schnittstellenempfehlungen gibt es in der Schicht 1?**
  X.21 für synchrone Duplexübertragung in digitalen Datennetzen, V.24 oder RS232-c für Analogleitungen, bei der Signalisierung und Daten über getrennte Leitungen laufen, serielle Übertragung mit 4 Zuständen (Bereit, Verbindungsaufbau, Datentransfer, Verbindungsabbau).

- **Sowohl Schicht 1 als auch Schicht 2 sollen Daten übertragen. Wo liegt der Unterschied?**
  In Schicht 2 werden Rahmenbildung, Fehlerüberwachung (Beschädigung, Verlust, Duplizierung) und Flusssteuerung (Empfänger und Sender gleichen Umfang und Geschwindigkeit ab) implementiert.

- **Wann werden PPP-Netze bzw. Broadcast-Netze eingesetzt?**
  Broadcast ist nur in kleinen (lokalen) Netzen sinnvoll. PPP ist ein Schicht-2-Protokoll im Internet, dass z.B. beim Tunneln eingesetzt wird.

- **Wie funktioniert SLIP?**
  SLIP sendet IP-Pakete, die mit einem speziellen Flag beginnen und enden. Kommt genau diese Zeichenfolge im Paket selbst vor, so wird sie gestopft.

- **Nachteile von SLIP?**
  Weder Fehlerkorrektur noch Fehlerbehebung, was eigentlich Aufgaben von Schicht 2 sind. Ist auf IP spezialisiert und Sender und Empfänger müssen ihre IP-Adresse selbst im Voraus wissen – es gibt also keine dynamische Adress-Zuweisung. Eine Authentifikation ist nicht vorgesehen.

- **Was ist PPP?**
  PPP bietet Rahmenerstellungsmethoden und Fehlererkennung. Es verwendet ein Verbindungsprotokoll namens LCP, das zum Testen und Anschalten von Leitungen dient. Zu Zeiten der Entwicklung von OSI gab es nur PPP-Netze – mit der Idee der Broadcast-Netze wurde die MAC-Schicht entworfen.

- **Wie funktioniert Ethernet?**
  Ethernet ist ein Broadcast-Netz, das eine Sicherungsschicht namens MAC (Mediumszugriffsschicht) enthält. Sie enthält Protokolle, die die Nutzung des gemeinsamen Übertragungsmedium organisieren. Ethernet ist der Standard für Broadcast-basierte LANs.

- **Welches Sendeprotokoll wird im Ethernet verwendet?**
  Beim **1-Persistent CSMA/CD** hören alle Stationen permanent den Kanal ab. Eine Station kann Daten übertragen, sofern der Kanal nicht belegt ist. Bei einer Kollision unterbricht jede Station die Übertragung und wartet einen zufällige Zeit vor dem Neustart der Übertragung. 1-Persistent bedeutet, dass busy waiting betrieben wird wenn die Leitung besetzt ist. Bei **Non-Persistent** wird eine zufällige Zeitstpanne gewartet.

- **Welche Dienstarten gibt es in verteilten Systemen?**
  Bei verbindungslosen (unzuverlässigen) Datagrammen übernehmen die

Hosts die Fehlerüberwachung und Flusssteuerung selbst. Bei verbindungsorientierten (zuverlässigen) Verbindungen wird dies von Schicht 2 übernommen. Im Internet sind Dienste verbindungslos, bei ATM immer verbindungsorientiert. Verbindungslose Dienste verlagern die Komplexität der Übertragung auf die Transportschicht (4).

- **Vergleiche verbindungsorientierte (virtuelle) und verbindungslose (Datagramm) Verbindungen!**
  Bei virtuellen Verbindungen wird beim Verbindungsaufbau eine Route zwischen den Teilnehmern festgelegt, die dann für alle Übertragungen beibehalten wird. Dementsprechend enthalten Pakete Verbindungsnummern anstelle von vollen Zieladressen. Dazu muss jeder Router einen Eintrag für jede offene virtuelle Verbindung speichern.
  Bei Datagramm Verbindungen werden keine Routen im voraus festgelegt. Pakete enthalten die volle Zieladresse und jedes Paket wird unabhängig von seinem Vorgänger befördert. Dieses Verfahren ist vor allem für kurze Verbindungen geeignet.

- **Wofür wird die Vermittlungsschicht benötigt?**
  Die Vermittlungsschicht dient der Adressierung von Zielsystemen über Transitsysteme hinweg. Desweiteren dient es der Flusskontrolle zwischen End- und Transitsystemen zur Verhinderung von Überlastungen der Übertragungswege.

Schicht 3

- **Nennen sie Protokolle der Schicht 3!**
  X.25 (drei Schichten: 1. digital oder analog; 2. durch HDLC; 3. Funktionen der Vermittlungsschicht) und LAPB.

