Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme

mobile and distributed systems group

Verteilte Systeme / Ubiquitous Computing

Skript zur Vorlesung im Sommersemester 2009

Prof. Dr. Claudia Linnhoff-Popien

Unter Mitarbeit von:
Michael Dürr, Tim Furche, Carmen Jaron, Matthias Roeckl und Diana Weiß

Inhaltsverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] http://www.3.ibm.com/pvc/pervasive.html.

[2] www.info.isog.org.

[3] www.netscape.com.

[4] Heinrich Abel. GPS: Global Positioning System - Funktionsweise und mathematische Grundlagen. http://www2.fht-esslingen.de/~abel/gps/Abel-GPS.htm.

[5] Coulouris.

[6] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *the Workschop on The What, Who, Where and How of Context-Awareness as a part of the 2000 Conference on Human Factors in Computing Systems*, April 2000.

[7] Hausmann et al, 2001.

[8] Rupp, Siegmund, and Lautenschlager, 2002.

[9] M. Samulowitz. *Dienstvermittlung in Pervasive Computing Systemen*. PhD thesis, LMU Munich.

[10] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. In *IEEE Personal Communications*, August 2001.

[11] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings Workschop on Mobile Computing Systems and Applications*. IEEE, December 1994.

[12] Wiesmann, 2000.

1

Motivation

►

Inhaltsangabe

Die Vorlesung Verteilte Systeme/Ubiquitous Computing ist eine der Kernvorlesungen des Lehrstuhls für Mobile und Verteilte Systeme. Woraus resultiert die aktuelle wirtschaftliche Bedeutung dieses Themas? Diese Frage wollen wir aus 3 Perspektiven betrachten.

1.1 Historische Entwicklungstendenzen

Von Zentralisierung zu Verteilung

Zur Motivation soll zunächst von Prozessoren und Rechenleistung abstrahiert werden und die Erfindung der Dampfmaschine zwecks Erzeugung von Energie betrachtet werden! Durch James Watt und seine Erfindung der Dampfmaschine wurde Energie im 19. Jahrhundert zunächst dort produziert, wo sie zwecks Erleichterung körperlicher Arbeiten auch genutzt wurde. Konzentrierte Arbeit in großen Fabriken prägte damaliges Leben. Die Weiterentwicklung der Technologie führte zu einem Übergang vom zentralen zum verteilten Arbeiten. 1882 wurde das erste Kraftwerk in Betrieb genommen (Thomas A. Edison, New York). Über Stromkabel wurde der Transport von Energie an verteilte Verbraucher möglich. Diese zweite Phase der Industrialisierung bedingte auch eine Flut neuer möglicher Anwendungen: von der E-Lok bis zum heutigen CD-Spieler. Energie ist heute speicherbar, bis in jeden Haushalt hinein allgegenwärtig und hinter die Anwendung selbst zurückgetreten. In der zweiten Hälfte des 20. Jahrhunderts führte die Entwicklung des Computers zu einer Unterstützung der geistigen Fähigkeiten. Damit wurde eine Speicherung und Verarbeitung von Informationen möglich, was in der Mainframe Ära zunächst zentral an Großrechnern erfolgte.

Technologische Einflussfaktoren

Der Beginn der „Modern Computer Ära“ ist um das Jahr 1945 herum anzusiedeln. Typisch für die Zeit waren (nach heutigen Maßstäben gerechnet) große und teure Computer, aufgebaut in der klassischen Von-Neuman-Architektur. In einem Unternehmen waren nur wenige Computer vorhanden, die vollständig von einander unabhängig arbeiteten (zentraler Ansatz). Aus dieser Zeit stammt eine Grundregel, welche als Grosch's Gesetz bekannt wurde. Sie besagt, dass sich die Rechenleistung einer CPU proportional zum Quadrat ihres Preises verhält. Investiert man also das doppelte in einen Rechner, so kann man nach Grosch's Gesetz das Vierfache an Leistung erwarten. Mit der Entwicklung der Mikroprozessortechnologie verlor das Gesetz von Grosch seine Gültigkeit. Ein genereller Trend zur Abkehr von dem zentralen Ansatz ist seit Mitte der 80er Jahre zu erkennen. Ausgelöst wurde er im Wesentlichen durch vier Entwicklungstendenzen:

1. **Hardware:**
   Im Bereich der Halbleiterchips kam es zu einer Leistungsexplosion. Die Rechenleistung von Mikroprozessoren hat sich im letzten Jahrzehnt ca. alle zwei Jahre verdoppelt, die Kapazität von Halbleiterspeichern alle drei Jahre vervierfacht. Stetig wachsende Leistung bei schrumpfenden Preisen und Abmessungen bildete die Grundlage dafür, dass immer mehr Rechner immer komplexere Software ausführen konnten.

2. **Netze:**
   Die Bereitstellung schneller, lokaler Datennetze bildet die ökonomische Voraussetzung dafür, Personal Computer und Workstations zu verbinden. Die Einführung der Ethernettechnik in den 70er Jahren kann als Wegbereiter für Verteilte Softwaresysteme gesehen werden.

3. **Software:**
   In den letzten vier Jahrzehnten sind auch erhebliche Fortschritte im Bereich der Softwaretechnik zu verzeichnen gewesen. Die Akzeptanz von programmiersprachlichen Konzepten wie Prozedur, Modul und Schnittstelle schuf die Voraussetzungen für die grundlegenden Mechanismen Verteilter Systeme. Konsequenzen waren der RPC (Remote Procedure Call, vgl. Kapitel 3.6) und die objektorientierte Modellierung Verteilter Systeme.

4. **Akzeptanz:**
   Die Abkehr von streng hierarchisch aufgebauten Organisationsformen in Unternehmen führt ganz allgemein zu einer Dezentralisierung und schafft flache Führungsstrukturen.

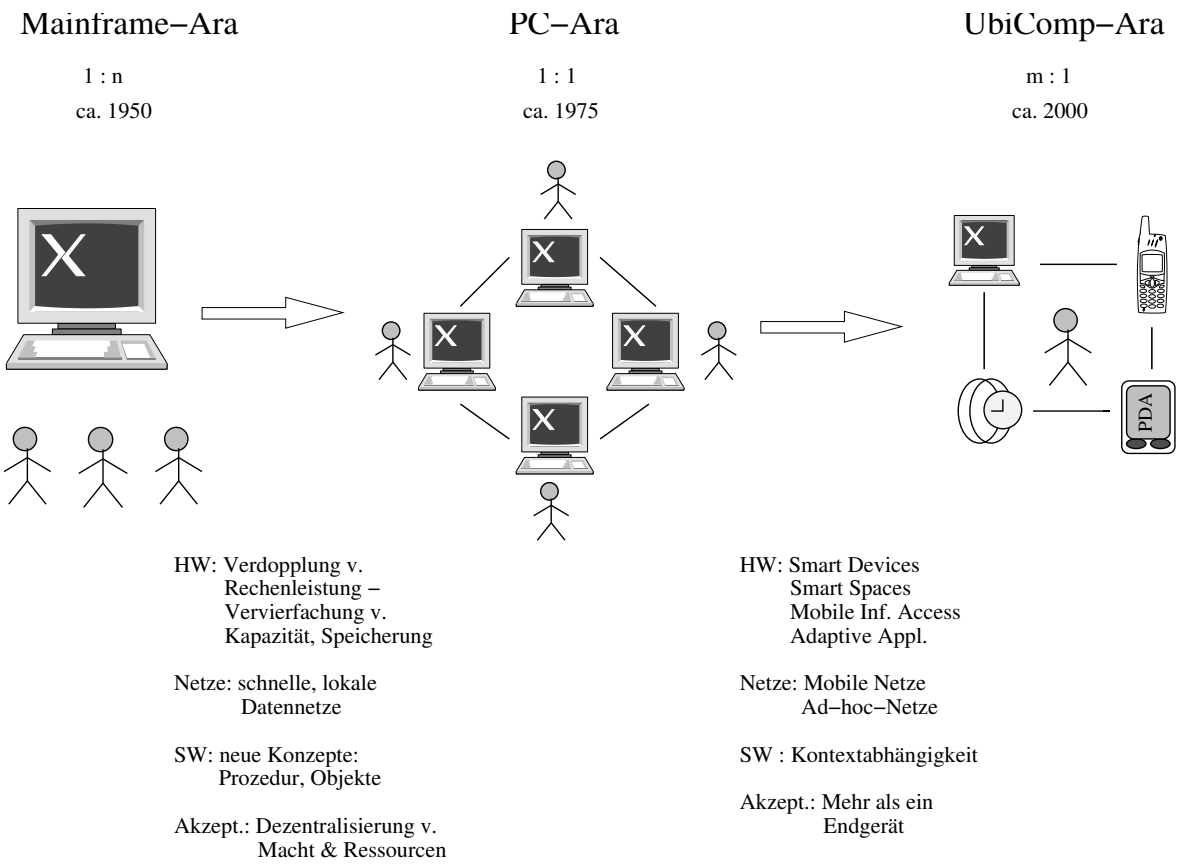


Abbildung 1.1: Modern Computer Ära

Eine Verteilung von Rechenleistung führte zur „PC Ära“. Datenverarbeitung war nun kein Privileg großer Unternehmen mehr, sondern der Mikroprozessor eroberte Büros und Haushalte. Diese Phase der Verteilung bedingte ebenfalls eine Flut neuer Anwendungen in den unterschiedlichsten Bereichen: Telekommunikation, Unterhaltung, Handel, Finanzwesen. Die Verbindung zwischen den Rechnern ist über das Internet realisiert.

	Jahr	Rechneranzahl	
interpoliert	1979	188	
	1989	130.000	
	1999	56.218.000	1 Computer / 100 Einwohner
	2009	15.180.000.000	3 Computer / 1 Einwohner
	2019	2.565.969.000.000	500 Computer / 1 Einwohner

Tabelle 1.1: Computer im Internet: [2]

Die Ubiquitous Computing Ära

Die Dezentralisierung von Rechenleistung soll nun fortgeführt werden. Nach einer $1 : n$ - Relation in der Mainframe Ära (auf einen Rechner kommen n Nutzer) und einer $1 : 1$ - Relation in der PC-Ära ermöglicht die immer billigere Hardware die Ausstattung eines Nutzer mit vielen Endgeräten:

- Laptop
- PDA
- Handy, Digitalkamera, Videokamera
- Wearable Devices
- Embedded Devices: Waschmaschine, HiFi-System, ...

Mit der Ubiquitous Computing Ära wird der Computer omnipräsent, d.h. Teil unseres Alltags zur Ausführung vieler privater und geschäftlicher Aufgaben.

1.2 Probleme der Verteilten Systemen

Klassische Szenerien der verteilten Systemen zeichnen sich durch eine sogenannte **Reaktivität** aus, d.h. der Nutzer (Anwender, User) des Systems **agiert** und das System reagiert. Agieren geschieht durch **Auslösen eines Ereignises**, z.B. Starten eines Programms oder Aktivieren eines Dienstes bzw. einer Komponente.
Solche reaktiven Dienste sind mittlerweile durch die Verteilung hinreichend komplex und bedingen eine Vielzahl von Problemen der Informatik.
Beispiel 1: Positionsbestimmung mittels GPS-Satellitennavigation
Problem: Uhrensynchronisation, d.h. weichen die Uhrzeiten zu sehr voneinander ab, ist Δt unbrauchbar.

Bsp 2: Handynavigation
Schritt 1: Selektion eines Point of Interest (PoI), z.B. Restaurant mit italienischer Küche in max. 1km Entfernung.
Schritt 2: Navigation zu diesem PoI über das Handydisplay
Probleme:

- beschränkte Prozessor/Speicherkapazität auf dem Handy

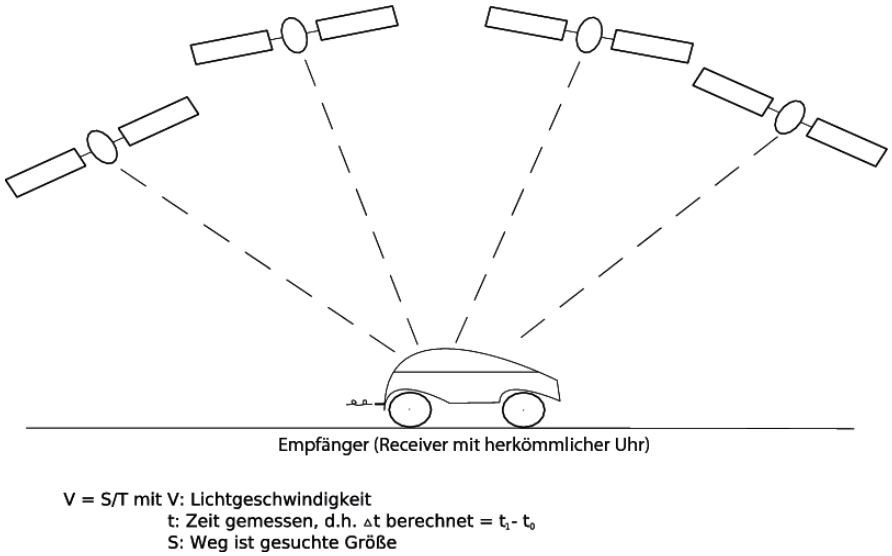


Abbildung 1.2: Positionsbestimmung mittels GPS

- Integration eines GPS-Receivers
- Laden des Kartenausschnitts von einem Server
- d.h. Outsourcen von Funktionalität
- Content Providing nötig
- Datenbanken für Kinos, Restaurants, etc. auswerten
- Navigation: Karte laden, interpretieren und Route berechnen
- d.h. große Heterogenität verschiedener Provider, die zusammenspielen müssen

Welche Bedeutung kommt derartigen Diensten zu?

1.3 Marktanalyse

Schauen wir uns dazu zunächst die gegenwärtige Situation der Informations- und Kommunikationsindustrie (ITK) an:

Laut Statistik des European Information Technology Observatory (EITO) wuchs der ITK-Markt in Europa in den letzten Jahren kontinuierlich. In Deutschland in 2004 um 2.6%, 2005 um 2.3% und 2006 um 1.5%. Nach den neuesten Zahlen wird für Deutschland in 2007 ein Wachstum von 1.6% und 2.0% für 2008 prognostiziert, sowie ein gesamteuropäisches Wachstum von 2.9% bei einem Marktwert von 668 Mrd. Euro.
Damit wächst der westeuropäische Markt für Informationstechnik und Telekommunikation schneller als die westeuropäische Gesamtwirtschaft.

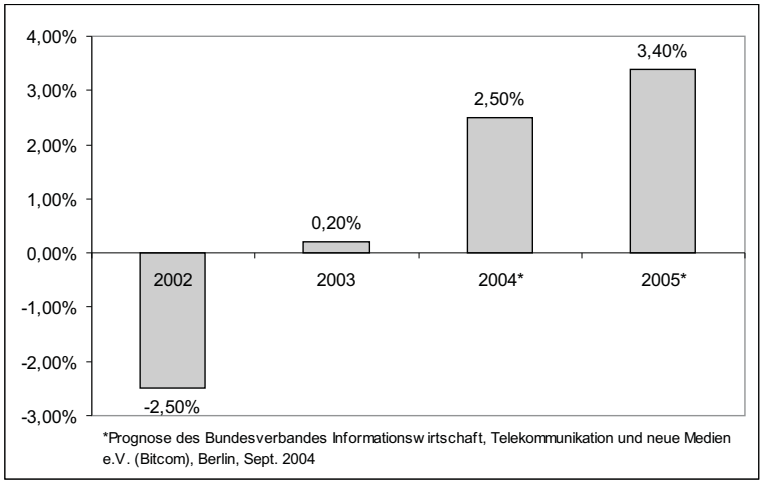


Abbildung 1.3: Wachstum der deutschen ITK in % des Vorjahres

Woher kommt aber dieses Umsatzwachstum im Einzelnen?
Laut Bitkom (2005) sollten Hardware (+1,9%) und Übertragungsdienste der Netzbetreiber (+3,2%) in 2005 nur relativ moderat wachsen. Dafür gaben IT-Services (+4,4%) und Software (+5,5%) dem Markt entscheidende Impulse.
Diese Tendenz wird auch aus einer anderen Betrachtungsweise sichtbar.
Betrachtet man das Umsatzwachstum der Netzbetreiber, so war in 2004 mit 3,2% und 60,9 Mrd. Euro zu rechnen. Diese gliederten sich in 35,3 Mrd. Euro für Festnetzdienste (25,5 Mrd. Euro der Dt. Telekom, 9,8 Mrd. Euro der neuen Wettbewerber) und 27,6 Mrd. Euro für Mobilfunkdienste (8,7 Mrd. Euro Telekom, 18,9 Mrd. Euro Neue Wettbewerber) auf.
Dabei wurden bei den Mobilfunkdiensten 81% des Umsatzes mit Sprachdiensten und 19% mit Datendiensten wie SMS und anderem gemacht. Und: Der prozentuale Anteil der Datendienste am Gesamtumsatz ist steigend.
Auch bei den Festnetzdiensten ist diese Tendenz zu beobachten: er Umsatz mit Sprachdiensten im Festnetz betrug 2003 noch 20,6 Mrd. Euro, für 2004 wurden 20,3 Mrd. Euro prognostiziert, für 2005 nur 19,9 Mrd.

Fazit:

- Beim Umsatzwachstum ist eine Gewichtung von der Hardware, d.h. den Engeräten (+1,9%) über die Netzübertragungsdienste (+3,2%) hin zu Software und Anwendungsdiensten (+5,5% bzw. +4,4%) zu beobachten.

- Der Umsatz mit Sprachdiensten nimmt ab - sogar bei einem Umsatzwachstum der Übertragungsdienste. Damit ist der Wachstumsmotor bei den Diensten in der Datenübertragung zu sehen.

Diese Beobachtungen wollen wir wissenschaftlich aus Sicht des Mobile Commerce einordnen.

1.4 Betriebswirtschaftliche Betrachtung des Mobile Commerce

Das Global Mobile Commerce Forum definiert Mobile Commerce offiziell als “The delivery of electronic commerce capabilities directly into the consumer’s device, anywhere, anytime via wireless networks.” (Mobile Internet, dpunkt, 2001)
Damit sind wir bei Tanenbaums Vision der Millionen mobiler Objekte, die jederzeit und von jedem Ort aus erreichbar sind.
D.h. der Anwender erhält standortunabhängig mittels drahtloser Technologien Zugang zum E-Commerce. Dabei werden sowohl B2B- als auch B2C-Geschäftsbeziehungen involviert, wobei B2B für Business-to-Business und B2C für Business-to-Consumer steht. Klassische Anwendungen für B2C sind Finanzdienstleistungen und Shopping, Beispiele für das B2B sind Fuhrpark- und Kundenmanagement.
Das Potential für M-Commerce resultiert primär aus der Verbreitung des Mobiltelefons. Waren Mobilfunknetze nach dem GSM-Standard ursprünglich für die Sprachübertragung konzipiert, so stoßen diese mit **Datenübertragungsraten** von 9,6 bis 14,4 Kbit/s bei Datenübertragungen schnell an ihre Grenzen. Mit der Einführung neuer Übertragungstechniken basierend auf der Paketübermittlung (GPRS, HSDPA) ließen sich Leistungssteigerungen auf wesentlich höhere Übertragungsraten realisieren. Die Entwicklung geht weiterhin zu noch schnelleren Verbindungstechniken (WiMAX, HSOPA, LTE).
Die **Attraktivität** der neuen paketvermittelten Techniken resultiert auch daraus, dass **Anwender immer online** sein können, **da nur die Menge der übertragenen Daten in Rechnung gestellt wird** und der langsame Verbindungsaufbau entfällt.
Aus wirtschaftlicher Sicht führt diese Entwicklung zu zwei Tendenzen:

– Es können **neue mobile Dienste** angeboten werden, multimediale Inhalte übertragen und mobile Portale betrieben werden

– Es entstehen neue Akteure wie Internet-Portal-Betreiber, Handelsfirmen und Online-Medienunternehmen

Weiterhin wird sich die wirtschaftliche Bedeutung der Wertschöpfungssegmente verändern: Während sich ursprünglich die Mehrwertleistung auf die Sprachübertragung konzentrierte, verschiebt sich die ökonomische Bedeutung hin zu Anwendungen, Informationen und Inhalten sowie der Aggregation von Diensten in mobilen Portalen.
Betrachtet man die Wertschöpfungskette hinsichtlich der wichtigsten Akteure am Markt, so ist eine Neupositionierung entlang dieser Wertschöpfungskette mit Verschiebung der Schwerpunkte nach hinten hin notwendig.

Abbildung 1.4: Wertschöpfungskette

Da letztendlich **kein Unternehmen** über ausreichende Ressourcen oder Kenntnisse verfügt, um erfolgreich die **komplette Wertschöpfungskette besetzen zu können**, müssen sich strategische **Allianzen** bilden. Letztendlich ist es nötig für einen kommerziellen Erfolg der mobilen Informationsgesellschaft, **attraktive Dienste und Anwendungen** zu entwickeln, die eine Nachfrage am Markt generieren und Marktpotenziale ausschöpfen.

Zurzeit sieht man das **größte Potenzial** für allgemeine Akzeptanz und wirtschaftlichen Erfolg in sogenannten **Context Aware Services (situationsbezogene Dienste)**, weloche eine Verallgemeinerung der **Location Based Services** sind. Damit entstehen Dienste und Informationen, die der mobile Nutzer personalisiert auf seinen aktuellen Standort und seine aktuelle Situation bezogen bereitgestellt bekommt. Diese haben für ihn in der Regel eine besondere Relevanz und Qualität.
Aktuell zu lösende Fragestellungen und **Probleme** sind zur Zeit in folgenden Punkten zu sehen:

- **Mangelnde Nutzerfreundlichkeit und Nutzeroberflächen** und dürftige Ergonomie existierender Lösungen sind Hindernisse bei einer schnellen Verbreitung, anstrebenswert sind hohe Nutzbarkeit, Navigation, Zielführung und Layout.

- Die **Genauigkeit der Standortangabe** ist momentan zu gering. Die Präzision der Lokalisierung mobiler Teilnehmer muss eine bessere Genauigkeit erreichen, welche die exakte Positionierung ermöglicht.

- Für die Verbreitung und Akzeptanz von M-Commerce-Anwendungen sind die **Sprach- und Volumengebühren** über die Luftschnittstelle momentan noch ein schwerwiegender Störfaktor. Hier sollten Netzanbieter kostengünstige Modelle (z.B. auch für das Verschicken einer SMS) entwickeln und sich mit Inhalteanbietern über die Aufteilung von Gebühren einigen.

1.5 Visionen des Ubiquitous Computing

Szenarien des Ubiquitous Computing zeichnen sich durch eine zusätzliche **Proaktivität** aus. Für den Nutzer entsteht der Eindruck: Das System denkt mit!
In Wirklichkeit spezifiziert er jedoch seine Anforderungen formal und programmiert das System, auf kontextuelle Ereignisse (gewisse Schwellwerte von Umweltbedingungen) geeignet zu reagieren.
In Anlehnung an [10] wollen wir im Folgenden zwei Beispiele betrachten, die die Möglichkeiten eines Ubiquitous Computing Systems (UbiComp-System) illustrieren:

Szenario 1: Tom wartet am Gate 23 des Münchener Flughafens auf seinen Flug. Er hat eine Reihe großer Dokumente bearbeitet, die er über eine Mobilverbindung per E-Mail noch verschicken möchte. Leider ist die Bandbreite gerade extrem schlecht, da viele Passagiere die Wartezeit nutzen, um noch im Web zu surfen.
Nun kommt das Pervasive Computing System Aura ins Spiel. Es erkennt, dass die verfügbare Bandbreite nicht ausreichen wird, bevor der Flieger geht. Nach einer Kommunikation mit dem Flughafenwetterservice und einer Auswertung des aktuellen Flugplans entdeckt Aura, dass die Bandbreite an Gate 15 exzellent ist und dass an Gate 15 und dessen Umgebung in der nächsten halben Stunde keine Flugzeuge starten oder laden werden. Ein Dialogfenster erscheint auf Toms Bildschirm, in dem ihm vorgeschlagen wird, zum nur drei Minuten entfernten Gate 15 zu gehen. Außerdem erfolgt die Anfrage nach einer Priorisierung der E-Mails, so dass die wichtigsten Daten zuerst verschickt werden können. Tom begibt sich zu Gate 15, vertreibt sich die Zeit mit einigen NTV-News im Fernsehen bis Aura mitteilt, dass das System fast fertig ist und Tom den Rück-

weg antreten kann. Während der Übertragung der letzten Daten ist er rechtzeitig zum Boarding auf Gate 23.

Szenario 2: Mang bereitet sich in ihrem Büro auf einen Vortrag mit Softwaredemonstration vor. Als es Zeit zum Aufbruch ist, sind die Vorbereitungen noch nicht ganz abgeschlossen.
Aura überträgt die Präsentation vom Laptop auf ihren PDA und ermöglicht damit eine Fertigstellung über Sprachkommandos auf dem Weg in den Konferenzraum. Gleichzeitig wird der Zielprojektor vorgewärmt. Über die Ortsangabe im Kalender und den Campus Location Tracking Service wird gleichzeitig der Weg angezeigt. Mit Betreten des Konferenzraums ist die Präsentation fertig gestellt und wird auf den lokalen Projektionscomputer geladen. Die Präsentation beginnt. Da bemerkt Aura auf einer Folie eine streng vertrauliche Budgetinformation und warnt Mang. Noch rechtzeitig erkennt sie dabei, dass Aura Recht hat und überspringt die Folie. Das Publikum honoriert beeindruckt den Vortrag mit heftigem Applaus.

Aus diesen beiden Szenarien sollen einige für das Ubiquitous Computing benötigte Konzepte abgeleitet werden:

– **Proaktivität**
Bevor die Übertragung von Toms Daten abgeschlossen ist, tritt er bereits den Rückweg an. Bevor eine vertrauliche Information gezeigt wird, erfolgt eine Warnung.

– **Wissenskombination (Combining Knowledge)**
Eine Abschätzung der Bandbreite für die Übertragung wird mit der Auswertung des Flugplans verknüpft. Hierbei kommen zwei ganz verschiedene Ebenen ins Spiel (niedrige für Datenübertragung und Anwendung auf Nutzerebene).

– **Zustandsübertragung**
Ein Pervasive Computing System sollte in der Lage sein, Ausführungszustände über Plattformgrenzen hinweg zu übertragen: vom Laptop auf den PDA, vom PDA auf den Präsentationscomputer.

– **Einbettung**
Alle Alltagsgeräte und -informationen sind digital und über das System verfügbar. Dies ist die Idee des Smart Spaces, in dem reale Welt und Rechnerwelt zusammengewachsen sind: der Online-Kalender, der Campus Location Tracking Service, der entfernt steuerbare Projektor.

Die beiden vorgestellten Szenarien haben auf den ersten Blick etwas noch Entferntes und hinterlassen einen Eindruck von Science-Fiction. Wir wollen den Objekten und ihren Wechselwirkungen jedoch etwas genauer auf den Grund gehen. Die genutzten Hardwaretechnologien existieren heute bereits: Laptops, Projektoren, PDAs, Mobilkommunikation. Gleiches gilt für Softwarekomponenten: Spracherkennung, Online Kalender, Location Tracking.
Der Science-Fiction-Eindruck kommt zustande, da das System extrem viel mehr kann als die Summe der Bestandteile: eine ebenen- und plattformübergreifende, nahtlose Integration von Komponententechnologien in ein ganzheitlich arbeitendes System.

Technische Entwicklungen, die Ubiquitous Computing ermöglichen Innerhalb der Informatik haben sich verschiedene Themenbereiche in Richtung Hardware, Netzwerke und Software entwickelt, die Ubiquitous Computing möglich machen. Drei wesentliche Einflussfaktoren sollen im Folgenden behandelt werden:

1. **Hardware: Smart Devices**
   Ein Smart Device ist allgemein ein Gerät mit eingebettetem Prozessor, Speicher und Netzwerkverbindung. Diese Post-PC-Generation umfasst vielfältige Bausteine, die auf bestimmte Aufgaben spezialisiert sind:

   - Kleidungsstücke, die Körpertemperatur und Blutdruck messen und in bestimmten Situationen einen Arzt verständigen.
   - Brillen, die Orte und Personen wieder erkennen.
   - Papier, das alles geschriebene digitalisiert.
   - Musikanlagen, die bei schlechter Laune passende Musik aussuchen oder Werbung leise drehen.
   - Digitale Hausverwaltungen, die über Daten eines Internetwetterdienstes entscheiden, ob Fenster geschlossen oder die Heizung angemacht wird.

   In diesem Zusammenhang heißt “Smart” nicht intelligent im Sinn von KI (Künstliche Intelligenz), sondern bezieht sich auf die Fähigkeit, digitale Informationen zu verarbeiten und mit anderen Einheiten auszutauschen. In der Regel besitzen solche Einheiten eine Benutzerschnittstelle und Sensoren.

   Probleme der Smart-Devices:

   - Sie müssen sehr robust sein. Selbst bei 99,9% Zuverlässlichkeit gäbe es in einem Haus mit 1000 Smart-Devices einen technischen Defekt pro Tag.
   - Sie sollten per Fernsteuerung gewartet werden können.
   - Batteriebetriebene Geräte sollten mit möglichst geringem Energieverbrauch betrieben werden können. Hier besteht momentan noch eine Schwachstelle. In den letzten Jahrzehnten hat sich die Energiekapazität von Akkumulatoren bezogen auf identisches Gewicht und Größe im Vergleich mit der Entwicklung der Mikroelektronik nur geringfügig erhöht. Anfang der 90er-Jahre wurde mit der kommerziellen Einführung des Lithium-Ionen-Akkus (Li-Ion) die Kapazität der bisher verfügbaren Nickel-Cadmium- (NiCD) und Nickel-Metallhydrid-Akkus (NiMH) in etwa verdoppelt. Seither wurden die bestehenden Akkumulatortechniken nur in kleinen Schritten verbessert und der Lithium-Polymer-Akku (LiPo) eingeführt, der zusammen mit dem Li-Ion-Akku derzeit die höchste alltagstaugliche Kapazität bietet. Etwa seit der Jahrtausendwende werden Akkus entwickelt, die auf dem Metall Zink basieren und im Optimalfall die doppelte bis dreifache Kapazität von Li-Ion-Akkus erreichen sollen. Seit Mitte 2008 sind die ersten Silber-Zink-Akkus kommerziell verfügbar und bieten etwa 40% mehr Kapazität als Li-Ion-Akkus.
   - Alternativ sollten Smart Devices aus der Umgebung Energie beziehen, z.B. über einen Federmechanismus im Schuh, der 6-8 Watt liefern könnte. Dies ist genug, um ein PDA oder Handy zu betreiben.

2. **Netze: Mobile und spontane Vernetzung**
   In der erreichten Entwicklungsstufe des Mobilfunks sind neben der Sprache auch multimediale Daten übertragbar. Dabei ist sowohl der Weitverkehrbereich als auch der Heimbereich abgedeckt. Bekannte Standards sind UMTS, WiMAX und HIPERLAN. Einen flexibleren Ansatz verfolgen die Ad-hoc-Netze. Sie verfügen über keine Basisstation. Geräte mit gleichem Übertragungsstandard konfigurieren sich für eine temporäre Zusammenarbeit, ohne sich a-priori kennen zu müssen. Diese Vernetzung bei Ad-hoc-Netzen ist relativ einfach realisiert. Die Komplexität ist vielmehr in die Endgeräte ausgelagert, da jede Station alle Mechanismen zum Kommunizieren bereitstellen muss, die sonst ggf. von der Basisstation übernommen werden.
   Ein typischer Standard ist Bluetooth. Bluetooth ist ein Ad-hoc-Netz zur spontanen Kommunikation zwischen verschiedenen Endgeräten wie Handys, PDAs und Notebooks. Die Kommunikation erfolgt über drahtlose Piconetze (drahtlose lokale Netze mit geringer Ausdehnung - typischerweise <10 m). Dabei können maximal 8 Bluetooth-Geräte in einem Netz spontan miteinander kommunizieren.
   Bluetooth - Geräte, die aktuell keinem Piconetz angehören, horchen laufend nach anderen Bluetooth-Geräten. Kommen diese genügend nahe, so identifizieren sich die neuen Geräte und werden selbst Mitglied des Pico-Netzes. Dadurch können bei Bedarf auch andere Geräte mit ihnen kommunizieren. Somit benötigen Bluetoothnetze keine separate Infrastruktur. Bluetooth ist gut geeignet, um z.B. UMTS-Netze zu ergänzen. So kann z.B. ein Foto von einer Digitalkamera über eine Bluetooth-Schnittstelle auf ein UMTS-Handy übertragen und dann weltweit verschickt werden.

3. **Software: Kontextabhängigkeit**
   Im klassischen lokalen oder Client/Server-System werden explizite Eingaben und Prozeduraufrufe gemacht, in deren Ergebnis explizite Angaben oder Ergebnisrückgaben erfolgen.
   In UbiComp-Systemen muss es im Gegensatz dazu möglich sein, auch ohne explizite Anforderung eine Prozedur zu starten. Der Impuls kann durch Daten erfolgen, die die Komponente selbst erfasst haben, z.B. vom Ort, von gemeinsamen Wetterdaten, von der Historie von Interaktionen, d.h. von Kontexten.
   Allgemein bezeichnet “Kontext” Informationen, die benutzt werden können, um die Situation einer Entität - d.h. einer Person, eines Orts, eines Objekts o.ä. - zu charakterisieren. Ein System heißt kontextabhängig, wenn es den Kontext verwendet, um den Benutzer mit relevanten Informationen und Diensten zu versorgen. Dabei ist die Relevanz von der Aufgabe des Benutzers abhängig.

1.6 Pervasive vs. Ubiquitous Computing

Der Begriff des **Ubiquitous Computing** (*ubiquitous: überall verbreitet, allgegenwärtig*) geht auf den Wissenschaftler Mark Weiser zurück. Obwohl Zitate bereits mit dem Jahr 1988 verbunden sind, ist die bahnbrechende Arbeit in Mark Weisers „The Computer for the 21st Century“ zu sehen, die im September 1991 in „Scientific American“ erschien. Hier erfolgte eine ausführliche Diskussion von Weisers Visionen. Heute kommt dem Ubiquitous Computing eine große Bedeutung im akademischen Forschungsbereich zu.
Von industrieller Bedeutung ist der von IBM geprägte Begriff des **Pervasive Computing**

(*pervasive: durchdringend, weit verbreitet*), siehe auch „What is pervasive Computing?“ unter [1]. Nach [1] fokussiert das Pervasive Computing eher das e-Business und web-basierte Geschäftsprozesse. Wir wollen jedoch beide Begriffe als Synonym verwenden. Eine ganz neue Qualität von Endgeräten macht den Informationszugang und die Informationsverarbeitung möglich „for everyone from everywhere at any time“. Charakteristiken dieser neuen Ära sollen exemplarisch anhand einiger Aspekte aufgeführt werden:

- Handheld Computer stellen eine zu Intelligenten Netzen komplementäre mobile Nutzerschnittstelle bereit;
- Smart Cards (im Handy, als Kreditkarte, als Magnetkarte für die Türöffnung sind als nicht kopierbare Datenträger Sicherheitstokens für Prozesse in vielfältigen Anwendungsbereichen);
- Set-Top-Boxes - interaktives Fernsehen und Spielkonsolen sind die Schnittstellen zwischen Home Entertainment Systemen und Entertainment Providern;
- Intelligente Steuerungen im vernetzten Haus erlauben den externen Zugang zwecks Bedienung von Heizung, Herd, Kühlschrank, Licht, ...
- Eingebettete Systeme wie z.B. Assistenzsysteme zur Auslösung von Notrufen, Navigationssysteme und Einbauhandys in der Automobilbranche ermöglichen ein ganz neues Potential an Kommunikation, Navigation und Sicherheit;
- Handys ermöglichen von überall her mobilen Zugang zu prinzipiell allen Computersystemen und Informationsdiensten.

2

Endgeräte und Betriebssysteme

►

Inhaltsangabe

Microsoft versteht unter dem Ziel des Pervasive Computing die „Convergence of Computer, Communication, Consumers Electronics, Content and Services“.
In diesem Kapitel wollen wir zunächst relativ losgelöst singuläre Komponenten betrachten, bevor sich das nächste Kapitel dann mit deren Kommunikation beschäftigt.
Komponenten des Pervasive oder Ubiquitous Computing lassen sich nach in drei Klassen einteilen:

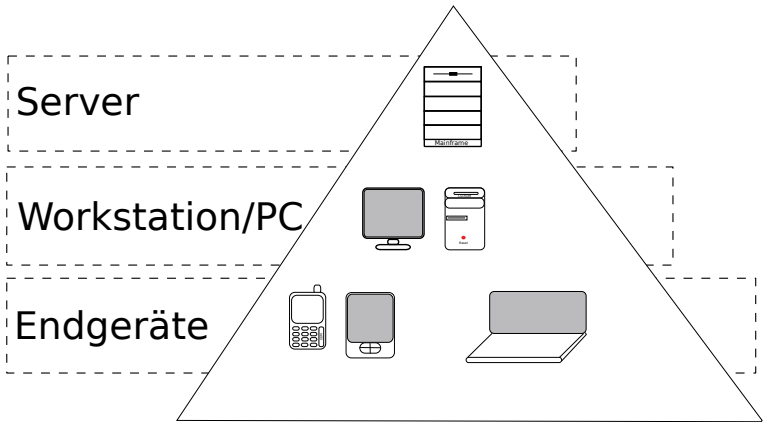


Abbildung 2.1: Komponenten des Ubiquitous Computing

Server

Server werden in Form von Webservern, Enterpriseservern und Mainframes (Großrechnern) verwendet. Diese und viele andere Ausprägungen haben das Ziel, große Informationsmengen zu speichern und zu verarbeiten, in dem große Datenbanken oder große Rechenkapazitäten verwendet werden.
Während die klassischen Großrechner, Workstations und PCs auch autonom und zentral arbeiten können, wollen wir uns im Folgenden mit der beim Ubiquitous Computing hinzukommenden Ebene der Endgeräte beschäftigen.

Workstations

Workstations sind eine optionale mittlere Ebene. Ein Beispiel für ein solches Gerät wäre der klassische PC. Er kann komplexe Informationen verarbeiten und Endgeräte verwalten, z.B. auf Daten vom PDA, Handy,... zugreifen. Sofern die Endgeräte in der Lage sind, direkt mit einem Netzanbieter zu kommunizieren, kann diese mittlere Ebene auch weggelassen werden. Alternativ kann der PC oder die Workstation auch direkt durch ein größeres Endgerät auf dieser Ebene ersetzt werden, z.B. durch eine Set-Top-Box als Gateway zu einem öffentlichen Netz.

Endgeräte

Drei Eigenschaften sind zu erfüllen. Beispiel Taschenrechner:

	Taschenrechner
eigener Prozessor	√
eigener Speicher	√
eigener Netzzugang	-

Endgeräte Sind dabei das Front End der Informationstechnologie, sie dienen der Generierung von Informationen oder sind beim Zugriff auf Infos behilflich. Für den Nutzer sind sie die sichtbare Schnittstelle seiner Kommunikation, in der Regel als kleinere Endgeräte.

Ubiquitäre Endgeräte Ubiquitäre (allgegenwärtige, überall verbreitete) Endgeräte beinhalten optimalerweise drei Klassen von Eigenschaften:

– decentralized: sie sind stark dezentralisiert

– diversified: sie werden diversifiziert, d.h. abwechslungsreich und vielfältig eingesetzt

– simple to use: sie sind für den Nutzer einfach handhabbar

Als historisch gesehen eines der ersten Geräte, das diese Eigenschaft erfüllt - noch lange bevor der Begriff des Ubiquitous Computing geprägt wurde - wird der Taschenrechner angesehen, auch wenn dessen klassische Version noch nicht die Bedingung des Netzzugangs erfüllt. Dennoch existiert er in vielen verschiedenen Formen, wird für vielfältige Zwecke genutzt, ist portabel und intuitiv ohne Handbuch anwendbar. Als Massenprodukt zielt er auf eine breite Nutzergruppe und hat folglich eine weite Verbreitung.
Heutige ubiquitäre Endgeräte sind wesentlich vielfältiger. Sie sollen zunächst in vier Klassen gruppiert werden, die im Folgenden detaillierter behandelt werden:

– Information Access Devices

– Smart Identification

– Embedded Systems

– Entertainment Systems

In der Praxis existieren zahlreiche Geräte, die auch in mehr als eine Klasse eingeordnet werden können. Der Einfachheit halber soll zunächst jedoch eine strikte Abgrenzung vorgenommen werden.

2.1 Information Access Devices

Die Geräte für den Informationszugang werden nochmals in drei Gruppen unterteilt.