- **Was genau ist X.25?**
  „Die ITU-T-Empfehlung X.25 beschreibt die Schnittstelle zwischen Datenendeinrichtungen DEE und Datenübertragungseinrichtungen DÜE an paketvermittelten Datennetzen. Der Funktionsumfang der Schnittstelle X.25 entspricht den Schichten 1 bis 3 des ISO/OSI-Basisreferenzmodells." [Schneider, Werner 2000, Kap. 6.1.7.3] Diese einzelnen Schichten werden oft auch als X.25/1, X.25/2 und X.25/3 bezeichnet, wobei sie folgende Aufgaben übernehmen:
  • X.25/1: Stellt einen transparenten Übertragungsweg am DEE/DÜE-Interface für Dateneinheiten der Schicht 2 bereit und sorgt für eine serielle asynchrone transparente Datenübertragung im Vollduplexbetrieb. Dabei werden physikalische Schnittstellen nach X.21, X.21bis oder V.-Empfehlungen verwendet mit Übertragungsgeschwindigkeiten bis 10 Mbit/s.
  • X.25/2: Die Datenübermittlungssteuerung erfolgt gemäß der HDLC-Prozedur und der Übermittlungsvorschrift LAPB mit Vollduplex, synchroner Datenübertragung, Fehlererkennung durch CRC-Prüfung und Fehlerkorrektur durch `Go-Back-N`.
  • X.25/3: Diese Schicht übernimmt das Multiplexen mehrerer logischer Kanäle auf einem Übermittlungsabschnitt. Dabei werden die virtuellen

Verbindungen überwacht und gesteuert durch Fehlerkontrolle, Fehlerkorrektur, Paketflusssteuerung und Reihenfolgesteuerung.

Schicht 4

- **Charakterisieren sie kurz die Schicht 4!**
  Die Transportschicht ist die erste Ende-zu-Ende-Schicht. Sie kennt nur DTE's und keine DEC's. Sie stellt das Bindeglied zwischen den Anwendungsprogrammen und dem Netz dar, sorgt für eine fachgerechte Übermittlung der Daten (zur Not mittels Fragmentierung) an die Schicht 3 und ist für Quality of Service zuständig. Diese werden durch Parameter beim Verbindungsaufbau charakterisiert und können Dauer, Ausfallwahrscheinlichkeit, Durchsatz, Transitverzögerung, Fehlerrate, Schutz und Priorität der Verbindung betreffen.

???

- **Was macht die Transportschicht?**
  Mit Hilfe der Schichten 1 bis 3 stellt die Transportschicht eine Endsystemverbindung zwischen Anwendungsinstanzen zur Verfügung.

- **Welche Protokolle werden dort benutzt?**
  X.25, IP.

Schicht 5

- **Was sind die Hauptaufgaben der Schicht 5?**
  Die Sitzungsschicht stellt ihren Nutzern (der Darstellungsschicht) Dienste zur Dialogführung und Synchronisation zur Verfügung. Dazu realisiert sie Dienste für die Herstellung einer Sitzungsverbindung und sorgt für die Synchronisation der Übertragung und Sitzungssteuerung. Genauso werden auch Dienste zum Sitzungsauf- und -abbau zur Verfügung gestellt.

- **Wie wird die Synchronizität gewährleistet?**
  Der Einsatz von Wiederaufsetzpunkten (Resycnhronization Points) ermöglicht es bei Bedarf die Kommunikation auf einen dieser Punkte zurückzusetzen.

- **Wie werden Dialogfunktionen gesteuert?**
  Durch Tokens (Activity Token, Data Token, Release Token).

- **Welche Arten von Synchronisationspunkten gibt es?**
  Hauptsynchronisationspunkte müssen gesetzt werden, Nebensynchronisationspunkte sind dagegen optional.

- **Was ist ein Dialog?**
  In diesem Zusammenhang ist ein Dialog eine logische Einheit zwischen zwei Hauptsynchronisationspunkten. Zwischen diesen Punkten können noch mehrere Nebensychronisationspunkte gesetzt werden, um den Dialog zu verfeinern – sobald ein Hauptsynchronisationspunkt gesetzt ist, ist das Zurücksetzen auf einen Nebensynchronisationspunkt nicht mehr möglich.

- **Was ist eine Aktivität?**
  Eine Menge von Dialogen.

- **Wer legt die Bedeutung der Synchronisationspunkte fest?**
  Die Anwendung.

- **Was sind die Hauptaufgaben der Schicht 6?**
  Damit zwei Anwendungen miteinander Daten austauschen können, müssen sie zunächst eine Menge von (abstrakten) Datentypen verfügen, in der sie ihre Daten repräsentieren können. Die Darstellungsschicht bietet Anwendungen diesen Dienst an. Ihre Hauptaufgabe ist die Umwandlung der Darstellungsart der Daten einer Anwendung in die einer anderen Anwendung durchzuführen.

- **Was wird in diesem Zusammenhang unter einer abtrakten Syntax verstanden?**
  Eine Menge von Datentypen einer Anwendung (z.B. String, float). Damit zwei Anwendungen miteinander kommunizieren können müssen sie die gleiche Menge von Datentypen haben.

- **Was ist LER?**
  Die Local Encoding Rules sind Regeln, denen der Decoder bzw. Encoder folgt, um die Transfersyntax in eine lokale Syntax zu dekodieren bzw. die lokale Syntax in die Transferyntax zu kodieren.

- **Was ist ASN.1?**
  Die Abstract Syntax Notation ist eine von der OSI vorgeschlagene Beschreibungssprache zur maschinenunabhängigen Beschreibung einer abstrakten Syntax strukturierter Daten einer Anwendung. Nach [Gora1998] ist *„eine abstrake Syntaxnotation ein von der [...] Zielumgebung unabhängiges Beschreibungsmittel, mit dem ausgedrückt werden kann, wie aus einzelnen Daten(-elementen) in korrekter Art und Weise komplexere Datenstrukturen und Informationsgebilde aufgebaut werden können."* Mittels ASN.1 ist eine offene, d.h. herstellerunabhängige Kommunikation möglich und soll Aufgaben der Schichten 6 und 7 im OSI-Modell übernehmen.