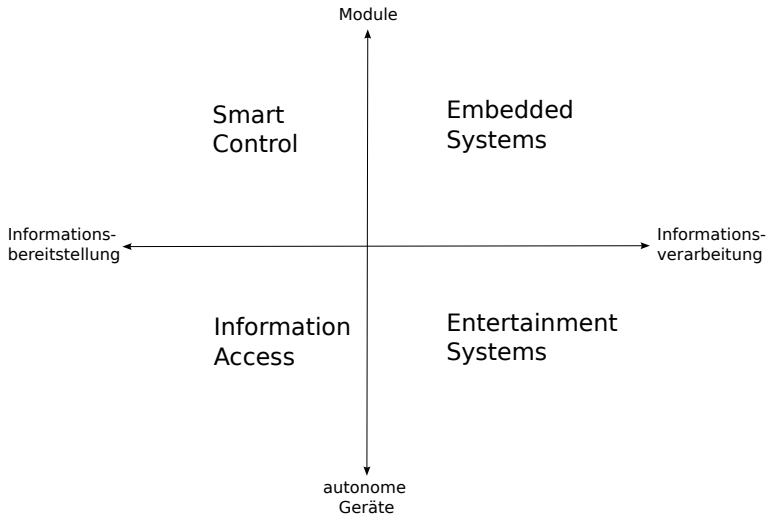


Abbildung 2.2: Einordnung von ubiquitären Endgeräten

2.1.1 Handheld Computers / PDA

Diese Geräte sind sehr klein, d.h. passen in quasi jede Tasche, und sind folglich auch sehr leicht. Die Eingabe erfolgt bei den neueren Geräten mittels Eingabestift über einen Touch Screen. Sofern das Gerät der Organisation persönlicher Daten dient, spricht man auch von einem Organizer oder Personal Digital Assistant (PDA). Traditionellerweise ersetzen PDAs das klassische Adressbuch mit Terminplaner, Papier und Bleistift und vereinigen alle diese Funktionalitäten in einer einzigen, handlichen und mobilen Verpackung. Attraktiv macht diese Geräte, dass sie alle Funktionen zusammenführen und überall und zu jeder Zeit verfügbar machen. Diese Funktionen fasst man auch unter dem Begriff Personal Information Management (PIM) zusammen, wie z.B. Kalender, Notizblock oder Adressbuch. Umfangreichere PIM-Applikationen beinhalten darüber hinaus noch Anwendungen wie Taschenrechner, Uhr, Tabellenkalkulation oder Spiele.
Die erste Generation der PDAs kam Mitte der 80er Jahre unter der Bezeichnung Organizer auf den Markt, in der Regel mit kleiner Tastatur. Der Durchbruch für die zweite Generation gelang Anfang der 90er Jahre. Diese Geräte ließen sich über einen Cradle via serieller Schnittstelle mit einem PC verbinden. Damit waren die Benutzer in der Lage, große Datenmengen am Computer einzugeben und dann zum Gerät zu schicken. Bei Batterieversagen ließen sich mittels des Synchronisationsmechanismus HotSync auch Backups erstellen. Ferner war damit indirekt eine Verbindung zum Internet gegeben.
Mitte der 90er Jahre setzten sich dann noch kleinere und leichtere Geräte durch, die über einen TouchScreen verfügen, über den Graffiti-Schrift eingegeben wird. Über einen Infrarot-Port besteht die Möglichkeit, Daten zwischen verschiedenen Geräten auszutauschen. Dieser Port kann auch für die Kommunikation mit einem Handy genutzt werden. Neuere Geräte zielen darauf ab, Internetzugang ohne zusätzliche Hard- und Software zu ermöglichen oder auf Mobilfunknetze zugreifen zu können.
Die Geräte nach der Jahrtausendwende verfügen über hochauflösende Farbdisplays und schnellere Prozessoren. Langsame Schnittstellentechnologien wie Infrarot (IrDA) wurden durch USB, Bluetooth und WLAN abgelöst, integrierte GPS-Empfänger ermöglichen Location Based

Services, weitere Anwendungen wie E-Mail, Webbrowsing oder IP-Telefonie wurden möglich. Durch Integration von Telefonie GSM und UMTS näherten sich PDAs den Handys an und werden zu Smart Phones, auch immer mehr Multimediafunktionen werden integriert.

Auf diesen Geräten laufen im Wesentlichen fünf alternative Betriebssysteme:

- Symbian OS
- BlackBerry OS
- Mac OS X (in angepasster Form auf dem Apple iPhone
- Windows Mobile (bzw. Windows CE)
- Linux in verschiedenen Modifiaktionen:
  - Android (Google)
  - webOS (Palm)
  - Maemo (Nokia)
  - Mobilinux (MontaVista)
  - Openmoko (Opensource)

Palm OS ist immer noch sehr verbreitet, allerdings mit stark abnehmender Tendenz, und soll hier mit Windows Mobile verglichen werden.
Palm OS wurde entwickelt von Palm Computing, 2002 ausgegliedert in PalmSource. Palm Computing wurde 1995 von U.S. Robotics aufgekauft, letzteres 1997 von 3com übernommen und 2000 wurde das Palm-Segment wieder in die Palm Inc. ausgegliedert. PalmSource wurde 2006 von dem japanischen Unternehmen Access aufgekauft und firmiert nun unter diesem Namen. Mit einem Marktanteil von zeitweise bis zu 70% war es das bekannteste Betriebssystem für Handhelds. Palm OS-basierte Endgeräte wurden gebaut von Palm und anderen Unternehmen wie IBM, Symbol und Handspring, die das Betriebssystem in Lizenz auf ihren eigenen Geräten verkauften. Inzwischen haben so gut wie alle namhaften Hersteller Palm OS den Rücken gekehrt und fast nur noch Palm selbst setzt es ein.
Windows Mobile wurde von Microsoft entwickelt und wird von Herstellern wie Casio, HP, HTC und inzwischen auch von Palm verwendet. Windows CE ist die interne und bis heute fortgeführte Bezeichnung von Microsoft für Windows Mobile. Im Gegensatz zu Palm OS ist die zugrundeliegende Philosophie, dass heutige PCs in Richtung mobiler Computer ergänzt werden sollen. Das bedeutet die Nutzer sollen in der Lage sein, dieselben Dinge auf großen als auch auf kleinen Endgeräten zu erledigen.
Die Meinung der Nutzer geht bezüglich dieser beiden Ansätze auseinander: manche sagen, es sei einfacher, mit Windows Mobile zu arbeiten, da die Nutzerschnittstelle bekannter ist und die Anwendungen ähnlich zu denen des PCs. Andere Nutzer empfinden diesen Ansatz als zu umständlich und halten Palm OS für die Lösung, die besser an das Endgerät angepasst ist.

Technisch resultiert aus den beiden Ansätzen, dass Windows Mobile mehr Prozessor- und Speicherressourcen benötigt als Palm OS. Damit sind Palm OS-Geräte billiger herzustellen und waren daher auch weiter verbreitet. Allerdings ermöglichen die Windows-Geräte oft Multimedia-Applikationen und Computerspiele. Sie haben Funktionen wie ein eingebautes

Bezeichnung	Casio Cassiopeia E-115	Palm IIIc	Palm Treo 750	HTC P3.600 Trinity	HP iPAQ 210
Marktstart	1999	2000	2007	2007	2008
Gewicht	256g	190g	178g	150g	190g
Prozessor	NEC VR4121 133MHz	Motorola Dragonball EZ 20MHz	Samsung SC32442 300MHz	Samsung SC32442 400MHz	Marvell PXA310 624 MHz
Auflösung	240x320 farbig	160x160 erstmals farbig	240x240 farbig $= \frac{2}{3}\cdot$ HTC	240x320 farbig	480x640 farbig
Schnittstellen	Seriell, IrDA	Seriell, IrDA	IrDA, Bluetooth, GSM, UMTS, USB	IrDa, WLAN 802.11b/g, Bluetooth, GSM, UMTS, USB	WLAN 802.11b/g, Bluetooth, USB
Digitalkammera	–	–	ja	ja	
Speicher	16 MB ROM, 32 MB RAM	2 MB ROM, 8 MB RAM	128 MB ROM, 64 MB RAM	128 MB ROM, 64 MB RAM	256 MB ROM 128 MB RAM
Betreibssystem	Windows CE 3.0	Palm OS 3.5	Windows Mobile5.0	Windows Mobile 5.0	Windows Mobile 6.0

Abbildung 2.3: Beispiel für verschiedenen Endgeräte.

Mikrofon zur Sprachaufzeichnung, Kameras, Audio- und Videowiedergabe, außerdem eine große Anzahl an Schnittstellen und ein Farbdisplay, im Gegensatz zu den zunächst meist nur mit monochromem Bildschirm ausgestatteten Palms.
Somit wird der ohnehin notwendige, stärkere Prozessor einerseits genutzt, andererseits motivieren die Zusatzfeatures auch den im Vergleich zum Palm höheren Preis. Für diese Entwicklung wurde auch ein neuer Name geschaffen: Seit Windows CE 3.0 heißen diese Geräte PocketPCs.

2.1.2 Subnotebooks

Bezüglich Größe, Gewicht, Rechenleistung und Funktionalität liegen die sogenannten Subnotebooks zwischen den Handhelds und den normalen Notebooks. Subnotebooks haben die meisten Funktionalitäten normaler Notebooks, allerdings verfügen sie etwas weniger Speicher und Rechenleistung, haben einen kleinen Bildschirm und eine separate Tastatur, über die die Dateneingabe einfacher wird als sie beim Touchscreen noch ist. Größenmäßig passen sie nicht mehr in jede Tasche, sind aber immer noch portabel und leichter als normale Notebooks.
Allerdings ist es auch genau dieser Kompromiss, der ihre Akzeptanz so schmälert: Sie sind nicht so bequem einsteckbar wie Handhelds - haben aber auch nicht die vollständige Funktionalität leistungsfähiger Notebooks. Weiterhin sind sie durch ihren nicht so großen Marktanteil verhältnismäßig teuer, deutlich teurer als größere, leistungsfähigere Notebooks. Die größte Verbreitung fanden sie zeitweise bei Außendienstmitarbeitern, die damit schnell Kundendaten speichern konnten.
Betriebssystemmäßig wurde Windows CE oder das EPOC32 Betriebssystem verwendet. EPOC war ein vielseitiges Betriebssystem, das für den Betrieb in verschiedenen Geräten entwickelt

wurde. Neben den Subnotebooks wurde es auch auf Handys angewendet. Die Entwicklung erfolgte durch das Symbian Konsortium, das von Ericsson, Motorola, Nokia und Psion gegründet wurde.
EPOC war zudem ein sehr stabiles Betriebssystem - Nutzer erfuhren einen Absturz nur alle paar Jahre. Von der Funktionalität her unterstützte es neben den PIM Applikationen einen Word Prozessor, Tabellenkalkulationsanwendungen, einen Webbrowser, der Java Applets unterstützt, Kommunikationsdienste wie Email, FAX und SMS sowie Backup und Synchronisation des Endgeräts mit einem PC. Sein Nachfolger ist Symbian.
In der Gegenwart kommen als Betriebssysteme gängige PC-Betriebssysteme wie Windows, MacOS X (iPhone) oder Linux (Android) zum Einsatz.

2.1.3 Handys

Als ubiquitäres Endgerät betrachtet haben sich Mobiltelefone von einer einfachen Person-zu-Person-Sprachschnittstelle zu einem leistungsfähigen Netz-Client entwickelt.
Sie ermöglichen z.T. auch PIM, Spiele und haben verschiedene Schnittstellen und Erweiterungsmöglichkeiten. Allerdings haben sie mit einem Problem zu kämpfen: Wie können nutzerfreundlich Daten eingegeben werden?
Unabhängig von einer sehr bequemen Zifferneingabe gibt es in der Regel die Möglichkeit, Sprache einzugeben und aufzuzeichnen. Texteingabe wird durch T9, also “Text on 9 keys”, realisiert, einer Mehrfachbelegung der Zifferntasten mit Buchstaben und Sonderzeichen. Die Software dekodiert die Folge der gewählten Tasten und ordnet ihr eine Bedeutung zu. Diese T9-Eingabe ist eine Schwachstelle.
Neben der einfachen Sprachkommunikation gibt es verschiedene Möglichkeiten, Daten zu übertragen und über ein Mobilnetz auf Daten zuzugreifen. Weit verbreitete Beispiele sind:

- SMS, der Short Message Service, ermöglicht die Übertragung von Textnachrichten mit bis zu 160 Zeichen zwischen mobilen Endgeräten.
- WAP, das Wireless Application Protocol, ist der Standard, der zur Übermittlung eines speziell formatierten Inhalts von einem Webserver auf Handys und andere drahtlose Endgeräte mit eingeschränktem Bildschirm verwendet wird. Ein WAP-Browser auf einem mobilen Endgerät ermöglicht folglich eine Interaktion mit dem Internet, z.B. für E-Commerce-Anwendungen (Aktien, Hotelreservierung, Fahrplan, Wetter).
- MMS, der Multimedia Message Service, verarbeitet und überträgt multimediale Objekte wie Bilder, Video, Audio und formatierte Textnachrichten.
- Mobile Multimedia geht noch einen Schritt weiter in Richtung Animation und Video Clips.
- Web- und E-Mail-Dienste ermöglichen bei fortgeschritteneren Geräten Zugriff auf das Internet, wie man es vom Arbeitsplatz oder vom heimischen PC aus gewohnt ist.

Bei letztgenannten Anwendungen kann die Geschwindigkeit der Datenübertragung allerdings ein Hindernis sein. Über das weit verbreitete GSM-Netz (Global System for Mobile Communications) können Daten mit 9,6 bis 14,4 kBit/s transportiert werden, was zu einem Engpass bei der Informationsabfrage führt. HSCSD (High Speed Circuit Switched Data) erreicht durch GSM-Kanalkopplung bis zu 115,2 kBit/s. Der parallel zu GSM installierte, paketorientierte

Datenübertragungsdienst GPRS (General Packet Radio Service) ermöglicht 171,2 kBit/s. In einem weiteren Evolutionsschritt wurde EGDE eingeführt (Enhanced Data Rates for GSM Evolution), das durch verbesserte Modulationsverfahren auf der Funkschnittstelle die maximale Datenrate auf 384 kBit/s steigert. Weiterentwicklungen von EDGE, die bisher (2008) noch kaum zum Einsatz kommen, ermöglichen bis zu 1 MBit/s im GSM-Netz.
Mit der großflächigen Einführung von UMTS etwa seit 2004 ist der Weg frei für noch größere Übertragungskapazitäten. UMTS überträgt standardmäßig bis zu 384 kBit/s und kann mit einem anderen, bisher wenig genutzten Multiplexverfahren bis zu 1920 kBit/s erreichen. Wie bei GSM mit GPRS steht auch bei UMTS mit HSDPA (High Speed Downlink Packet Access) ein paketorientierter Datenübertragungsdienst zur Verfügung, welcher 13,98 MBit/s übertragen kann. In der Praxis sind die Datenraten allerdings niedriger, da sie von der Empfangsqualität am jeweiligen Standort und der aktiven Teilnehmerzahl der UMTS-Funkzellen abhängen. Gängig sind Downloadgeschwindigkeiten von 1,8 bis 7,2 MBit/s.
Als reines Datenübertragungsnetz etabliert sich WiMAX (Worldwide Interoperability for Microwave Access). In Deutschland wurde 2007 mit dem Netzausbau begonnen, mit einer Flächendeckung kann jedoch erst in mehreren Jahren gerechnet werden, falls sich der Standard durchsetzt. Theoretisch sind über WiMAX 70 MBit/s möglich, in der Praxis ergeben sich aus denselben Gründen wie bei HSDPA abgestufte Datenraten von 2 bis 12 MBit/s.
Mit DVB-H (Digital Video Broadcasting - Handhelds) existiert auch ein Netz zur Übertragung von Fernsehprogrammen auf mobile Endgeräte. Seit etwa 2007 sind Endgeräte mit Unterstützung für DVB-H verfügbar, 2008 wurde mit der Einführung des Standards in Deutschland begonnen.
In Entwicklung befindet sich auch bereits der Nachfolgestandard von UMTS/HSDPA namens HSOPA (High Speed OFDM Packet Access), besser bekannt als LTE (3GPP Long Term Evolution). Er wartet mit Geschwindigkeiten von 100-150 MBit/s auf und wurde kommerziell erstmals 2009 in Stockholm und Oslo in Betrieb genommen.

Generell ist also eine Technologieentwicklung erkennbar, welche den Engpass auf der Luftschnittstelle beseitigt. Die Datenübertragungsgeschwindigkeit nähert sich immer mehr den etablierten Verfahren für kabelgebundene Datenübertragung auf dem Kupferkabel an bzw. überholt diese teilweise. Zum Vergleich: ISDN bietet 64 bzw. 128 kBit/s mit Kanalbündelung, der aktuelle DSL-Standard ADSL2+ (Asymmetric Digital Subscriber Line) ermöglicht 24 bzw. 48 MBit/s mit Kanalbündelung, praktisch sind derzeit (2008) in Deutschland bei optimaler Verbindungsqualität 16 MBit/s üblich. Mit VDSL2 (Very High Speed Digital Subscriber Line) sind theoretisch maximal 100 MBit/s erreichbar, die Deutsche Telekom bietet seit 2006 Anschlüsse mit 50 MBit/s bei optimaler Leitungsqualität an. Aktuelle Lösungen über das Fernsehkabelnetz erlauben bis zu rund 100 MBit/s.
Neben dem Zugang zum Internet haben Handys in der Regel auch eine Bluetooth-Schnittstelle, um Daten lokal zwischen Geräten auszutauschen, z.B. um einen Adressbucheintrag zu verschicken oder Daten mit PCs zu synchronisieren. Bluetooth hat eine Reichweite von bis zu 10m und ermöglicht die Bildung von Ad-hoc-Piconetzen mit anderen Endgeräten. Selten ist auch noch eine Infrarot-Schnittstelle (IrDA) zu finden, die nur eine langsamere Eins-zu-Eins-Kommunikation ermöglicht.
Es ist eine deutliche Konvergenz der mobilen Endgeräte erkennbar. Tendenziell entwickeln sich Handys zu Smart Phones, die immer umfangreichere Funktionalität der PDAs verschmilzt mit der Technologie der Handys und es entwickeln sich All-in-one-Kommunikationssysteme, die teilweise auch schon die Funktionalität konventioneller PCs bieten. Durch die kontinuier-

liche technologische Evolution sind nicht nur Telefonie sondern auch Dienste wie ortsunabhängiger, schneller Internetzugriff und mobiles digitales Fernsehen möglich.
Ein prominentes Beispiel für diese Konvergenz ist das 2007 von Apple eingeführte Multimediatelefon iPhone sowie der verwandte iPod touch ohne Telefoniefunktionen. Basierend auf dem Betriebssystem Mac OS X integriert das iPhone Telefonie, SMS und E-Mail, Kamera und Fotoverwaltung, Audio- und Videoplayer, einen vollständigen Webbrowser, PIM, Synchronisation, Navigationssystem mit Landkarten und Positionsbestimmung und kann beliebig mit weiteren Anwendungen erweitert werden. Es ist ausgestattet mit einem großen, farbigen 3,5-Touchscreen, GSM mit EDGE, WLAN, Bluetooth und USB und bis zu 16 GB Flash Memory.

2.1.4 Betriebssysteme für Information Access Devices

Ein grundlegendes Problem für die Entwicklung und den Betrieb von ubiquitären Anwendungen auf den Information Access Devices ist in dem Vorhandensein unterschiedlichster Programmierschnittstellen und proprietärer Betriebssysteme zu sehen.
Zur Lösung dieses Problems gibt es zwei unterschiedliche Ansätze:

Java Byte Code Die Java Community setzt sich dafür ein, standardisierten Java Byte Code als Integrationsschicht zwischen den verschiedenen Betriebssystemen und plattformunabhängigen Anwendungen zu verwenden.

Win32 API auf Windows CE Windows CE wird heute als Windows Mobile bezeichnet. API steht für Application Programming Interface. Microsoft versucht, eine Standardisierung direkt auf der Betriebssystemebene zu erreichen. Auf der Basis der Akzeptanz von Windows als De-facto-Standard für PCs hat Microsoft im Jahre 1996 Windows CE eingeführt, um ein einheitliches Betriebssystem für möglichst alle nur denkbaren ubiquitären Computer zur Verfügung zu stellen (Handhelds, Industriesteuerungen, Entertainmentsysteme, Embedded Systems, Wireless Devices,...).
Die Win 32 API definiert dabei eine einheitliche Schnittstelle für alle Anwendungen, wobei die Windows CE Komponenten die gesamte hardwarespezifische Funktionalität kapseln, so dass eine plattformunabhängige Schnittstelle entsteht.
Im Gegensatz zu den Windows-Betriebssystemen für PCs ist Windows CE nicht darauf ausgelegt, als ganzer Block an die Endnutzer verkauft zu werden. Microsoft stellt vielmehr eine Menge von Funktionalitäten, d.h. modularen Building Blocks, zur Verfügung, wobei jeder Gerätehersteller dann sein Windows CE Betriebssystem für eine spezifische Prozessorplattform auf dem speziellen Endgerät zusammenstellt. So hat z.B. ein Handheld die Möglichkeit, mehr Funktionalität aufzunehmen als ein SmartPhone.
Gewöhnlich wird das Betriebssystem auf einem ROM ausgeliefert, der dann in das Endgerät eingebaut wird.
Der Vergleich von Java und Windows CE ist in Abbildung 2.4 dargestellt:

1. **Das Windows CE-Betriebssystem Konfiguration des Betriebssystems** Mit dem bereits erwähnten Konzept der Building Blocks von Windows CE kann die **Größe eines Betriebssystems auf die Größe eines Endgeräts angepasst werden**. Dieser flexible Ansatz ist ähnlich zu den konfigurierbaren Java Subsets der Java 2 Micro Edition.

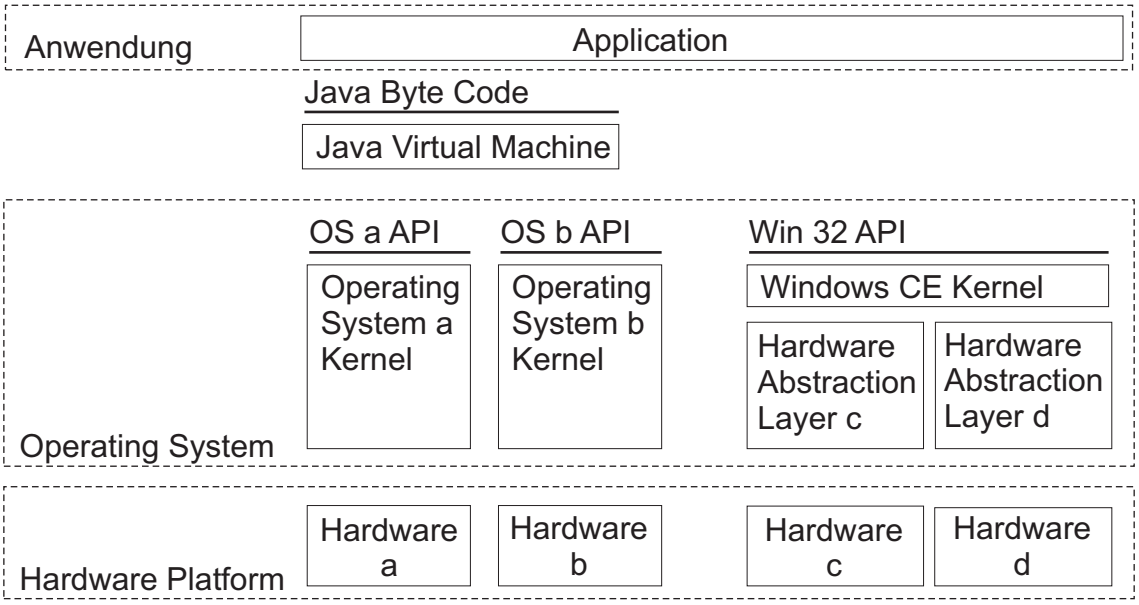


Abbildung 2.4: Vergleich von Java und Windows CE

In Abhängigkeit von der beinhaltenden Funktionalität werden **drei Gruppen von Betriebssystemkonfigurationen** für Windows CE unterschieden:

- Handheld PC und Handheld Professional PC (H/PC) sind Endgeräte der Subnotebook-Klasse, die mit Tastatur, Maus, VGA-Display und USB Port ausgestattet sind. Sie verfügen in der Regel über mehr als 16MB RAM und bieten Pocketversionen zum Beispiel von Microsoft Word, Excel, PowerPoint, Access, Internet Explorer und Outlook von an.
- Pocket PCs (P/PC) sind entsprechend kleiner mit einfacheren PIM-Anwendungen
- Automotive PCs (Auto/PC) beinhalten in der Regel eine Sprachschnittstelle, ein oder mehrere optische Laufwerke für Daten und Musik, USB- und Bluetooth-Schnittstellen, Empfänger für Radio, Fernsehen und GPS und einen Farbbildschirm. Auto/PC-Anwendungen sind Kontrollfunktionen, Navigation, Diagnose, Entertainment und PIM.

Dieser universelle Ansatz unterscheidet sich vollständig vom konkurrierenden Palm OS, das für jede Klasse von Endgeräten exklusiv entwickelt und optimiert wurde.

Speichermanagement Alle Arten von Speicher (Flash Speicher, Festplattentreiber, ROM oder RAM) werden durch Windows CE in der gleichen Art und Weise verwaltet. Der Zugriff ist über die Win32 API möglich. Der verfügbare Speicher wird dabei in zwei getrennte Blöcke unterteilt: Programm- und Datenspeicher.

Der **Datenspeicher** beinhaltet die persistenten Daten, die ein gewöhnlicher PC in der Regel auf einer Festplatte speichern würde. Er nutzt sowohl RAM als auch ROM.

Der **Programmspeicher** dient unter anderem der Realisierung von Stack und Heap aller laufenden Anwendungen. Der Programmspeicher wird als virtueller Speicher organisiert. Eine Memory Management Unit (MMU)(=Page Table) innerhalb des Kernels verwaltet die Abbildung virtueller Speicheradressen auf aktuelle physische Adressen. Dieses klassische Konzept macht die Speicherzuordnung von physikalischen Konfigurationen unabhängig, in dem ein kontinuierliches, lineares Speichermedium simuliert wird, unabhängig davon, wie die Daten auf RAM oder ROM aufgeteilt werden, z.B. mittels verschiedener Flashspeicherkarten oder Festplatte. Die Seitentabelle wird nicht innerhalb des Betriebssystems gespeichert, sondern direkt auf der Hardware. Die Seiten sind dabei ein bis vier Kilobyte groß, abhängig von der jeweiligen Windows CE Implementierung auf einem speziellen System.

Beispiel: Der gesamte virtuelle Adressraum von Windows CE ist 4 GB groß. Ein Teil ist durch 32 Speicherslots belegt, die jeweils 32 MB groß sind. Diese können bis zu 32 simultan laufenden Prozessen zugeordnet werden - für jeden Prozess steht ein Slot zur Verfügung. Ein weiterer 32 MB großer Slot wird für den aktuell laufenden Prozess benutzt.

Der verbleibende virtuelle Adressraum ist für den Kernel reserviert sowie als Shared Memory zwischen verschiedenen Prozessen.

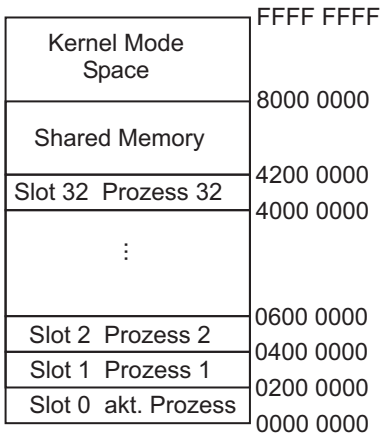


Abbildung 2.5: Aufteilung des virtuellen Adressraumes von Windows CE

Prozesse, Threads und Unterbrechungen Windows CE arbeitet als Multitasking-Betriebssystem. Wie schon aus dem Speichermanagement bekannt, unterstützt der Kernel bis zu 32 Prozesse. Innerhalb jedes Prozesses kann eine unbeschränkte Anzahl von Threads laufen, solange freier RAM verfügbar ist.

Threads werden insbesondere benutzt, um auf asynchrone Ereignisse zu reagieren, wenn diese beobachtet (monitored) wurden. Threads können acht Prioritäten haben, wobei die höchstpriorisierten Threads nie unterbrochen werden. Diese oberste Priorität wird für Echtzeitanwendungen und Gerätetreiber verwendet. Die mittleren Prioritäten werden für Anwendungen und Betriebssystem-Kern-Prozesse verwendet. Die niedrigsten Prioritäten werden für Hintergrundoperationen genutzt.

Basierend auf diesen Prioritäten bekommt jeder Thread Zeitscheiben an Ausführungszeit zugeteilt, wobei die Threads mit niedrigerer Priorität mit der Ausführung warten müssen, bis alle höherprioren Threads abgearbeitet sind.

Die Priorität eines Threads bleibt während dessen Abarbeitung konstant. Die einzige Ausnahme besteht dann, wenn ein höherpriorisierter Prozess auf die Ressource eines niedrigerpriorisierten Prozesses wartet - dann wird dessen Priorität auf die des wartenden Prozesses heraufgestuft.

Unterbrechungen werden verwendet, um dem Betriebssystem externe Ereignisse mitzuteilen bspw. von Peripheriegeräten. Eine sogenannte Hardwareunterbrechungsanfrage (Hardware Interrupt Request, IRQ) wird in Windows CE in zwei Schritten ausgeführt:

– Zunächst wird jede IRQ **einem Interrupt Service Router (ISR)** zugeordnet. Der ISR ist eine sehr schnelle, im Kernelmodus laufende Routine, welche die Anfrage auf einen entsprechenden Interrupt Service Thread (IST) abbildet und den entsprechenden Thread Identifier dem Kern mitteilt.

– Der Kern sendet dem wartenden IST-Thread eine **Nachricht**, welche die Verarbeitung der Unterbrechung initiiert. Nachdem der IST-Thread gestartet wurde, kann der ISR unmittelbar mit der Verarbeitung der nächsten Unterbrechung beginnen. So werden Verzögerungen bei der Unterbrechungsbehandlung auf ein Minimum reduziert.

Da ein IST ein ganz normaler Thread ist, kann diesem eine Priorität zugeordnet werden. Um das Eventhandling zu beschleunigen, werden in der Regel hohe Prioritäten genutzt.

Dieses Verfahren separiert die eher zeitintensive Unterbrechungsbehandlung von der Unterbrechungszuordnung und -weiterleitung, d.h. vom Interrupt Routing. Damit kann eine obere Schranke für die Verzögerung bei der Unterbrechungsbehandlung angegeben werden, d.h. es kann garantiert werden, dass Unterbrechungen immer in dieser maximalen Zeit erkannt werden. Und: Bei Unterbrechungen mit hoher Priorität kann auch garantiert werden, dass diese innerhalb einer bestimmten Zeit abgearbeitet werden.

So erfüllt Windows CE die Anforderungen an ein **Echtzeit-Betriebssystem**. Damit wird für die kleinen Endgeräte ein breites Anwendungsfeld zeitkritischer Applikationen möglich wie z.B. Switching von Telekommunikationsprozessen, Kontrolle von Herstellungsprozessen (z.B. am Fließband) oder Navigation.

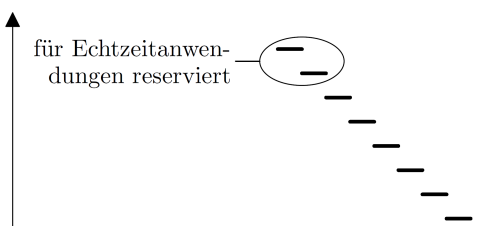


Abbildung 2.6: Priorität der Prozesse bei Windows CE

2. **Palm OS**
   Das Palm OS wurde durch die Firma Palm entwickelt. Obwohl es ein proprietäres Betriebssystem ist, hatte es **um die Jahrtausendwende** einen Marktanteil von 70% auf den Handhelds.

   Speichermanagement Der gesamte Speicher eines Palm-Geräts befindet sich auf sogenannten **Speicherkarten (Memory Cards)**. Eine solche Karte ist eine logische Einheit von **RAM, ROM oder beidem**. Der maximale Adressraum einer **Speicherkarte** beträgt theoretisch 256 MByte. Der Speicher, der insgesamt verfügbar ist, wird in mehrere Bereiche aufgeteilt:

   – **einen einzelnen sog. dynamischen Speicherbereich** von z.B. 96 kByte und
   – **mehrere Datenbereiche** für Daten und Anwendungsprogramme, die auf dem ROM gespeichert werden sowie Betriebssystem-Software, die auf einem RAM gespeichert wird.

   Damit ist dieses Konzept der Unterteilung in dynamischen Speicher- und Datenbereich sehr ähnlich zum Windows CE-Konzept, das Programm- und Datenspeicher unterscheidet.

   Bei Palm OS werden **beide Arten des Speichers getrennt verwaltet**.

   Durch die Palm-Anwendungen bedingt, verwendet Palm OS keine traditionellen Dateisysteme. Stattdessen werden die **Daten in Datenbanken** gespeichert. Jede Datenbank besteht wiederum aus **mehreren Datensätzen** (Records), die vom Data Base Manager verwaltet werden.

   Datensätze einer Datenbank werden als sogenannte Chunks innerhalb des Datenbereichs gespeichert. Chunks sind dabei auf 64 kByte limitiert.

   Vorteilhaft an diesem Konzept ist, dass **kein Paging** erforderlich ist, sondern Operationen auf den Datensätzen (wie z.B. Hinzufügen, Löschen) unmittelbar ausgeführt werden können.

   Events Auf einem Palm-Gerät kann zu jedem Zeitpunkt nur eine Applikation laufen. Es gibt **keine Threads**. Allerdings sind Unterprogrammaufrufe möglich in dem Sinne, dass z.B. eine Suchanfrage andere Applikationen aufrufen kann, um z.B. eine Anfrage auf einer Datenbank auszuführen. Nach Ausführung der Anfrage wird die Kontrolle an die aufrufende Einheit zurückgegeben.

   Palm OS ist Event-getrieben. Events werden sog. Event Handlern zugewiesen. Das Grundkonzept der Arbeitsweise besteht darin, Codeblöcke zu vermeiden, die viel CPU-Zeit verbrauchen, während auf ein Event gewartet wird. Z.B. werden Polling Intervalle verwendet anstatt kontinuierlich zu Pollen.

3. **Symbian EPOC** Symbian ist laut eigenen Angaben der weltweit führende Anbieter von Betriebssystemen bei Handhelds. Symbian hat im 2. Quartal 2004 (April-Juni) seinen Marktanteil beim Betriebssystem Symbian für mobile Geräte (Smartphones und PDAs)

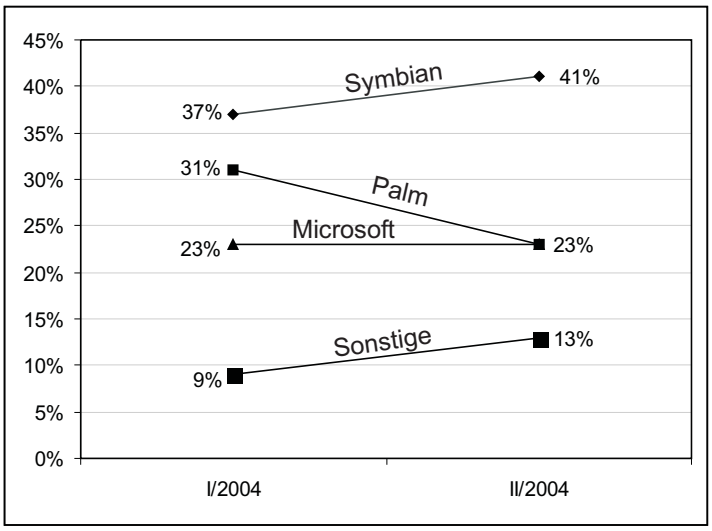


Abbildung 2.7: Marktanteil (in %) der Betriebssysteme mobiler Endgeräte (Analyse der US-Marktforscher der Fa- Canalys)

weltweit um 4% auf 41% gesteigert. Microsoft stagnierte bei 23%, der Anteil von Palm ging um 8% auf 23% zurück.

Symbian hat dabei Potentiale in den Märkten wie Europa, Naher Osten, Afrika aber auch Japan.

Lizenznehmer, die ihre Handys mit dem Symbian OS ausstatten, sind u.a. Nokia, Siemens, Sony Ericsson, Motorola, Fujitsu, LG und Samsung. Potential ist auf dem chinesischen Mobilfunkmarkt zu sehen bei den asiatischen Lizenznehmern Arima, BenQ und Sendo.

Allein im Oktober 2004 haben sieben Lizenznehmer 27 neue Geräte an mehr als 200 Netzbetreiber ausgeliefert.

Weltweit wurden bis Ende 2004 mehr als 15 Millionen Symbian-Handys verkauft. Für das Jahr 2009 wird diese Zahl auf rund 200 Millionen prognostiziert.

Für die Weiterentwicklung des Symbian-Betriebssystems hat der Chef des Symbian-Konsortiums, David Levin, die Zahl der Entwickler von 900 auf 1200 bis Ende 2005 aufgestockt. Künftig will Symbian gemeinsam mit Chiphersteller Intel UMTS-Handys entwickeln und zwar mit einer stromsparenden Technologie. Außerdem drängt Symbian auf die Ablösung herstellerspezifischer Betriebssysteme zugunsten einheitlicher Standards.

Aufbauend auf dem Betriebssystem wird Potential für den Verkauf von Spielesoftware gesehen sowie für Programme, mit denen Unternehmen z.B. die Aktivitäten ihrer Außendienstmitarbeiter besser steuern können.

Interessant ist auch, dass jeder Besitzer eines Symbian-Handys in 2004 durchschnittlich 18% mehr Downloads gekauft hat, als Kunden mit anderen Betriebssystemen. In der Literatur wird die Wurzel des Symbian OS in Form von EPOC beschrieben.

EPOC ist ein vielseitiges Betriebssystem, das für den Betrieb in verschiedenen Geräten entwickelt wurde. Neben dem Einsatz auf Sub-Notebooks wird es auch auf Handys angewendet. Die Entwicklung erfolgte durch das Symbian-Konsortium, welches von Ericsson, Motorola, Nokia und Psion gegründet wurde.
EPOC ist ein **Echtzeitbetriebssystem**, ist **objektorientiert** in C++ geschrieben und unterstützt **preemtives** Multitasking. Der Kern von EPOC besteht aus den vier Hauptkomponenten GUI/System, Grafik, Engine Support und Basis. Zudem ist EPOC sehr stabil, Nutzer erfahren einen Absturz nur alle paar Jahre. Es bietet neben PIM-Applikationen eine Textverarbeitung, Tabellenkalkulation, Kommunikationsdienste wie E-Mail, Fax und SMS, einen Webbrowser mit Java-Support sowie Backup und Synchronisation des Endgeräts mit einem PC.

Der Kern von EPOC besteht aus vier Hauptkomponenten.

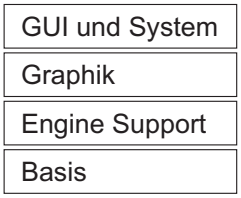


Abbildung 2.8: Komponenten von EPOC

2.2 Smart Identification

Smart Identification ist das gleiche wie Smart Control. Es wird nichtflüchtiger Speicher benötigt. Diese Technik des Smart Identification wollen wir noch mal in zwei Arten unterteilen, zuvor jedoch noch Grundlagen betrachten.

2.2.1 Klassifikation von Speicherbausteinen

Speicherbausteine mit wahlfreiem Zugriff (Random Access Memory, RAM) gibt es in zwei Varianten: statisch und dynamisch.

static RAM (SRAM) wird intern mit Schaltungen realisiert, die strukturell einem Flip-Flop-Speicherbaustein entsprechen. Diese Speicher haben die Fähigkeit, ihre Inhalte zu behalten, solange sie mit Strom versorgt werden. SRAM ist sehr schnell, typische Zugriffszeiten betragen nur wenige Nanosekunden. Aus diesem Grund wird SRAM als Level 1 und 2 Cache eingesetzt.

ddynamic RAM (DRAM) besteht demgegenüber nicht aus Flip-Flops, sondern aus einer Reihe von Zellen, die jeweils einen Transistor und einen winzigen Kondensator enthalten. Kondensatoren können be- und entladen werden, so dass sich Nullen und Einsen speichern lassen. Da elektrische Ladung jedoch zu Kriechverlusten neigt, muss jedes Bit in DRAM alle paar Millisekunden aufgefrischt, d.h. nachgeladen werden.

Aktuelle Betriebssysteme für mobile Endgeräte *(Stand Februar 2008)*
Gegenwärtig existieren neben den proprietären Lösungen einiger Mobiltelefonhersteller (z.B. RIM BlackBerry OS) noch folgende herstellerübergreifende Betriebssysteme:

- **Symbian OS** (Nokia)
  Symbian OS wird von einem Konsortium bestehend aus Ericsson, Nokia, Panasonic, Samsung, Siemens und Sony Ericsson entwickelt, Nokia hält die mit Abstand größten Anteile. Historisch gesehen ist Symbian aus dem Betriebssystem EPOC der Firma Psion hervorgegangen. Gegenwärtig werden verschiedene graphische Nutzerschnittstellen angeboten, die auf einer identischen Betriebssystemschicht aufbauen. In der aktuellen Version 9.5 sind dies UIQ (Endgeräte mit Touchscreen und Stifteingabe), die sehr verbreitete Series 60 (Mobiltelefone mit Zifferntastatur und klassischen "Barrenlayout") und die Series 80 (Endgeräte mit voller QWERTZ-Tastatur), sowie Unterstützung aktuelle Digitalfernsehstandards und eine integrierte Datenbank.