- **Was ist BER?**
  Die Basic Encoding Rules wurden zur Ergänzung von ASN.1 festgelegt und stellen eine Menge maschinenunabhängiger Kodier- und Dekodierregeln dar, durch deren Anwendung auf in ASN.1 dargestellte Datentypen sich eine maschinenunabhängige Transfersyntax ergibt. BER wird also zu Kodierung und Dekodierung von ASN.1-Konstrukten verwendet.
  Die Kodierung eines Datenwertes in ASN.1 mit Hilfe der BER gliedert sich in maximal 4 Komponenten: Das Bezeichnerfeld (Identifikationsfeld), das Längenfeld oder ein Hinweis auf ein EOC am Ende des Wertes, das Inhaltsfeld und schliesslich das optionale EOC.

- **Welche Datentypklassen gibt es in ASN.1?**
  UNIVERSAL umfasst vordefinierte Datentypen wie Integer, Boolean, ISO646String etc.
  APPLICATION ist die Klasse, die zur Erzeugung selbstdefinierter Typen

benutzt wird. Dazu gibt es Konstrukte wie Sequence, Set, Choice etc. PRIVATE ist eine Klasse mit Typ-Definitionen innerhalb einer Organisation. Typen beginnen mit Großbuchstaben, Werte mit Kleinbuchstaben.

- **Was ist ein Tag in ASN.1?**
  Jedem Datentyp ist eine eindeutige Kennzeichnung zugeordnet – ein Tag. Es wird mit übertragen, damit ein Datentyp eindeutig identifiziert werden kann und besteht aus dem Klassennamen und einer Tagnummer.

- **Was ist TVL?**
  TVL (tag, value, length) beschreibt die Struktur gemäß der BER kodierte Daten abgelegt werden. Der Code eines Datentyps besteht gemäß BER aus drei 1-Byte-Feldern für Typbezeichnung (2 Bits für Klasse und 6 Bits für die Tag-Nummer), Länge des Datentyps und seinem Wert.

- **Was ist eine Schnittstelle?**
  Die Abstraktion vom Verhalten eines Objektes.

- **In welcher Schicht des OSI Referenzmodells findet die Konvertierung von Daten statt?**
  In der Darstellungsschicht (Schicht 6).

- **Stellen sie die Protokolle der TCP/IP-Welt und des OSI Referenzmodells gegenüber und vergleichen sie beide miteinander!**

OSI	TCP/IP
Hauptkonzepte: Dienst, Schnittstelle, Protokoll.	Keine Unterscheidung.
Wurde vor der Entwicklung der Protokolle entworfen: allgemeinerer Ansatz.	Wurde nach der Entwicklung der Protokolle entworfen: praxisorienterte Einfachheit statt Universalität.
7 Schichten.	5 Schichten.
Top-down-Entwicklung mit wenig Bezug zur Praxis.	Bottom-up-Entwicklung getrieben aus der Praxis.
Redundaten Funktionalität, schlechtes Timing, schlechte Implementierung, schlechte Technologie.	Schwächen insbesondere in der IP-Schicht.
Trennt klar zwischen Dienst, Schnittstelle und Protokoll.	

- **Was macht IP?**
  IP ist ein Protokoll der Schicht 3 und sorgt für eine logisch eindeutige Adressierung von Hosts im Netz. Gleichzeitig wir die logische Netztopologie und das Routing übernommen.

- **Erklären sie TCP!**
  Als Protokoll der Schicht 4 bietet TCP einen verbindungsorientierten Dienst an, bei dem sich die Anwendung um die Fehlerkorrektur oder die Reihenfolge der Pakete keine Gedanken machen muss. Über sog. SAPs (Ports) können verschiedene Dienste auf einem Rechner eindeutig angesprochen werden.

- **Nennen sie Schicht 2 Protokolle aus der Internet-Welt!**
  PPP und SLIP.

- **Was ist ODP?**
  Mit dem Open Distributed Processing sollte eine Standardisierung der Technologie verteilter Systeme stattfinden. Beteiligt waren die ISO und die ITU-T sowie diverse nationale Standardisierungsgremien. Ziel war die Schaffung eines Referenzmodels, das als generisches Framework für die Standardisierung dienen sollte, und Standards für Komponenten und Beschreibungshilfsmittel typischer Dienste und FDTs. Da bisherige Standardisierungsbemühungen der OSI nicht zufriedenstellend waren, wurde diese Eigeninitiative gegründet.
  Die Struktur des Referenzmodells teilte sich in Umfang und Überblick, Deskriptives Modell, Präskriptives Modell und Architekturelle Semantik auf. Die Modellierung basiert dabei auf einem objektorientierten Ansatz, dem ein verteiltes System mit einer Menge kooperierender Objekte zugrunde liegt. Diese Objekte interagieren über Nachrichten und Schnittstellen, wobei verschiedene Interaktionsformen unterstützt werden.
  Ein wichtiger Bestandteil waren die Transparenzfunktionen, die einen einfacheren Umgang mit verteilten Systemen ermöglichen sollten.