- **Windows Mobile** (Microsoft)
  Windows Mobile in der aktuellen Version 6.0 basiert auf Windows CE, das ursprünglich ausschließlich für sogenannte Embedded Devices entwickelt worden war. Das Betriebssystem liegt in zwei konkreten Ausprägungen vor: Windows Mobile und Windows Mobile for Smartphones, welches die klassischen Organizerfähigkeiten um Telefonie- und SMS-Funktionalität erweitert. Als Synchronisationsprotokoll kommt das proprietäre ActiveSync zum Einsatz, welches einen Abgleich des mobilen Geräts mit einem PC ermöglicht.

- **Palm OS** (Access, früher PalmSource)
  Access bietet momentan Palm OS 5 (Garnet) und Palm OS 6.1 (Cobalt) parallel an, da bisher nur sehr wenige Endgerätehersteller das neue Betriebssystem lizenziert haben. Aktuell verliert Palm OS auch kontinuierlich an Bedeutung im Bereich der mobilen Endgeräte. Selbst die ursprüngliche Mutterfirma PalmOne produziert mittlerweile Endgeräte mit Windows Mobile als Betriebssystem. Gründe sind u.a. die frühere und bessere Unterstützung von Multimedia-Anwendungen und farbigen Displays.

- **BREW** (Qualcomm)
  Das "Binary Runtime Environment for Wireless"(BREW) wurde von Qualcomm primär für CDMA-Mobiltelefone entwickelt, unterstützt aber mittlerweile alle gängigen Übertragungsstandards, wie GSM/GPRS oder UMTS. Es kann sowohl das alleinige Betriebssystem auf einem Mobiltelefon sein, als auch neben einem nativen OS koexistieren. Im letzteren Fall ist seine Rolle vergleichbar mit der Plattform J2ME. Lizenznehmer sind u.a. Sony Ericsson und Motorola.

- **MacOS X** (Apple)
  Mit dem iPhone brachte Apple 2007 erstmals ein fast vollständiges Rechnerbetriebssystem auf einem mobilen Endgerät unter. MacOS X basiert ursprünglich auf dem Unix-Derivat BSD und wurde nur um die für die mobilen Anwendungen nicht benötigten Komponenten reduziert. In 2008 wurde das SDK für MacOS X auf dem iPhone veröffentlicht, womit leistungsfähige Anwendungen, wie sie auch auf Desktoprechnen möglich sind, realisiert bzw. einfach portiert werden können. Sollte Apple sich entschließen, diese Variante von MacOS X allgemein verfügbar zu machen, wäre eine weitere moderne Plattform am Markt. Das Betriebssystem ist auf diversen Hardwareumgebungen lauffähig.

- **Android** (Google) Android wurde im Februar 2008 von der Open Handset Alliance (bestehend aus 34 ITK-Unternehmen) vorgestellt. Nokia ist nicht beteiligt, da es mit Symbian OS in Konkurrenz zu Android steht. Das Betriebssystem wurde unter der Federführung von Google entwickelt und soll eine leistungsfähige und offene Plattform für mobile Anwendungen werden, der Quellcode wird als Open-Source veröffentlicht. Basierend auf Linux 2.6 und einer speziell entwickelten, optimierten Java VM (Dalvik) und mit diversen Erweiterungen für Grafik, Multimedia und Datenbanken, lassen sich damit die meisten Fähigkeiten aktueller PCs implementieren.

- verschiedene **Linux-Varianten** (u.a. Motorola)
  Diese Betriebssysteme basieren auf klassischen Linux-Distributionen, die um die entsprechenden Telefonie- und Übertragungsfunktionalitäten erweitert wurden. Gegenwärtig befinden sich allerdings diese Lösungen noch in einem sehr frühen Stadium und wurden nur vereinzelt in einigen Produktivgeräten implementiert.

Um die Auffrischung muss sich eine externe Logik kümmern, so dass DRAM komplexere Schnittstellen erfordert als SRAM. Während der Auffrischung hat die CPU keinen Zugriff auf den DRAM. Dadurch ist DRAM langsamer als SRAM, zig Nanosekunden werden als Zugriffszeit benötigt. Dafür hat DRAM jedoch größere Kapazitäten und eine sehr hohe Dichte, also viele Bits pro Chip. Aus diesen Gründen werden Hauptseicher in der Regel als DRAM realisiert und optimalerweise mit SRAM als Cache kombiniert.
DRAM ist in verschiedenen Typen erhältlich: Die erste Generation namens **FPM-DRAM** (Fast Page Model) wurde 1987 eingeführt und basiert auf der Anordnung der RAM-Speicherzellen in einer Matrix bzw. Tabelle. Dadurch lässt sich die Adressierung sehr einfach gestalten. Nach jedem Takt wird zwischen der Angabe von Spaltenadresse (RAS, Row Address Signal) und Zeilenadresse (CAS, Column Address Signal) hin und her geschaltet und somit die angesprochene Speicherzelle ausgewählt.
Die Optimierung dieses RAS/CAS-Verfahrens erlaubte einen bis zu dreimal schnelleren Zugriff auf die Daten als bei herkömmlichem DRAM. Während eines fortlaufenden Speicherzugriffs wird das Anlegen der immer gleichen Zeilenadresse gespart. Es genügt, die Zeilenadresse einmal anzugeben und die jeweilige Spaltenadresse anzugeben und auszuwerten. Der Zugriff erfolgt erheblich schneller.
Diese Speicherart wurde 1995 durch **EDO-RAM** abgelöst. Hinter der Bezeichnung EDO (Extended Data Output) steckt eine Technik, mit der die Spannung in den Kondensatoren länger aufrecht erhalten bleibt. Dadurch kann der Speicherzustand einer Speicherzelle länger beibehalten werden. Damit stehen die Daten am Ausgang länger bereit und die Häufigkeit des Speicherrefreshs verringert sich. Während die Daten noch gelesen werden, wird bereits die nächste Adresse an den Speicherbaustein angelegt. Bei aufeinander folgenden Lesezugriffen wird somit eine höhere mittlere Lesegeschwindigkeit erreicht. Dadurch ist zwar eine einzelne Speicherreferenz nicht schneller, allerdings verbessert sich die Speicherbandbreite, d.h. es können mehr Wörter pro Zeiteinheit durchgeschleust werden. Dadurch ist zwar eine einzelne Speicherreferenz nicht schneller, allerdings verbessert sich die Speicherbandbreite, d.h. es können mehr Wörter pro Zeiteinheit durchgeschleust werden.
Während FPM- und EDO-Chips asynchron arbeiten, also ohne einheitlichen Taktgeber, ist **SDRAM** (Synchronous DRAM, seit 1997) eine Mischung aus statischem und dynamischem RAM. Er wird durch einen einzelnen, synchronen Taktgeber gesteuert. Intern besteht SDRAM aus zwei Speicherbänken. Der Zugriff erfolgt abwechselnd, so dass die benötigte Erholzeit zwischen den Zugriffen scheinbar entfällt. Einen zusätzlichen Geschwindigkeitsvorteil bringt das Pipeline-Verfahren. Darüber hinaus ist SDRAM programmierbar und so universell einsetzbar.
Eine weitere Verbesserung hinsichtlich der Datenrate ermöglicht **DDR-SDRAM** (Double Data Rate SDRAM, seit 2000). Das Verfahren ist ganz einfach, es wird sowohl die aufsteigende als auch die absteigende Flanke des Takts für Speicherzugriffe genutzt, so dass sich eine Bandbreitenverdoppelung ergibt.
In einer weiteren Entwicklungsstufe wurde 2004 **DDR2-SDRAM** konzipiert, der pro Taktschritt auf vier Datenworte zugreifen kann. Seit 2007 existiert **DDR3-SDRAM**, das in einem Takt acht Datenworte übertragen kann. DDR2- und DDR3-SDRAM sind neben **FB-DIMM** (Fully Buffered Dual In-line Memory Module) aus dem professionellen Workstation- und Serversegment die zurzeit (2008) gängigen Hauptspeichertypen.
Statt DDR-SDRAM verwendet man häufig auch nur die Abkürzung **DDRAM** (Double Date Random Access Memory). Während dieses Wiederaufladens hat die CPU keinen Zugriff auf den DRAM. Dadurch ist der DRAM langsamer als der SRAM.

RAM habt den entscheidenden Nachteil, dass die Daten bei ausgeschaltetem Strom verloren gehen. In vielen Anwendungen wie z.B. Spielzeug, Elektrogeräten und Autos, PDAs und Handys müssen das Programm und ein Teil der Daten auch bei ausgeschaltetem Strom gespeichert bleiben. Außerdem werden weder das Programm noch die Daten nach der Installation jemals geändert.

Diesen Anforderungen wird **ROM** (Read Only Memory) gerecht, das weder absichtlich noch aus Versehen geändert oder gelöscht werden kann. Die Daten werden bei der Herstellung auf der Chipoberfläche eingebrannt. Bei einer Änderung muss der gesamte Chip ersetzt werden. ROM ist zudem billiger als RAM wenn es in großen Mengen hergestellt wird, weil sich dann die Kosten für eine notwendige Maske amortisieren. Problematisch ist jedoch, dass zwischen Herstellung und Auslieferung oft viele Wochen vergehen und Daten dann mitunter schon veraltet sind.

Um es den Unternehmen zu vereinfachen, neue ROM-basierte Produkte zu entwickeln, wurde **PROM** (Programmable ROM) erfunden. Dieser Speicher arbeitet wie ROM, das einmal durch Durchbrechen winziger Sicherungen programmiert wird, wodurch sich die Vorlaufzeit erheblich verkürzt.

Das nachfolgend entwickelte **EPROM** (Erasable PROM) kann sowohl programmiert als auch gelöscht werden. Wird das Quarzfenster eines EPROM-Chips etwa 15 Minuten lang einem starken ultravioletten Licht ausgesetzt, sind alle Bits auf 1 gesetzt. Im Falle eines iterativen Designzyklus kann EPROM dann weitaus wirtschaftlicher als PROM sein, da es wieder verwendbar ist.

Eine nochmalige Verbesserung ist **EEPROM** (Electrically EPROM), der statt ultraviolettem Licht durch Stromimpulse gelöscht werden kann. Es kann vor Ort gelöscht und neu programmiert werden, ohne dass der verwendete Chip wie beim EPROM in ein spezielles Programmiergerät eingelegt werden muss. Er ist allerdings wesentlich kleiner und langsamer als EPROM. Wäre der Vorteil der Nicht-Flüchtigkeit nicht gegeben, so könnte EEPROM auch nicht mit RAM konkurrieren, da es langsamer und viel teurer ist.

Schließlich wollen wir noch **Flash Memory** betrachten, den leistungsfähigsten nichtflüchtigen Speicher, der die älteren Bauarten mehr und mehr verdrängt. Er kann blockweise gelöscht und wieder programmiert werden. Die einzelnen Flashspeicherzellen unterliegen bei Schreibzugriff durch die notwendige Löschung einem Verschleißprozess, so dass sie im Durchschnitt nach 10.000 bis 100.000 Löschzyklen ausfallen. Flash-Medien sind daher mit einem intelligenten Defektmanagement ausgestattet. Jedes Flash-Medium enthält Fehlererkennungsmechanismen, die bei Erkennen eines Defektes die betroffene Speicherzelle ausblenden und durch eine neue aus einem Pool zusätzlicher Speicherzellen ersetzen, die in einem ausgeblendeten Bereich des Mediums vorgehalten werden. Dadurch werden bei regulärem Gebrauch Millionen von Schreibzyklen möglich, derzeit typische Haltbarkeitsangaben sind beispielsweise 2 Millionen Stunden MTBF (Mean Time Between Failures).
Anwendungsgebiete für Flash Memory sind beispielsweise das Speichern von Bildern in Digitalkameras und Daten in mobilen Endgeräten. Dort wird der Speicher entweder fest integriert oder steht auf kompakten, wechselbaren Speicherkarten zur Verfügung. Auch die weit verbreiteten USB-Sticks mit Kapazitäten von mehreren Gigabytes basieren auf Flash Memory.

Flash Memory ist inzwischen soweit entwickelt, dass es traditionelle Festplatten im PC auf absehbare Zeit als Speichermedium ablösen kann. Seit Anfang 2008 wächst das Angebot für Flash-basierte Festplatten rasant, erste Notebooks werden bereits mit Flash-Festplatten ausgeliefert (Apple Mac Book Air). Da für die Flash-Festplatten die Schnittstellen und Formfaktoren aktueller Festplatten übernommen werden, gestaltet sich deren Einsatz sehr einfach. Die leistungsfähigsten aktuellen Flash-Festplatten überflügeln mit einer Kapazität von 1,6 TB und kontinuierlicher Datenübertragung mit 230 MB/s bei 30-100 Âµs Zugriffszeit die derzeit größten (1,0 TB) und schnellsten (170 MB/s, 3 ms) Festplatten bei weitem - bei fast identischen Abmessungen des Mediums. Zusätzlich verbrauchen Flash-Medien in der Regel weniger Strom als Festplatten und sind deutlich robuster was die Langlebigkeit, die Umgebungsbedingungen als auch die mechanische Empfindlichkeit angeht und sie arbeiten geräuschlos, da sie keine mechanischen Bauteile enthalten. Da gegenüber Festplatten inzwischen ausschließlich Vorteile zu verzeichnen sind, spricht nur der momentan noch deutlich höhere Preis des Flash-Memory gegen die Ablösung der Festplatten durch Flash-Medien (Stand 2008).

Eine Gegenüberstellung der Speichermedien wird abschließend in Tabelle 2.9 vorgenommen.

Speichertyp	Nutzungsart	Löschung	Bytes änderbar	flüchtig	übliche Verwendung
SRAM (Static RAM)	Lesen/ Schreiben	Elektrisch	Ja	Ja	Level-2-Cache
DRAM (Dynamic RAM)	Lesen/Schreiben	Elektrisch	Ja	Ja	Hauptspeicher
ROM (Read Only Memory)	Nur Lesen	Nicht möglich	Gar nicht	Nein	Großvolumige Geräte
PROM (Programmable ROM)	Nur Lesen	Nicht möglich	Einmal programmierbar, aber nicht löschbar	Nein	Kleinvolumige Geräte
EPROM (Erasable ROM)	Vorwiegend Lesen, Schreiben sehr aufwändig	In spezieller Kammer mittels UV-Licht	Wiederverwendbar durch Neuprogrammierung	Nein	Herstellung von Geräteprototypen
EEPROM (Extended EPROM)	Vorwiegend Lesen, Schreiben sehr aufwändig	Elektrisch mittels Impulsen, innerhalb der Schaltung byteweise löschbar	Mit speziellem Gerät	Nein	Herstellung von Geräteprototypen
Flash-Speicher	Lesen/Schreiben	Elektrisch blockweise loschbar	Blockweise änderbar	Nein	Speichermedium für Digitalbilder

Abbildung 2.9: Klassifizierung von RAM- und ROM-Typen

2.2.2 Smart Card

Smart Cards sind kleine Endgeräte mit einem Prozessor, die - auf einer Plastikkarte aufgebracht - in der Regel kontaktbehaftet mit ihrer Umgebung kommunizieren können.
Historisch gesehen geht diese Entwicklung auf Karten mit magnetischen Streifen zurück, die Informationen speichern können. Das macht eine solche Karte maschinenlesbar und ermöglicht die Verarbeitung der Informationen. Der Nachteil bei diesm ist, dass er störanfällig ist.
Allerdings resultiert das Problem, dass jeder mit der nötigen Ausrüstung die Daten lesen und ggf. auch schreiben kann. Die Idee dahinter: Daten von der Karte werden von einem Prozessor verarbeitet und auf dem Kommunikationsweg zwischen Speicher(=Magnetstreifen) und externen Prozessor abgahört.

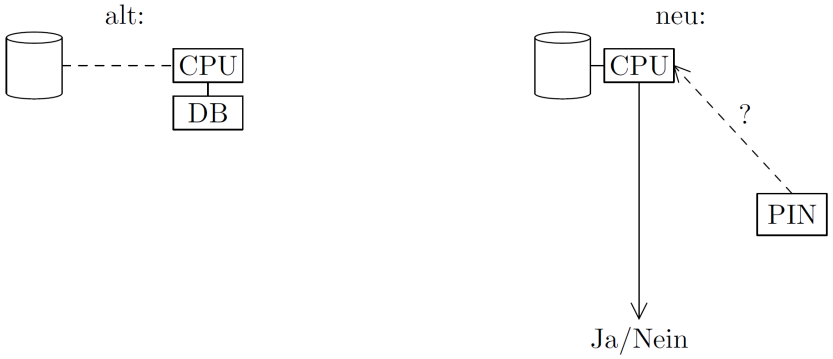


Abbildung 2.10: Vorteil Smart Card

1968 wurde ein Patent für eine Identifizierungskarte mit integrierter Schaltung eingereicht - das Grundkonzept der heutigen Smart Cards.
Dieses Konzept hat im Gegensatz zum Magnetstreifen den Vorteil, dass es nicht kopierbar ist und dann missbraucht werden kann.
Aus diesem Grund sind die Smart Cards von großer Bedeutung, z.B. für Finanzanwendungen. Smart Cards verwendet heute jeder im alltäglichen Leben - es gibt sie als Telefonkarten, Geldkarten, Karten der Krankenkasse. Bis 1998 hat die Deutsche Bank alleine 50 Millionen Geldkarten basierend auf diesem Konzept ausgegeben. Jedes Mobiltelefon, das auf dem GSM-Standard basiert, enthält eine Smart Card zur Authentifikation des Eigentümers. Weitere Anwendungen liegen in Türöffnungskarten, die ggf. auch gleichzeitig zum firmeninternen Bezahlen (z.B. in der Cafeteria) verwendet werden können.
Smart Cards sind fälschungssicher und nicht duplizierbar. Sie bestehen aus einem Single-Chip Microcomputer, der auf einer Plastikkarte in Standard-Kreditkartengröße befestigt ist.
Der Chip ist maximal 25qmm groß. Sein integrierter Prozessor ist in der Lage, selbständig zu arbeiten, Passwörter zu kontrollieren, bevor gespeicherte Daten gelesen werden können und kryptographische Algorithmen auszuführen.
Durch dieses Prinzip können anwendungsabhängig teure Verbindungen zu einem Hostsystem vermieden werden.

Hardware einer Smart Card Eine Smart Card enthält einen Computer mit CPU und Speicher. Dabei haben heutige Smart Cards etwa die selbe Rechenleistung wie die ersten IBM PCs.

Der Aufbau des Chips einer Smart Card ist in Abbildung 2.11 dargestellt.

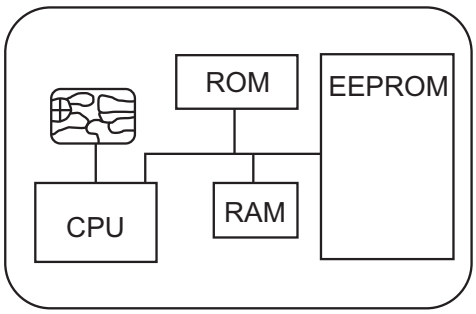


Abbildung 2.11: Aufbau des Chips einer Smart Card

Hardwaresicherheit Smart Cards werden genutzt, um zu speichernde Daten physikalisch sicher zu machen. Da Prozessor und Speicher auf demselben Chip kombiniert sind, ist es extrem schwierig, die zwischen Prozessor und Speicher ausgetauschten Signale abzugreifen. Hinzu kommen Hardware- und Softwaremechanismen, die zum Datenschutz angewandt werden. Her Nachteil hierbei ist, dass die Kommunikation kontaktbehaftet ist.

Kartenlesegeräte Kartenlesegeräte variieren von einfachen Kartenlesern bis hin zu (z.B. Bank-) Terminals mit Monitor und Tastatur für eine bequeme Nutzerführung, wie sie z.B. in Geldautomaten realisiert werden. Außerdem sind viele Kartenleser in Endgeräte eingebettet, z. B. in Handys zum Lesen der SIM-Karte, in Set-Top-Boxes.

Smart Card Software Im allgemeinen besteht eine Smart Card-Anwendung aus

– einer Off-Card- Anwendung und

– einer On-Card-Anwendung.

Die Off-Card-Anwendung wird vom Hostcomputer oder Terminal abgearbeitet, das mit der Smart Card über das Kartenlesegerät verbunden ist. Der On-Card-Teil ist auf dem Chip der Smart Card gespeichert. Dieser Teil kann aus Daten und ggf. ausführbarem Code bestehen. Sofern ausführbarer Code vorhanden ist, wird dieser vom Smart Card Betriebssystem ausgeführt und kann insbesondere Betriebssystemdienste wie z. B. kryptographische Anwendungen nutzen.
Smart Cards verfügen i.d.R. über ein Dateisystem zum Speichern von Daten.
Für die meisten Typen aktueller Smart Cards hat der Anwendungsentwickler keinen Einfluss auf die Entwicklung des ausführbaren Codes auf der Karte, d.h. auf den On-Card-Code. Die Entwicklung solchen Codes kann nur durch den Entwickler des Karten-Betriebssystems vorgenommen werden. Der Code muss in die ROM-Maske integriert werden, bevor die Karte hergestellt wird.

2.2.3 Smart Labels / RFID-Tags

Barcode $\rightarrow$ Smart Label
||
ID für Ware mit Zusatzinfos(Marke, Gewicht, ...)

Die Radio Frequency Identification (RFID - Funkfrequenzidentifizierung) ist eine Methode, um Daten **berührungslos** und **ohne Sichtkontakt lesen und speichern zu können**, vgl. http://www.lexikon-definition.de/Rfid.html.
Die mit der BarCode-Technik (kein Schreiben möglich) vergleichbare Technologie dient der Identifikation von Objekten. Wie BarCode-Systeme bestehen auch RFID- oder Transpondersysteme aus mindestens einem Schreib-/ Lesegerät und den an den Objekten montierten Datenträgern, den Transpondern. Ferner benötigt die Infrastruktur Server, auf denen Dienste ausgeführt werden.
BarCode- und Transpondertechniken unterscheiden sich jedoch durch die Kommunikation zwischen Lesegerät und Datenträger. Während BarCode-Systeme zur Gruppe der optischen Übertragungssysteme gehören, kommunizieren Lesegerät und Datenträger der Transpondersysteme über ein Funksignal.
Hieraus leiten sich auch die Vorteile der Transpondertechnik ab:

- es ist keine Sichtverbindung zum Lesegerät nötig, deshalb können Transponder auch vollständig in die Objekte integriert werden
- Objekte lassen sich auch auf große Entfernung erkennen und
- In einem Durchgang ist auch die Identifizierung von mehreren Objekten gleichzeitig möglich.
- Transponder sind unempfindlich gegen Nässe, Schmutz und Licht - sie können auch nicht verkratzt oder übermalt werden.

Baugröße und Bauformen Ist der Chip auch sehr klein, so wird die Baugröße doch maßgeblich durch die Antenne (in Abhängigkeit von Frequenz bzw. Wellenlänge) und das Gehäuse (das den Anforderungen einer bestimmten Schutzklasse entsprechen muss) bestimmt.
Transponder können die Größe von Büchern besitzen und andererseits so klein sein, dass sie sich z.B. in Geldscheine integrieren lassen.
Transponder für den Handel werden als aufklebbare Etiketten hergestellt, als Wegfahrsperren werden sie als Schlüsselanhänger produziert, für die Tieridentifikation als Glasröhrchen, für die Palettenidentifikation als Nägel oder für die Zutrittskontrolle als Chipkarten.

Historie

- **60er Jahre:** Erste kommerzielle Vorläufer der RFID-Technologie werden als elektronische Warensicherungssysteme auf den Markt gebracht, um Diebstähle zu unterbinden. Basierend auf der Mikrowellentechnik oder Induktion konnte nur ein Bit gespeichert und übertragen werden.
- **70er Jahre:** RFID-Technologie entwickelte sich weiter und wurde zur Kennzeichnung von Tieren, zum Einsatz in der Automatisierung sowie zur Identifizierung von Fahrzeugen im Verkehr verwendet.

- **80er Jahre:** Mehrere amerikanische Bundesstaaten sowie Norwegen setzen RFID im Straßenverkehr für Mautsysteme ein
- **90er Jahre:** RFID für Mautsysteme setzt sich in den USA verstärkt durch, RFID wird für Zugangskontrollen, bargeldloses Zahlen, Skipässe, Tankkarten entwickelt
- **2000er Jahre:** Durch günstige Massenproduktion kommt es zum Preisverfall der Technik, RFID-Tags werden auch **in Verbrauchsgegenstände** eingebracht.

Im November 2002 erwirbt der Rasierklingen-Hersteller Gillette eine Option auf 500 Millionen RFID-Tags des Unternehmens Alien Technologie. Die Einzelhandelskette Wal-Mart kündigt an, RFID-Tags in der Versorgung ihrer Supermärkte einzusetzen.
Die Metro "Future Store" malt die Vision, den gut gefüllten Einkaufswagen einfach aus dem Laden zu schieben und dabei per Funk mit der Karte aus der Hosentasche zu bezahlen.

Weitere Visionen: Ladenbesitzer werden durch einen höheren Gewinn durch automatische Überwachung von Verkaufsregalen motiviert. Der Sensor fordert das Ladenpersonal zum Nachfüllen auf oder meldet Sortierbedarf an, wenn eine DVD an den falschen Platz zurückgelegt wurde. RFID-markierte Kleidungsstücke können ohne Kassenbon umgetauscht werden und die intelligente Waschmaschine liest die Reinigungshinweise der anvertrauten Wäsche direkt aus den Etiketten ab.
Die EZB hat vorgeschlagen, RFID-Tags auf Geldscheinen anzubringen, um damit gefälschte Banknoten einfacher erkennen zu können.
In der neuen Wiener Hauptbücherei dienen Tags der Bestandskontrolle. Einige RFID-Lesegeräte können mittels Pulklesung stapelweise Bücher auf einmal lesen. An den Türen und Ausgängen befinden sich Gates, die wie Sicherheitstore in Kaufhäusern die korrekte Entleihung überwachen.
Weitverbreitet kommt in Asien die berührungslose, wiederaufladbare Fahrkarte zum Einsatz, z.B. in Hong Kong unter dem Namen Octopus-Karte, mit der auch auf Parkplätzen, in Fast-Foodketten und in Läden bezahlt werden kann. In Singapur, Taipei und Tokyo ist sie reine Fahrkarte.
Kunden- und Kreditkarten verschiedener Schweizer Banken sollen mit RFID-Chips bestückt werden, um den zuvor gebuchten Zugang zu Konzerten, Sportveranstaltungen oder Skigebieten zu ermöglichen.

Kosten Der Preis von einfachen (passiven) RFID-Tags bewegt sich bei einer Auflage von einer bis zehn Milliarden Stück zwischen fünf und zehn Cent pro Stück. Bei einer Auflage von ca. 10.000 Tags belaufen sich die Kosten je nach Größe auf 50 Cent bis einen Euro.

Technologie Der Aufbau eines RFID-Tags sieht prinzipiell vor:

- eine **Antenne**,
- einen **Transponder**, d.h. einen analogen Schaltkreis zum Senden und Empfangen,
- einen **digitalen Schaltkreis**, der bei komplexeren Modellen ein Von-Neumann-Rechner ist und
- einen **permanenten Speicher**, der ggf. auch beschreibbar ist.

Energieversorgung Das deutlichste Unterscheidungsmerkmal stellt die Art der Energieversorgung der RFID-Transponder dar. $Energie = Abstand^2$
Hierbei unterscheidet man zwei Arten:

- **Passive RFID-Tags** kleine, batterielose Funkchips besitzen **keine eigene Energieversorgung** und müssen ihre Versorgungsspannung durch Induktion aus den Funksignalen der Basisstation gewinnen. Dadurch reduzieren sich Gewicht und Kosten, es sinkt jedoch die Reichweite. Die gespeicherten Daten können **nur gelesen** werden. Außerdem ist die Menge der speicherbaren **Daten relativ gering**. Dieser Speicher wird üblicherweise genutzt, um eine eindeutige Identifikationsnummer zu hinterlegen. Passive RFID-Tags haben in der Regel eine nahezu **unbegrenzte Lebensdauer**, brauchen jedoch eine **stärkere Leseeinheit**.

- **Aktive RFID-Tags** Transponder **mit eigener Energieversorgung** erzielen eine erheblich höhere Reichweite, besitzen einen größeren Funktionsumfang, sind aber auch erheblich teurer. Deswegen werden sie dort eingesetzt, wo die Transponder eine lange Lebensdauer haben, damit sich deren Einsatz rechnet, z.B. in der Containerlogistik oder bei der Mauterfassung. Sie können typischerweise sowohl **gelesen als auch beschrieben werden**. Die meiste Zeit befinden sie sich im Ruhezustand, d.h. sie **senden keine Informationen aus**. Wird ein spezielles **Aktivierungssignal empfangen so aktiviert sich der Sender**. Der interne Speicher kann - je nach Modell - bis zu einer Million Byte aufnehmen. Im Vergleich zu den passiven RFID-Tags haben sie eine geringere Lebensdauer, die auch maßgeblich durch die **Batterieleistung** beeinflusst wird.

Frequenzbänder Für den Einsatz werden drei Frequenzbänder vorgeschlagen:

- **Niedrige Frequenzen von 30 - 500 kHz**
  Diese Systeme besitzen eine geringe Reichweite (120cm), lange Übertragungszeiten, sind aber günstig in der Anschaffung. Dadurch eigenen sie sich z.B. für Zugangskontrollen, Wegfahrsperren und in der Lagerverwaltung.

- **Mittlere Frequenzen von 10 - 15 MHz**
  Diese Systeme besitzen eine kurze bis mittlere Reichweite (3-11m), mittlere Übertragungsgeschwindigkeit und mittlere bis günstige Preisklasse. In diesem Bereich arbeiten auch die sogenannten Smart Labels mit meist 13,5 MHz.

- **Hohe Frequenzen von 850 - 950 MHz, 2,4 - 2,5 GHz sowie 5,8 GH**
  Diese Systeme besitzen hohe Reichweiten bis max. 30 Meter (nur aktiv). Sie haben schnelle Lesegeschindigkeiten. Allerdings steigen die Preise rapide bei höherer Leistung der Systeme. Diese Systeme kommen z.B. im Bereich der automatisierten Mautsysteme zum Einsatz oder bei der Güterwagenidentifikation. Typische Frequenzen sind 433 MHz, 868 MHz, 915 MHz, 2,45 GHz (Industrial, Science and Medicine = ISM) und 5,8 GHz.

  Beispiele zur Erläuterung: Wenn ein Transpondersystem zur Identifikation von Mehrwegcontainern – wie Gasflaschen, IBCs (Industrial Bulk Containers) oder ULDs (Unit Load Devices) – eingesetzt werden soll, muss dieses System über eine hohe Reichweite verfügen, auf Metall eingesetzt werden und ganze Objektgruppen erkennen können. Hier bietet sich ein **2,45-GHz-Produkt** an.

	Niedrige Frequenzen	Mittlere Frequenzen	Hohe Frequenzen
	30-500 KHz	10-15 MHz	850-950 MHz 2.4-2.5 GHz 5.86 GHz
RFID Frequenzen	100-135 KHz	13.56 MHz	2.45 GHz
Reichweite	bis 120 cm	3-11 m	max. 30 m
Energieversorgung	passiv	passiv	(semi)passiv, aktiv
Lebensdauer	abhängig von Belastung	abhängig von Belastung	bis 10 Jahren, nutzungs-unabhängig
Geswindigkeit	3 m/s	3 m/s	3 m/s
Bauformen	vielfältige	Folien, laminierbar	vielfältige
Leserichtung	kreisförming	abhängig Antenne	gerichtet
Pulkfähigkeit	realisiert	16/s, unbegrenzt	12/s, unbegrenzt
Materialdurchdringung	hohe Eindringungstiefe	hohe Eindringungstiefe	materialabhängig
Einsatz auf Metall	beschränkt	beschränkt	sehr gut
Beschreibbarkeit	realisirt	realisirt	realisirt
		Smart Labels	

Abbildung 2.12: Eigenschaften unterschiedlicher RFID-Frequenzbänder

Sollen dagegen Bücher identifiziert werden, ist die Reichweite weniger wichtig als die Bauform. diese sollte dagegen flach und in Form einer Klebefolie realisiert werden können. Auch in diesem Beispiel ist die Gruppenerkennung von Bedeutung. Es würde daher ein **13,56-MHz-Produkt** in Frage kommen.

Quelle: I.D. SystemsAG, Köln; www.idsystems-ag.de

Arten der Anwendung Als Grundprinzip gibt es die 3-Pear-Anwendung:

- Tag
- Lesegerät
- Server

RFID-Anwendungen werden verwendet, um mobile Güter zu orten:

1. Tag im mobilen Gut
   Infrastrukturbasierte Ortung

2. Lesegeräte im mobilen Gut z.B. Museumsanwendungen
   Terminalbasierte Ortung

2.2.4 Betriebssysteme für die Smart Identification

1. **Java Card** Die Java Card Plattform ist ein Betriebssystem für die Smart Cards, das in Java geschriebene Anwendungen ermöglicht, die auf der Smart Card abgearbeitet werden. Die Java Card Spezifikation Version 2.1.1 wurde im Mai 2000 vom Java Card Forum vorgestellt, die Rechte daran hat Sun. Die Java-Programme, die „on-card“ ausgeführt werden können, heißen Card Applets und bestehen aus einem Java Card spezifischen Bytecode, der durch die Java Card Runtime Environment ausgeführt werden kann.

   Das Ziel besteht darin, dass die Java Card Applets in jeder Java Card laufen. Es ist leider noch nicht ganz erreicht, da aktuelle Implementierungen noch leicht voneinander abweichen. Der Java Card Software Stack sieht dabei wie folgt aus:

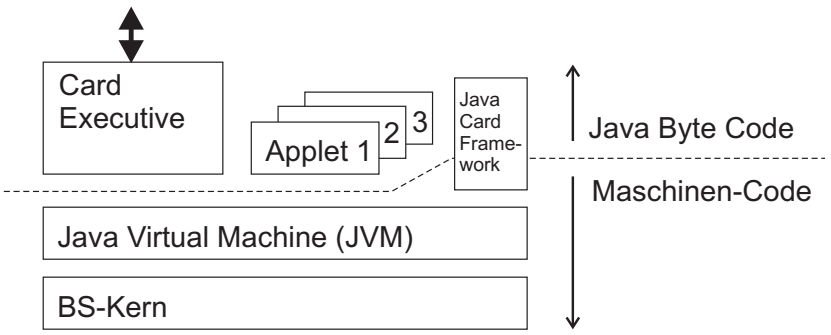


Abbildung 2.13: Aufbau des Chips einer Smart Card

   - Das Modul Card Executive ist die Schnittstelle nach außen zum Off-Card-Code.
   - Die JVM führt den Bytecode der Applets und der Bibliotheksfunktionen, die benötigt werden, aus.
   - Das Java Card Framework stellt Bibliotheksfunktionen zur Verfügung.
   - Die Applets haben theoretisch eine unendlich lange Lebenszeit. Sie müssen ansonsten explizit deinstalliert werden.

2. **Windows für Smart Cards** 1999 hat Microsoft sein Windows für Smart Card Betriebssysteme vorgestellt. Dieses stellt eine wohldefinierte API zur Verfügung, die alle prozessorspezifischen Details verbirgt und eine kartenunabhängige Funktionalität bereitstellt.

   In Analogie zu Java Card wird die Entwicklung einer Kartenanwendung durch eine höhere Programmiersprache unterstützt.

   Allerdings wird nicht Java verwendet, sondern ein Byte Code basierend auf Visual Basic, der in einer Runtime Environment auf der Karte ausgeführt wird.

Des weiteren besteht eine Analogie zum Microsoft-Produkt Windows CE: das Betriebssystem wird kundenspezifisch für spezielle Prozessoren zusammengestellt. D.h. der Kartenhersteller ist in der Lage zu konfigurieren, welche der bereitgestellten Features er in sein Betriebssystem integrieren will.

Die auf der Karte gespeicherten Applets werden - wie bei den Java Cards - auf der Karte selbst ausgeführt.

2.3 Embedded Systems

Embedded Systems kommen z.B. im intelligenten Haus und im Auto zur Anwendung

2.3.1 Intelligentes Haus

Im Netz sind **Sensoren** und **Aktoren** vorhanden.
Sensoren sammeln Informationen aus ihrer Umgebung und sind mit einem Netzwerk verbunden. Das Netzwerk kann verdrahtet (wired) oder drahtlos (wireless) sein.
Aktoren nehmen Signale vom Netz auf und agieren unmittelbar in ihrer Umgebung.
Sensoren, Aktoren und CPU werden auch unter dem Begriff Smart Control Steuerelemente (Intelligente Steuerelemente) zusammengefasst. Das Grundprinzip funktioniert wie in Abbildung 2.14 dargestellt.

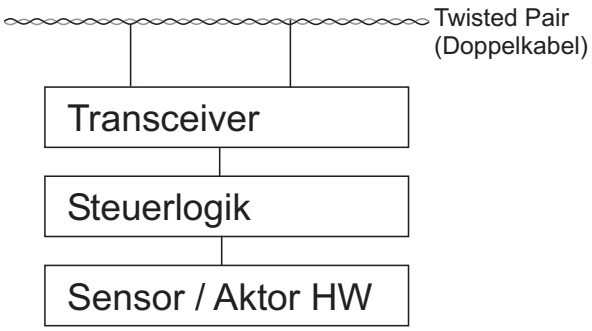


Abbildung 2.14: Aufbau eines Sensor-/ Aktor-Netzwerkes

Der **Transceiver** (Sende-/ Empfangseinheit) verbindet das Steuerelement mit dem Netzwerk. Er erzeugt bzw. interpretiert die physikalischen Signale fürs bzw. vom Netz und führt Datenprotokolle der unteren Schichten aus.
Die **Steuerlogik** (Control Logic) ist in der Regel ein Mikroprozessor, der dem Gerät quasi die Intelligenz liefert. Sie interpretiert die Anweisungen, die über das Netz erhalten werden, verarbeitet sie und sendet ggf. entsprechende Antworten zurück. Bei der Steuersignalverarbeitung wird der Aktor in der Regel ausgeführt. Bei einer asynchronen Nachrichtenverarbeitung können außerdem Alarmnachrichten erzeugt werden, wenn kritische Situationen auftreten.
Ein Beispiel für die Steuerung eines Intelligenten Hauses ist der Busch-Jaeger-Elektroinstallationsbus EIB (www.busch-jaeger.de).

2.3.2 Elektronik im Auto

Laut Angeben der Firma BMW stecken **40% der Herstellungskosten** eines Fahrzeugs **in der Elektronik und Steuerung**. Dabei existieren im Fahrzeug bis zu 70 Steuergeräte, deren Entwicklung von 2200 Mitarbeitern der Business IT betrieben wird. (Zitat BMW: „Wir suchen aber nicht den reinen Informatiker, sondern den Experten mit Verständnis beider Welten.“)
Computer, Steuergeräte und In-car-Networks arbeiten im Auto zusammen, um das Fahren einfacher, sicherer und komfortabler zu gestalten.
Die zentrale Architektur ist dabei das CAN, das sogenannte **CarAreaNetwork** oder auch Controller Area Network. Beim neuen **BMW 5er** kommunizieren dabei **48 Steuergeräte** über eine Busarchitektur, in der **über 500 Nachrichten** in Form so genannter **Bordnetztelegramme** versendet werden können. Dieser Bus ist relativ schmalbandig.
Für die gewünschte Steuerung sind **4500 Parameter per Codierung einstellbar** – und zwar werden diese im **EEPROM** gespeichert. Ein Beispiel wäre eine Einstellung für Fahrzeuge im skandinavischen Raum, das Abblendlicht bei laufendem Motor immer eingeschaltet zu haben.
Die Software für die Fahrzeugsteuerung wird auf **Flashspeichern** gespeichert. Dabei fallen **für alle Steuergräte zusammen 80 MByte an Daten** an.
Am Bus ist eine Datenübertragung mit 1 KByte/s möglich. Diese sind sicherheitsrelevante Informationen.
Beispiele für Steuergeräte (Aktoren / Sensoren), die am CAN anliegen sind

- Türelektronik
- Klimaanlage / Heizungsmodul / Sitzheizung
- Einparkhilfe (PDC)
- Regensensor / Scheibenwischermodul
- Alarmanlage
- Schiebedach
- Car Communication Computer

Neben dem CAN gibt es noch eine weiteres Netz: das **Media Oriented System Transport (MOST)-Netz** für breitbandige Informationen.

Dabei kommen **Lichtwellenleiter** für die **Übertragung von Video- und Audiodaten** zum Einsatz.

Angeschlossene Geräte sind z.B. **Telefoninterface, Schnittstelle für Spracheingabe, Audio-CD-Wechsler, Kopfhöreranschluss, Antennensteuerung, Navigationssystem.**
Die Steuereinheit hat im Auto folgenden prinzipiellen Aufbau:

2.4 Entertainment Systems

Moderne Haushalte sind in der Regel an verschiedene Netze angeschlossen: Telefonnetz / Internet auf der einen Seite und Radio / TV über Kabel, Satellit oder die Luftschnittstelle auf der anderen Seite.

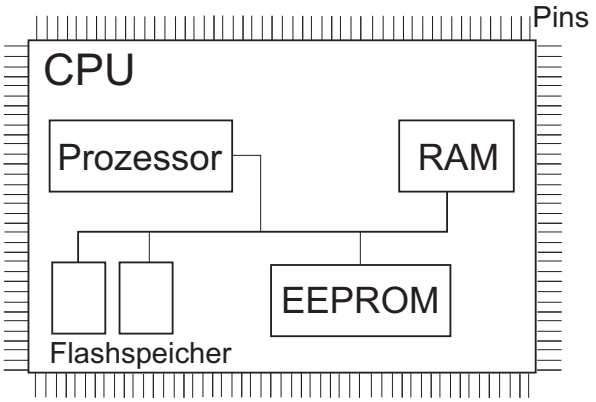


Abbildung 2.15: Aufbau der Steuereinheit im Auto

Während über das Telefonnetz / Internet eine bidirektionale Kommunikation von / zum Kunden möglich ist, haben wir bei Radio / TV eine unidirektionale Kommunikation via Broadcasting.
Es liegt auf der Hand, dass diese beiden Netze zusammenwachsen werden.

2.4.1 Digital Video Broadcasting

Das Digital Video Broadcasting (DVB) Projekt startete 1993 mit dem Ziel, das Broadcasten der Fernsehprogramme in eine digitale Übertragung zu transformieren

DVB-T (Terrestrial) Mit DVB-T soll der Fernsehempfang über die terrestrische Antenne auf einen digitalen Standard umgestellt werden. Bis 2010 muss diese Umstellung vollzogen sein.
Vorteile für den Nutzer sind zusätzliche Programmangebote bei insgesamt 20-30 Programmen, lokale und regionale Inhalte sowie begleitende Datendienste bei deutlich besserer Empfangsqualität als beim heutigen analogen Fernsehen. Zum Empfang von DVB-T kann die vorhandene Antenne weiter verwendet werden – oder auch nur eine kleine Stabantenne. Ein weiterer Vorteil liegt im portablen und mobilen Empfang. Das heißt, digitale Fernsehbilder oder Datendienste können auch von kleinen Minifernsehern im Auto oder von Taschencomputern über eine bleistiftgroße Stabantenne empfangen werden. So kann der Fernseher dann wie ein Kofferradio an einer beliebigen Stelle im Haus aufgestellt werden.
Bei DVB-T werden - wie auch im sonstigen Digital-TV - die Video- und Audiosignale mittels **MPEG2-Standard** komprimiert. Dadurch lassen sich auf einem TV-Kanal anstelle eines Programms bis zu vier Programme ausstrahlen, indem ein Multiplex-Bitstrom verwendet wird. Bei Geräten mit integrierter Festplatte ist eine direkte Aufzeichnung des digitalen Signals ohne erneute Kompression möglich.
Für den PC gibt es DVB-T-PCI-Karten, für unterwegs DVB-T-USB-Boxen, die i.d.R. noch ein separate Stromversorgung erfordern.
Zum Empfang von DVB-T ist ein digitaler Empfänger erforderlich, der als Set Top Box erhältlich ist.

DVB-C (Cable) Die ersten Breitband Kabelnetze entstanden in Europa vor mehr als 15 Jahren. Diese Technik war ursprünglich auf 30-50 Programme ausgelegt. Jetzt wird daran gearbeitet, sie auf digitale Technik umzurüsten - dann können bis zu 500 Programme gleichzeitig in gleich bleibend guter Qualität übertragen werden.
Neben Fernsehprogrammen sind Daten übertragbar, Mulimedia-Anwendungen, Internet und Telefon (mit Rückkanal ist dann ein interaktives Fernsehen möglich.)

DVB-S (Satellit) Mit einer einzigen Satellitenschüssel sind schon jetzt über 1000 TV- und Radioprogramme digital über Satellit empfangbar. Daneben ist es möglich, auch Internetinhalte und Multimediadienste abzurufen.

DVB-H (Handheld) DVB-H ist die jüngste der DVB-Entwicklungen und zielt auf einen Empfang für **batteriebetriebene, kleine Endgeräte** ab wie z.B. Handys, PDAs u.a.
Problematisch ist dabei die Batterielebenszeit. Aus diesem Grund wird der Stromverbrauch dahingehend reduziert, dass **Time Slicing** eingeführt wird:
Im Gegensatz zu den kontinuierlichen Datenübertragungen bei DVB-T verwendet DVB-H einen Mechanismus, bei dem Bursts (Haufen) von Daten zu gewissen Zeiten empfangen werden. Dies wird **IP Datacast Carousel** genannt. Dabei ist der Receiver die meiste Zeit inaktiv und kann so **90% der Energie sparen**.
DVB-H kann mit DVB-T koexistieren, da beide in geeigneter Weise durch **dasselbe Multiplexverfahren** verarbeitet werden.
In Berlin wird ein DVB-H-Versuchsfeld aufgebaut. Parallel dazu wird die Standardisierung weiter vorangetrieben.

2.4.2 Multimedia Home Platform

Das Ziel der Multimedia Home Platform (MHP) besteht in der Realisierung einer Schnittstelle zwischen digitalem TV-Receiver und einem Netz mit bidirektionalen Kommunikationsmöglichkeiten, so dass interaktive Dienste möglich sind.
Da MHP aus dem DVB gewachsen ist, nutzt es die definierten DVB-Protokolle und **ermöglicht** den Herstellern **die Entwicklung einheitlicher Endprodukte, die gleichermaßen auf DVB-T, -C und -S laufen**.
Falls MHP auf einer anderen Plattform als DVB laufen soll, so kann auf Globally Executable MHP (GEM) zurückgegriffen werden.
Die Fähigkeit zur Interaktion kann entweder im TV-Gerät selbst implementiert werden, im PC realisiert werden oder aber separat in einer so genannten Set-Top-Box.
Damit unterstützen Set-top-boxes multimediale Spiele, http-Browsing und ermöglichen Ein- und Ausgaben über einen USB-Port, CR ROM, Tastatur und Telefonmodem.
Der prinzipielle Aufbau ist wie folgt:

2.5 Connectivity

Es gibt zahlreiche Möglichkeiten der Kommunikation mobiler Endgeräte mit anderen Objekten. Zwecks Vertiefung dieser Thematik soll auf entsprechende Vorlesungen wie Mobilkommunikation verwiesen werden.
Mobile Endgeräte können für den Weitverkehrsbereichkommunizieren über

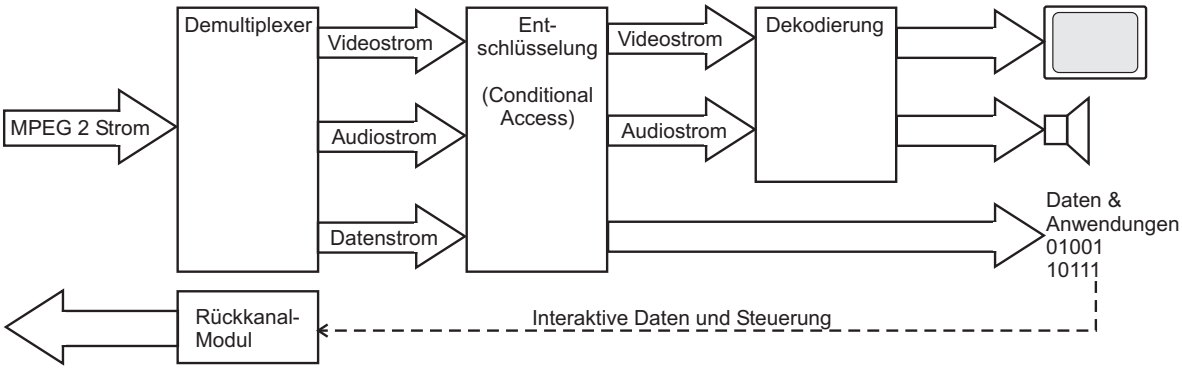


Abbildung 2.16: Schematische Darstellung von DVB

- GSM
- GPRS und
- UMTS
- WLAN

Im Nahverkehrsbereich stehen zur Verfügung

- DECT
- Bluetooth
- IrDA (Infrarot).

3

Kommunikation in Verteilten Systemen

►

Inhaltsangabe

Coulouris/Dollimore/Kindberg definieren ein Verteiltes System als ein System, in dem Komponenten auf vernetzten Computern dadurch miteinander kommunizieren und ihre Aktionen koordinieren, indem sie Message Passing durchführen, d.h. Nachrichten weiterleiten. Alle Verteilten Systeme haben gemein, dass sie miteinander verbundene Computer bzw. Prozessoren umfassen, die räumlich voneinander getrennt sind.
Diese Definition führt zu drei Charakteristiken eines Verteilten Systems:

1. **Nebenläufigkeit von Komponenten / Parallelität von Prozessen**
   bedingt höheren Durchsatz, kürzere Wartezeiten und damit auch kürzere Antwortzeiten. Es können sich ggf. Möglichkeiten der Echtzeitverarbeitung ergeben (Real Time).

2. **Unabhängige Fehler in den Komponenten**
   Durch die Verteilungstransparenz kann ein Fehler oft verschiedene Ursachen haben. Denn: Die positive Eigenschaft der Ausfalltransparenz hat nachteilig zur Folge, dass Komponentenfehler oft nicht erkannt bzw. nicht eindeutig zugeordnet werden können.

3. **Fehlen einer globalen Uhr**
   Beispiel: Satellitennavigation (vgl. Kapitel 7.1)

Die unabhängigen Komponenten bewirken bei einem Ausfall einer Komponente eine höhere Fehlertolleranz des Gesammtsystems. Auf verschiedenen Rechnern können Programme parallel abgearbeitet werden, wobei Ressourcen wie Web Pages oder Dateien gemeinsam genutzt werden. Die Kapazität dieser Ressourcen kann durch Hinzunahme von Ressourcen noch erhöht werden. Beispiele sind Rechner, Speicherplatz und Drucker. Die Kooperation von Programmen über Message Passing stößt oft auf Zeitstempel, d.h. benötigt die Uhrzeit, zu der eine Aktion ausgeführt wird. Innerhalb eines Netzes gibt es in der Regel jedoch keine korrekte oder allgemein gültige Zeit. Deshalb müssen Uhren synchronisiert werden, wofür lediglich das Senden von Nachrichten als Hilfsmittel genutzt werden kann. Damit es nicht zu Deadlocks kommen kann, muss eine Komponente eine Resource erst freigeben, bevor sie eine andere anfordern darf.
Jeder Rechner kann ausfallen, aber innerhalb eines Verteilen System sind Komponentenausfälle von ganz neuer Qualität mit ganz neuen Konsequenzen. Insbesondere unterscheidet sich die Sichtbarkeit des Ausfalls eines Einzelrechners in einem Verteilten System signifikant vom Ausfall eines einzelnen, nicht vernetzten Rechners. Beispiele von Verteilten Systemen sind Intranets, das Internet oder mobile bzw. ubiquitäre Systeme. Detaillierter betrachtet sind darunter auch zu verstehen Mobilfunknetze, Firmennetze, Campusnetze, In-car-Networks innerhalb eines Autos usw.

3.1 Beispielszenarien

Exemplarisch sollen einige der Verteilten Systeme betrachtet werden.

Das Internet Das Internet ist ein riesiges Netz unterschiedlichster, miteinander verbundener Rechner. Mitte der 60er Jahre - auf der Höhe des kalten Krieges - wollte das US-Verteidigungsministerium (DoD, Department of Defense) ein Kommando- und Steuernetz aufbauen, das einen Atomkrieg überleben könnte. Die konventionellen leitungsvermittelnden Telefonnetze galten als zu verletzlich, da der Ausfall einer Leitung oder eines Vermittlers mit Sicherheit

alle Gespräche beenden würde, die über diese oder jenen laufen. Um dieses Problem zu lösen, wandte sich das Ministerium an seinen Forschungsbereich ARPA, die Advanced Research Projects Agency.
Die ARPA war wiederum als Reaktion auf den Start des Sputniks durch die Sowjetunion im Jahr 1957 mit der Aufgabe ins Leben gerufen worden, ausgefeilte Technologien zu entwickeln, die für das Militär von Nutzen sein würden. Die ARPA hatte weder Wissenschaftler, noch Labors. Sie bestand aus nichts als einem Büro und einem - nach Pentagon-Maßstäben - kleinem Budget. Sie bestritt ihre Aufgabe durch die Ausschreibung von Verträgen an Universitäten und Unternehmen, deren Ideen viel versprechend aussahen.
So gingen verschiedene Verträge anfänglich an Universitäten für die Erforschung der damals als radikale Vorstellung geltenden Paketvermittlung, die Anfang der 60er Jahre veröffentlicht worden war. Und so entschied die ARPA, das Netz für das US-Verteidigungsministerium als paketvermitteltes Netz zu realisieren, das aus einem Teilnetz und Hostrechnern bestehen sollte.
Das Teilnetz wiederum sollte aus Minicomputern namens IMP, d.h. Interface Message Processors, bestehen, die über Übertragungsleitungen zusammengeschlossen wurden. Um eine hohe Zuverlässigkeit sicherzustellen, sollte dabei jeder IMP an mindestens zwei IMPs angeschlossen werden. Falls einige Leitungen und IMPs zerstört würden, sollten Nachrichten automatisch über Ausweich- oder Schutzpfade geführt werden. An einen IMP konnte höchstens ein Host angeschlossen werden.
1968 wurde nach entsprechender Ausschreibung der Zuschlag für die Entwicklung des Teilnetzes und der dazugehörenden Software erteilt, wobei dieser zweigeteilt wurde - in Software für das Teilnetz sowie den Anschluss der Hosts und in Software außerhalb des Teilnetzes, d.h. die Anwendungssoftware für die Host/Host-Kommunikation.
So entstand im Dezember 1969 ein experimentelles Netz mit Knoten an vier Universitäten, die alle eine große Anzahl von ARPA-Verträgen hatten. Danach wuchs das Netz schnell, und immer mehr IMPs wurden ausgeliefert und installiert.
Später wurde die IMP-Software geändert. Mehrere Hosts pro IMP wurden möglich sowie Hosts, die mit mehreren IMPs kommunizierten. Satelliten und mobile Hosts wurden getestet. Das Experiment zeigte, dass die vorhandenen ARPA-Protokolle nicht für den Betrieb über mehrere Netze geeignet waren. Bei der Suche nach neuen Protokollen wurde 1974 das TCP/-IP-Modell erfunden, das zur Abarbeitung und Übertragung über mehrere verbundene Netze entwickelt wurde.
Wissenschaftliche Mitarbeiter an der Berkeley-Universität entwickelten eine bequeme Programmierschnittstelle zum Netz, die Sockets, und schrieben viele Anwendungen sowie Programme, um die Vernetzungsaufgaben zu vereinfachen. So wurde es einfach, bestehende LANs mittels TCP/IP an das ARPANET anzuschließen.
1983 war dieses Netz stabil und mit über 200 IMPs und Hunderten von Hosts auch sehr erfolgreich. An diesem Punkt übergab die ARPA das Management des Netzes der DCA (Defense Communications Agency), die den militärischen Teil mit 110 IMPs in den USA sowie weiteren 50 IMPs in anderen Ländern in ein separates Netz, das MILNET, abtrennte.
Die Netze, Maschinen und Nutzer, die an das ARPANET angeschlossen wurden, nahmen rapide zu, nachdem TCP/IP am 1. Januar 1983 das einzige offizielle Protokoll wurde. Hinzu kam, dass mit dem NSFNET (NSF- US National Science Foundation) ein Zusammenschluss erfolgte. NFS war dabei ein Netz, das primär für die Unis und Forschungseinrichtungen gedacht war, die keinen Forschungsvertrag mit der ARPA hatten und sich damit nicht in das ARPANET einklinken durften. Nun nahm der Umfang des Netzes expotentiell zu. Verbindungen wurden

auch mit Kanada, Europa und dem pazifischen Raum hergestellt.
Obwohl es nie eine offizielle Einweihung gab, wurde die entstehende Sammlung von Netzen etwa ab Mitte der 80er Jahre als ein Netzverbund betrachtet - das Internet. 1990 zählte es 3.000 Netze mit 200.000 Rechnern; 1992 wurde der Millionste Host angeschlossen; die Größe des Internets verdoppelte sich Mitte der 90er Jahre etwa jedes Jahr.
Das Bindeglied, das das Internet zusammenhält, ist dabei das TCP/IP-Modell mit dem TCP/IP-Protokollstapel. Was bedeutet es eigentlich, am Internet zu hängen? Ein Rechner hängt am Internet, wenn er TCP/IP ausführt, eine IP-Adresse hat und die Fähigkeit, IP-Pakete an alle anderen Rechner im Internet zu senden. Traditionell umfasst das Internet vier Hauptanwendungen:

- E-Mail: Die Möglichkeit, elektronisch Post zu schreiben, zu senden und zu empfangen
- News: Spezielle Foren, in denen Benutzer mit gemeinsamen Interessen Nachrichten austauschen können
- Remote Login: Über Telnet, rlogin oder andere Programme können sich Internetnutzer von irgendwo an einem anderen Computer anmelden, für den sie Zugriffsrechte haben
- File Transfer: Mit dem FTP-Programm können Dateien von einer Maschine zu einer anderen Maschine im Internet kopiert werden. Auf diese Weise sind riesige Mengen an Artikeln, Datenbanken und anderen Informationen verfügbar.

Während bis Anfang der 90er Jahre das Internet hauptsächlich Wissenschaftlern vorbehalten war, änderte eine neue Anwendung des Internets, das World Wide Web (kurz WWW), alles mit einem Schlag und brachte Millionen von Nutzern außerhalb des akademischen Bereiches ins Netz. Dabei wurde keine zugrunde liegende Einrichtung geändert, sondern lediglich eine nutzerfreundlichere Bedienung ermöglicht.
Programme werden auf den Computern ausgeführt, die über Message Passing miteinander kommunizieren können. Dabei kann ein Programm quasi überall (anywhere) abgearbeitet werden und Nachrichten an jedes andere Programm (anywhere else) schicken. Das Internet ermöglicht die Nutzung verschiedener Dienste: E-mail, Datei Transfer (File Transfer), World Wide Web. Tatsächlich wird im umgangssprachlichen Gebrauch das Internet oft mit dem World Wide Web gleichgesetzt.
Die Abbildung 3.1 zeigt das Internet als Zusammenhang von Intranets, d.h. Teilnetzen, die von Firmen oder Organisationen betrieben werden. Internet Service Provider (ISP) stellen Modemverbindungen an einzelne Nutzer oder kleinere Unternehmen bereit. Die Intranets sind über Backbones miteinander verbunden, d.h. über Netzverbindungen mit einer hohen Übertragungskapazität wie z.B. optische Fasern oder Satellitenverbindungen mit hoher Bandbreite.
Das Internet ist etwas anderes als das www. Letzteres ist ein Anwendungsdienst auf dem Internet. Bei neuen Technologien wie z.B. dem Internet dauer es länger, bis diese Technologie sich durchsetzt, als man denkt.

Das Intranet Als Intranet wird ein begrenzter Teilbereich des Internets bezeichnet, der separat administriert wird und lokalen Sicherheitspolicies genügt.
Die Abbildung 3.2 zeigt ein Intranet, das aus verschiedenen Local Area Networks (LANs) besteht, die über Backbones miteinander verbunden sind. Über einen Router ist das Intranet

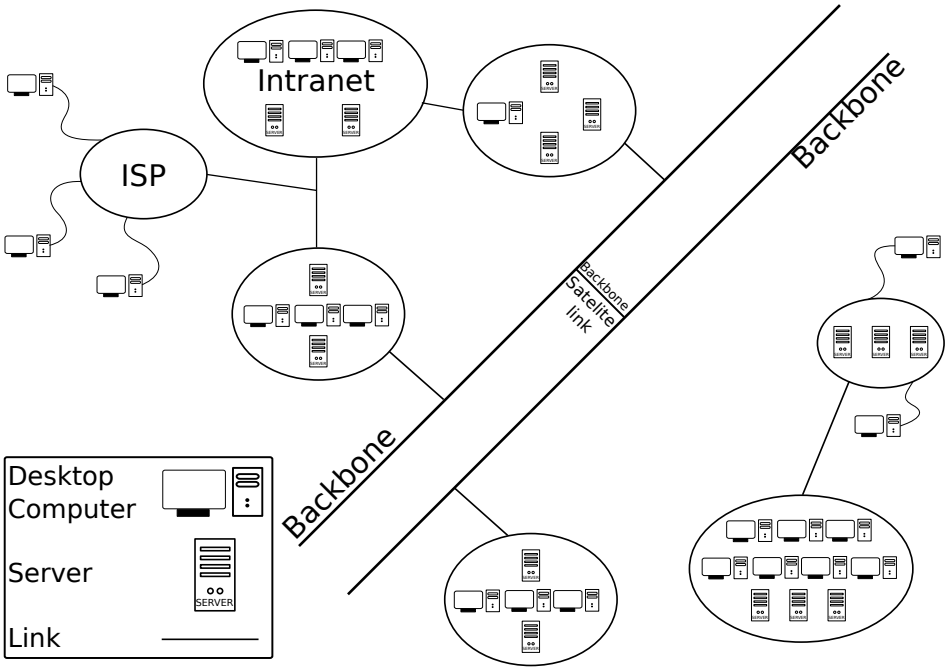


Abbildung 3.1: Beispiel für einen Bereich des Internets

mit dem Internet verbunden. Damit können weitere Dienste angeboten und genutzt werden. Allerdings ist ein Schutz vor unautorisierter Nutzung und Missbrauch nötig!
Eine Firewall dient dem Schutz des Intranets in dem Sinne, daß der Ein- und Ausgang unautorisierter Nachrichten verboten wird. Implementierungsseitig werden ein- und ausgehende Nachrichten danach gefiltert, woher sie kommen und wohin sie gehen.
Ganz strenge Sicherheitsvorschriften können nur dadurch umgesetzt werden, daß ein Rechner oder Teilsystem gar nicht ans Netz angeschlossen wird, z.B. bei militärischen Organisationen insbesondere in Kriegszeiten.

Ubiquitous Computing Die technologischen Fortschritte in der Geräteminiaturisierung und der drahtlosen Kommunikation haben zu einer Integration kleiner und tragbarer Kommunikationsgeräte in die Verteilten Systeme geführt.
Mobile Computing umfasst die Ausführung von Prozessen, während sich der Nutzer bewegt oder außerhalb seiner gewohnten Umgebung, d.h. seines Heimatintranets, aufhält. Ubiquitous Computing zielt insbesondere auf kleine, billige Geräte ab, die überall verfügbar, d.h. allgegenwärtig sind (vgl. Abbildung 3.3).
Eine andere Möglichkeit ist es die Netzt nach Reichweite zu klassifizieren: WAN (Wide Area Network) - LAN (Local Area Network) - PAN (Personal Area Network, z.B. Bloototh).

3.2 Das Client/Server-Prinzip

„Client“ und „Server“ sind Rollen. Sie konnen nach Definition wechseln, sind kontextabhängig, temporär, charakterisieren die Stellung zu einem Dienst und dessen Erbringung. Eine der

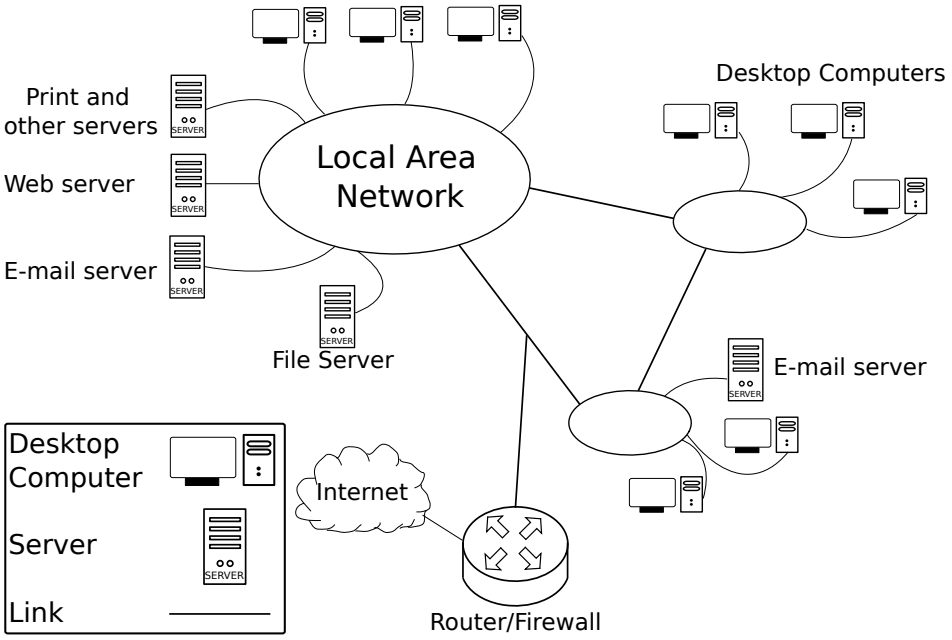


Abbildung 3.2: Beispiel für ein Intranet

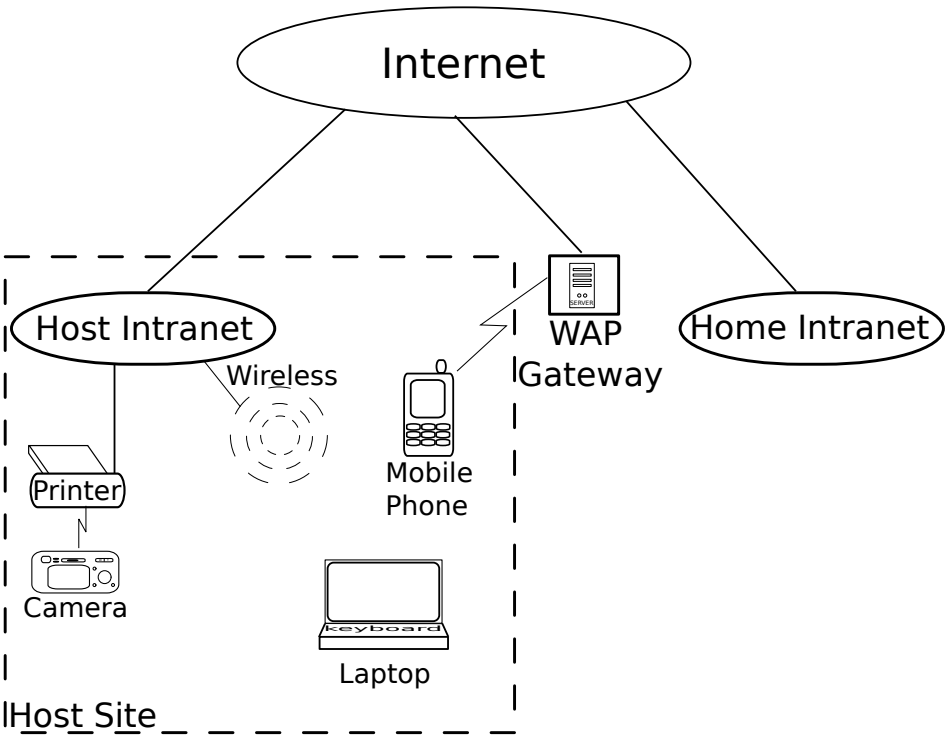


Abbildung 3.3: Beispiel eines einfachen ubiquitären Systems

wesentlichen Motivationen für das Entstehen Verteilter Systeme ist das gemeinsame Nutzen von Ressourcen, das auch als „Resource Sharing“ bezeichnet wird.
Hardware Ressourcen wie z.B. Scanner, Disketten und Drucker werden ganz selbstverständlich genutzt, Datenressourcen wie Dateien, Web-Seiten oder Programme wie z.B. Suchmaschinen können allerdings auch gemeinsam genutzt werden. Dieses Teilen spart Kosten. Im Folgenden soll zu Ressourcen auf höheren Ebenen übergegangen werden.
An dieser Stelle soll auf einem abstrakteren Level der Begriff des Dienstes eingeführt werden. Ein **Dienst** ist ein bestimmter Teil eines Computersystems, der eine Ansammlung von miteinander in Beziehung stehenden Ressourcen verwaltet und ihre Funktionalität Nutzern und Anwendungen bereitstellt.
Z.B. wird auf gemeinsam genutzte Dateien durch einen File Service zugegriffen. Dokumente werden mittels eines Print Services auf Druckern ausgedruckt. Güter werden auf elektronischem Wege über einen Electronic Payment Service gekauft. Dabei besteht der Zugriff auf den Dienst ausschließlich über die Menge der Operationen, die der Dienst exportiert. Bspw. stellt ein File Service die Operationen Read, Write und Delete zur Anwendung auf Dateien bereit. Diese Zurückführung der Dienst- bzw. Ressourcennutzung auf eine Menge wohldefinierter Operationen ist alltägliche Praxis, spiegelt aber auch die physikalische Organisation eines Verteilten Systems wieder. Ressourcen sind gekapselt und können mit Programmen verwaltet und über Kommunikationsschnittstellen in Anspruch genommen werden.
Unter einem **Server** versteht man ein laufendes Programm, d.h. einen Prozess auf einem vernetzten Computer, der Anfragen von Prozessen anderer Rechner zwecks Dienstausführung akzeptiert und geeignet beantwortet. Ein Server bietet einen Dienst an und ist in der Regel kontinuierlich laufend.
Die anfragenden Prozesse werden als **Clients** bezeichnet. Die Anfragen werden als Nachrichten gesendet, ebenso sind die Antworten ebenfalls Nachrichten. Der Client nimmt einen Dienst in Anspruch und ist in der Regel am Dienst temporär interessiert.
Eine Interaktion zwischen Client und Server von der Anfrage (Request) bis zur Antwort (Response) wird dabei als **Remote Invocation** bezeichnet.
Client und Server sind Rollen, die von Komponenten bei einer Anfrage angenommen werden. Sie können sich von Zeit zu Zeit ändern. Ebenso ist es möglich, daß eine Komponente sowohl Client als auch Server ist. Server sind dabei kontinuierlich laufend, man sagt auch, sie sind passiv. Clients sind aktiv und dauern nur so lange, wie die Anwendung, von der sie ein Teil sind. Client und Server sind Prozesse. Ist ein Verteiltes System in einer objektorientierten Sprache geschrieben, so können Ressourcen durch Objekte gekapselt werden, auf die von Clientobjekten aus zugegriffen wird. Man sagt dann auch, ein Clientobjekt fordert eine Methode, d.h. eine Funktion eines Serverobjekts an. Viele Verteilte Systeme können in Form interagierender Clients und Server konstruiert werden. WWW, E-Mail und vernetzte Drucker sind Beispiele, die in dieses Modell passen.
Das Web-Beispiel soll im Folgenden verdeutlicht werden:

Beispielstudie: Das World Wide Web Das WWW ist ein System für die Publikation und den Zugriff auf Ressourcen und Dienste innerhalb das Internets, z.B. Dokumente, Audio, Video und Dienste.
Auf der Clientseite wird ein Webbrowser ausgeführt. Allgemein verfügbare Software dazu ist z.B. Netscape und der Internet Explorer. Ressourcen im Web sind Web Pages oder andere Arten von Inhalt, sprich Content, der in einer Datei gespeichert und dem Nutzer präsentiert

werden kann, z.B. Programme, Audio, Video, Postscript- oder Portable Document Format (PS- oder PDF-Dokumente).
Ohne seine grundlegende Struktur zu ändern, hat sich das Web von der Daten - zur Dienstbereitstellung geändert. Beispiel für Dienste sind der elektronische Handel (Electronic Commerce) oder dynamische Aktieninformationen.
Das Web basiert auf einer Hypertextstruktur, d.h. Dokumente enthalten Links, also Referenzen auf andere Dokumente und Ressourcen des Webs. Das Web ist ein offenes System - d.h. basierend auf frei verfügbaren Standards kann es erweitert oder geändert werden.

Das Web basiert auf drei technologischen Komponenten:

1. **Die HyperText Markup Language (HTML)** Diese Sprache spezifiziert den Inhalt und das Layout von Webpages, wie sie durch Webbrowser angezeigt werden. HTML wird auch genutzt, um Links und die damit verbundenen Ressourcen zu definieren. HTML-Text wird in einer Datei gespeichert, auf die ein Webserver zugreifen kann. Ein Browser holt sich den Inhalt dieser Datei vom Webserver durch Aufruf einer Webseite. Er liest den Inhalt und übersetzt ihn in eine formatierte Webseite. D.h. nur der Browser - nicht aber der Webserver - interpretiert den HTML-Text. An der Dateicharakterisierung (File name extension) „.html“ erkennt der Browser die Art der Datei.

2. **Der Uniform Resource Locater (URL)** URLs identifizieren Dokumente und andere Ressourcen, die als Bestandteil des Webs gespeichert werden. Sie werden von den Browsern verwendet, um Ressourcen auf den Webservern zu lokalisieren. Der Browser ruft eine entsprechende Ressource auf, wenn ein Nutzer eine URL explizit im Browser angibt, wenn er sie über ein Bookmark aufruft oder aber über einen Link aktiviert.
   Jede URL besteht in ihrer allgemeinen Form aus zwei Komponenten:
   scheme: scheme-specific-location
   Dabei deklariert die erste Komponente, d.h. „scheme“, den Typ der URL.
   Beispiel:

   – mailto:linnhoff@informatik.uni-muenchen.de
   – ftp://ftp.downloadIt.com/software/aProg.exe
   – http://www.informatik.uni-muenche.de

   Das Web ist offen, aber nur soweit, wie der Zugriff auf Ressourcen unterstützt wird. D.h. wenn eine Firma ein neues Schema erfindet und das zugehörige Protokoll für den Zugriff darauf entwickelt und verbreitet/veröffentlicht, dann kann die ganze Welt über die URL
   Schema_neu : ...
   darauf zugreifen. Dies ist jedoch nur dann möglich, wenn die Browser das Schema_neu-Protokoll unterstützen bzw. über ein Plug-in diese Funktionalität bereitgestellt bekommen.
   Die URL hat zwei Aufgaben. Sie spezifiziert einerseits den Webserver, der eine Ressource bereitstellt. Andererseits identifiziert sie die eigentliche Ressource. Letzteres kann durch zusätzliche Angabe eines Pfadnamens oder eines Arguments geschehen. Vgl. auch Kapitel 4.

3. **Das HyperText Transfer Protocol (HTTP)** Das HyperText Transfer Protocol definiert die Art und Weise, in der Browser und andere Clients mit Webservern interagieren. An

dieser Stelle können nur die wesentlichen und grundlegenden Charakteristiken angedeutet werden:

- HTTP ist ein „request-reply“ -Protokoll. Der Client sendet eine Request-Nachricht an den Server. Diese Nachricht enthält die URL der angeforderten Ressource. Dabei benötigt der Server lediglich den Teil, der nach seinem eigenen Namen folgt. Sofern die referenzierte Datei existiert, sendet der Server deren Inhalt in einer Reply-Nachricht an den Client zurück. Existiert die Datei nicht, wird eine Fehlernachricht gesendet.
- Contenttypen werden unterstützt. Browser unterstützen in der Regel nicht jeden Contenttyp. In einer Browseranfrage kann eine Liste von präferierten Typen mit übertragen werden, z.B. Bilder im Gif-Format können unterstützt werden, nicht aber Bilder im Jpeg-Format. Der Server kann dies berücksichtigen, wenn er Inhalte an den Browser zurücksendet.
- In HTTP Version 1.0 fordert der Client genau eine Ressource pro HTTP-Anfrage an. Enthält eine Webseite bspw. 4 Bilder, so führt der Webbrowser insgesamt 5 separate Anfragen aus, um den Inhalt der Webpage vollständig darstellen zu können. Typischerweise können Browser verschiedene Anfragen nebenläufig ausführen, um die für den Nutzer entstehende Verzögerung so gering wie möglich zu halten.
- Als Default kann in der Regel jeder Nutzer auf jede Ressource zugreifen. Soll der Zugriff eingeschränkt werden, so müssen entsprechende Zugriffsrechte definiert werden, z.B. kann die Angabe eines Passworts verlangt werden.

Basierend auf diesen grundlegenden technologischen Eigenschaften stellt das Web ein paar fortgeschrittene Konzepte bereit:

Web Formen für dynamisch generierte Seiten Ein Web Formular ist eine Webseite, die Anwendungen für den Nutzer enthält, sowie Eingabemöglichkeiten wie Textfelder oder Checkboxes. Wenn der Nutzer das Formular einreicht, dann sendet der Browser eine HTTP-Anfrage an den Webserver, welche die vom Nutzer eingegebenen Werte enthält. Der Server muss den Input verarbeiten. Dazu spricht die URL ein Programm auf dem Server an - nicht eine Datei. Das Programm generiert einen HTML-Text, den der Server an den Browser zurückschickt.
So ist es für den Browser und letztendlich auch für den Nutzer transparent, ob der Inhalt einer Webpage statisch über eine Datei oder dynamisch über eine Parameterübergabe für ein Programm generiert wurde.
Das auf dem Server laufende Programm wird oft als Common Gateway Interface (CGI) Programm bezeichnet. Es besitzt anwendungsspezifische Funktionalität und produziert Inhalte eines bestimmten Typs (in der Regel HTML-Text) in Abhängigkeit der vom Client bereitgestellten Argumente. In der Regel wird eine Abfrage durch Auswertung einer Datenbank bearbeitet.
Mit einer Webseite kann auch in Javascript [3] geschriebener Code heruntergeladen werden, der innerhalb des Browsers läuft und für den Nutzer qualitativ höherwertige Interaktionen bereitstellt.
So können z.B. Teile der Webseite geupdatet werden, ohne eine vollständig neue Seite zu laden, oder im Falle von ungültigen Einträgen kann der Nutzer eine Fehlermeldung erhalten.

Applets Ein Applet ist eine Anwendung, die bei Anforderung einer Webseite automatisch geladen wird. Auf diese Applets wird später zurückgekommen.

3.3 Architekturmodell Verteilter Systeme

Ein Architekturmodell beschreibt, wie Komponenten eines Systems miteinander interagieren und welche Schichtenstruktur die dazu verwendete Software des Verteilten Systems besitzt. Die Softwarearchitektur eines Verteilten Systems soll als Schichtenmodell beschrieben werden, das auf jedem einzelnen Rechner vorhanden ist.

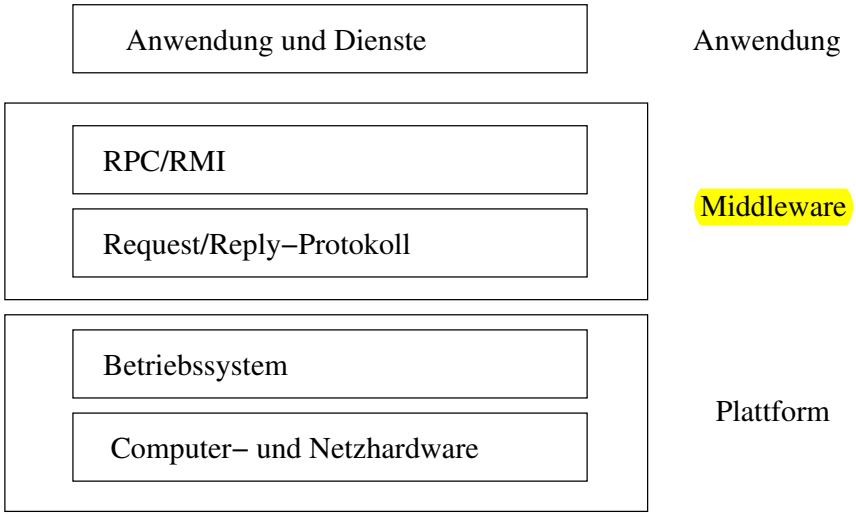


Abbildung 3.4: Schichtenmodell eines Verteilten Systems

Die Plattform umfasst die Hard- und Software auf den niedrigen Ebenen eines Computers, die im Kontext Verteilter Systeme gebraucht werden. Sie bietet der darüberliegenden Schicht Dienste an, ist aber in jedem Computer unabhängig realisiert.
Beispiele für Plattformen sind Intel486/Windows, SUN SPARC/Sun OS, Power PC/Mac OS, Intelx89/Linux u.s.w.
Unter der Middleware versteht man eine zwischen Betriebssystem, Entferntheit und Anwendung bei Verteilten Systemen neu hinzukommende Schicht, die Heterogenität verbergen und Verteilungstransparenz ermöglichen soll. Das Verstecken von Heterogenität bezieht sich im optimalen Fall auf

- das zugrunde liegende Netz
- die Hardware
- das Betriebssystem
- die Programmiersprache

In der Regel ist Middleware basierend auf dem Internet Protocol implementiert. Damit bleiben eventuelle Unterschiede in den zugrundeliegenden Netzen verborgen. Hardware- und

Betriebssystemheterogenität sollte von jeder Middleware abgedeckt werden. Wenn Microsoft Distributed Component Object Model (DCOM) lediglich auf Microsoftprodukte ausgerichtet ist, dann soll dieses Produkt nicht als Middleware im eigentlichen Sinne aufgefasst werden. Ein echtes Beispiel von Middleware ist dagegen CORBA. Diese Architektur unterstützt Verteilte Anwendungen in verschiedenen Programmiersprachen, die auf unterschiedlichen Betriebssystemen und unterschiedlicher Hardware laufen. Ein Beispiel ist in Abbildung 3.5 aufgeführt.