- **Vergleichen sie ODP mit OSI!**
  Nach [Spaniol1991] ist OSI ein Vorgänger von ODP. In [Popien et al. 1995] wird als ein Hauptunterschied angeführt, dass sich das OSI-Modell in der Anwendungsschicht mit seiner kommunikationsorientierten Sicht vor allem den Anforderungen an die Modellierung verteilter Anwendungen aus der Kommunikationsperspektive nähert. Damit spezifiziert es ausschließlich Verhalten, das mit von außen sichtbarer Kommunikation verbunden ist und nicht mit einer internen Struktur des Systems. Im Gegensatz dazu stellt ODP einen Rahmen für die Standardisierung aller Aspekte der verteilten Verarbeitung zu Verfügung, insbesondere werden sie Syntax und Semantik aller Begriffe für die Spezifikation verteilter Systeme und Anwendungen definiert.
  Das OSI Referenzmodell beschäftig sich nicht mit dem internen Design von Systemen, sondern lediglich mit der „äusseren" Kommunikation, die in der ODP-Sichtweise durchaus eingesetzt werden kann.

- **Was gibt es für kryptographische Methoden?**
  Es gibt die Verschlüsselung (symmetrisch und asymmetrisch) und die Signatur (mittels Hash-Funktionen). Mittels dieser beiden Methoden lassen sich alle technischen Sicherheitsaspekte eines verteilten Systems abdecken.

- **Welche Sicherheitsaspekte gibt es, welche Anforderungen bestehen?**
  Im Allgemeinen werden 5 Aufgabenbereiche für Sicherheit identifiziert:

  1. Authorisierung (mittels (a)symmetrischer Verschlüsselung)
  2. Authentisierung (mittels (a)symmetrischer Verschlüsselung)
  3. Vertraulichkeit (mittels (a)symmetrischer Verschlüsselung)
  4. Integrität (mittels (a)symmetrischer Verschlüsselung und Signatur)
  5. Nachweisbarkeit (mittels (a)symmetrischer Verschlüsselung und Signatur)

- **Erklären sie die Funktionsweise von Kerberos!**
  Kerberos ist ein Schema zu Authentisierung von Benutzern in einem verteilten System. Ziel bei der Entwicklung von Kerberos war die Entwicklung eines verteilten Sicherheitsdienstes. Dabei sollten Sicherheit, Zuverlässigkeit, Transparenz und Skalierbarkeit gewährleistet sein. Auf Basis des Needham-Schroeder-Protokolls wurde mit Kerberos ein Authentisierungssystem erfunden, das zwischen Kerberos-Akteuren und Kerberos-Elementen unterscheidet. Ziel war der single-logon, so dass zwei Dienste eingesetzt wurden: der authentication service und der ticket granting service.
  Wichtig ist, dass in dem Protokoll niemals das Passwort über das Netz übertragen wird. Im Prinzip lässt sich der Client seine Identität vom AS bestätigen (mittels eines Challenge-Response-Verfahren) und wendet sich dann an den Ticket-Granting-Server, der ihm daraufhin ein „Ticket" für den Server, der angesprochen werden soll, ausstellt, das Benutzer ID und Gruppenzugehörigkeiten enthält. Mit diesem Ticket kann der Client dann auf den Server zugreifen, wobei der Server selbst in seiner ACL prüft, welche Rechte der Benutzer beim Zugriff hat.

???

- **Was für Konzepte beinhaltet Kerberos?**
  Das Konzept hinter Kerberos ist die Authentifizierung eines Clients gegenüber den Servern. Dazu authentifiziert sich der Client gegenüber dem AS und erbittet von ihm ein Ticket für den TGS. Dieser stellt ihm ein Ticket und ein PAC für den gewünschten Server aus, der die ID und Gruppenzugehörigkeiten enthält. Damit ist der Benutzer authentifiziert und der Server kann auf Basis dieser Informationen und seiner lokalen ACL entscheiden, welche Rechte er dem Client gewährt.
  Damit der Benutzer nicht bei jeder Aktion sein Passwort eingeben muss, bekommt der Client ein Ticket-gewährendes Ticket ausgestellt, mit dem er Tickets „nachbestellen" kann – ohne das Passwort wieder neu einzugeben: Er meldet sich beim AS an und erhält ein Ticket für den TGS (mit Verfalsdatum), bei dem er sich dann ein Ticket für den entsprechenden Server holen kann.

- **Wie werden Replay-Attacken in Kerberos verhindert?**
  Durch Zeitstempel, Verfallsdaten, Sitzungsschlüssel und Nounces.

Um Änderungen und Verfälschungen zu verhindern wird das Ticket verschlüsselt und enthält neben der Server- und Client-ID auch die Netzwerk-Adresse des Clients, was aber nur einen schwachen Schutz darstellt.

- **Wie funktioniert die Schlüsselverteilung bei Kerberos?**
  Der Authentifizierungs-Server hat alle ihm bekannten Passwörter in einer zentralen Datenbank gespeichert. Zusätzlich besitzt der AS für jeden Server einen eindeutigen geheimen Schlüssel. Die Verteilung der Schlüssel erfolgt entweder physisch oder sonst irgendwie geschützt – Vorgaben werden hierzu keine gemacht.
  Um Replay-Attacken zu verhindern und die Kommunikation sicherer zu machen werden Sitzungsschlüssel eingesetzt: Nach der Authentifizierung beim AS erhält C einen Sitzungsschlüssel $K_{C,TGS}$ im Ticket mitgeliefert, mit dem fortan die Kommunikation mit dem TGS verschlüsselt wird. Durch Einsatz eines Sitzungsschlüssels kann jetzt auch der Server durch Hochzählen eines Nounces seine Identität beweisen.