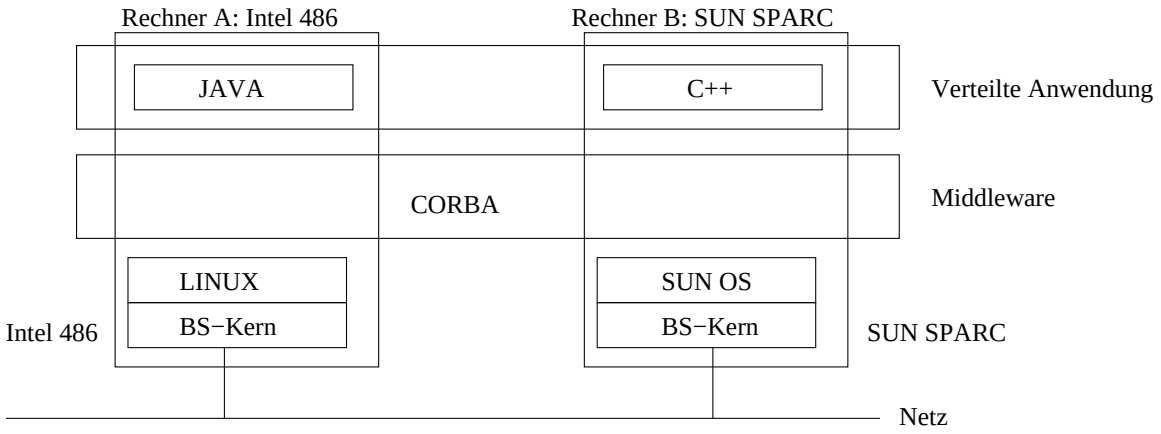


Abbildung 3.5: Beispiel für den Einsatz einer Middleware

Bezüglich der Programmiersprache unterstützt Java RMI beispielsweise lediglich eine einzige Programmiersprache.
Transparenz ist eine zentrale Anforderung, die den Umgang mit Verteilten Systemen so weit wie möglich erleichtern soll. Insbesondere umfasst die Verteilungstransparenz eine Reihe einzelner Aspekte:

- Zugriffstransparenz, die spezielle Zugriffsmechanismen für einen lokalen bzw. entfernten Dienstaufruf versteckt
- Ortstransparenz, die die Systemtopologie als solche verschattet,
- Migrations - und Replikationstransparenz halten Objektbewegungen bzw. -kopien verborgen
- Fehlertolleranz
- Ausfalltransparenz
- Abarbeitungstransparenz
- Gruppentransparenz

Hieraus resultiert eine Reihe von Anforderungen bzw. Aufgaben für die Middleware, die in der Regel durch eigene Dienste unterstützt werden.
Schließlich stellt Middleware noch eine Programmierabstraktion bzw. ein einheitliches Modell für den Programmierer bereit. Damit kann ein Programm, das auf einem Rechner abgearbeitet wird, auf einem anderen Rechner eine Prozedur aufrufen.

Die Middleware verbirgt, daß Nachrichten über das Netz gesendet werden, um Anfragen und Antworten zu senden. Dieses Modell wird auch als entfernter Objektaufruf oder entfernter Prozeduraufruf bezeichnet und ist in der oberen Teilschicht der Middleware implementiert.
Im Folgenden soll die Interaktion von Komponenten eines Verteilten Systems betrachtet werden. Dabei werden zwei grundsätzliche Varianten unterschieden: Das Client/Server-Modell und das Peer-to-Peer-Modell.

– **Client/Server-Modell**
  Client und Server sind unterschiedliche Rollen. Bei der Betrachtung Verteilter Systeme ist dies das meist genutzte Modell. Es treten zwei Arten von Prozessen auf : Client und Server. In seiner einfachsten Struktur besteht eine Interaktion zwischen verschiedenen Clients und einem Server, wobei die Clients Aufrufe an Serverprozesse schicken und als Antwort ein Ergebnis erhalten.

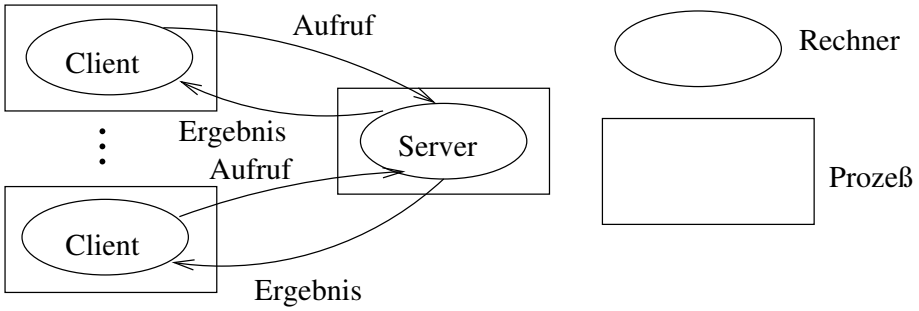


Abbildung 3.6: Client/Server-Modell

– **Peer-to-Peer-Modell**
  In diesem Modell existiert nur eine Art von Prozess - die Peers, die alle eine ähnliche Rolle einnehmen und kooperativ eine verteilte Aktivität realisieren. Es gibt quasi keinen Server. Im allgemeinsten Fall können n Peer-Prozesse miteinander interagieren.

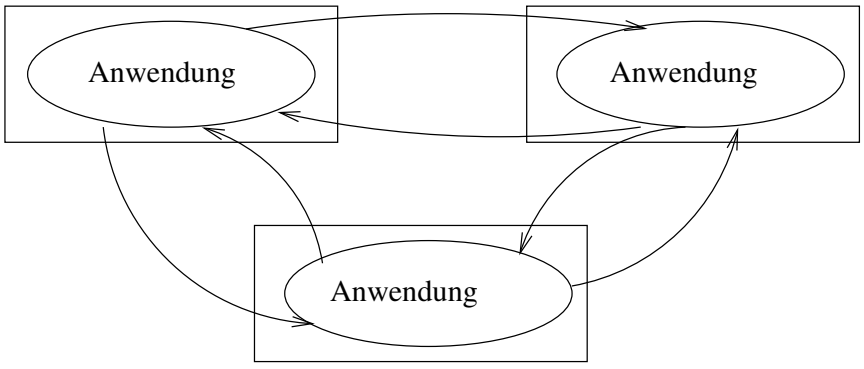


Abbildung 3.7: Peer-to-Peer-Modell

Als Beispiel kann man sich ein verteiltes Whiteboard vorstellen, d.h. eine Anwendung, die Nutzern auf verschiedenen Rechnern das Lesen und interaktive Modifizieren von Bildern, Text

und Daten ermöglicht. Aus Implementierungssicht wären die Informationen auf allen Rechnern redundant vorhanden. Änderungen würden über Ereignisse ebenfalls an alle Rechner verschickt werden, so daß zu jedem Zeitpunkt überall konsistente Versionen der Datenbestände vorhanden sind.
Im Folgenden soll auf das Client/Server-Modell (C/S) fokussiert werden. Dabei sind verschiedene Varianten möglich.

C/S-Variante 1: Rekursive Serveraufrufe Server können selbst wieder Clients anderer Server sein. Z.B. ist ein Webserver oft Client eines lokalen Fileservers, der die Dateien verwaltet, die den Inhalt von Webseiten speichern.

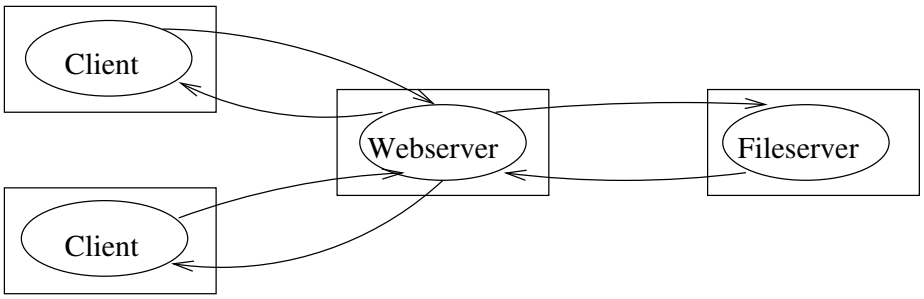


Abbildung 3.8: Rekursive Serveraufrufe

Viele Internetdienste sind auch Clients von Namensdiensten, die Internet Domain Names auf Netzadressen abbilden (vgl. Kapitel [**?**]).

C/S-Variante 2: Multiple Server Dienste können auch durch mehrere Serverprozesse auf verschiedenen Rechnern implementiert werden, die miteinander interagieren, um einen Dienst bereit zu stellen.

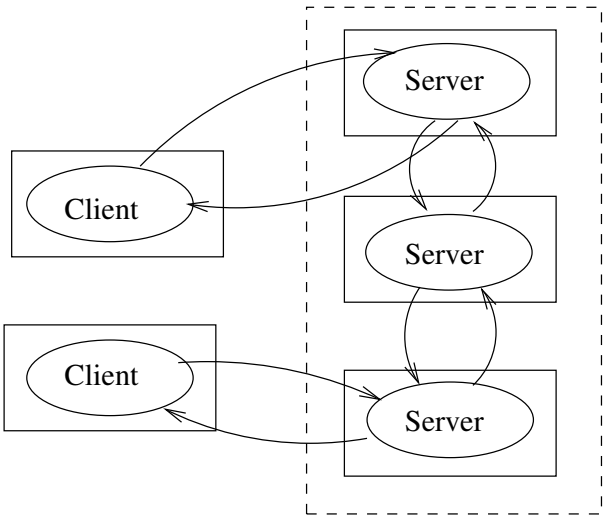


Abbildung 3.9: Multiple Server

Dabei kann die Menge der Objekte, die einen Dienst realisieren, auf verschiedene Server aufgeteilt werden. Alternativ können auch Replikate von Serverobjekten auf verschiedene Rechner kopiert werden. Dies nennt man vertielte Datenhaltung.
Das WWW ist ein Beispiel für die erste Möglichkeit - jeder Webserver verwaltet seine eigenen Ressourcen.
Alternativ können Datenbanken repliziert und konsistent verwaltet werden, um die Leistungsfähigkeit beim Zugriff zu erhöhen.

C/S-Variante 3: Proxy Server Wir wollen unsere Überlegungen mit dem Konzept des Caches beginnen. Ein Cache speichert gerade genutzte Datenobjekte zwischen, wobei der Zugriff auf den Cache schneller geht als auf die Datenobjekte selbst. Wird ein neues Datenobjekt benutzt, so wird es dem Cache hinzugefügt. Falls nötig werden dafür andere Datenobjekte ersetzt.
Ruft ein Client Daten auf, so wird zunächst der Cache auf die Existenz dieser Daten geprüft. Beinhaltet der Cache keine aktuelle Kopie der Daten, so wird eine solche erzeugt.
Caches replizieren bzw. duplizieren die Daten. Der Vorteil ist eine kürzere Antwortzeit und eine reduzierte Ausfallwahrscheinlichkeit. Der Nachteil ist die Konsistenz zu nennen, also die Datenbankaktualität.
Caches können jedem Client zugeordnet sein, bzw. in Proxyservern realisiert werden, auf die verschiedene Clients zugreifen können. In der Praxis werden die Caches sehr häufig verwendet. Als Beispiel soll ein Webserver betrachtet werden, der einen Cache der gerade abgerufenen Webseiten und anderer Webressourcen verwaltet. Dieser Cache wird im lokalen Dateisystem des Clients realisiert, wobei ein spezieller HTTP-Aufruf genutzt wird, um mit dem Webserver abzuklären, dass die zwischengespeicherten Daten aktuell sind. Diese Kontrolle ist vor dem Zugriff auf Daten jedes Mal nötig.
Durch Einführung eines Proxyservers wird ein gemeinsam genutzter Cache für gerade aufgerufene Webressourcen bereitgestellt.

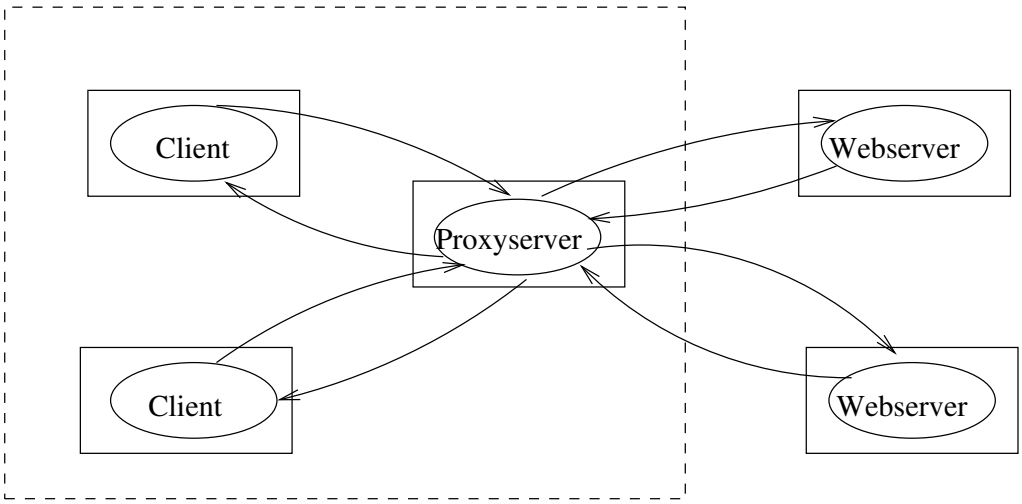


Abbildung 3.10: Proxy Server

Dadurch wird die Verfügbarkeit und Performance des Dienstes erhöht und gleichzeitig die

Netz- und Webserverlast reduziert. Proxyserver können gleichzeitig andere Aufgaben übernehmen, z.B. entfernte Webserver über eine Firewall ansprechen.

C/S-Variante 4: Einsatz Mobilen Codes Mobiler Code ist ein Programmcode, der von einem Rechner auf einen anderen gesendet werden kann und auf dem Zielrechner zur Ausführung kommt.
Da die Befehlsmenge eines Rechners von seiner Hardware abhängt, ist Maschinencode in der Regel nicht als Mobiler Code geeignet. Dieses Problem der Nichtausführbarkeit von verschicktem Code ist oft auch bei E-Mail-Attachments sichtbar. In Abhängigkeit von Betriebssystem und Anwendungssoftware sind Dateien dann unbrauchbar.
Zur Ausführbarkeit von Code auf einem anderen Rechner soll das Prinzip der Virtuellen Maschine betrachtet werden. Ein Compiler für eine bestimmte Programmiersprache generiert keinen Code für eine bestimmte Hardware, also keinen speziellen Maschinencode, sondern Code für eine Virtuelle Maschine, die allerdings für jede Art von Hardware einmal implementiert werden muß.
Beispiel: Ein Javacompiler generiert Code für die Java Virtual Machine, auf der Javaprogramme abgearbeitet werden können.
Diese Lösung ist im allgemeinen Fall nicht auf alle anderen Programmiersprachen übertragbar.
Als Beispiel für mobilen Code wollen wir nun Applets betrachten. Applets sind kleine Standalone Anwendungen, die an einen Client gesendet und in dem Adressraum seines Prozesses ausgeführt werden können. Bei Webapplets wird über den Webbrowser ein Link auf ein Applet ausgewählt und der Code vom Webserver in den Browser geladen sowie dort ausgeführt.

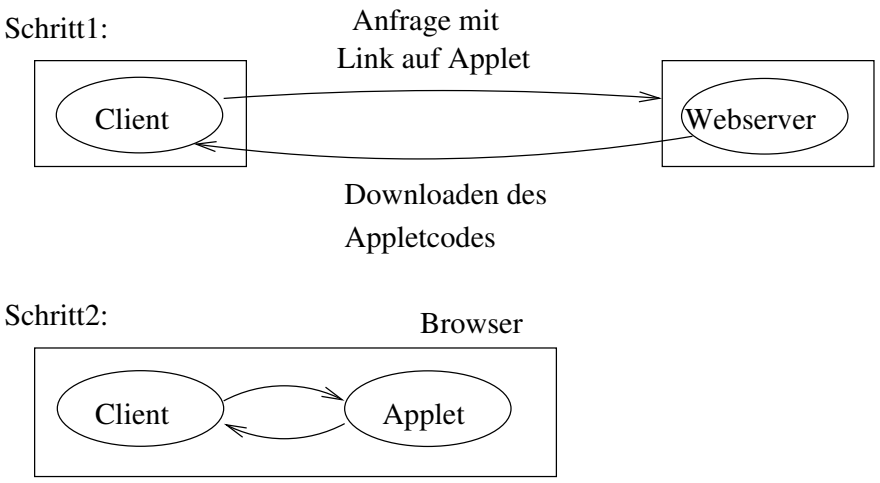


Abbildung 3.11: Webapplet

Vorteil dieser lokalen Programmausführung ist die gute Antwortzeit, die bei schwankenden Bandbreiten - z.B. im Mobilfunknetz - optimal ist.
Sofern eine Anwendung allerdings auf veränderliche Daten zugreifen muss, die der Webserver dynamisch bereitgestellt bekommt (z.B. Aktienkurse), reicht die reine Appletfunktionalität nicht mehr aus. Auch zusätzlicher Code, der mit dem Server kommuniziert, kann eine Datenaktualisierung nur bedingt bereitstellen. Denn: Interaktionen zwischen Client und Ser-

ver werden im Normalfall stets vom Client initiiert.
Die Lösung besteht in zusätzlicher Software, die nach einem Push-Modell arbeitet. Dabei ergreift nicht der Client, sondern der Server die Initiative der Interaktion.
Als Anwendung sind bedingte Kaufoptionen eines Aktienhändlers denkbar, beispielsweise wenn der Kurs unter einen bestimmten Schwellwert fällt. Für diesen kundenorientierten Dienst (customized service) muß der Kunde ein spezielles Applet downloaden, das Updates vom Aktienserver empfangen und anzeigen kann und ferner automatische Kauf- und Verkaufoperationen ausführen kann, die durch kundendefinierte Bedingungen ausgelöst werden, die lokal auf dem Kundenrechner gespeichert werden.
Wichtig ist es, in diesem Zusammenhang das Thema Sicherheit zu erwähnen. Entfernte Zielrechner müssen vor Manipulationen geschützt werden.

C/S-Variante 5: Einsatz Mobiler Agenten Mobiler Code kommt vom Server zum Client; bei Mobilen Agenten ist es genau anders herum. Ein Mobiler Agent ist ein laufendes Programm, das sowohl Code als auch Daten umfasst. Der Agent bewegt sich innerhalb eines Netzes von einem Rechner zum nächsten. Dabei führt er eine Aufgabe aus, die von irgendjemandem oder irgendeiner Komponente gestellt wurde, z.B. Informationen zu sammeln und mit einem bestimmten Ergebnis zurückzukehren.

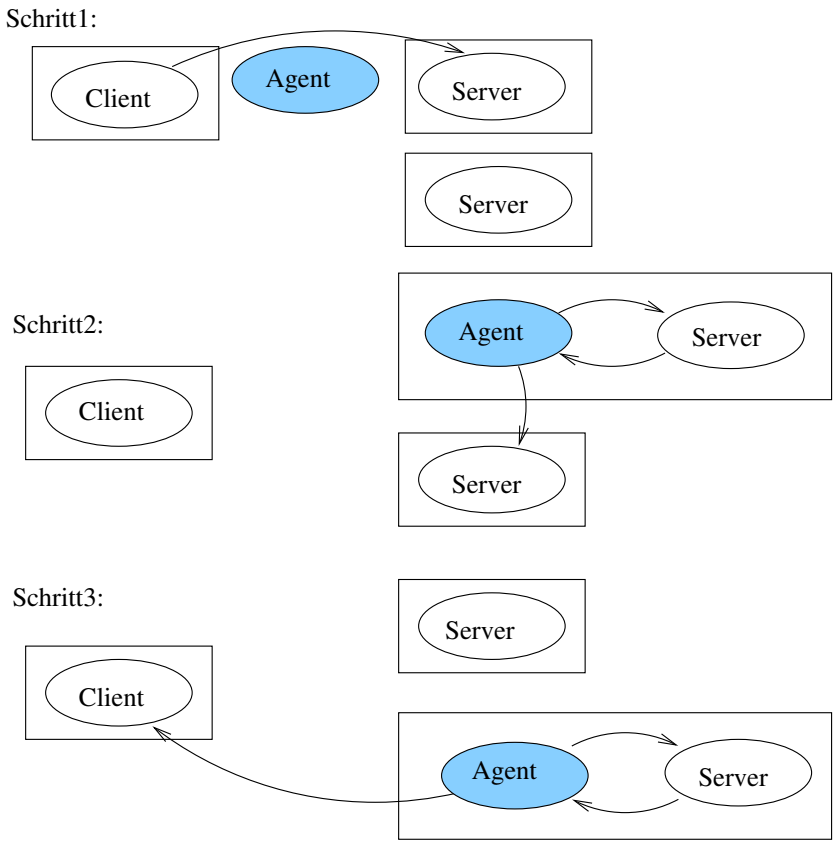


Abbildung 3.12: Mobiler Agent

Bei jedem Rechnerbesuch kann der Agent mehrere Aufrufe tätigen, z.B. Datenbankeinträge

lesen. Dabei werden entfernte Informationen auf lokale Interaktionen zurückgeführt und so Kommunikationskosten und -zeit eingespart. Der Vorteil hierbei ist die geringere Netzlast. Allserdings geht diese Variante nur, wenn der Client nicht interagieren muss.
Anwendungsbeispiele sind die Installation und das Warten von Software auf Computern innerhalb einer Organisation oder Preis- und Produktvergleiche bei konkurrierenden Herstellern. Sicherheit ist hier wieder ein wichtiges Thema. Der Agent ist gleichermaßen verwundbar wie auch der Zielrechner.
Weitere Varianten des Client/Server-Prinzips wie ThinClients, Mobile Devices und Spontaneous Networking werden in Kapitel 6 betrachtet.

3.4 Netzwerkgrundlagen

Hier gehen wir davon aus, dass das Netzwerk bis Schicht 4 des OSI-Referenzmodells schon implementiert ist.
Für die genauere Betrachtung des Architekturmodells Verteilter Systeme (vgl. Abschnitt 3.3) soll eine schichtenweise Detaillierung von unten nach oben vorgenommen werden.
Damit ist zunächst die Plattform von Interesse. Für die Wirkungsweise der Computerhardware sei auf das Skript „Technische Grundlagen der Informatik“ verwiesen. Eine genauere Betrachtung des Betriebssystems kann dem Skript „Informatik III“ entnommen werden.
Die für Verteilte Systeme grundlegendsten Prinzipien der Netzfunktionsweise sollen im Folgenden erläutert werden. Für eine dieser Thematik angemessenen Detaillierung sei auf die Vorlesung „Rechnernetze“ verwiesen.
Wir betrachten zunächst einen **verbindungsorientierten Dienst**. Dieser ist nach dem Telefonsystem ausgelegt, d.h. wir haben einen Verbindungsaufbau durch 1. Hörer abnehmen, 2. Nummer wählen, können dann über eine Verbindung mit der anderen Seite sprechen, und müssen zum Verbindungsabbau dann den Hörer auflegen. In der Informatik ist der File Transfer ein Beispiel für einen verbindungsorientierten Dienst.
Im Gegensatz dazu ist ein **verbindungsloser Dienst** nach dem Postsystem gestaltet. Jede Nachricht (Brief) trägt die volle Adresse und wird unabhängig von allen anderen durch das System geschleust. Normalerweise kommt diejenige von zwei Nachrichten, die früher abgeschickt wurde, auch früher an. Es ist aber nicht ungewöhnlich, dass Nachrichten sich auch verzögern können, so dass die zweite vor der ersten ankommt. Bei einem verbindungsorientierten Dienst ist eine solche Eigenschaft unmöglich.
Die Aufgabe der Internetprotokollschicht ist es, den Hosts zu ermöglichen, Pakete in jedes beliebige Netz zu übertragen und unabhängig an das - in der Regel in einem anderen Netz befindliche - Ziel zu bringen. Pakete können dabei möglicherweise sogar in einer anderen Reihenfolge ankommen, als sie abgeschickt wurden. In diesem Fall ist es die Aufgabe der höheren Schichten, sie wieder richtig anzuordnen, falls eine geordnete Zustellung gewünscht wird.
Die IP- oder Vermittlungs- oder auch Netzwerk-Schicht definiert ein offizielles Paketformat und ein Protokoll, das so genannte Internet Protokoll. Sie hat die Aufgabe, IP-Pakete richtig zuzustellen. Das Paket-Routing ist hier deutlich die wichtigste Frage, ebenso das Vermeiden von Überlastungen.
Die Schicht oberhalb der Internet-Schicht wird als Transportschicht bezeichnet. Sie soll den Partnereinheiten an den Quell- und Zielhosts die Kommunikation ermöglichen. Dazu werden zwei Ende-zu-Ende-Protokolle definiert, die alternativ verwendet werden können.

Das Transmission Control Protocol, sprich das TCP-Protokoll, ist ein zuverlässiges verbindungsorientiertes Protokoll, durch das ein Bytestrom von einer Maschine fehlerfrei einer anderen Maschine im Internet zugestellt wird. Es zerlegt den eingehenden Bytestrom in einzelne Nachrichten und leitet sie an die Internet-Schicht weiter. Am Ziel werden sie vom empfangenden TCP-Prozess wieder zu einem Ausgabestrom zusammengesetzt. TCP handhabt auch die Flusssteuerung, um sicherzustellen, dass ein langsamer Empfänger nicht von einem schnellen Sender mit Nachrichten überschwemmt wird.
Das alternative Protokoll auf dieser Schicht wird User Datagramm Protocol genannt, kurz UDP. Dabei handelt es sich um ein unzuverlässiges verbindungsloses Datagramm Protokoll für Anwendungen, die anstelle der Abfolge oder Flusskontrolle von TCP diese Aufgabe lieber selbst bereitstellen. UDP wird dabei im Wesentlichen als IP-Replikat auf Transportschicht realisiert. Ein UDP-Datagramm ist innerhalb eines IP-Paketes eingekapselt. Es hat einen kleinen Header, der u.a. die Quell- und Zielportnummern enthält. Dieses Protokoll wird vorwiegend für einmalige Abfragen und Anwendungen in Client/Server-Umgebung benutzt, in denen die Schnelligkeit der Zustellung wichtiger ist als ihre Genauigkeit, z.B. für die Übertragung von Sprache und Video.
Bei der Einordnung der Transportschicht in die Schicht vier der sieben Schichten des OSI-Modells ist dies die niedrigste Schicht, auf der Nachrichten, d.h. Messages im Sinne des Message Passing gehandhabt werden. Die Nachrichten werden an Kommunikationsports adressiert, die bestimmten Prozessen zugeordnet sind.

In der Diskussion des Architekturmodells Verteilter Systeme soll nun zu den beiden Teilschichten der Middleware übergegangen werden.

3.5 Das request/reply-Protokoll

Basierend auf der Plattform soll im Folgenden die Middleware sukzessive betrachtet werden. Dabei wird mit der Teilschicht des Anfrage/Antwort - oder request/reply-Protokolls begonnen. Dies ist in OSI bei Schicht 5 einzuordnen.

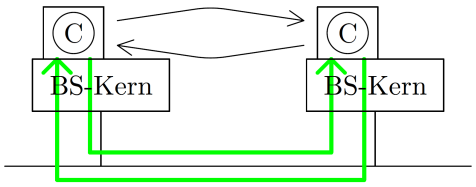


Abbildung 3.13: Request/Reply durch den OSI-Stack hindurch

Das Prinzip der Interprozesskommunikation Eine Kommunikation zwischen zwei Prozessen wird dadurch realisiert, dass ein Prozess eine Nachricht, d.h. eine Folge von Bytes, an ein Ziel sendet. Dort empfängt ein anderer Prozess diese Nachricht. Mit jedem Ziel ist in der Regel eine Warteschlange verbunden. Der sendende Prozess bewirkt, dass die Nachricht der entfernten Warteschlange hinzugefügt wird. Und der dortige empfangende Prozess entnimmt die Nachricht seiner lokalen Warteschlange.
Die Kommunikation zwischen dem sendenden und dem empfangenen Prozess kann entweder **synchron** oder **asynchron** realisiert werden. Im ersten Fall synchronisieren sich die Prozesse

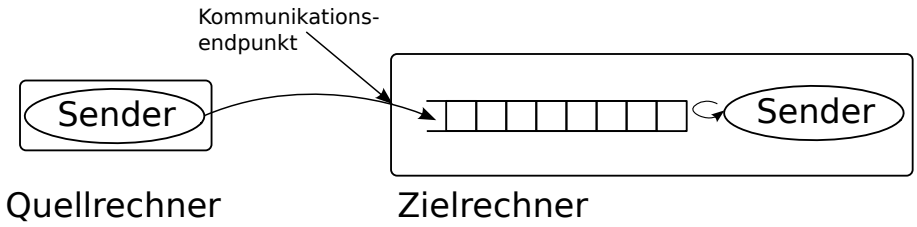


Abbildung 3.14: Interprozesskommunikation

bei jeder Nachrichtenübertragung. Dabei sind sowohl die Operation *send* als auch *receive* als blockierende Operationen realisiert.
Bei der asynchronen Kommunikation wird das Senden nichtblockierend realisiert, so dass die zu sendende Nachricht aus einem lokalen Puffer übertragen wird und parallel dazu mit der Abarbeitung des sendenden Prozesses weiter fortgefahren werden kann.
Wir wollen nun das Ziel des Nachrichtenverschickens genauer betrachten. Dieses Ziel wird auch als Endpunkt der Kommunikation bezeichnet oder als Port. Der Port wird vom lokalen Betriebssystem verwaltet und ist eindeutig einem laufenden Prozess zugeordnet. Ein Port kann mehrere Sender haben, aber besitzt genau einen Empfänger. Aus Sicht eines Prozesses können ihm auch mehrere Ports zugeordnet sein, von denen er Nachrichten erhält. Portnummern werden in der Regel vom Server publiziert, damit Clients Nachrichten direkt dorthin senden können. Jeder Prozess, der eine Portnummer eines anderen Prozesses kennt, kann diese für eine Nachrichtenkommunikation nutzen.
Das resultierende Problem der Ortstransparenz soll später betrachtet werden. Hier sind dynamisches Binden zur Laufzeit und ortsunabhängige Identifikationen, die auf Adressen abgebildet werden, von Bedeutung.
Als Alternative bieten sich an:

– Broadcasten von Anfragen

– Directory

Da es nur 2^{16} Ports gibt, ist die Portnummer nicht ausreichend: Es ist nicht geklärt, auf welchen Rechner sich die Nachricht bezieht, wenn bei zwei Rechnern der geliche Port aktiv ist.

Sockets Ein Socket kann man sich als den lokalen Port eines Rechners und dessen Internetadresse vorstellen.
Beide Protokolle auf der Transportschicht, d.h. TCP und UDP, nutzen das Konzept der Sockets. Jeder Socket ist dabei aber mit genau einem der beiden Protokolle assoziiert, also TCP oder UDP.
Sockets sind ein Kommunikationsendpunkt zwischen Prozessen. Das Konzept wird in eigentlich allen moderneren Betriebssystemen verwendet: UNIX, Linux, Windows NT, Macintosh OS. Eine Anwendung kann Daten an einen Socket geben - man sagt auch schreiben -, diese Daten werden über das zugrundeliegende Netz an den Socket eines anderen Prozesses gesendet. Damit dieser andere Prozess die Daten empfangen kann, muß sein Socket an einen lokalen Port und die Internetadresse des Rechners, auf dem er läuft, gebunden sein.
Sockets können sowohl für das Senden als auch für das Empfangen von Nachrichten genutzt werden. Ferner kann jeder Prozess Daten von beliebig vielen Ports empfangen, aber ein Port

ist immer nur einem Prozess zugeordnet.

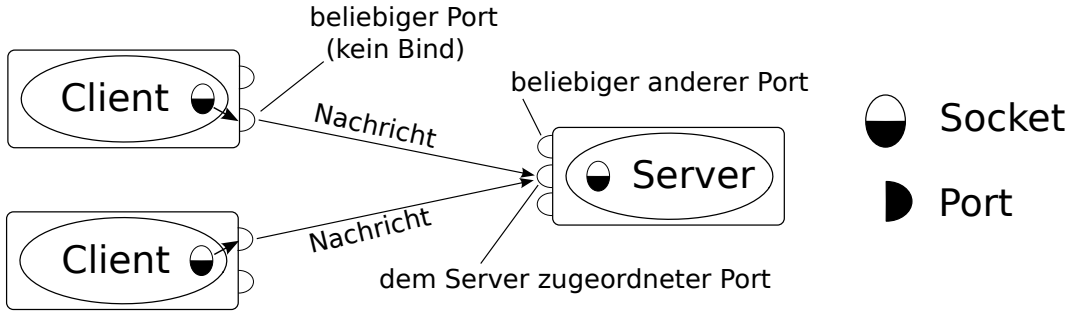


Abbildung 3.15: Sockets

In Abbildung 3.16 soll das Prinzip der Sockets für TCP skizziert werden.

Für die Programmierung der Sockets stehen für TCP/IP acht Kommunikationsprimitive bereit: Socket, bind, listen, accept, connect, send, receive und close.
Mit dem Aufruf von *Socket* generiert der aufrufende Prozess, d.h. der Client oder Server, einen neuen Kommunikationsendpunkt für ein spezifisches Transportprotokoll. Betriebssystemintern werden Ressourcen reserviert, um die zu sendenden und zu empfangenden Nachrichten handhaben zu können.
Mit **bind** wird eine lokale Adresse mit dem neuen Socket assoziiert, diese beinhaltet die IP-Adresse des Rechners und die Nummer eines lokalen Ports. Damit weiß das Betriebssystem, welche Nachrichten der entsprechende Prozess erhalten möchte.
Im Fall einer verbindungsorientierten Kommunikation wird das Primitiv **listen** aufgerufen. Damit wird Speicherplatz für eine gewisse Anzahl von Verbindungen reserviert, die vom Prozess akzeptiert werden.
Der Aufruf von **accept** blockiert den aufrufenden Prozess, bis eine Nachricht eingetroffen ist. Auf der Clientseite ist ein explizites Binden an eine lokale Adresse nicht notwendig. Dem Socket kann dynamisch ein Port zugewiesen werden, wenn die Verbindung aufgebaut wird. Allerdings muss die Adresse des Senders auf Transportschicht angegeben werden. Diese Adresse ist das bekannte (IP-Adresse, Portnummer)-Paar, das mittels **connect** spezifiziert wird. Der Client wird blockiert, bis die Verbindung erfolgreich aufgebaut wurde, danach kann der Nachrichtenaustausch durch die Primitive write und read bzw. **send** und **receive** erfolgen. Mit **close** erfolgt ein ordnungsgemäßer Verbindungsabbau.
Welche Besonderheit gibt es bei der Request/Response-Abarbeitung?

– Adressierung: Woher kennt der Client die Adresse des Servers?
  - Die Addresse wurde als Socket zuvor vom Server publiziert. → Client weiß sie.
  - Der Client kennt sie nicht: Er broadcastet ein Lokalisierungspacket. Diese Methode skaliert gut, wenn wenige Clients nach dem Server fragen.
  - Es existiert ein Directory-Server. Diese Methode ist effikiv, wenn viele Clients fragen, wo der Server ist.

– Blokieren/Puffern: Wenn sowohl gepuffert als auch blockert wirt, dann ist es redundant. Wenn weder das eine noch das andere gemacht wird, gehen Nachrichten verloren. ⇒

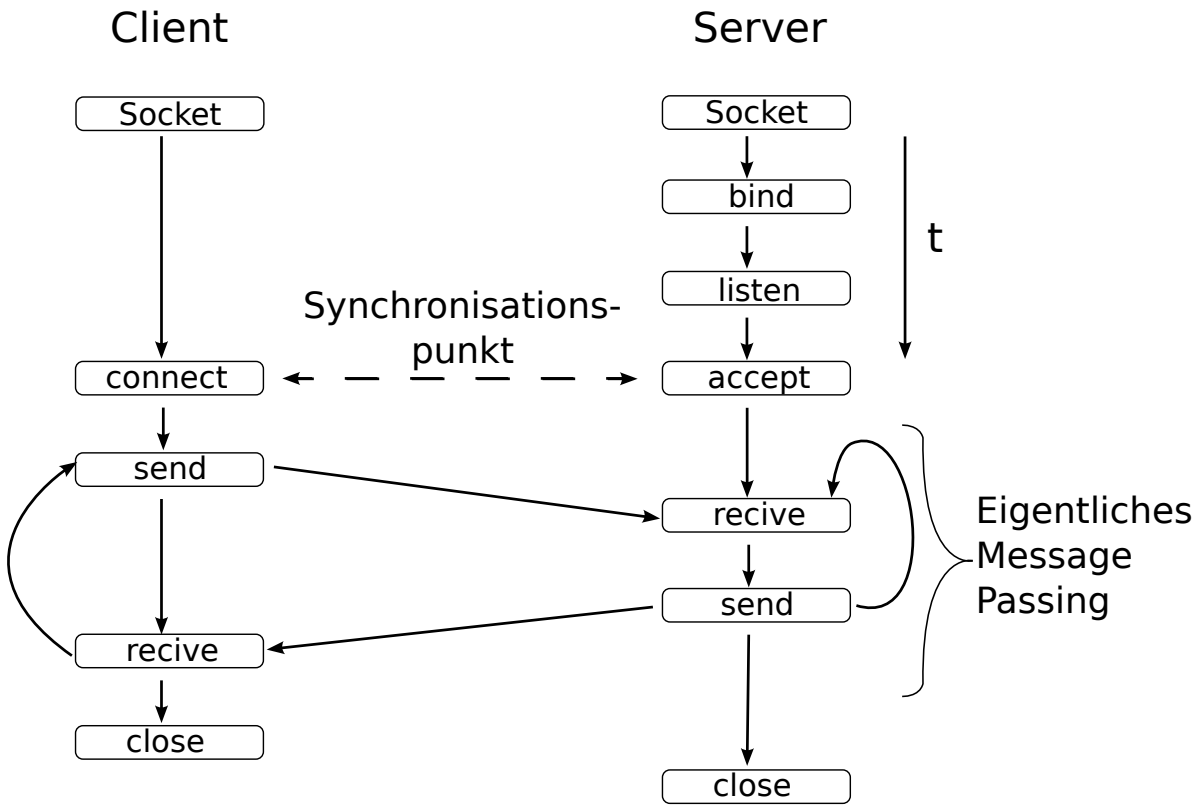


Abbildung 3.16: Kommunikationsprimitive bei TCP/IP

man nimmt eine der beiden anderen Mögligkeiten.

– Zuverlässigkeit:

  - egal: Es ist kein Problem, falls nichts ankommt („Prinzip der Deutschen Bundespost“).
  - vollständig: Es werden ständig ACKs hin- und hergeschickt.
  - Optimierung: Der Server sendet als Antowrt ein ACK zurück und der Slient sendet ein zweites ACK zurück.

  Bei idempotenten (vgl.:$f''(x) = f'(x) = f(x)$) Diensten macht es nichts, wenn etwas mehrmals oder gar nicht passiert, z.B. Konto lesen. Überweisen hingegen ist nicht idempotent.

UDP Datagram - Kommunikation Die Übertragung basiert auf dem Senden (send) eines Prozesses und dem Empfangen (receive) eines anderen Prozesses. Dabei muss jeder sendende oder empfangende Prozess zunächst einen Socket generieren, der an eine Internetadresse und einen lokalen Port gebunden ist. Ein Server bindet seinen Socket an einen Serverport, der den Clients bekannt gegeben wird, so dass sie Nachrichten an diesen Port senden können. Ein Client dagegen bindet seinen Socket an irgendeinen freien lokalen Port. Beim Aufruf einer receive-Operation können die IP-Adresse und Portnummer des Senders genutzt werden, um

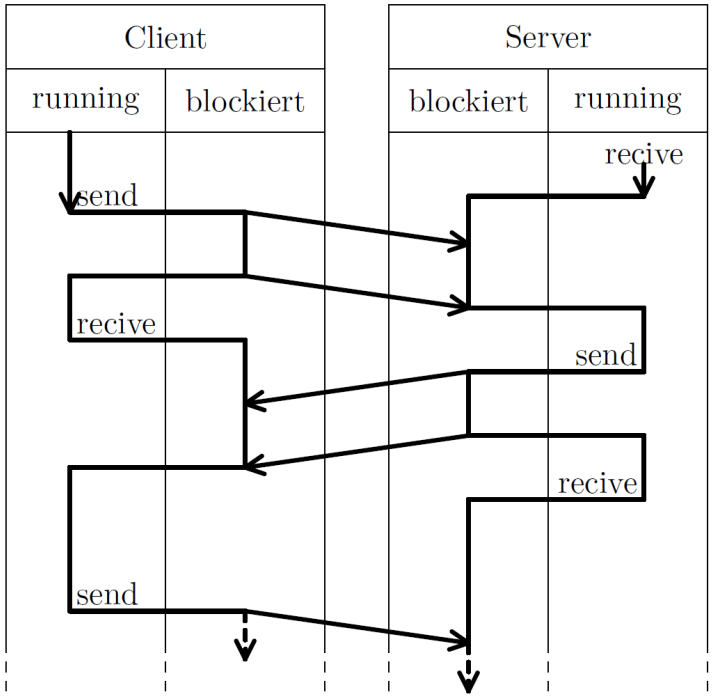


Abbildung 3.17: eigentliches Message Passing

eine Anfrage zu replyen.
Ein mittels UDP versendetes Datagramm wird vom sendenden Prozess an den empfangenden Prozess ohne jede Quittung oder Bestätigung verschickt. Falls ein Fehler auftritt, kann die Nachricht komplett verloren gehen.