- **Wie ist bei RPC Sicherheit gewährleistet?**
  Beim RPC werden verschiedene Schutzklassen unterschieden, je nachdem, ob eine Authentisierung nur einmal zu Beginn, bei jedem Aufruf oder bei jedem Übertragungspaket stattfinden soll. Zusätzlich kann die Übertragung verschlüsselt stattfinden sowie die Integrität der Nachricht durch Signaturen gewährleistet werden. All diese Funktionen sind per API wählbar. Zur Auswahl stehen:
  - AUTH_NULL: Keine Authentifizierungsmechanismen
  - AUTH_UNIX: Es wird lediglich die Unix-Benutzerkennung mit überragen
  - AUTH_SHORT: Es wird eine Kennung für alle weiteren Aufrufe mitübertragen
  - AUTH_DES: Die Authentifizierungsinformationen werden zusätzlich verschlüsselt.

- **Wo werden beim RPC die Daten konvertiert?**
  Die Konvertierung der Daten übernimmt beim RPC der Stub. Diese Aufgabe wird auch „Zusammenstellen der Nachricht" oder einfach „Marshalling" genannt. Weitere Aufgaben von ihm sind das Senden und Überwachen der korrekten Übertragung der Nachricht.

- **Wo sitzt das Thread-Package bei DCE?**
  Das Thread-Package bei DCE sitzt direkt auf dem Betriebssystem auf.

- **Wie ist die Verteilung bei Linda, DACNOS und OSF DCE organisiert?**
  Bei Linda steht ein sog. Object Space zur Verfügung, über den Prozesse Objekte austauschen und gemeinsam bearbeiten könen. Bei DACNOS stellt das Object-Sharing einen ähnlichen Mechanismus zur Verfügung, während bei OSF DCE kein ähnliches Konstrukt vorgesehen ist.
  Dafür ist ein verteiltes Dateisystem ein inhärenter Bestandteil von DCE, während etwas Vergleichbares bei DACNOS oder Linda nicht vorgesehen

ist.
Man könnte schliessen, dass DACNOS und Linda die Verteilung von Daten über Objekt-Räume organisieren, während bei DCE die Verteilung auf einem verteilten Dateisystem beruht.

- **Ist RSC (DACNOS) objektorientiert?**
  Nein, DACNOS ist aber objektbasiert, so stellt der RSC eine objektbasierte Programmierschnittstelle zur Verfügung. Die Basis des Client-Server-Betriebs stellt das Object-Sharing dar, die Grundlage für die Kooperation. RSC-Objekte präsentieren sich als typische BS-Konstrukte, denen Ereignisse zugeordnet und spezifiziert werden können und zu Ereignislisten zusammengefasst werden können. Dabei werden 7 Objekttypen unterschieden: PORT, CARRIER, WINDOW, LOCK, NOTICE, EVENT-LIST, PROCESS.

- **Welche Namens-/Verzeichnisdienste kennen sie?**
  DNS, X.500, LDAP, Mirosoft Active Directory Service, Novell Directory Service. Während Verzeichnisdienste, die Namen attribut-basiert auflösen, als „gelbe Seiten" bezeichnet werden, heissen konventionelle Namensdienste, die lediglich Namen auflösen, „weisse Seiten".

- **Welchen Hauptanforderungen müssen Verzeichnisdienste genügen?**
  Sie müssen eine beliebige Anzahl von Namen verwalten können, eine lange Lebenszeit besitzen und hochverfügbar sein. Fehler müssen isoliert und Misstrauen toleriert werden können.

- **Worin unterscheiden sich X.500 und DNS?**
  Während DNS streng hierarchisch gegliedert ist, kann bei X.500 eine Adressierung sowohl über Attribute als auch hierarchisch stattfinden.

- **Erläutern sie X.500 und seine Bestandteile!**
  X.500 ist ein Standard der ITU und der ISO. Ziel war die Entwicklung eines Dienstes für den Zugriff auf Informationen über „Entitäten der realen Welt". Genausogut kann es auch für Hard- oder Software-Dienste eingesetzt werden. Es stellt die Basis für LDAP dar.
  Die Daten auf X.500-Servern werden in einer Baum-Struktur, dem **Directory Information Tree** (**DIT**) abgelegt, die aus benamten Knoten besteht, die zusätzlich noch Attribut-Eigenschaften speichern. Die gesamte Struktur, die den Baum und die mit den Knoten assoziierten Attributen umfasst heisst **Directory Information Base** (**DIB**). Es besteht die Intention eine globale DIB zu bilden, sodass alle lokalen, individuellen DIBs Teil von dieser sind. Fragt ein Client nach einer Information auf einem Server nach, und diese kann nicht gefunden werden, so werden entweder andere Server befragt oder der Client weitergeleitet.
  Die Terminologie von X.500 spricht von **Directory Service Agents** (**DSU**), die Server, und den **Directory User Agents** (**DUA**), den Clients. Jeder