- **Blockierung:** Bei der UDP Datagramm-Kommunikation werden nicht-blockierende send- und blockierende receive-Primitive verwendet. Bei der Ankunft einer Nachricht auf dem Zielrechner wird die Nachricht in eine Warteschlange für den Socket gespeichert, die an den Zielport gebunden ist. Bei einem bereits anstehenden oder späteren receive auf diesen Socket kann die Nachricht von der Warteschlange, d.h. Queue, ausgelesen werden.

- **Timeout:** Eine Blockierung kann gegebenenfalls erfolgen, bis ein Timeout abgelaufen ist.

- **Threads:** Falls der receive ausführende Prozess noch andere Aufgaben zu erledigen hat, so kann der receive-Aufruf in einem separaten Thread ausgeführt werden. Trotz Blockierung kann der Prozess dann während des Wartens auf eine eintreffende Nachricht aktiv sein.

- **Reihenfolge:** Bei der Verwendung von UDP ist es protokollspezifisch, dass Nachrichten in einer anderen Reihenfolge ankommen können als sie ursprünglich gesendet wurden. Trotzdem hat UDP seine Daseinsberechtigung, da es Overhead vermeidet, z.B. hinsichtlich Reihenfolgeerhaltung und Bestätigungen.

Client/Server-Kommunikation Wir wollen nun das Prinzip der Client/Server-Kommunikation betrachten, das im Kern auf den Aufruf der in Abbildung 3.18 dargestellten Kommunikationsprimitive des Message Passings zurückgeführt werden kann.

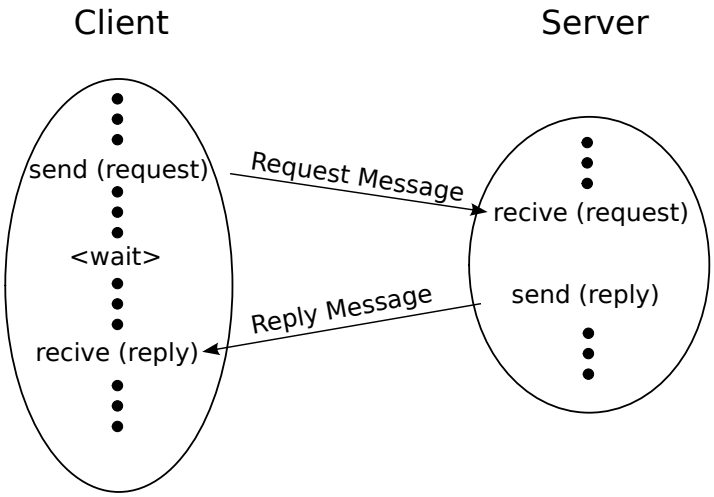


Abbildung 3.18: Client/Server-Kommunikation

Beispiel: Java API für UDP-Datagramme Im Folgenden soll die Funktionsweise des request/reply-Protokolls anhand der Programmiersprache Java verdeutlicht werden. Zunächst erst einmal keine Panik, falls Sie kein Java-Experte sein sollten! Ziel der Beispielprogramme ist es, die Wirkungsweise von send- und receive-Primitiven zu verdeutlichen, auch wenn die Client- und Server-Programme doch deutlich darüber hinausgehen.
Wir beginnen mit zwei Klassen, wobei jede Klasse die Zusammenfassung mehrerer Elemente zu einem neuen Objekt ist, das unter einem gemeinsamen Namen angesprochen wird.

Datagramm Packet Instanzen der Klasse DatagramPacket werden von einem Prozess zu einem anderen transportiert, wenn ein Prozess ein Packet sendet und ein anderer Prozess dieses Packet empfängt.
Diese Klasse besteht aus folgenden Bestandteilen:

Nachricht	Länge der Nachricht	Internetadresse	Portnummer

Dabei ist der Teil Nachricht selbst ein Feld, das aus Bytes besteht, die die eigentliche Nachricht enthalten. Die anderen 3 Teile sind Integerzahlen.
Internetadresse und Portnummer beziehen sich auf die jeweils entfernten Sockets. In diesem Sinne kann ein Konstruktor verwendet werden, der im Falle von receive aus einer Nachricht ein DatagramPacket generiert, in dem er die anderen 3 Bestandteile automatisch ableitet. Umgekehrt kann mit den Methoden getData, getPort und getAddress der Inhalt der benötigten Felder gelesen werden.

Datagram Socket Zur Unterstützung des Sendens und Empfangens von UDP Datagrammen wird die Klasse DatagramSocket genutzt. Diese Klasse kann in zwei Fällen, d.h. durch

Verwendung zweier Konstruktoren, verwendet werden. Entweder wird eine Portnummer als Argument verwendet - wenn Prozesse einen speziellen Port nutzen müssen, z.B. bei Servern. Oder es wird ein argumentloser Konstruktor genommen, d.h. das System kann in diesem Fall einen beliebigen freien Port wählen, was auf der Clientseite sinnvoll ist. Dabei muss eine SocketException angegeben werden. Diese interne Unterbrechung oder Ausnahme kommt dann zur Anwendung, wenn ein Port bereits genutzt wird oder reserviert ist. Die Klasse DatagramSocket stellt eine Reihe von Methoden bereit, von denen die wichtigsten im Folgenden betrachtet werden sollen.

- **Send:**
  Diese Methode wird zum Abschicken eines Datagramms von einem Socket an einen anderen Socket genutzt. Das Argument der Methode send ist eine Instanz der Klasse DatagramPacket, die eine Nachricht, deren Länge und ihr Ziel enthält. Diese Methode muss eine IOException berücksichtigen.

- **Receive:**
  Diese Methode wird zum Empfangen eines Datagrams an einer Socket genutzt. Das Argument der Methode receive ist ein leeres DatagramPacket, in das die Nachricht und ihre Länge und Quelle hineingeschrieben werden können. Eine IOException muß ebenfalls berücksichtigt werden.

- **SetSoTimeout:**
  Diese Methode erlaubt das Setzen eines Timeouts. Dies ist insbesondere beim blockierenden Empfangen von Nachrichten von Bedeutung, um unendliches blockierendes Warten zu vermeiden, sondern nur eine spezifizierte Zeit zu warten und dann ggf. eine Interrupted IOException auszulösen.

- **Connect:**
  Diese Methode kann bei einer Klasse DatagramSocket aufgerufen werden, um die entsprechende Socket mit einem speziellen Ziel - angegeben durch eine entfernte Portnummer und Internetadresse - zu verbinden. Dann ist die Socket nur in der Lage, Nachrichten an diese Adresse zu senden und von dieser Adresse zu empfangen.

Auf der Basis dieser beiden Klassen DatagramPacket und DatagramSocket sollen nun die clientseitigen und serverseitigen Programme betrachtet werden. Das im Folgenden betrachtete Clientprogramm kreiert einen Socket, sendet eine Nachricht an einen Serverprozess, der an den Port 6789 gebunden ist und wartet schließlich auf das Empfangen einer Antwort.

UDP-Client:

```
1 // importiert die entsprechenden Klassen java.net.* und java.io.*
  import java.net.* ;
3 import java.io.* ;

5 // zwecks Zusammenarbeit muessen andere Module Zugriff auf diese
  // Klasse UDPClient haben koennen, daher explizite Gewaehrung eines
7 // Zugriffs auf die Klasse UDPClient durch Exportierung mittels des
  // Zusatzes public
9 public class UDPClient {
```

```
  // deklariert main() - Methode
11  // die main()-Methode unterscheidet eine Anwendung von einer Klasse
  // Insbesondere kann eine main()-Methode in eine Klasse eingefuegt
13  // werden. Wichtig ist die Argumentuebergabe durch args[]. Hierbei
  // werden 2 Elemente referenziert:
15  // Die Message mit Index 0 und die IP-Adresse mit Index 1.
  public static void main (String args[]){
17    // Definition eines try-Blocks, um Ausnahmen (Exceptions) handhaben
      // zu koennen, die am Ende durch Catch-Klauseln behandelt werden
19    // koennen.
      try {
21      // erstellt eine neue Instanz der Klasse DatagramSocket und
        // speichert eine Referenz darauf in der Variablen aSocket ab -
23      // damit ist ein neuer Socket kreiert. Nun: Basierend auf einer
        // gegebenen Nachricht bzw. Message muessen die Bestandteile fuer
25      // die Zusammensetzung eines DatagramPackets bereitgestellt
        // werden.
27      DatagramSocket aSocket = new DatagramSocket();
        // Aus dem Element 0 des Arguments wird ein Bytestring erzeugt,
29      // der unter der Variablen m abgespeichert wird. Dies ist die
        // eigentliche zu uebertragende Nachricht.
31      byte[] m = args[0].getBytes();
        // Aus dem Element 1 des Arguments wird die IP-Adresse des
33      // Zielrechners generiert.
        InetAddress aHost = InetAddresss.getByName(args[1]);
35      // Deklaration einer Konstanten 6789 als Portnummer des
        // entfernten Server, was durch Bekanntgabe des Servers als
37      //Info vorliegt.
        int serverPort = 6789;
39      // Erstellt eine neue Instanz der Klasse DatagramPacket und
        // speichert eine Referenz darauf in der Variablen request ab -
41      // dieses DatagramPacket beinhaltet im ersten Feld die
        // Nachricht m, im Zweiten Feld deren Laenge und dann in aHost
43      // und serverPort die Internetadresse sowie Portnummer der
        // Zielsocket.
45      DatagramPacket request =
          new DatagramPacket(m, args[0].length(), aHost, serverPort);
47      // die oben kreierte Instanz aSocket ruft die Methode send zum
        // Abschicken des gerade kreierten Datagrams request auf.
49      aSocket.send(request);
        // Es wird ein Speicherplatz der Groesse 1000 Byte alloziiert, um
51      // fuer die zu empfangene Antwort Speicherkapazitaet
        // bereitzustellen.
53      byte[] buffer = new byte[1000];

55      // Erstellt eine neue Instanz reply der Klasse DatagramPacket
        //zum Empfang der Antwort. Dabei sind nur 2 der 4 Felder in den
57      // Argumenten enthalten, so dass eine Referenz auf das
        // Speicherarray und dessen Groesse angegeben werden. Die Adresse
59      // des Absenders, d.h. des Servers, ist ja bekannt und
        // interessiert deshalb nicht.
61      DatagramPacket reply =
          new DatragramPacket(buffer, buffer.length);
```

```
63        // Die Instanz aSocket ruft nun die Methode receive auf und kann
          // dabei die empfangene Nachricht in den bereitgestellten
65        // Speicherplatz ablegen.
          aSocket.receive(reply);
67        // GetData ist eine Methode, die auf DatagramPacket angewandt
          // werden kann, hier speziell auf die Instanz reply. Damit
69        // erhaelt man also die zurueckgeschickte Nachricht, die als
          // String ausgedruckt wird.
71        System.out.println("Reply:" + new String(reply.getData()));
          // Beendet die Verbindung gemaess der bei den Sockets
73        // vorgestellten Kommunikationsprimitive.
          aSocket.close();
75        // Ausnahmebehandlung bei Sockets, z.B. falls ein bereits
          // reservierter oder belegter Port durch den Client angesprochen
77        // wird.
      }
79    catch(Socket Exception e) {
        System.out.println("Socket:" + e.getMessage());
81    }
      // Ausnahmebehandlung durch Ein/Ausgabe, z.B. im Zusammenhang mit
83    // Aufruf von send und /oder receive- Methoden
      catch(IO Exception e) {
85      System.out.println("IO:"+ e.getMessage());
      }
87  }
}
```

Damit kreiert der UDP-Client zunächst einen neuen Socket, verschickt über diesen ein Datagramm an einen Serverprozess, wartet auf den Empfang einer Nachricht und beendet die Verbindung.
Nun soll die Serverseite betrachtet werden. Der zugehörige Server kreiert einen Socket, der an seinen Port 6789 gebunden ist, wartet dann auf den Empfang von Nachrichten von Clients, die er ganz einfach dadurch beantwortet, dass dieselbe Nachricht zurückgeschickt wird.

UDP-Server:

```
import java.net.* ;
2 import java.io.* ;

4 public class UDPServer{
    public static void main(String args[]){
6     try{
        // Hierbei Verwendung des Konstruktors, der eine spezielle
8       // Portnummer als Argument verwendet, damit der Server an genau
        // diesen Port gebunden ist.
10      DatagramSocket aSocket = new DatagramSocket(6789);
        // Bereitstellung von 1000 Byte Speicherplatz
12      byte[] buffer = new byte[1000];
        while(true){
14        // Request als neue Instanz der Klasse DatagramPacket hat
          // Speicherplatz fuer die Nachricht und deren Laenge.
16        DatagramPacket request =
```

```
            new DatagramPacket(buffer, buffer.length);
18          // An der oben generierten Socket-Instanz aSocket koennen
            // Nachrichten empfangen werden, die im DatagramPacket request
20          // abgespeichert werden koennen.
            aSocket.receive(request);
22          // Es wird eine neue Instanz der Klasse DatagramPacket
            // kreiert, die als reply bezeichnet wird und alle - aus dem
24          // erhaltenen DatagramPacket request - angekommenen Teile
            // liest und in dieses DatagramPacket zurueckschreibt.
26          // Beachte:
            // Die Methode receive hatte allerdings nicht die Server-IP-
28          // Adresse und -Portnummer aus der gesendeten Nachricht
            // uebernommen, sondern durch die Quellangaben, d.h.
30          // Ortsangaben des Clients, ersetzt.
            // Somit ist die neu generierte Nachricht nicht mit der
32          // erhaltenen Nachricht identisch. Dies gilt lediglich fuer die
            // eigentliche Message m und deren Laenge, d.h. das 1. und
34          // 2. Feld des DatagramPackets
            DatagramPacket reply =
36            new DatagramPacket(request.getData(),
              request.getLength(),request.getAddress(),
38            request.getPort()
            );
40          // Zuruecksenden der gerade beschriebenen reply-Nachricht bzw.
            // des reply-DatagramPacket
42          aSocket.send(reply);
          }
44      }
        catch(SocketException e) {
46        System.out.println(Â¿Socket:Â¿ + e.getMessage());
        }
48      catch(IOException e) {
          System.out.println("IO:" + e.getMessage());
50      }
    }
52  }
```

Einige Bemerkungen zu diesem request/reply-Protokoll.

TCP Stream Kommunikation Die bisherigen Betrachtungen bezogen sich auf die UDP-Kommunikation. In ähnlicher Form ist auch eine TCP-Kommunikation realisierbar. Diese TCP-Kommunikation wird unter anderem bei folgenden Anwendungen benutzt

- Hyper Text Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- Terminal Session to a remote computer (Telnet)
- Simple Mail Transfer Protocol (SMTP)

Die Wirkungsweise des TCP-Clients und TCP-Servers kann Abschnitt 4.2 aus [5] entnommen werden.

Datenrepräsentation und Marshalling Beim Aufruf entfernter Dienste besteht ein Problem: Informationen, die in laufenden Programmen enthalten sind, werden durch Datenstrukturen repräsentiert, z.B. in Form von Objekten. Werden Informationen jedoch übertragen und zu diesem Zweck in Nachrichten verpackt, so besteht sie aus einer Folge von Bytes.
Daraus resultiert das Problem, dass Datenstrukturen transformiert, d.h. geflattet, werden müssen. Das heißt, daß Daten zum Übertragen in eine Folge von Bytes konvertiert und nach der Übertragung wiederhergestellt werden. Hinzu kommt, dass verschiedene Rechner auch ganz unterschiedliche Datendarstellungen verwenden, z.B. Little-Endian und Big-Endian für die Reihenfolge. Für den Datenaustausch zwischen zwei Computern gibt es zwei grundsätzliche Alternativen:

1. Es wird ein einheitliches externes Format vereinbart, in das alle zu übertragenden Daten konvertiert werden. Lediglich wenn beide Rechner vom selben Typ sind und dies auch wechselseitig bekannt ist, dann kann auf die Transformation verzichtet werden.

2. Die Daten werden im Format des Senders übertragen, wobei eine Spezifikation dieses Formats zusätzlich angegeben wird. Der Empfänger konvertiert die Daten dann in das von ihm benötigte Format.

Ein vereinbarter Standard für die Darstellung von Datenstrukturen wird als externe Datenrepräsentation (External Data Representation) bezeichnet. Der Prozess der Umwandlung von Daten in eine für die Übertragung geeignete Form, d.h. in eine Nachricht, wird als Marshalling bezeichnet.
Demzufolge ist Marshalling auch die Übersetzung von strukturierten Daten in eine externe Datenrepräsentation. Die Rückübersetzung wird als Unmarshalling bezeichnet.
Marshalling und Unmarshalling werden von einer Middleware ausgeführt, ohne den Anwendungsprogrammierer zu involvieren. Das heißt, diese beiden Aktivitäten sind für ihn transparent.

Beispiel: Common Data Representation (CDR) als Externe Datenrepräsentation von CORBA 2.0
CDR kann alle Datentypen repräsentieren, die CORBA als Argumente entfernter Aufrufe zulässt. Dies sind 15 grundlegende Typen, z.B. Boolean, Character, Short, long, float, double, unsigned long und eine Reihe zusammengesetzter Typen wie String oder Sequence, die aus einer Länge vom Typ unsigned long gefolgt von Characters oder anderen Elementen besteht.
Bezüglich der Datenaustauschalternativen wird stets die Ordnung des Senders spezifiziert und genutzt. Es wird sowohl Little-Endian als auch big-endian unterstützt.
Als Datenstruktur soll folgendes Beispiel betrachtet werden:

```
Struct Person {
  String name;
  String place;
  long year;
};
```

Dieser Datenstruktur sollen nun durch Person p = new Person(“Sarah“, “München“, “2001“) die Werte “Sarah“, “München“, 2001 zugeordnet werden. Die geflattete Form wird in einer 28 Byte langen Nachricht wie folgt dargestellt.

Bytenummer	Inhalt der 4 Byte-Felder	Bemerkungen
0-3	5	Stringlänge
4-7	Sara	String mit
8-11	h_ _ _	3 Byte leer
12-15	7	Sringlänge
16-19	Münc	String mit
20-23	hen_	1 Byte leer
24-27	2001	Unsigned long

Als CORBA CDR Nachricht ergibt sich folglich:
0005Sarah_ _ _ 0007Müchnen_2001
Hierbei steht 1 Byte pro Zeichen zur Verfügung. Eine Kodierung könnte beispielsweise in ASCII erfolgen.
Die Ausführung des Marshallings und Unmarshallings wird bei CORBA durch den Interface Compiler ausgeführt. Dieser wandelt die Argumente und Ergebnisse bei entfernten Methodenaufrufen in Nachrichten um und umgekehrt.
Das Analogon bei Java RMI ist die Objektserialisierung (Object Serialization). Im Fall von HTTP übernimmt dieses Protokoll das Marshalling. Es wird in ASCII transformiert, ggf. auch komprimiert.

Entfernte Objektreferenzen und Adressierung Wenn ein Client eine Methode auf einem entfernten Objekt aufruft, wird an den Serverprozess, der das entfernte Objekt beherbergt, eine Invocation Message geschickt. Diese Nachricht spezifiziert, von welchem speziellen Objekt eine Methode aufgerufen wird. Eine Invocation ist mit einem Rückgabewert, also ist sie bidirektional, im Gegensatz zum Announcement, das ohne Rückhabewert und daher unidirektional ist.
Das entfernte Objekt wird dabei durch eine Remote Object Reference identifiziert, die im gesamten Verteilten System gültig ist. Diese Referenz ist eindeutig über Zeit und Raum, d.h. über alle Prozesse in den verschiedenen Rechnern des Verteilten Systems.
Eine Variante einer Remote Object Reference ist die Verknüpfung verschiedener Bestandteile:

32 Bit	32 Bit	32 Bit	32 Bit	Beliebig viele Bits
Internetadresse	Portnr.	Zeit	Objektnr.	Schnittstelle d. entfernten Objekts

Hier ist Zeit nicht als Reihenfolgeidentifizierung sondern für Versionsidentifizierung zu sehen. In einfachen RMI-Implementierungen existieren entfernte Objekte nur in den Prozessen, die sie kreiert haben, und sind hinsichtlich ihrer Lebenszeit an das Laufen dieser Prozesse gebunden.
Dann kann die Remote Object Reference als Adresse des entfernten Objekts genutzt werden. Sie ist eine Form der Maschine.Prozess-Adressierung, die jedoch keine Ortstransparenz bereitstellt und wenig flexibel ist, z.B. bei einer Migration des Objekts. Alternativen der Adressierung sind das Broadcasten eines Lokalisierungspackets und die Verwendung von Suchdiensten.

Gruppenkommunikation Im Gegensatz zum bislang betrachteten paarweisen Nachrichtenaustausch gibt es Anwendungen, bei denen ein Multicast sinnvoller ist.
Dabei ist der Begriff der Gruppe von Bedeutung. Eine Gruppe ist eine Menge von Prozessen,

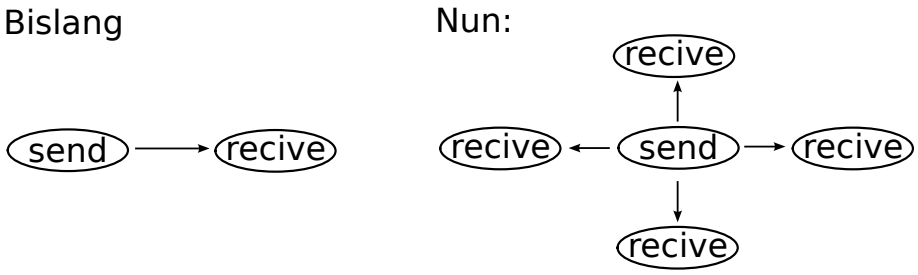


Abbildung 3.19: Punkt-zu-Punkt- und Punkt-zu-Mehrpunkt-Kommunikation

die miteinander kooperieren. Eine Multicast-Operation ist eine Operation, die eine einzelne Nachricht von einem Prozess an jedes Mitglied einer Gruppe von Prozessen sendet. In der Regel ist die Mitgliedschaft innerhalb der Gruppe dabei für den Sender transparent.
Es gibt verschiedene Varianten einer Multicast-Operation. Der einfachste Fall liefert keine Garantien für das Verschicken von Nachrichten und deren Ordnung. Beispiel sind News-Gruppen oder E-Mail-Gruppen.
Es gibt aber auch ganz andere Arten von Beispielen:

1. **Fehlertoleranz basierend auf replizierten Diensten**
   Ein replizierter Dienst ist mit einer Gruppe von Servern gleichzusetzen. Clientanfragen sind dann Multicastoperationen an alle Gruppenmitglieder, die identische Operationen ausführen. Vorteil: Auch wenn einige der Gruppenmitglieder ausfallen, kann der Dienst noch erbracht werden.

2. **Auffinden eines Discovery Servers in spontanen Netzen**
   Spontane Netze oder Adhoc-Netze treten in der Ära des Ubiquitous Computing auf und sind durch eine hohe Dynamik gekennzeichnet. Ein Discovery Service ist dabei eine - oft zentrale - Komponente, die Adressen von Servern für gewisse Anwendungszwecke bereitstellen kann. Damit ein in das Netz neu hinzukommender Client oder Server aber erst einmal die Adresse eines solchen Discovery Servers erfährt, kann er eine Multicast-Nachricht verschicken. Dieser Ansatz wird in der klassischen Ära der Verteilten Systeme auch als Broadcasten eines Lokalisierungspackets bezeichnet. Ist die Adresse eines Discovery Services somit bekannt geworden, kann sie von Servern genutzt werden, um anzubietende Schnittstellen registrieren zu lassen und von Clients, um nach Schnittstellen oder Diensten gezielt zu suchen.

3. **Leistungsverbesserung durch replizierte Daten**
   Wenn Datenreplikate in Anwendungscomputern platziert werden, kann oft Netzlast reduziert werden, indem ein lokaler Datenzugriff erfolgt. Ändern sich jedoch die Datenbestände, so ist dies durch eine Multicastoperation an alle Replikate zu aktualisieren.

4. **Ereignisanzeige**
   Tritt ein Ereignis ein, das für eine Gruppe von Usern von Interesse sein kann, so ist dieses durch eine Multicastnachricht bekannt zu geben. Ein solches Ereignis könnte eine Mitteilung für ein Newssystem sein oder auch eine neue Schnittstelle oder ein Dienst für einen bestimmten Kundenkreis.

Multicastoperationen werden auf OSI-Schicht 3 basierend auf dem Internet-Protokoll durch IP-Multicast unterstützt. Da IP-Pakete immer an Computer geschickt werden - nicht aber an Ports oder Prozesse! - ermöglicht das IP-Multicast es einem Sender, ein einzelnes IP-Packet an eine Menge von Computern zu schicken, die eine Multicastgruppe bilden.
Der Sender hat weder Ahnung von der Größe der Gruppe noch von den einzelnen Mitgliedern. Die Adressierung einer Multicastgruppe soll nicht weiter verteilt werden - sie nutzt Internetadressen der Klasse D, d.h. die mit den 4 Bits 1110 in IPv4 beginnen.
Ist ein Computer ein Mitglied einer Multicastgruppe, so kann er an diese Gruppe gesendete IP-Pakete empfangen. Seine Mitgliedschaft zu dieser Gruppe ist dynamisch, d.h. berechtigte Computer können jederzeit beliebigen Gruppen beitreten oder sie wieder verlassen.
Andererseits kann ein Rechner auch ein IP-Paket an eine Gruppe schicken, bei der es selbst kein Mitglied ist. Auf der Schicht der Anwendungsprogrammierung ist IP-Multicast nur über UDP möglich. Auf der IP-Schicht gehört ein Computer zu einer Multicastgruppe, wenn mindestens einer seiner Prozesse Sockets benutzt, die zu dieser Multicastgruppe gehören. Kommt eine Multicastnachricht an einem Computer an, so werden Kopien dieser Nachricht an alle lokalen Sockets geforwardet, die diese spezielle Multicastadresse nutzen und an die spezifische Portnummer gebunden sind. Die Java API stellt eine Schnittstelle zum IP-Multicast durch die Klasse *MulticastSocket* bereit.

3.6 Remote Procedure Call

Das Client/Server-Modell bietet einen brauchbaren Weg, ein Verteiltes System zu strukturieren. Trotzdem hat es eine extreme Schwachstelle: das Basisparadigma, auf dem die gesamte Kommunikation aufbaut, ist die Ein- und Ausgabe von Daten, die mittels *send* und *receive* verschickt und empfangen werden. Ziel in Verteilten Systemen ist es jedoch, verteiltes Arbeiten für den Benutzer genauso aussehen zu lassen wie zentrales Arbeiten. Dazu sind aber komfortablere Mechanismen notwendig.
Der erste und wahrscheinlich auch bestbekannte Mechanismus war die Erweiterung des konventionellen lokalen Prozeduraufrufs (Local Procedure Call) hin zum entfernten Prozeduraufruf (Remote Procedure Call - RPC), der es Clientprogrammen ermöglicht, Prozeduren in Serverprogrammen aufzurufen, die in separaten Prozessen und in der Regel auf anderen Computern laufen.
Jüngere Mechanismen haben das objektbasierte Programmiermodell erweitert, um es Objekten in unterschiedlichen Prozessen zu ermöglichen, mittels Remote Method Invocation (RMI) miteinander zu kommunizieren. RMI ist dabei eine Erweiterung der Local Method Invocation, d.h. des lokalen Methodenaufrufs, dahingehend, dass es einem Objekt in einem Prozess ermöglicht wird, eine Methode eines Objekts in einem anderen Prozess aufzurufen. Bemerkung: RMI ist dabei generischer zu verstehen als das Java RMI.
Im Folgenden soll der klassische RPC betrachtet werden. Er geht auf einen Vorschlag von Birell und Nelson aus dem Jahr 1984 zurück. Die grundlegende Idee besteht darin, dass ein Programm auch dann ein Unterprogramm aufrufen können soll, wenn sich dieses auf einem anderen Computer befindet. Dabei wird der aufrufende Prozess so lange suspendiert, bis die Ausführung auf dem entfernten Computer beendet und der - für den Programmierer unsichtbare - Nachrichtenaustausch fertig ist. Dieses Prinzip soll ausgehend von der Wiederholung eines lokalen Prozeduraufrufs genauer betrachtet werden.

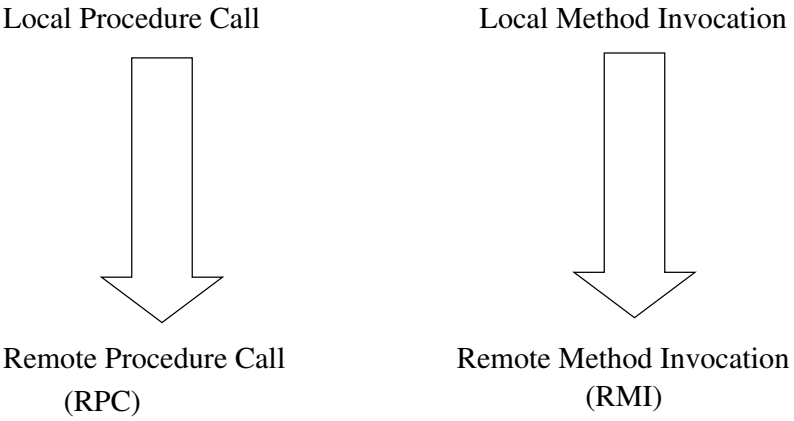


Abbildung 3.20: RPC - RMI

Konventionelle lokale Prozeduraufrufe Angenommen das Volumen eines Quaders soll mit dem konventionellen Unterprogrammaufruf *Volumen(Länge, Breite, Höhe)* berechnet werden, dann passiert folgendes:
In einem traditionellen System wird *Volumen* aus einer Bibliothek entnommen und durch den Binder in das ausführbare Programm eingebunden.
Die Routine *Volumen* ist in der Regel ein in Assembler geschriebenes Unterprogramm. Im Folgenden soll der Stack, d.h. der Kellerspeicher des Prozesses betrachtet werden. Vor dem Aufruf von *Volumen* enthält er lediglich lokale Variablen des Hauptprogramms. Für den Prozeduraufruf werden die Parameter in der Regel in umgekehrter Reihenfolge auf dem Stack abgelegt, dann die Rücksprungadresse gespeichert und das Unterprogramm ausgeführt. Es wird angenommen, daß der Ergebniswert in ein Register übertragen wird und nicht auf dem Stack abgelegt wird. Damit ist die Abarbeitung des Unterprogramms beendet. Die Rücksprungadresse wird vom Stack entfernt, und die Kontrolle geht wieder an den Aufrufer. Dieser nimmt die Parameter vom Stack, so dass der ursprüngliche Zustand vorliegt.

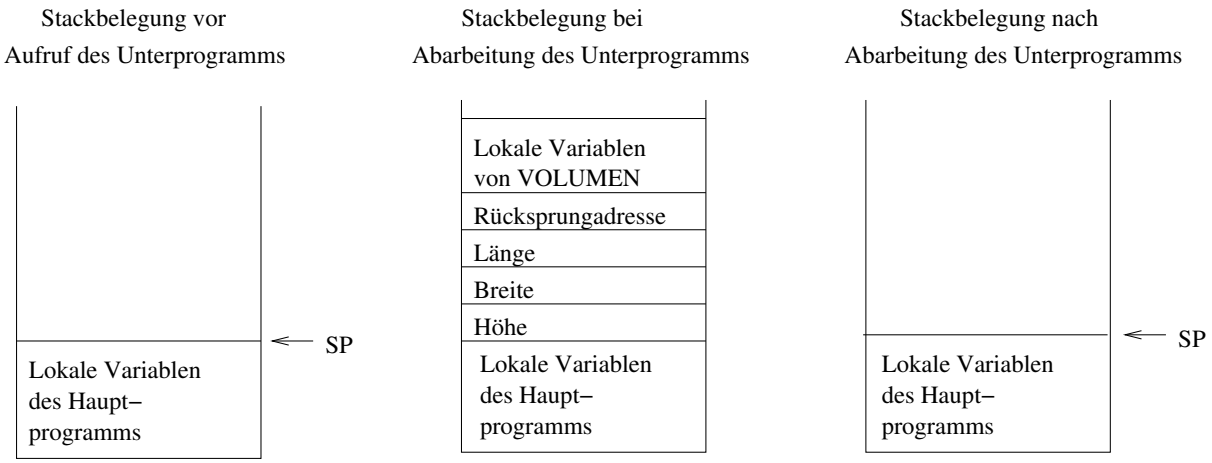


Abbildung 3.21: Stackbelegung bei lokalem Prozeduraufruf

Das Prinzip dieses lokalen Prozeduraufrufs kann auch wie in Abbildung 3.22 dargestellt werden.

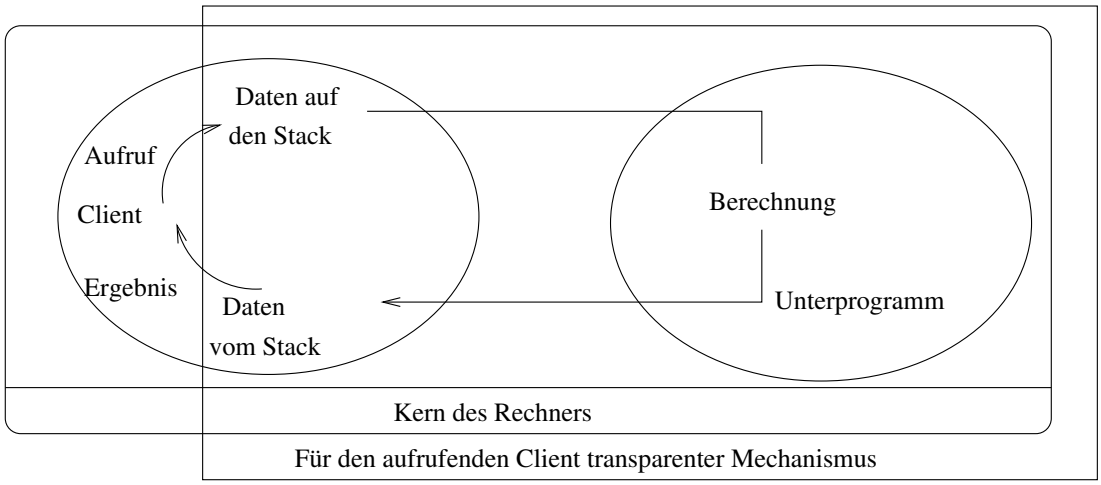


Abbildung 3.22: Lokaler Prozeduraufruf

Entfernte Prozeduraufrufe Ist *Volumen* nun ein entferntes Unterprogramm, so ist in diesem Fall eine andere Version von *Volumen*, ein so genannter Client-Stub in der Bibliothek enthalten. Der Aufruf erfolgt in Analogie zum lokalen Prozeduraufruf. Allerdings werden die Parameter nicht auf den Stack kopiert, sondern die Parameter werden in eine Nachricht verpackt (Marshalling) und der Kern durch das send-Primitiv beauftragt, die Nachricht an den Server zu schicken. Im Anschluss daran ruft der Client-Stub ein receive auf, und es erfolgt eine Blockierung, bis die Antwort eintritt.
Auf der Serverseite wird die eingetroffene Nachricht vom Kern an einen so genannten Server-Stub übergeben, der an den aktuellen Server gebunden ist. Der Server-Stub hat in der Regel gerade ein receive aufgerufen und wartet auf ankommende Nachrichten. Der Server-Stub packt die in der Nachricht enthaltenen Parameter aus (Unmarshalling) und ruft das Unterprogramm lokal auf. Parameter und Rücksprungadresse werden wie gewohnt abgelegt. Nach der Ausführung des Unterprogramms erfolgt die Rückgabe der Ergebnisse an den Server-Stub, der diese wieder verpackt (Marshalling) und die entstandene Nachricht mittels send an den Client zurückschickt. Zum Abschluss kann der Server-Stub wieder receive aufrufen und ist damit bereit, die nächste Nachricht zu empfangen.
Auf der Clientseite wird die eintreffende Nachricht dem Client-Stub übergeben. Damit ist die Blockierung auf Clientseite beendet, und die Nachricht wird entpackt (Unmarshalling). Dann kann das Ergebnis z.B. in Registern abgelegt werden. Wenn der Aufrufer von *Volumen* die Kontrolle zurückerhält, so ist ihm nur bekannt, dass Daten vorliegen. Es bleibt jedoch transparent, dass dieser Aufruf entfernt ausgeführt wurde.
Analog zum lokalen Prozeduraufruf ist das Prinzip des entfernten Prozeduraufrufs in Abbildung 3.23 zusammengefasst.
Auf diese Art kann ein entfernter Prozeduraufruf auf zwei lokale Prozeduraufrufe zurückgeführt werden, die vom Client an den Client-Stub bzw. vom Server-Stub an den Server gerichtet sind.

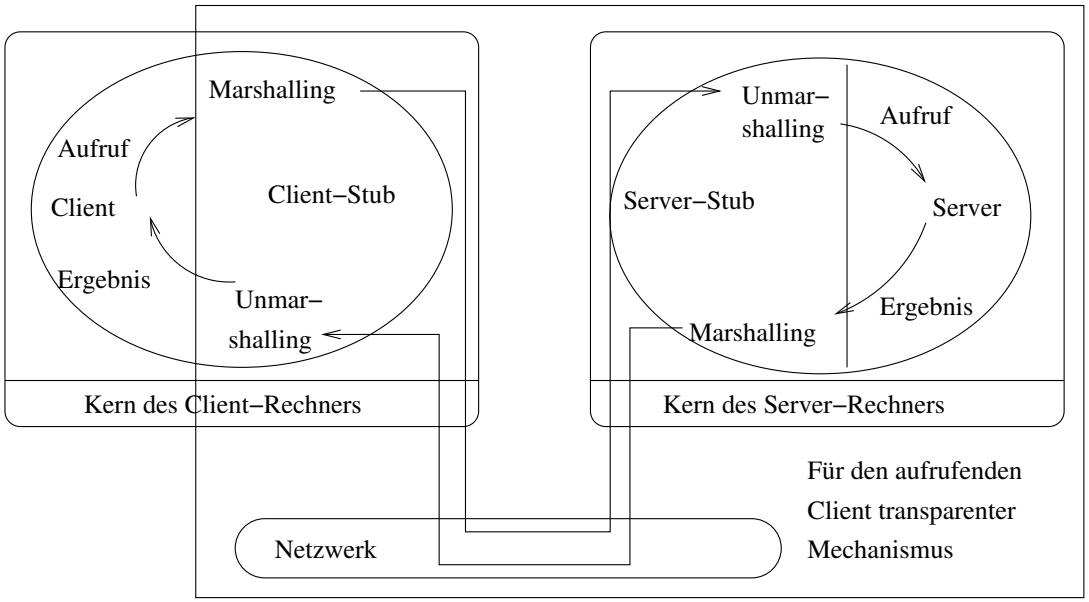


Abbildung 3.23: Entfernter Prozeduraufruf

Dynamisches Binden Zu Beginn einer Serverausführung erfolgt der Aufruf einer Initialisierungsmethode. Diese bewirkt das Exportieren der Schnittstelle des Servers.
Der Server sendet hierzu eine Nachricht an ein Programm, das Binder genannt wird, und gibt damit seine Existenz bekannt. Dieser Vorgang heißt auch Registrierung des Servers. Dabei übergibt der Server dem Binder zwecks Verwaltung in z.B. einer Tabelle

- seinen Namen
- ggf. die Versionsnummer und/oder einen eindeutigen Bezeichner sowie Remote Object Reference
- ein Handle, d.h. eine Referenz oder Adresse, die zur Lokalisierung des Servers dient.

Möchte der Server keinen Dienst mehr anbieten, so läßt er sich entregistrieren. Mit diesen Voraussetzungen ist dynamisches Binden möglich, das folgendermaßen geschieht:

- Der Client ruft zum ersten Mal ein entferntes Unterprogramm, z.B. *Volumen* auf.
- Der Client-Stub erkennt daraufhin, dass der Client an noch keinen Server gebunden ist.
- Der Client-Stub sendet eine Nachricht an den Binder, um z.B. die Version 1.0 der Schnittstelle *quader-server* zu importieren.