Eintrag in der DIB besteht aus einem Namen und einer Menge von Attributen und korrespondiert mit einem Pfad durch den DIT vom Wurzelknoten des Baums an. Es sind sowohl absolute als auch relative Pfadnamen erlaubt. Ein Attribut besteht aus einem Typ und einem oder mehreren Werten. Neue Typbezeichnungen können definiert werden, wenn sie noch nicht vorhanden sind und beinhaltet eine Bezeichnung des Typs und eine Syntax-Definition in ASN.1 für alle erlaubten, obligaten und optionalen Werte dieses Typs.
Die Einträge einer DIB ähneln einer Objekt-Klassen-Struktur, wie sie in objektorientierten Programmiersprachen Verwendung findet. Jeder Eintrag hat ein `objectClass`-Attribut, das die Klasse(n) des Objektes beschreibt. Bis auf die **topClass** müssen alle Objekte ein `objectClass`-Attribut führen, das die Namen ein oder mehrerer Klassen haben muss. Bei mehreren Klassen erbt das Objekt die obligaten und optionalen Attribute aller dieser Klassen.
Die Einordnung in dem DIT ist determiniert durch die Auwahl eines oder mehrerer Attribute als **distinguished attributes**. Demgemäß werden Attribute zum Beschreiben eines Objektes als **Distinguished Name** (**DN**) bezeichnet. Folgende Operationen sind möglich:
`read`: auf relativen oder absoluten Namen, liefert die Attribute eines Eintrags zurück.
`search`: Attributbasierte Suche mit einem **Base Name** und Filterausdrücken.
Als Standard schreibt X.500 keine Implementierung vor.

- **Was sind X.500 Standard-Attribute?**
  Country (`c=de`), Organisation (`o=uni`), Organisational Unit (`ou=uni-ffm`), Common Name (`cn=Michael Jaeger`).

- **Erläutern sie LDAP!**
  Die Standard-Schnittstelle zum Zugriff auf X.500 beinhaltet obere Schichten des ISO Protokoll-Stacks. LDAP ist nun ein leichtgewichtiger Ansatz zum Zugriff auf X.500 über TCP/IP. Ausserdem vereinfacht LDAP den Zugriff auf X.500 über eine relativ einfache API und die ASN.1-Kodierung wird zu Gunsten einer textuellen aufgegeben.
  Obwohl LDAP auf X.500 aufbaut wird diese nicht unbedingt benötigt. Eine Implementierung von LDAP kann durchaus auch auf anderen Verzeichnis-Diensten aufbauen wie z.B. Microsoft ADS.
  LDAP liefert einen weitläufig akzeptierten Standard für den Zugriff vor allem auf Intranet-Dienste, wobei der Zugriff auf die Verzeichnis-Daten durch Authentikation gesichert ist.

- **Wie werden X.500-Namen erzeugt?**
  Über Attribute.

- **Welche Schwierigkeiten würden sie bei einer Implementierung von X.500 auf einer relationalen Datenbank erwarten?**

Ein X.500-Verzeichnis ist dynamisch: Neue Attribute können schnell angelegt werden, was bei einer relationalen Datenbank gleich eine komplette Änderung des Tabellendesigns nach sich ziehen würde. Ein eindeutiger Schlüssel liesse sich auch schwer finden und müsste erzeugt werden, hier kann auch nicht der distinguished name verwendet werden, da sich dieser aus den Knoten im X.500-Baum von der Wurzel bis zu dem Element zusammensetzt und je nach Position im Baum unterschiedlich lang ist und aus unterschiedlich vielen Knoten zusammengesetzt ist.
Die inhärent objektorientierte Basis von X.500 liesse sich am besten in einer objektorientierten Datenbank (OODBMS) abbilden.

- **Was ist der Unterschied zwischen einem Objekt und einer Funktion?**
  Ein Objekt besitzt neben Methoden auch Attribute und somit einen „Zustand", der durch diese Attribute definiert ist.

- **Wie sind Internet-Namen aufgebaut?**
  Internet-Namen sind streng hierarchisch aufgebaut: von rechts nach links beginnt ein Internet-Name mit dem Namen der Top-Level-Domain gefolgt von einem Punkt und der Second-Level-Domain. Ab da können jeweils durch Punkte getrennt beliebig viele Sub-Domain-Namen kommen, allerdings muss der letzte Name ein Host-Name sein. Beispiel: `www.cs.uni-frankfurt.de` mit TLD `de`, SLD `uni-frankfurt`, Sub-Domain `cs` und Hostname `www`.

- **Wie funktioniert ARP?**
  Mittels des Address Resolution Protocols (ARP) versucht ein Host eine logische Host-Adresse (z.B. IP-Adresse) in eine physische Hardware-Adresse (bei Ethernet die MAC-Adresse der Netzwerkkarte) aufzulösen. Dabei wird im lokalen Netz ein Broadcast mit der logischen Adresse geschickt, auf die der Rechner mit seiner physischen Adresse antwortet, der diese logische Adresse besitzt. Ist die Adresse nicht im lokalen Netzwerk vorhanden, so antwortet entweder der Router, der dieses Netzwerk bedient mit seiner MAC-Adresse oder der Client bemerkt das selbst und schickt das Paket direkt an den Router (heute gängig und einfach handhabbar, da jedem Client neben seiner Netzwerkadresse auch die Netzmaske konfiguriert wird).

- **Was ist eine Internet-Adresse?**
  Ein Kommunikationsendpunkt und eine logische Netzwerkadresse. Zusammen mit der Netzmaske kann so das Netzwerk „berechnet" werden, in dem sich der Rechner aufhält.

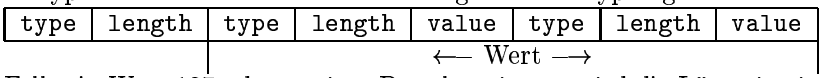
- **Kann ein Rechner mehrere IP-Adressen haben?**
  Ja, z.B. Router.