Der Binder überprüft, ob ein oder mehrere Server eine Schnittstelle mit diesem Namen und dieser Versionsnummer exportieren. Falls nicht, schlägt die Operation fehl. Existiert aber ein passender Server, so liefert der Binder das Handle und den eindeutigen Bezeichner an den

Client-Stub zurück. Der Client-Stub nutzt das Handle als Adresse, an die er die Anfragenachricht samt Parametern sendet.
Der Vorteil des dynamischen Bindens ist die hohe Flexibilität. Der Binder kann

– den Server in regelmäßigen Abständen überprüfen und ggf. entregistrieren und

– bei mehreren identischen Servern die Auslastung steuern (Load Balancing) sowie

– Authentifikation unterstützen.

Nachteile sind demgegenüber darin zu sehen, dass

– das Im- und Exportieren von Schnittstellen zusätzlichen Aufwand bedeutet und

– der Binder zum Engpass werden kann.

– Ausfall des Binder-Dienstes führt zum Systemausfall.

Für den Zugriff auf entfernte Objekte muss eine explizite Schnittstelle angegeben werden, ein so genanntes Interface. Dieses Interface spezifiziert die Prozeduren und Variablen, auf die von anderen Modulen aus zugegriffen werden kann. Bei der Implementierung von Modulen werden alle Informationen verborgen außer denen, die an der Schnittstelle sichtbar sind. Ebenso kann die Implementierung ausgetauscht werden, solange das Interface erhalten bleibt. Für das Funktionieren eines entfernten Zugriffs ist eine formale Spezifikation oder Definition der Schnittstelle notwendig.

Beispiel: Die Spezifikation eines Servers, der Operationen für Quader bereitstellt, kann prinzipiell wie folgt angegeben werden:

Specification of quader_server, version 1.0;

Volumen (in int länge, in int breite, in int höhe, out int raum);
Grundfläche (in int länge, in int breite, out int stellplatz);
Gewicht (in int länge, in int breite, in int höhe, in float dichte, out float schwere);

end;

Bei den spezifizierten Prozeduren oder Methoden werden Parameter und zugehörige Typen (in, out oder in/out) angegeben. Diese Typen beziehen sich immer auf den Server. Ein in-Parameter (z.B. länge) wird vom Client an den Server gesendet, geht also als Wert in die Berechnung eines Serverprogramms ein. Ein out-Parameter (z.B. raum) wird vom Server an den Client gesendet, der entsprechende Wert geht also aus der Berechnung des Unterprogrammaufrufs hervor. Ein in/out-Parameter wird schließlich vom Client an den Server gesendet, dort modifiziert und anschließend an den Client zurückgesendet.
Falls die verwendete Programmiersprache bereits eine geeignete Notation für die Definition

von Schnittstellen bereitstellt, bei der Input/Output-Parameter auf den normalen programmiersprachlichen Gebrauch abgebildet werden können, so ist dieser Mechanismus für den entfernten Zugriff auf Prozeduren geeignet.
Java RMI ist ein Beispiel dafür. Insbesondere ist dieser Ansatz realisierbar, da alle Teile einer entfernten Anwendung in derselben Programmiersprache geschrieben sind und da diese Sprache sowohl für lokale als auch für entfernte Aufrufe benutzt wird.
Nun kann es vorteilhaft sein, neben Java auch Dienste, die in anderen Programmiersprachen geschrieben sind, und ggf. schon existieren, in einem Verteilten System zu erlauben. Für einen solchen programmiersprachenübergreifenden Aufruf ist ein neues Konzept notwendig, die Interface Definition Language (IDL). Eine IDL stellt eine Notation für Schnittstellendefinitionen bereit, bei der jeder Parameter einer Methode zusätzlich zu seinem Typ als Input- oder Outputparameter beschrieben werden kann. Die CORBA IDL spezifiziert einen Namen und eine Menge von Methoden, die ein Client aufrufen kann. Für die Methoden werden Parameter angegeben, die durch die Schlüsselworte in, out oder inout als Input- bzw. Outputparameter oder beides charakterisiert werden.

```
1  //In file Person.idl

3  struct Person {
     String name;
5    String place:
     long year;
7  };

9  interface PersonList {
     readonly attribute string listname;
11   void addPerson(in Person p);
     void getPerson(in String name, out Person p);
13   long nummer();
   };
```

PersonList ist dabei eine Schnittstelle, die die entfernt aufrufbaren Methoden *addPerson* und *getPerson* samt zugehörigen in- bzw. out-Parameter spezifiziert. Ferner können in der IDL auch Exceptions angegeben werden, dies wird mittels Schnittstellen realisiert, die eigene Methoden repräsentieren.

Von der IDL zur Implementierung IDL-Spezifikationen werden mit IDL-Compilern weiterverarbeitet. Solche IDL-Compiler stehen für verschiedene Plattformen und verschiedene Programmiersprachen zur Verfügung. Sie generieren aus der allgemeinen IDL-Spezifikation 3 Module: einen Header sowie den Client- und Server-Stub. In manchen Verteilungsplattformen wie z.B. CORBA wird ein Server-Stub auch als Skeleton bezeichnet. Stubs und Skeletons sind dann nur auf einer bestimmten Plattform verwendbar.
Der Programmierer muss nun noch die Client- und Serverprogramme schreiben, die eine Funktion aufrufen oder realisieren. Dabei werden Funktionsaufrufe lokal behandelt, mittels der Stubs und der zugrundeliegenden Middleware wird die Kommunikation zwischen entfernten Objekten gewährleistet.

Im letzten Schritt werden die Programmcodes und die vom Compiler generierten Module gebunden. In der Praxis geschieht dies in der Regel über so genannte Makefiles.

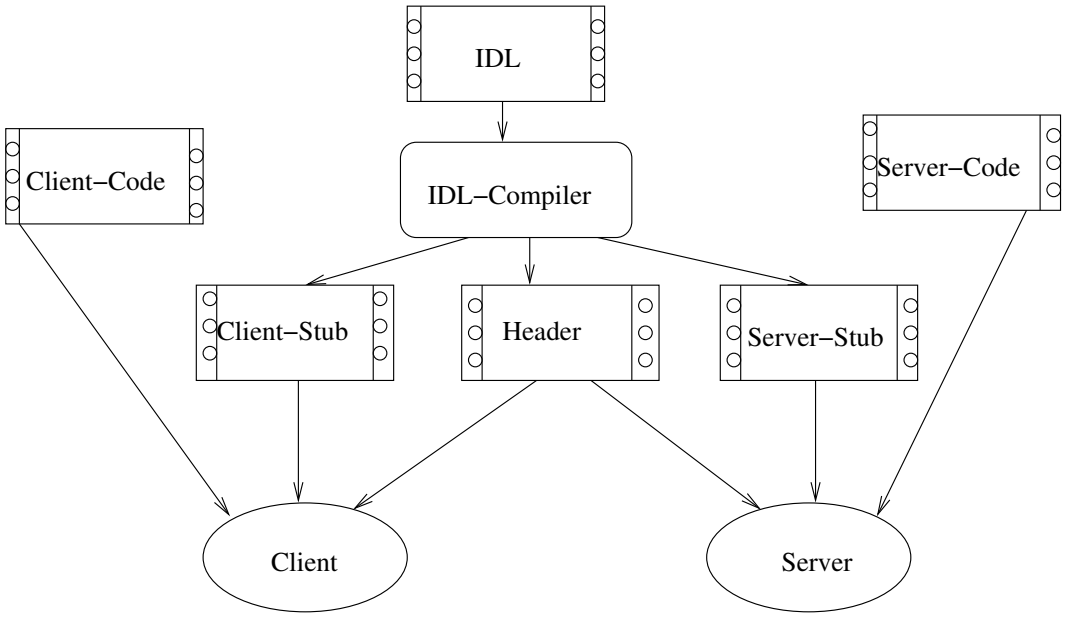


Abbildung 3.24: Grundlegendes Prinzip der Erstellung von Client und Server

4

Naming-, Directory und Lokalisierungsdienste

►

Inhaltsangabe

4.1 Motivation

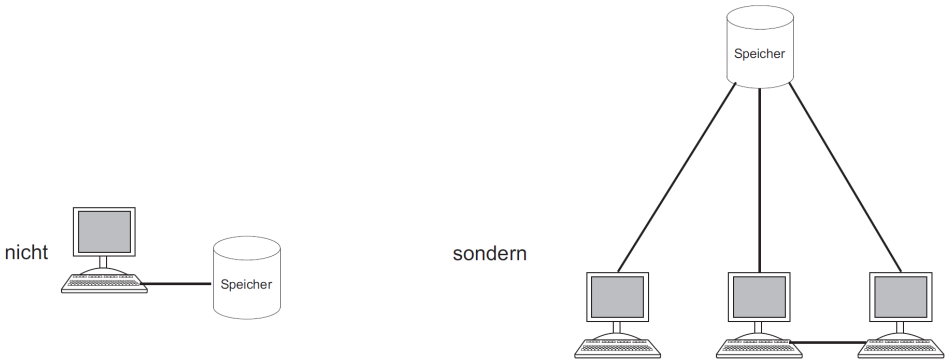


Abbildung 4.1: Motivation

Ein wichtiges Ziel verteilter Systeme ist die gemeinsame Nutzung von Ressourcen. Informationen, die in Form von Dateien im Netz vorliegen, werden durch Verteilte Dateisysteme unterstützt. Ein Dateiclient bietet dabei Zugriff auf Dateien, die auf einem Server abgelegt sind. Dabei wird eine ähnliche, z.T. sogar bessere Leistung und Zuverlässigkeit erzielt, als wenn diese auf lokalen Festplatten gespeichert werden würden.
Bezüglich Anlegen und Wiederfinden ergeben sich folgende Fragestellungen:

- Ablegen:
  Wie spreche ich die Dateien an? Name:
  - eindeutig
  - eineindeutig (= eindeutig in beide Richtungen)
  - abhängig von der Umgebung
- Wiederfinden:
  Wonach suchen?
  - Kenne ich die Resource
  - Weiß ich, wo sie liegt?
  - Gibt es sie überhaupt?
- Unaghängig davon kann es sein, dass unterschiedliche Situationan (Kontext) unterschiedliche Dienstwünsche bedingen (Ort, Wetter, etc.).

Das Verteilte Dateisystem ermöglicht es Programmen, entfernte Dateien genauso zu speichern und auf sie zuzugreifen wie dies lokal der Fall wäre. Dateisysteme sind verantwortlich für die Organisation, das Speichern, das Abrufen, das Benennen, die gemeinsame Nutzung und den Schutz von Dateien.
So können Benutzer von jedem Computer aus auf ihre Dateien zugreifen. Der Dateidienst ermöglicht es Programmen, auf entfernte Dateien genauso zuzugreifen wie auf lokale, so dass die Benutzer von jedem Computer im Netz aus auf ihre Dateien zugreifen können. Ferner befreien sie den Programmierer davon, sich mit den Details der Speicherreservierung und des

Speicherlayouts zu beschäftigen, so dass Dateien auf Festplatten oder anderen nicht flüchtigen Speichermedien abgelegt werden können.
Dateien verfügen sowohl über Daten als auch Attribute. Die Daten werden byteweise abgespeichert. Die Attribute werden in einem Datensatz festgehalten, sie umfassen bspw. Zeitstempel und Dateigröße, aber auch Dateityp, Benutzer-ID und Zugriffskontrolllisten sowie Eigentümer. Eigentümer, Dateityp und Zugriffskontrolllisten können von den Benutzer-Programmen aktualisiert werden. Die anderen Attribute werden vom Dateisystem verwaltet.

Vorteil: Diese Konzentration des persistenten Speichers auf wenige Server bedeutet, dass weniger lokaler Festplattenspeicher erforderlich ist und dass Einsparungen hinsichtlich der Verwaltung und Archivierung sowie Sicherung der Daten eines Unternehmens erzielt werden können.

Nachteile: Die zentrale Speicherung von Dateien hat aber auch Nachteile:

- Organisation
- Speichern/Backup/Benennen
- für eine sehr große Anzahl von Computern ist ein vollständig zentraler Ansatz schwer skalierbar
- Aufrufen/gemeinsamme Nutzung
- lokale Organisationen wollen ihre einen Dateisysteme verwalten

Diese Betrachtung von Dateien soll nun auch auf andere Ressourcen des Internets übertragen werden.
Ziel: Zugriff remote und lokal
Organisation von Daten: Dateien: Daten + Attibute (Zeitstempel, Größe, Typ, Nutzer-ID, etc.)
Trend: Konzentration von persistenten (=dauerhaftem) Speicher auf wenige Server. Vorteil: weiger Administration. Nachteile: Skalierbarkeit, „Befindlichkeit“ (= Kontrolle über eigene Daten).

4.2 Namen, Adressen und Identifikatoren

Ein **Name** in einem Verteilten System ist eine Folge von Bits oder Characters, die auf eine Entität verweisen, also auf eine Ressource wie z.B. einen Rechner, einen Drucker, eine Datei, einen Nutzer, Mailboxes, Newsgroups, Nachrichten, Prozesse oder Netzverbindungen.
Um eine Entität benutzen zu können, ist ein Zugang zu ihr notwendig. Dies geschieht über einen **Zugangspunkt (Access Point)**. Dieser wird auch als Adresse der Entität bezeichnet. Beispiele sind 48-Bit-lange Ethernetadressen oder 32-Bit- bzw. 64-Bit-lange Speicheradressen.

Beachten Sie, dass eine Entität auch mehr als einen (bzw. auch 0) Zugangspunkt und folglich mehr als eine Adresse haben kann (Telefonnummern, Portnummern eines Prozesses). Ferner ist es möglich, dass sich Zugangspunkte über die Zeit ändern. Wenn ein mobiler Computer seinen Ort ändert, so erhält er i.d.R. eine neue IP-Adresse, oder wenn ein Nutzer seinen Internet Service Provider ändert, so ist er über eine andere E-Mail-Adresse erreichbar.

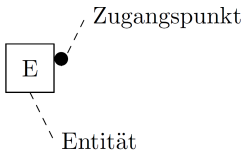


Abbildung 4.2: Schema von Entität und Zugangspunkt

Im Falle mehrerer Zugangspunkte ist es schwierig, eine Referenz für ein Objekt anzugeben. Aus diesem Grund ist es einfacher und flexibler, einen Namen pro Entität anzugeben, der eine eindeutige Identifikation ermöglicht. Das Ziel ist es eindeutig auf die Entität zugreifen zu können
Ein solcher **Identifikator (Identifier)** hat folgende Eigenschaften:

1. Ein Identifikator referenziert höchstens eine Entität, d.h. es gibt IDs, die nicht referenzieren.

2. Jede Entität wird durch höchstens einen Identifikator gleichzeitig referenziert. D.h. es gibt Entitäten ohne IDs.

3. Ein Identifikator referenziert immer dieselbe Entität, er wird auch zu einem späteren Zeitpunkt nicht für andere Zwecke wiederverwendet. D.h. eine Entität kann zu verschiedenen Zeiten duch verschiedene IDs referenzeirt werden (z.B. Personalausweisnummer).

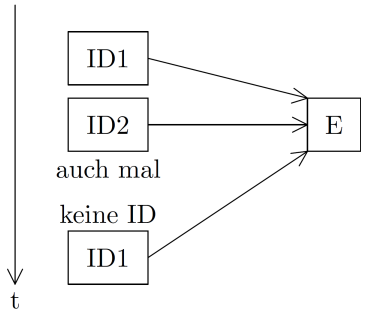


Abbildung 4.3: Verschiedene IDs zu verschiedenen Zeiten

Aus diesen drei Eigenschaften folgt, dass die Relation zwischen Identifikator und Entität nicht eineindeutig ist - es kann nämlich vorkommen, dass eine Entität mit der Zeit ihren Identifikator ändert.
Dieses Prinzip der Namensgebung, das unabhängig von der Betrachtung spezieller Adressen ist, wird **ortsunabhängig (location independent)** genannt. Es liefert einen Beitrag zur Realisierung von Ortstransparenz in Verteilten Systemen und später zum Lösen des Problems der Lokaliesierungsdienste.
Identifikatoren haben einen weiteren Vorteil: Während Adressen in der Regel eine maschinenlesbare Form verwenden (Bitstrings), sind Identifikatoren eher „human-friendly“, d.h. nutzerfreundlicher, da sie durch Schriftzeichen dargestellt werden.

Beispiel: Dateien in UNIX-Systemen können durch Name mit bis zu 255 Zeichen referenziert werden.

4.3 Namensräume und Namensauflösung

Namen werden in Verteilten Systemen innerhalb von Namensräumen organisiert.
Ein **Namensraum (Name Space)** ist ein markierter, gerichteter Graph mit zwei Arten von Knoten: Blattknoten und Verzeichnisknoten.
Ein **Blattknoten (Leaf Node)** hat keine abgehenden Kanten. Er repräsentiert eine Entität, die einen Namen besitzt. Der Blattknoten speichert in der Regel noch Informationen über die Entität, die er repräsentiert, z.B. ihre Adresse, so dass ein Client darauf zugreifen kann, oder ihren Zustand.
Ein **Verzeichnisknoten (Directory Node)** kann über abgehende Kanten, von denen jede mit einem Namen markiert ist, verfügen. Jeder dieser Knoten wird wiederum als eine andere Entität in einem Verteilten System betrachtet, verfügt auch über einen zugeordneten Identifikator und ist mit einem Namen markiert.
Ferner speichert jeder Verzeichnisknoten eine Tabelle ab, in der den Markierungen der abgehenden Kanten die Identifikatoren der nachfolgenden Knoten zugeordnet werden. Solch eine Tabelle wird auch als **Verzeichnistabelle (Directory Table)** bezeichnet.

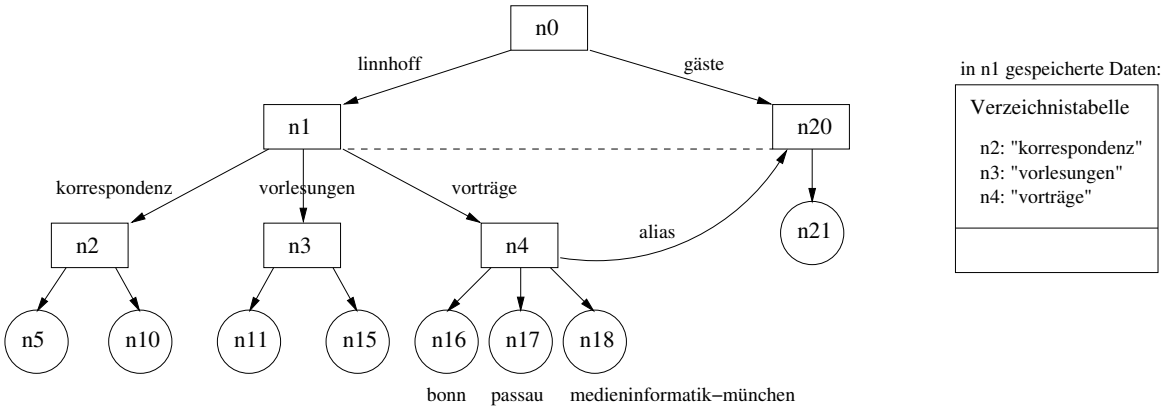


Abbildung 4.4: Namensraum

In der Abbildung 4.4 hat *n0* lediglich abgehende, aber keine eingehenden Kanten. Ein solcher Knoten wird als **Wurzelknoten (Root Node)** bezeichnet. Prinzipiell ist es möglich, dass Namensgraphen mehrere Wurzelknoten haben. Der Einfachheit halber haben die Namenssysteme jedoch in der Regel nur einen solchen Wurzelknoten.
Jeder Knoten im Namensgraphen kann nun durch eine Folge von Marken angegeben werden, die den Kanten als Pfad von einem Knoten N bis hin zum ausgewählten Knoten entsprechen. Dabei wird die Syntax

N: <Marke 1, Marke 2, ... , Marke n>

verwendet, wobei N den ersten Knoten des Pfades bezeichnet. Solch eine Folge wird **Pfadname (Path Name)** genannt.
Ist der erste Knoten die Wurzel, so spricht man von einem absoluten Pfadnamen (Absolute

Path Name), anderenfalls von einem relativen Pfadnamen (Relative Path Name).
Beispiel für den Knoten *n16* ist der absolute Pfadname:

n0: <linnhoff, vorträge, bonn>

Diese Vorgehensweise ähnelt der in vielen Dateisystemen. Allerdings verwenden Dateisysteme eine andere Syntax. Als Trennzeichen wird ein Slash genutzt, so dass sich im Beispiel ergibt:

/linnhoff/vorträge/bonn

Links dürfen verwendet werden, die Graphen sollten jedoch azyklisch bleiben.
Im Falle des Namensgraphen des Betriebssystems UNIX stellt ein Verzeichnisknoten ein Dateiverzeichnis dar. Ein Blattknoten repräsentiert eine Datei. Es gibt einen Wurzelknoten. Die Implementierung muss eine Fülle von Details berücksichtigen.
Können mehrere Pfade zum gleichen Knoten führen, z.B. durch Aliase, so gibt es auch verschiedene Pfadnamen, z.B. /linnhoff/vorträge/gäste/boger und /gäste/boger.
In der Regel sind sie jedoch strikt hierarchisch, so dass es immer genau einen absoluten Pfadnamen gibt. Allerdings sind Namensgraphen immer azyklisch, d.h. auch im Falle von Querverweisen können keine Zyklen auftreten.
Wir wollen Namensräume nun nutzen, um Informationen über Entitäten mit Hilfe von Namen zu speichern und abzurufen. Der Prozess des Heraussuchens eines Namens wird dabei auch als **Namensauflösung (Name Resolution)** bezeichnet. Basierend auf dem Pfadnamen *N*:<*Marke 1*, ... , *Marke n*> beginnt die Namensauflösung am Knoten *N*. In der Verzeichnistabelle des Knotens *N* wird nach der *Marke 1* gesucht und der Identifikator des nachfolgenden Knotens zurückgegeben. In dessen Verzeichnistabelle wird nach *Marke 2* gesucht usw. bis man den Identifikator des Knotens erhält, auf den die Kante mit *Marke n* zeigt.
Wichtig ist dabei die Kenntnis über den Startpunkt und der Zugang zu den Verzeichnistabellen. Dabei unterscheidet man lokale und globale Namen.
Ein **globaler Name (Global Name)** ist ein Name, der immer dieselbe Entität bezeichnet. Er wird immer bezüglich desselben Verzeichnisknotens interpretiert, ganz egal wo im System dieser Name genutzt wird.
Bei einem **lokalen Namen (Local Name)** hängt die Interpretation vom Ort seiner Benutzung ab. Aus Sicht des Pfadnamens ist der Name relativ, wobei der zugehörige Ausgangsknoten implizit bekannt ist.
Bislang sind wir davon ausgegangen, dass die Namensauflösung in einem einzigen Namensraum stattfindet. Nun wollen wir jedoch eine Menge von Namensräumen betrachten, die über verschiedene Computer verteilt sind. Dabei vermischen sich die Namensräume.
Der (lokale) Knoten, der einen entfernten Identifikator speichert, wird **Mountpunkt (Mount Point oder Einfügepunkt)** genannt. Dies ist nicht mit dem Mounting Point zu verwechseln, welcher in der Regel die Wurzel ist. Der entsprechende Verzeichnisknoten des entfernten Namensraums, auf den verwiesen wird, heißt **Mountingpunkt (Mounting Point)**. Er ist in der Regel ein Wurzelknoten.
Um einen entfernten Namensraum innerhalb eines Verteilten Systems mounten zu können, müssen folgende Informationen vorliegen:

1. Der Name eines Zugangsprotokolls, d.h. eines Kommunikationsprotokolls, über das auf den entfernten Namensraum zugegriffen werden kann.

2. Der Servername, d.h. aufgelöst in eine Adresse, über die der Server erreicht werden kann.

3. Der Name eines Mountingpunkts im entfernten Namensraum, d.h. wieder aufgelöst in einen Identifikator für den Knoten des entfernten Namensraums.

Durch die Verknüpfung der Namensräume entsteht ein beliebig großer Namensraum. Für ein weltweites Verteiltes System mit Millionen von Entitäten, die über große geographische Gebiete verteilt sind, muss damit der Namensraum auch über mehrere Nameserver verteilt und geeignet strukturiert werden.
Gewöhnlich wird eine hierarchische Organisation zugrunde gelegt, bei der von einem einzigen Wurzelknoten ausgegangen wird. Um eine effiziente Implementierung zu ermöglichen, wird der Namensraum in logische Schichten eingeteilt. Seit 1989 unterscheidet man dabei die folgenden 3 Schichten:

– **Global Layer**: beinhaltet die Knoten auf höchstem Level, d.h. Länder und große bedeutende Organisationsformen, die sehr stabil sind, deren Verzeichnistabellen sich also kaum ändern.

– **Administrational Layer**: beinhaltet die Verzeichnisknoten, die von einzelnen Organisationen verwaltet werden. Dabei gehören Gruppen von Entitäten zur selben Verwaltungseinheit. Daten sind hier relativ stabil, ändern sich aber häufiger als auf dem Global Layer.

– **Managerial Layer**: besteht aus den Knoten, die sich i.d.R. häufig ändern. Dazu gehören Knoten die Hosts, Dateien und Verzeichnisse repräsentieren. Im Gegensatz zu den anderen beiden Schichten werden diese nicht nur von Systemadministratoren, sondern auch von Endnutzern verwaltet.

Beispiel: Es resultieren hohe Anforderungen an die Verfügbarkeit und Leistungsfähigkeit der entsprechenden Nameserver. Hierzu werden Server auch repliziert und clientseitig ein Caching durchgeführt. Auf Konsistenzfragen der Replikate wird in Kapitel 9 zurückgekommen.
Ein Vergleich der Anforderungen und Eigenschaften der einzelnen Schichten ist in der Tabelle 4.1 enthalten.
In derartig großen Namensräumen gibt es zwei prinzipiell unterschiedliche Möglichkeiten der Namensauflösung. Dabei soll angenommen werden, dass die Nameserver nicht repliziert vorliegen und kein Caching verwendet wird. Jeder Client hat aber Zugriff auf einen Namensauflöser (Name Resolver), d.h. eine Komponente, die für die Ausführung der Namensauflösung

	Global Layer	Administrational Layer	Managerial Layer
Geographischer Umfang:	weltweit	Organisation	Institut
Anzahl der Knoten:	wenige	viele	enorm viele
Dauer einer Suchanfrage:	Sekunden	Millisekunden	sofort
Anzahl der Replikate:	viele	keine oder wenige	keine
Clientseitiges Caching:	ja	ja	manchmal

Tabelle 4.1: Global - Administrational - Managerial - Layer

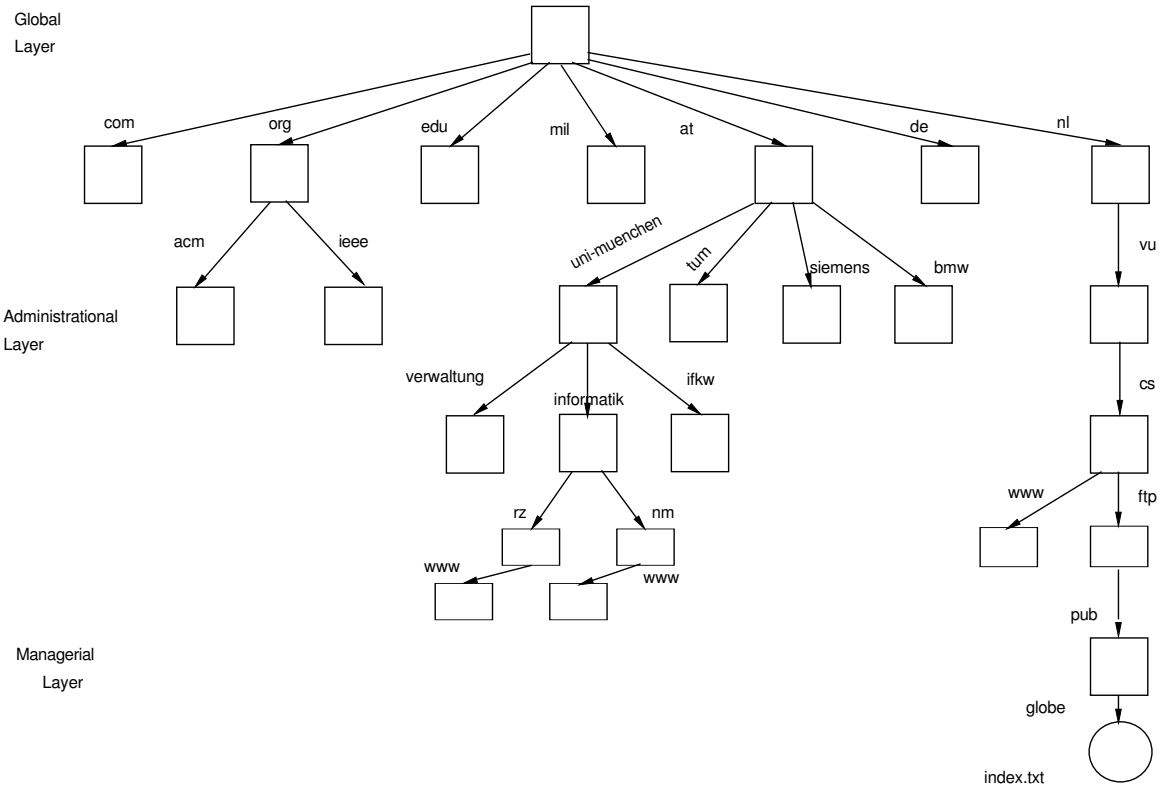


Abbildung 4.5: Beispiel für die Schichtenstruktur des Internets

zuständig ist.
Dabei gehen wir von folgendem absoluten Pfadnamen aus:

root:<*nl*, *vu*, *cs*, *ftp*, *pub*, *globe*, *index.txt*>
ftp:<*pub*, *globe*, *index.txt*>
oder in URL-Notation: *ftp://ftp.cs.vu.nl/pub/globe/index.txt*

Es gibt zwei prinzipielle Möglichkeiten, die Namensauflösung zu implementieren:

Iterative Namensauflösung Es wird angenommen, dass der Namensauflöser die Adresse des Root Servers kennt und ihm den vollständigen absoluten Pfadnamen schicken kann.
Der Root Server kann nun beispielweise die Marke *nl* auflösen und schickt dem Client die Adresse des assoziierten Nameservers zurück. Dieser Nameserver erhält den verbleibenden Pfadnamen

nl:<*vu*, *cs*, *ftp*, *pub*, *globe*, *index.txt*>

usw. bis die Adresse des ftp-Servers erhalten wird. Der Client kontaktiert den ftp-Server, der die verbleibenden Marken auflöst und die angeforderte Datei an den Client zurückschickt.
Beachte, dass der absolute Pfadname bei diesem Vorgehen in zwei Teile gesplittet wird. Der Namensauflöser des Clients nutzt

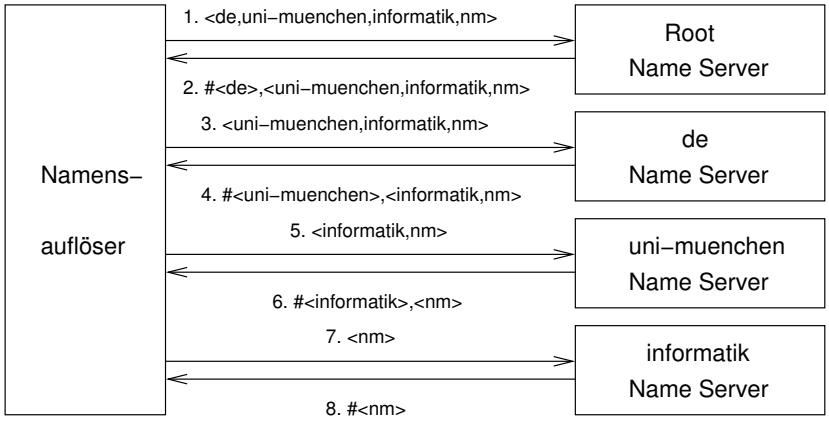


Abbildung 4.6: Iterative Namensauflösung

root:<*nl*, *vu*, *cs*, *ftp*>

um die Adresse des ftp-Servers zu erhalten. Separat davon kontaktiert der Clientprozess den ftp-Server danach mit dem Pfadnamen

ftp:<*pub*, *globe*, *index.txt*>

um die Datei zu erhalten.
Der Vorteil ist, dass keine Infrastukturmaßnahme erforderlich ist. Der Nachteil dabei ist viel Netzverkehr. Clientseitiges Caching ist daher evtl. sinnvoll.

Rekursive Namensauflösung Anstatt jedes Zwischenergebnis an den Namensauflöser des Clients jedes mal zurückzugeben, wird das Ergebnis nun direkt an den nächsten Nameserver weitergeleitet. Dabei kann die Aufgabe des Namensauflösers auch wieder darin bestehen, die Adresse des ftp-Servers zu erhalten, so dass der Client diesen in einem separaten Prozess direkt kontaktiert.
Diese rekursive Namensauflösung hat zwei Vorteile:

– Es kann ein effektives Caching bei den Nameservern eingesetzt werden, da sich die Entitäten der Globalen und Administrationsschicht in der Regel kaum ändern. Insbesondere ist dies sinnvoll, wenn viele Clients dieselben Anfragen stellen.

– Sie reduziert die Netzlast zwischen dem Namensauflöser des Clients und den Nameservern.

Allerdings erfordert sie eine höhere Performance der Nameserver und die Unterstützng von diesem für diese rekursive Auflösung.
Bei der erst genannten iterativen Namensauflösung ist Caching innerhalb der Nameserver wenig sinnvoll. Hier ist lediglich clientseitig Caching von Vorteil.

4.4 Namens- und Verzeichnisdienste

Domain Name System (DNS) Einer der größten verteilten Namensdienste ist das Internet **Domain Name System (DNS)**. DNS wird vorwiegend zur Suche von Hostadressen und Mail-

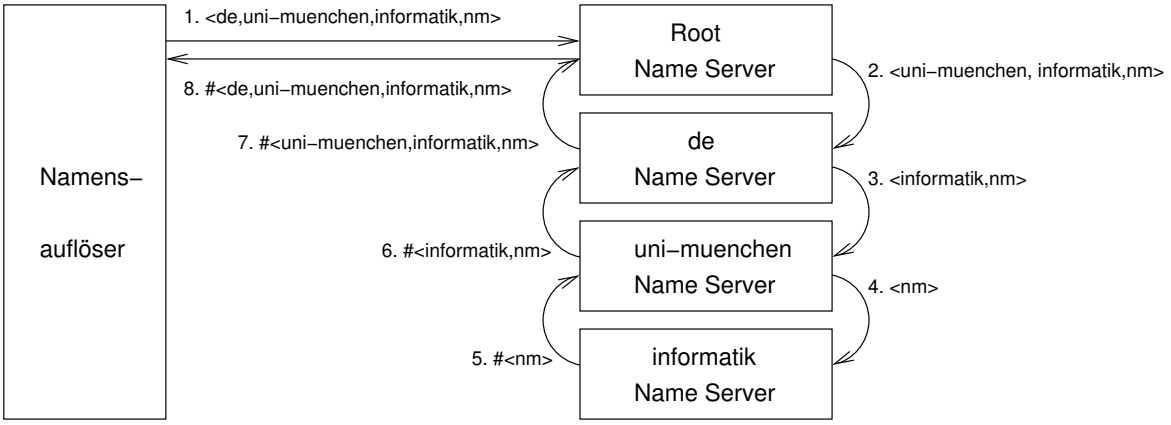


Abbildung 4.7: Rekursive Namensauflösung

servern genutzt.
Der DNS Namensraum ist organisiert als Baum, in dem es eine Wurzel gibt und in dem jeder Knoten genau eine eingehende Kante hat. Ausnahme ist die Wurzel; sie hat keine eingehende Kante. Die Marken der Kanten werden ebenfalls zur Bezeichnung der Knoten verwendet.
Eine Marke besteht aus maximal 63 Zeichen. Ein kompletter Pfadname ist auf 255 Zeichen beschränkt. Die Darstellung des Pfadnamens besteht aus der Auflistung der Marken, die durch Dots, d.h. Punkte getrennt werden. Z.B. *root:<de,uni-muenchen,informatik>* wird dargestellt als *informatik.uni-muenchen.de,* wobei der die Wurzel anzeigende letzte Punkt in der Regel weggelassen wird.
Ein Teilbaum wird **Domäne (Domain)** genannt. Ein Pfadname zu seiner Wurzel heißt **Domänenname (Domain Name)**. In Analogie zu den Pfadnamen kann ein Domänenname entweder relativ oder absolut sein.
Der Inhalt eines Knotens wird durch eine Menge so genannter **Resource Records** dargestellt. Diese Resource Records können verschiedene Typen besitzen, je nachdem welche Art von Informationen sie speichern.
Beispiel:

– Der Resource Record vom Typ A bezieht sich auf eine Host-Entität und enthält die IP-Adresse des entsprechenden Hosts.

– Der Typ MX (Mail eXchanger) bezieht sich auf die Entität der Domäne und referenziert einen Mailserver, um Mails, die an diesen Knoten geschickt werden, handhaben zu können.

Der DNS-Namensraum wird in disjunktive Teile eingeteilt, die als **Zonen (Zones)** bezeichnet werden. Eine Zone ist somit Teil des Namensraumes, und zwar der Teil, dem ein separater Nameserver zugeordnet ist.
Resource Records vom Typ NS sind einer Zone zugeordnet und referenzieren den Nameserver, der die Zone zur Verfügung stellt. Dabei kann ein Knoten in einem DNS Namensraum auch gleichzeitig eine Domäne und eine Zone repräsentieren, z.B. auf der Ebene einer Organisation.
DNS ist ein Beispiel eines traditionellen **Naming Services**. Bei gegebenem Namen löst DNS

diesen zu einem Knoten im Namensgraphen auf und gibt den Inhalt des Knotens in Form des Resource Records zurück. In diesem Sinne ist DNS vergleichbar mit dem Suchen von Telefonnummern in einem Telefonbuch (Weiße Seiten).
Ein ganz anderer Ansatz wird durch einen **Directory Service** realisiert. An Stelle eines vollen Namens ist als Sucheingabe die Beschreibung einer oder mehrerer Eigenschaften einer Entität gegeben. Dieser Ansatz ist daher eher mit den Gelben Seiten vergleichbar.

Der X.500 Namensraum

X.500 ist ein Directorydienst und basiert ebenfalls auf Records, diese werden als **Verzeichniseinträge (Directory Entries)** bezeichnet. Sie sind vergleichbar mit den Resource Records in DNS.
Ein X.500 Record besteht aus einer Sammlung von (Attribut, Wert)-Paaren, wobei single-valued- und multiple-valued-Attribute unterschieden werden, je nachdem, ob einem Attribut genau ein Wert oder auch eine Liste von Werten zugeordnet werden kann.
Die Menge aller Verzeichniseinträge wird in X.500 **Directory Information Base (DIB)** genannt. Dabei ist jeder Record eindeutig benannt, so dass nach ihm gesucht werden kann. Die Hierarchie der Einträge heißt **Directory Information Tree (DIT)**.
Durch das Betrachten der ersten Namensattribute (C, L, O, OU und CN) entstehen global eindeutige Namen pro Record, so genannte **Globally Unique Names**.
Ein global eindeutiger Name ist so beispielweise:

/C=DE/O=Ludwig-Maximilians-Uni/OU=Institut für Informatik

Was dem informatik.uni-muenchen.de im DNS entspricht.
Ein Namingattribut wird dabei auch als **Relative Distinguished Names (RDN)** bezeichnet. Diese führen als Folge zu global eindeutigen Namen.
Zum Suchen stellt X.500 zwei Operationen bereit:

– *read* wird genutzt, um einen einzelnen Record zu lesen und

– *list* gibt die Namen aller ausgehenden Kanten an.

Attribut	Abkürzung	Wert
Country	C	DE
Locality	L	München
Organization	O	Ludwig-Maximilians-Uni
Organizational Unit	OU	Institut für Informatik
Common Name	CN	Main Server
Mail-Servers	-	Listen der
FTP-Servers	-	zugehörigen
WWW-Servers	-	IP-Adressen

Tabelle 4.2: Beispiel für einen X.500 Verzeichniseintrag eines Knotens

4.5 Lokalisierungsdienste

Die bislang beschriebenen Namensdienste eignen sich primär für Entitäten, die einen festen Ort haben. Mobile Entitäten zeichnen sich dagegen durch häufige Änderungen der Name-to-Address-Abbildungen aus. Diese Eigenschaften sollen durch erweiterte Konzepte realisiert werden.
Bei den stationären Objekten konnten - über die Zeit gesehen - verschiedene Identifier, d.h. verschiedene Namen, zur Referenzierung einer Entität verwendet werden.
Durch Namensauflösung wurde der entsprechende Identifier auf den Access Point, d.h. die Adresse der Entität abgebildet.