- **Erläutern sie ASN.1 und BER!**
  Die Abstract Syntax Notation ist ein ISO Standard, der unter anderem eine Darstellung für Daten definiert, die über ein Netzwerk gesendet werden. Der darstellungsspezifische Teil von ASN.1 wird als Basic Encoding

Rules (BER) bezeichnet.
ASN.1 unterstützt das C-Typensystem ohne Funktionszeiger, definiert eine kanonische Zwischenform und benutzt Typ-Tags.

- **Was wird damit tatsächlich umgesetzt?**
  Transparenz der Darstellung von Daten über verschieden Programmiersprachen und Rechner-Plattformen hinweg.

- **Wofür benötigt man ASN.1, erkläre die Tags?**
  Im Grunde genommen in ASN.1 ein Schicht-6-Protokoll, das die Darstellung von Datenelement beschreibt. Dazu wird jedes Datenelement als ein Tripel der Form `<tag, length, value>` dargestellt, wobei `tag` normalerweise ein 8-Bit-Feld ist (ASN.1 lässt aber auch Multi-Byte-Tags zu). `Length` spezifiziert, wieviele Bytes `value` umfasst. Zusammengesetzte Datentypen können durch Verschachtelung von Basistypen gebildet werden:



  Falls ein Wert 127 oder weniger Byte lang ist, so wird die Länge in einem einzigen Byte spezifiziert: eine 32-Bit-Ganzzahl wird also mit einem Byte für `type`, einem Byte für `length` und 4 Bytes für `value` dargestellt. Im Gegensatz dazu wird bei XDR bei Ganzzahlen `type` und `length` weggelassen, was es effizienter macht und den Verarbeitungsaufwand beim (Un-)Marshalling verringert. Zudem bedeutet Längenangabe bei ASN.1, dass der Datenwert höchstwahrscheinlich nicht auf eine natürliche Byte-Grenze (z.B. Wort-Länge) fällt, was die (De-)Kodierung aufwendiger macht. Siehe auch [Peterson et al. 2000, Kapitel 7.1].

- **Wie überträgt man BER?**
  Da BER in der Schicht 6 anzusiedeln ist, sind für die Übertragung der Daten die unteren Schichten zuständig (Schicht 1 bis 4).

- **Erklären sie den Nachteil von TLV!**
  In der Regel kennt die Anwendung bereits die Datentypen, die die Anwendung erwartet, trotzdem wir der Typ und die Länge kodiert. Bei Ganzzahlen ist das in der Regel nicht notwendig. Die zusätzliche Längenangabe spart zwar u.U. Speicherplatz, erfordert jedoch einen Mehraufwand beim (De-)Kodieren, da die übertragenen Datenwerte dann in der Regel nicht den „natürlichen" Byte-Grenzen (Wortlängen) genügen.

- **Kann man ASN.1 als IDL verwenden?**
  Die IDL ist eine Sprache zur Beschreibung von Schnittstellen, während ASN.1 eine Sprache zur Datenrepräsentation ist. Aus diesem Grund kann ASN.1 meiner Ansicht nach nicht als IDL verwendet werden. Andererseits werden in der IDL auch nur Schnittstellen und Datentypen definiert. Würde man eine Schnittstelle als Datentyp interpretieren, so könnte man ASN.1 vielleicht auch als IDL einsetzen. ???

- **Gibt es ASN.1 und BER auch in CORBA?**
  Ja, indirekt durch die IDL.

- **CORBA-IDL: Wozu wird sie verwendet?**
  Die CORBA-IDL beschreibt die Schnittstelle eines Objektes. Diese Beschreibung wird benötigt, damit Stubs und Skeletons für Clients und Server automatisch generiert werden können und so durch das Language Mapping und die Datentransformation eine Programmiersprachen- und Plattformunabhängigkeit erreicht wird. Durch Ablage der Schnittstellendefinitionen im IR wird so auch das dynamische Auffinden von Objekten über ihre Schnittstelle ermöglicht.
  Im Prinzip schafft die IDL eine Transparenz, was die Funktionsweise eines Objektes angeht, sowohl was die Implementierung und die Programmiersprache als auch die Plattform und den Zugriff angeht.

- **Wie funktioniert der RPC?**

  1. Der Stub versucht sich an den Server zu binden, d.h. ihn zu lokalisieren. Dazu schaut er z.B. in einer Datei nach, in der alle Serveradressen gespeichert sind.
  2. Dann verpackt der Stub die Daten (Marshalling) in das NDR-Format.
  3. Die Daten werden über das Transportsystem (Client-Runtime) verschickt.
  4. Auf dem Ziel empfängt das Transportsystem (Server-Runtime) die Daten.
  5. Der Stub packt die Argumente aus.
  6. Der Server bearbeitet die Daten und verschickt sie auf dem gleichen Weg wieder an den Client zurück.

- **Funktioniert der RPC mit ASN.1?**
  Nein, es gibt keinen Stub-Compiler.

- **Erkläre die Konvertierung von Daten anhand von ASN.1!**
  Die Daten werden gemäß der BER und der ASN.1-Beschreibung in ein neutrales Format umgewandelt und dann über das Netzwerk verschickt. Am Ziel werden sie wieder mittels BER und ASN.1 ausgepackt und dann mit den LER in das lokale Datenformat konvertiert.

- **Nachteile von RPC?**
  Synchronität, Blockade (kann mit Threads umgangen werden).

- **Wie findet ein Client einen Server?**
  Entweder mit dem Trader oder der Server wird dem Programm mit übergeben.

- **Gibt es in DCE noch andere Möglichkeiten einen Server zu finden?**
  Explizites Binden, Namensdienst als rudimentärer Trader.

- **Woher kennt der Client beim expliziten Binden den Server?**
  Über Konfigurationsdateien oder Verzeichnisdienste oder beim Internet z.B. über well-known Ports.

- **Kann so ein Port auch mit anderen Anwendungen besetzt sein?**
  Klar.
- **Erläutern sie das Zugriffsverfahren bei Ethernet!**
  CSMA/CD: Jeder Teilnehmer hört permanent das Medium ab, und entscheidet, ob es eine Nachricht animmt oder nicht. Möchte ein Teilnehmer eine Nachricht verschicken, so wartet er so lange, bis das Medium frei ist und verschickt dann seine Nachricht. Kommt es nun zu einer Kollision, so merken das alle beteiligten Sender (CD) und warten eine zufällige Zeit, um dann wieder zu warten, bis das Medium frei ist, um dann die Nachricht zu verschicken.
- **Welche Komponenten gibt es in CORBA?**
  Hufeisenmodell: ORB, OA, Stub (IDL), Skeleton (IDL), DII.
- **Was ist ein Objekt?**
  Ein Objekt besitzt einen Zustand, Verhalten und Identität, wobei der Zustand und die Operationen, die auf einem Objekt möglich sind, eng miteinander zusammenhängen und nach außen hin gekapselt werden.
- **Was ist OMA?**
  Die Object Management Architecture ist ein Standard der OMG und besitzt die Komponenten Application Objects, ORB, Object Services und Common Facilities. Der CORBA-ORB ist hier das zentrale Element.
- **Welche Kommunikationsprotokolle sind in CORBA üblich?**
  GIOP als generisches Protokoll, das auf verschiedenen Transportprotokollen implementiert werden kann. IIOP als obligatorische Implementierung seit Version 2.0, aber auch andere Implementierungen (ESCIOP) möglich.
- **Wofür braucht man IIOP?**
  Interoperabilität über ein weit verbreitetes Protokoll, vorher Wildwuchs bei Implementierungen auf Transportprotokollen.
- **Wie funktioniert Sicherheit bei CORBA?**
- **Erklären sie die Funktionsweise der IDL!**
  Die IDL ist eine deklarative Schnittstellenbeschreibungssprache – sie hat also keinen funktionalen Teil! Mit Hilfe der IDL wird eine Schnittstelle beschrieben, also der Rückgabewert, Parameter (in, out, inout) und Datentypen. Über das Language Mapping wird vom Stub Compiler ein Stub für den Client und den Server in der gewünschten Programmiersprache erzeugt, der dann in das Programm eingebunden wird. Auf diese Weise kann der Programmierer die Schnittstelle benutzen ohne sich um die Programmiersprache, in der das Server-Programm geschrieben wurde, oder auf welcher Plattform es läuft Gedanken zu machen.
- **Wie werden Stubs und Skeletons erstellt?**
  Die Stubs und Skeletons werden aus der IDL von einem Stub-Compiler erzeugt.

- **Welche Middleware-Konzepte finden sich im OSI-Schichtenmodell wieder?**
  Schicht 6 (Darstellungsschicht) entkoppelt Anwendungen von der Datenrepräsentation.

???

- **Worin unterscheiden sich DCE und CORBA?**
  Zunächsteinmal haben DCE und CORBA viel gemein. Zwar ist DCE prozedural angelegt und CORBA objektorientiert, jedoch wird auch jede objektorientierte Sprache in prozeduralen Code übersetzt, so dass der Unterschied hier nicht fußt. Die Hauptunterschiede sind eher, dass bei DCE der Abstratkionsgrad niedriger liegt als bei CORBA und die Entwurfsziele bei CORBA vor allem Interoperabilität und Integration waren. CORBA und DCE sind sogar so ähnlich, dass DCE-Komponenten durchaus auch als Basis-Komponenten von CORBA eingesetzt werden könnten.

Literatur

[Tanenbaum1995] A. S. Tanenbaum: „Verteilte Betriebssysteme", Prentice Hall 1995.

[Stallings2001] W. Stallings: „Sicherheit im Internet", Addison Wesley 2001.

[JaegerMiddle] M. Jaeger: „Zusammenfassung Middleware SS2001", http://www.in-flux.de/lehre_informatik.html.

[Spaniol1991] O. Spaniol: „ODP (Open Distributed Processing): Yet another Viewpoint", Achener Informatik-Berichte, Nr. 91-21, 1991.

[Popien et al. 1995] Popien, Schürmann, Weiß: „Verteilte Verarbeitung in Offenen Systemen", Teubner Stuttgart, 1995.

[Geihs1995] K. Geihs: „Client/Server-Systeme", 1995.

[Sloman et al. 1989] M. Sloman, J. Kramer: „Verteilte Systeme und Rechnernetze", Prentice Hall 1989.

[Peterson et al. 2000] L. Peterson, B. Davie: „Computernetzwerke", dpunkt.Verlag 2000.

[Gora1998] W. Gora: „ASN.1 – Abstrac Syntax Notation One", Fossil-Verlag GmbH 1998.

[Schneider, Werner 2000] U. Schneider, D. Werner: „Taschenbuch der Informatik", Carl Hanser Verlag 2000.