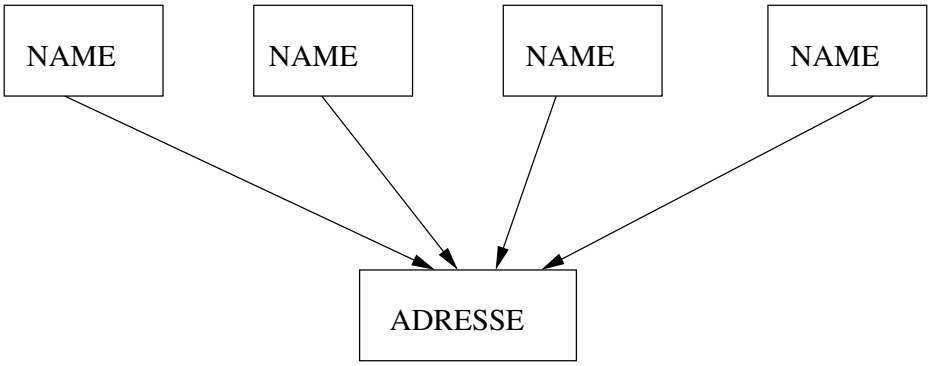


Abbildung 4.8: Name-to-Address-Abbildung bei stationären Entitäten

Nun werden mobile Entitäten betrachtet, bei denen sich zusätzlich zum Namen die physikalische Adresse ändern kann.

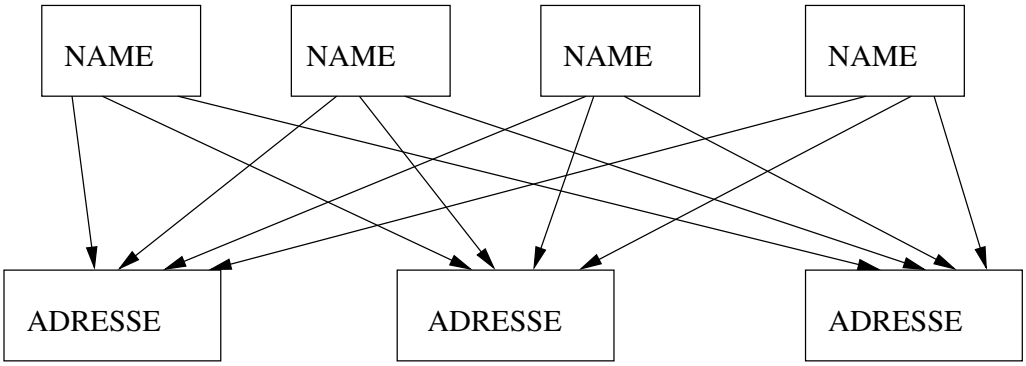


Abbildung 4.9: Name-to-Address-Abbildung bei mobilen Entitäten

Traditionelle Namensdienste basieren auf einer statischen Abbildung zwischen Namen und Adressen, und diese Abbildung muss mit jeder Namens- oder Adressänderung auch geändert werden. Damit ergibt sich eine recht umständliche Verwaltung und der Wunsch nach einem anderen Konzept.
Eine bessere Lösung besteht in dem Einführen eines eindeutigen Identifiers, der statisch immer derselben Entität zugeordnet ist. Ändern sollen sich in diesem Fall nur die Namen, die

auf diesen Identifier abgebildet werden. Diese Lösung ist also 2-stufig.
Neben diesem Naming wird dann die Lokalisierung der Entität über eine Abbildung des Identifiers auf eine Adresse realisiert.

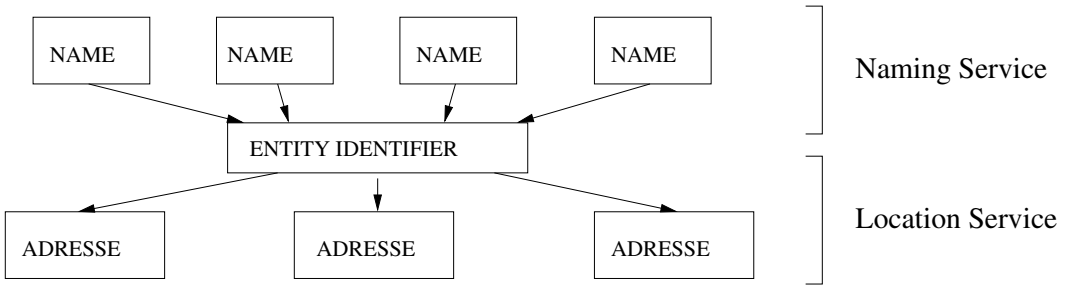


Abbildung 4.10: Name-to-Identifier-to-Address-Abbildung

In diesem Fall sollte der Identifier von seiner Struktur her eine maschinenfreundliche Darstellung haben und sich nicht ändern! Ändern kann sich jedoch die Adresse, weshalb diese nicht mit einem Identifier gleichgesetzt werden sollte.
Der **Naming Service** sucht nun zu einem gegebenen Namen einen Identifier heraus. Dann ist ein **Location Service** notwendig, der zu einem Identifier die aktuelle Adresse angibt. Im Fall replizierter Objekte können auch mehrere Adressen angegeben werden.
Dieser Location Service ist Gegenstand der nachfolgenden Betrachtungen. Zunächst sollen zwei sehr einfache Realisierungen eines Location Services betrachtet werden, die jedoch nur in Local Area Networks mit überschaubarer Größe angewendet werden können. Diese Realisierungen können kombiniert werden, z.B. GSM: In der Regel wird jedesmal pro Hirairchieebene (in verteilten Systemen wird Skalierbarkeit durch Einfühung von Hirarchien erhalten) in Lokalisierungsdienst verwendet. GSM hat folgende 3 Ebenen: Provider, Localisation Area und Funkzelle.

1. **Broadcasting und Multicasting**
   Eine Nachricht, die den Identifier einer Entität enthält, wird an jeden Rechner des LANs gebroadcastet. Der Rechner, der einen Zugangspunkt zu der Entität bereitstellen kann, sendet eine Antwort mit der Adresse des Zugangspunkts. Vorteil: einfach (realisierbar ohne Infrastruktur), oft alternativlos
   Es ist jedoch ineffizient, dieses Verfahren zu skalieren. Zum einen wird Bandbreite (insbesondere bei vielen Quellen) verschwendet, zum anderen kosten die Anfragen Prozessorzeit (z.B. AD-Hoc-Netze).

2. **Forwarding Pointers**
   Dieses Prinzip ist extrem einfach: Wenn eine Entität von Rechner A nach Rechner B migriert, so wird auf A eine Referenz auf den neuen Ort auf B hinterlassen. Dadurch ergeben sich Ketten von Forwarding Pointers.
   Vorteil: Bei geringer Mobilität besser skalierbar als Broadcast und Multicast.
   Von Nachteil ist wieder die Skalierbarkeit, d.h. das Enstehen zu langer Ketten bei sehr mobilen Entitäten und die Fehleranfälligkeit bei Ausfall einer Komponente.
   Im Folgenden sollen Ansätze betrachtet werden, die auch für etwas größere Netze angewendet werden können.

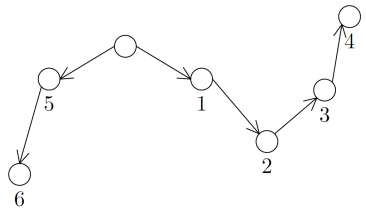


Abbildung 4.11: Forwarding Pointers

3. **Home Location**
   Vorraussetzung: Für eine Entität existiert eine "Home Location" (in der Regel der Ort, wo das Gerät geschaffen wurde). Die Home Location ist der Ort, an dem eine Entität geschaffen wurde. Sie speichert zu jeder Zeit den aktuellen Ort der Entität. Dieser Ansatz wird auch als Rückfalllösung für die Forwarding Pointers genutzt und ist außerdem seit 1997 eine Grundlage für Mobile IP.

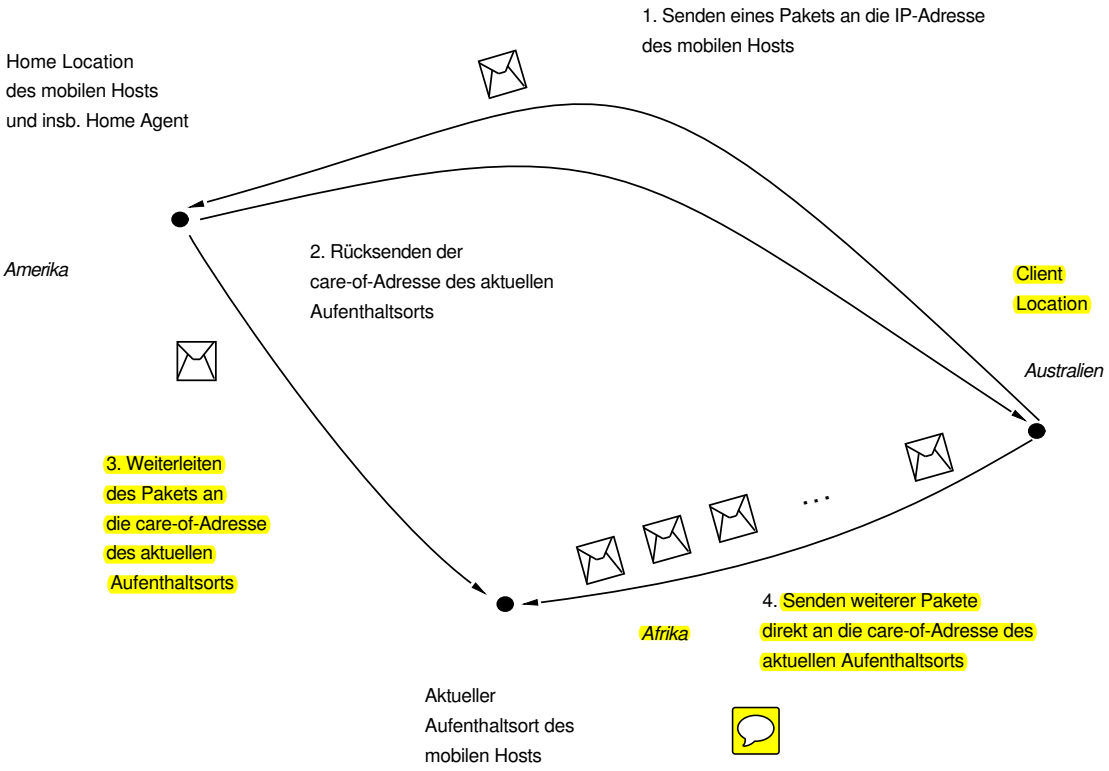


Abbildung 4.12: Prinzip von Mobile IP

Dies ist besser skalierbar als Broadcast (4 statt n Anfragen) und braucht keine Zentrale, sondern eine Komponente pro Resource.

Das Prinzip von Mobile IP Jeder mobile Host nutzt eine feste IP-Adresse. Die gesamte Kommunikation an diesen mobilen Host oder genauer die zugeordnete IP-Adresse geht an einen **Home Agent** (Heimatagenten, Homeagenten), der sich an der Home Location des mobilen Hosts befindet. Die Netzwerkadresse des Home Agents ist dabei gleich der IP-Adresse des mobilen Hosts.
Bewegt sich der mobile Host in ein anderes Netz, so fordert er eine temporäre Adresse an, die er für Kommunikationszwecke benötigt. Sie wird auch **Care-of-Adresse** genannt. Diese Adresse wird im Home Agent registriert.
Erhält der Home Agent nun ein Paket für den mobilen Host, so sucht er den aktuellen Aufenthaltsort des mobilen Agenten heraus.

- Befindet sich der Host im lokalen Netz, so wird das Paket einfach geforwardet.

- Anderenfalls wird es als IP-Paket verpackt an die Care-of-Adresse gesendet. Gleichzeitig wird der ursprüngliche Sender des Pakets über den aktuellen Aufenhaltsort des mobilen Hosts informiert, so dass nachfolgende Pakete direkt vom Client an die Care-of-Adresse gesendet werden können.

Problem 1: Nachteilig ist, dass - auch dann, wenn sich Client und mobiler Host im selben Netz befinden sollten - immer erst die Home Location kontaktiert wird. Falls der Anfragende und der Angefragte sich in einem Teilnetz befinden, so entsteht ein weiläufiger Traffic (Initialtraffic), der gar nicht nötig wäre.
In der Mobilkommunikation wird daher ein auf die Wissenschaftler Mohan und Jain (1994) zurückgehender, zweistufiger Ansatz verwendet, wobei erst geprüft wird, ob ein mobiles Gerät lokal verfügbar ist und dann der Home Agent angefragt wird.
D.h. lokale Skalierbarkeit ist nicht vorhanden. Das Ziel ist, dass ich nur über Heimat- oder Wurzelknoten geht, falls sich der Empfänger nicht in meiner Umgebung befindet. Die nächste Idee ist also, dass man zuert abklärt, ob das Ziel in meiner Nähe ist. Und falls nicht wird ein zentraler Knoten kontaktiert.

Problem 2: Problematisch ist außerdem die Existenz und Erreichbarkeit der Home Location. Falls die Adresse der Home Location nicht bekannt ist, müsste ferner zunächst ein Naming Service genutzt werden.

4. **Globe Location Service**
Der wohl beste Ansatz geht auf das Globe-System und eine Arbeit von Maarten von Steen aus dem Jahr 1998 zurück. Er wird in der Literatur auch als hierarchischer Ansatz bezeichnet, da er auf einer geschichteten Architektur basiert.

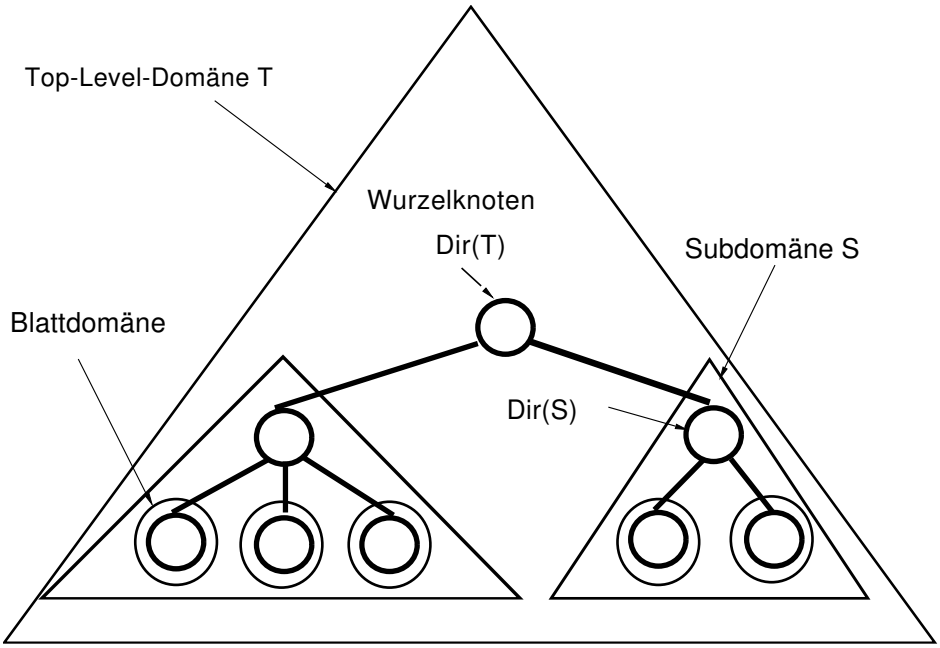


Abbildung 4.13: Baumstruktur des Globe Location Services

Idee: Ein Directory, das den Aufenthaltsort aller mobilen Entitäten speicher (sehr schlecht skalierbar).
Problem: Wenn ich jede Ortsänderung dort aktualisiere (Location Update), dann entsteht ein extremer Kommunikationsverkehr an der Konponente (entspricht zentralem Directory). Die Abhilfe lautet:
Vorraussetzung: Die Home-Location ist hier egal.
Dieser Ansatz erweitert den zweistufigen Home Location Ansatz hin zu mehreren Stufen. Es wird zunächst von einer DNS-ähnlichen Organisation des Netzes ausgegangen, wobei das Netz in eine Menge von Domänen aufgeteilt ist. Jede Domäne kann in mehrere, kleinere Subdomänen weiter aufgeteilt werden. Eine Domäne auf unterster Ebene wird auch **Blattdomäne (Leaf Domain)** genannt. Sie entspricht i.d.R. einem LAN in den Computernetzen oder einer Zelle im Mobilnetz.
Analog zu DNS hat jede Domäne *D* einen assoziierten Verzeichnisknoten dir(D), der auch Wurzelknoten der Domäne ist und alle Entitäten der Domäne kennt.
Zwecks Verwaltung der Entitäten wird zu jeder Entität, die sich aktuell in einer Domäne *D* befindet, ein **Location Record** in Dir(D) gespeichert. Dieser Location Record enthält die aktuelle Adresse der Entität in der Domäne. Die übergeordnete Domäne legt ebenfalls einen Location Record an, speichert darin aber lediglich einen Zeiger auf den Verzeichnisknoten Dir(D) ab. Folglich speichert der Wurzelknoten für jede Entität einen Location Record, der jeweils einen Zeiger auf den Verzeichnisknoten der nächstniedrigeren Domäne enthält. Falls eine Entität repliziert ist, so hat sie mehrere Adressen. Falls die Replikate in zwei unterschiedlichen Domänen liegen, so hat die kleinste, übergeordnete Domäne, die diese beiden Domänen enthält, in ihrem Location Record zwei Zeiger (vgl. Abb. 4.14).

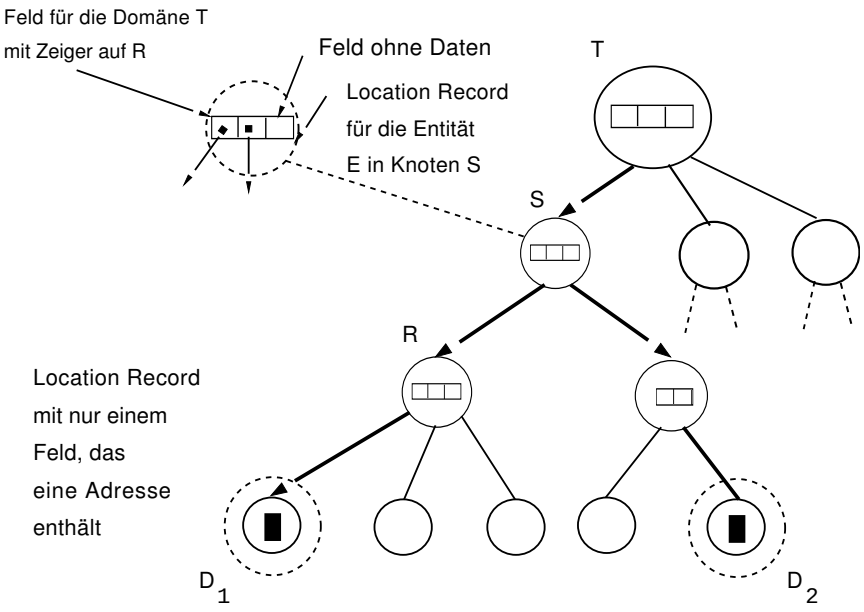


Abbildung 4.14: Beispielszenario für replizierte Entitäten

Für die Lokalisierung von Entitäten wird nun von einem ähnlichen Szenario ausgegangen (vgl. Abb. 4.15).

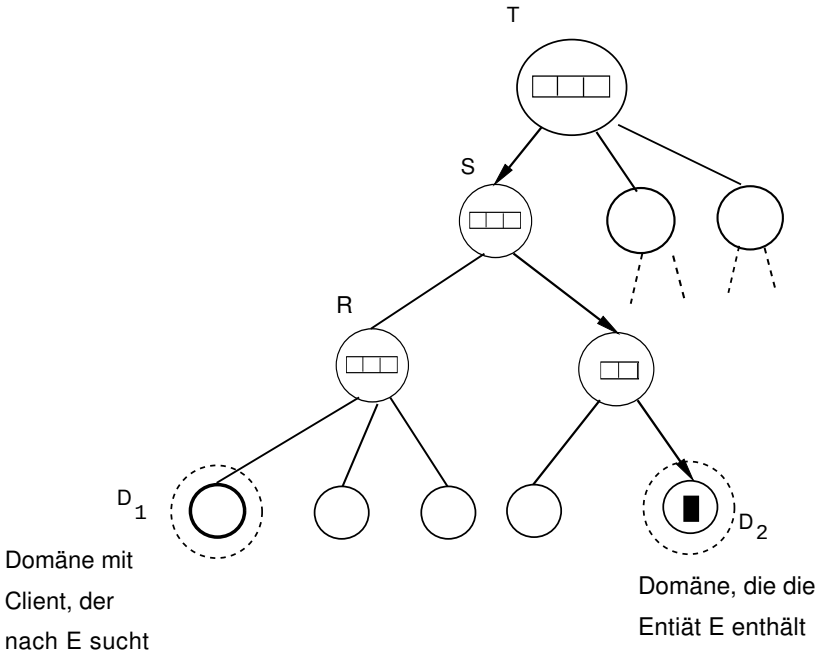


Abbildung 4.15: Beispielszenario für eine Suchanfrage

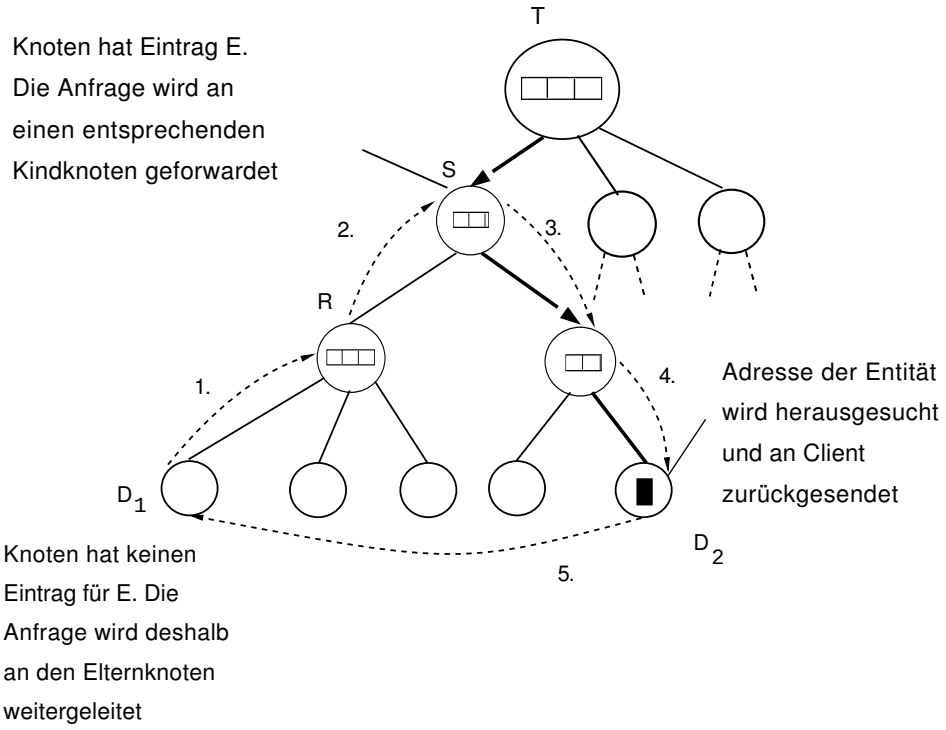


Abbildung 4.16: Schrittweise Suchanfrage

Der Client kontaktiert zunächst den Verzeichnisknoten seiner Domäne D_1. Sofern dort kein Eintrag für die Entität *E* enthalten ist, wird der Verzeichnisknoten der übergeordneten Domäne - hier *R* - kontaktiert usw. bis ein Verzeichnisknoten - hier *S* - mit Eintrag eines Location Records für *E* gefunden wurde.
Nun ist bekannt, dass *E* sich irgendwo in der Domäne *S* befindet, genauer in der Subdomäne, auf die der Zeiger des Location Records verweist.
Die Suchanfrage wird an den Verzeichnisknoten dieser Subdomäne weitergeleitet, der einen Zeiger auf die nächste Subdomäne enthält usw., bis in einer Blattdomäne die Adresse der Entität erhältlich ist. Diese Adresse wird dem Client zurückgegeben.
Vorteil: Dieses Suchen ist durch Lokalität geprägt, sofern die Entität lokal verfügbar ist. Nur im schlechtesten Falle muss über den Wurzelknoten gesucht werden.
Das Einfügen eines Replikats geschieht in ähnlicher Weise. Angenommen, in der Domäne D_1 wird ein Replikat eingefügt, so wird mittels der Schritte (1) und (2) in den Verzeichnissen der Domänen *R* und *S* eine Ergänzung der Einträge vorgenommen.
Und ein anderer Fall soll betrachtet werden: Falls die Entität *E* von der Domäne D_2 in eine andere Domäne migriert, so ist ein Update vorzunehmen. Dieses besteht aus zwei Teilen. Am Zielort der Entität ist eine Aktualisierung für das Einfügen eines Replikats vorzunehmen, und am ursprünglichen Ort sind die Zeiger auf die nicht mehr vorhandene Entität zu löschen. Wenn erst aktualisiert und dann gelöscht wird, kann die Kette von Löschungen so lange durchgeführt werden, bis in einem Verzeichnis noch ein Zeiger auf eine andere Subdomäne vorhanden ist.
Die Alternative, die man beim Einfügen eines Replikats oder der Aktualisierung der Ein-

träge eines migrierenden Objekts hat, besteht darin, ob die Installierung der Kette von Zeigern bottom-up oder top-down vorgenommen wird.
Für den top-down-Ansatz spricht, dass die Entität schneller lokalisierbar ist und folglich ggf. weniger Suchanfragen unbeantwortet verlorengehen. Allerdings würde eine eher wahrscheinliche Nichterreichbarkeit eines Elternknotens eher für ein bottom-up-mässiges Vorgehen sprechen.

Inwiefern ist ein Caching sinnvoll?
Sofern ein mobiles Objekt nur zwischen Knoten migriert, die alle zu einer gewissen Domäne *D* gehören, kann ein Zeiger auf den Verzeichnisknoten *dir(D)* gesetzt werden.
Diese Vorgehensweise wird auch **Pointer Caching** genannt.
Bezüglich der Skalierbarkeit dieses Ansatzes ergibt sich ggf. ein Engpass im Wurzelknoten. Einerseits sind dort Location Records aller Entitäten zu speichern, was im Millionenbereich schnell zu großen Datenmengen führen kann. Andererseits laufen jedoch alle nicht lokalen Anfragen über diesen Knoten, so dass viele Such- und Updateoperationen dort auszuführen sind.

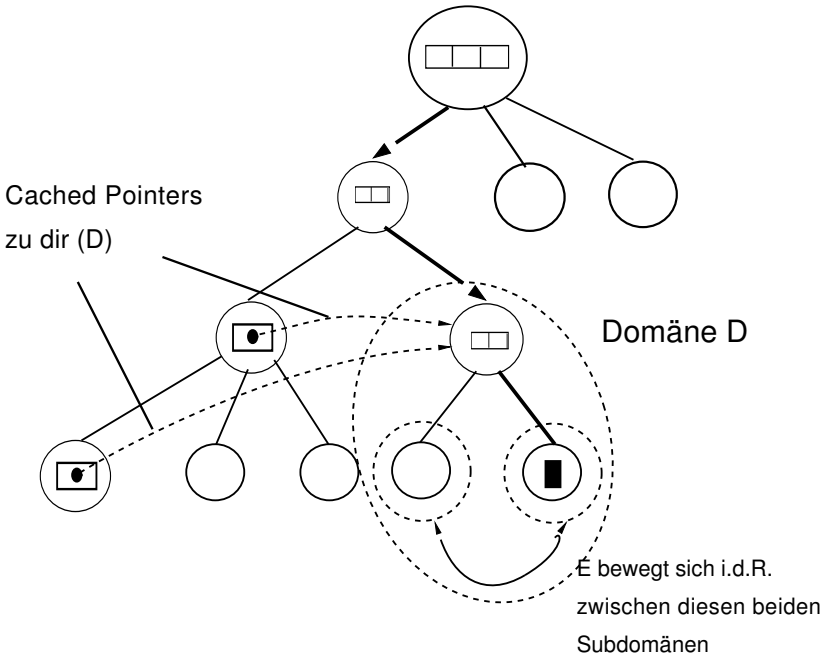


Abbildung 4.17: Pointer Caching

Ein weiteres Problem besteht im Löschen unreferenzierter Entitäten. Sobald auf eine Entität nicht mehr zugegriffen werden kann, sollte der Verweis auf sie gelöscht werden. Falls dies nicht explizit geschieht, muss eine Möglichkeit gefunden werden, solche Entitäten zu erkennen und automatisch zu löschen. Dieser Prozess wird auch als „Distributed Garbage Collection“ bezeichnet. Dies soll aber an dieser Stelle nicht weiter vertieft werden.
Die lokale Skalierbarkeit ist wie folgt: Bei Anrufverhalten erfolgt die Lokalisierung lokal.

Bei Migrationsverhalten ändert das mobile Objekt seinen Ort. Dies ist generell aufweniger als Home-Location, außer bei lokalen Bewegungen. Dort ist es etwa vergleichbar.

Fallbeispiel: GSM
Skalierbarkeit durch Hirarchien (räumlich: Zellen, Location Areas, Gesammtnetz)

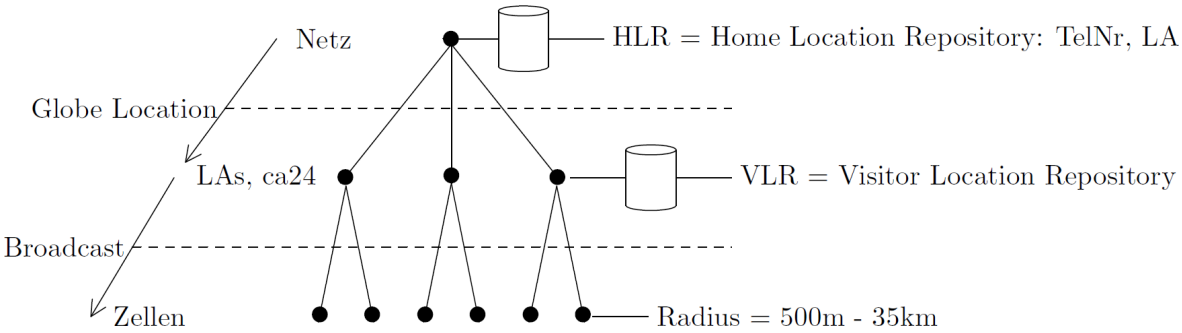


Abbildung 4.18: GSM

Vorgenensweise bei einem Anruf vom Festnetz zu Handy, wenn das Handy einen Eintrag im dazugehörigen VLR hat: HLR → VLR
Wenn nicht: HLR → VLR → Broadcast → ins VLR eintragen
Warum zwei verschiedene Lokalisierungsstrategien: Grundüberlegung: Wie verhaält sich das Anrufverhalten im Verhältnis zum Mobilitätsverhalten?

				Modell	Verhalten
1.	wie oft angerufen werden	<<	wie oft Änderung der Zelle	Rentner	Position update
2.	wie oft angerufen werden	<<	wie oft Änderung der Zelle	Geschäftsreisende	Paging

Abbildung 4.19: Beispiel für verschiedenen Endgeräte.

Frage: Paging (vgl. Broadcast in 4.18) oder Position Update(=PU) (Datenbankeintrag aktualisieren, vgl. HLR in 4.18).

4.6 Verteilte Speicherbereinigung

Naming- und Location Services stellen letztendlich einen Referenzdienst für Entitäten bereit. Verweist ein Dienst auf eine Entität, so ist ein Zugriff auf diese Entität möglich und dann auch deren Nutzung.

1. Was aber, wenn der Zugriff auf eine Entität nicht mehr möglich ist, z.B. weil diese nicht mehr existiert? Dann sollte auch die auf sie verweisende Referenz gelöscht werden.

2. Was ist, wenn eine Entität existiert, auf die keinerlei Referenz mehr besteht? Dann sollte die entsprechende Entität gelöscht werden, da sie nicht mehr auffindbar ist, z.B. weil

sie auch nicht mehr benötigt wird. Dieses Problem gewinnt insbesondere in Verteilten Systemen an Bedeutung.

Annahme: Im Folgenden gehen wir davon aus, dass der Zugriff auf ein Objekt nur dann möglich ist, wenn es eine Referenz auf dieses Objekt gibt.
Wir gehen ferner davon aus, dass in unserem Modell Referenzen auf Objekte durch gerichtete Kanten in einem Graphen dargestellt werden. Objekte, die einen systemweiten Referenzdienst bereitstellen, werden als Wurzelknoten dargestellt. Dies können auch mehrere Objekte sein, die zu einer sog. Wurzelmenge zusammengefasst werden. Auf diese Objekte sind keine Referenzen vorhanden. Alle anderen Objekte sollten durch Kanten oder durch Pfade von Kanten von diesen Objekten aus erreichbar sein.
Objekte, für die es keine direkten oder indirekten Referenzen von Objekten der Wurzelmenge gibt, sollten folglich gelöscht werden.

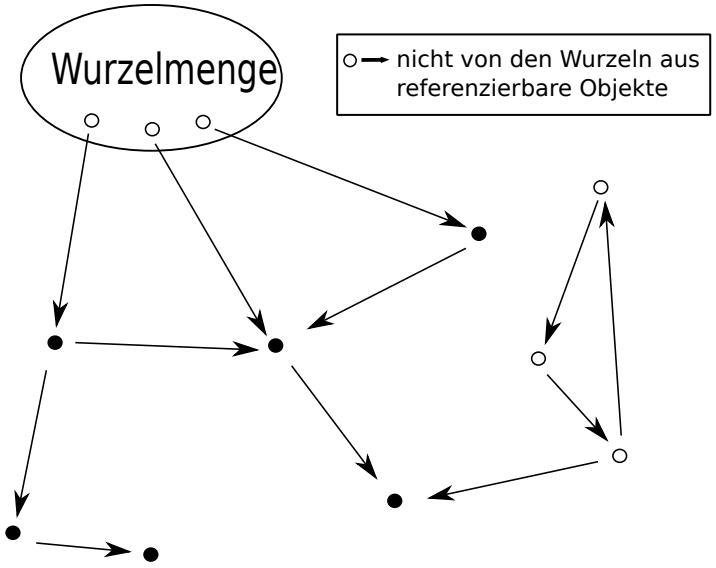


Abbildung 4.20: Wurzelmenge

Im Folgenden wollen wir Lösungsmöglichkeiten betrachten, wie Objekte erkannt werden können, auf die keine Referenz besteht, um sie nachfolgend zu löschen.

Referenzzähler Zunächst wollen wir uns das Prinzip der Referenzzähler für **Einprozessorsysteme** veranschaulichen.
Die Methode der Referenzzähler wird wie folgt implementiert: Immer wenn eine Referenz auf dieses Objekt erzeugt wird, wird der zugehörige Referenzzähler inkrementiert. Wird später eine Referenz wieder gelöscht, so wird der Referenzzähler dekrementiert. Hat der Zähler den Initialwert 0 wieder erreicht, so kann das Objekt gelöscht werden.
Die **Anwendung** dieses einfachen Verfahrens **auf Verteilte Systeme** führt zu Problemen, die im Wesentlichen durch die Unzuverlässigkeit der Kommunikation bedingt sind.
Das Problem, einen korrekten Referenzzähler bei unzuverlässiger Kommunikation zu verwalten, soll im Folgenden veranschaulicht werden.

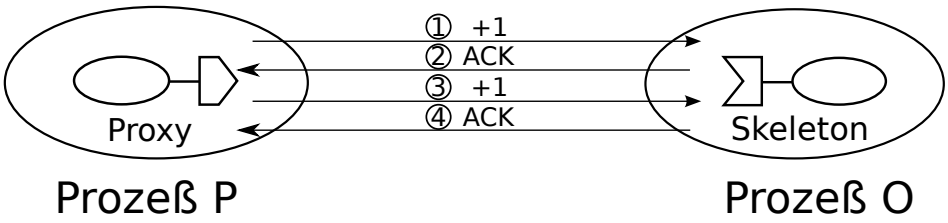


Abbildung 4.21: Doppelte ACK- Nachricht

Der Proxy von P dient der Erzeugung einer Referenz auf ein entferntes Objekt.
Da die ACK- Nachricht (Nummer 2) verloren geht, wird die Referenz von P auf O beim Referenzzähler von O doppelt gezählt und ist folglich nicht mehr korrekt.
Dieses Problem ist lösbar, z.B. indem Nachrichten als idempotent implementiert werden, so dass doppelte Verschickungen erkannt werden können.

Ein weiteres Problem tritt auf, wenn eine entfernte Referenz in einen anderen Prozess kopiert oder migriert wird.

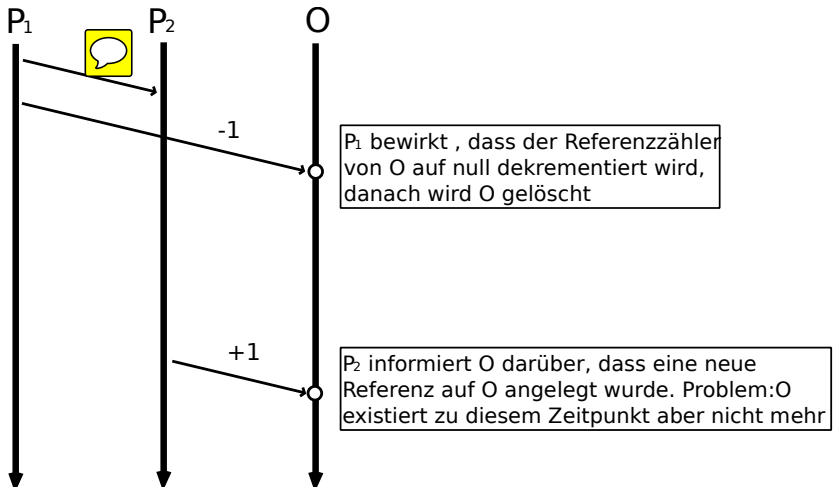


Abbildung 4.22: Synchronisationsproblem

Eine Lösung dieses Problems besteht darin, dass Skeleton von O darüber zu informieren, dass die Referenz auf P_2 migriert wird.

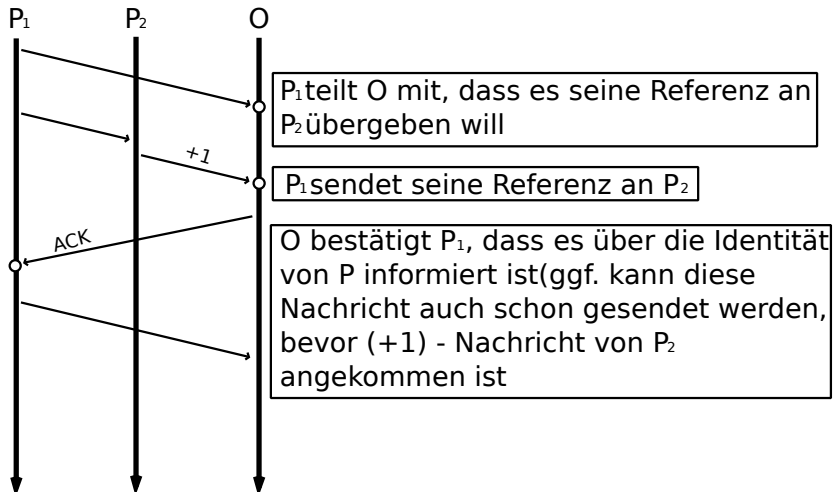


Abbildung 4.23: Synchronisationsproblem

Die Idee dieses Ansatzes besteht darin, dass P_1 das Skeleton von O nicht auffordern darf, seinen Referenzzähler zu dekrementieren, solange P_1 nicht sicher ist, dass O über die Migration der Referenz informiert ist.
Dieses Verfahren ist sicherer in der Vermeidung von Problemen, es führt jedoch zu zwei additiven Nachrichten, so dass die Netzlast steigt.
Eine Variation zur Migration ist die Replikation von Referenzzählern. Falls dieses häufiger vorkommt, so existiert eine Methode, die relativ wenig Netzlast verursacht. Zu diesem Zweck werden so genannte **gewichtete Referenzzähler** eingeführt.
Das Grundprinzip besteht darin, dass Replikate von Verweisen installiert werden können, ohne dass eine Kommunikation mit dem Objekt O erforderlich ist.
Zu diesem Zweck erhält jedes Objekt O eine Gesamtgewichtung, z.B. eine Zweierpotenz wie 128. Bei Erstellen einer kopierten Referenz wird die Hälfte der Gewichtung, die dem Original der Referenz zugeordnet ist, an die Kopie übergeben. Der Originalwert wird auf die andere Hälfte zurückgesetzt.
Beim Löschen einer Referenz wird eine Dekrementierungsnachricht an das Skeleton gesendet und der Wert von der Gesamtgewichtung subtrahiert.
Der Rest bleibt wie gehabt - ist die Gewichtung auf diese Zweierpotenz, z.B. 128, zurückgesetzt, kann das Objekt gelöscht werden.

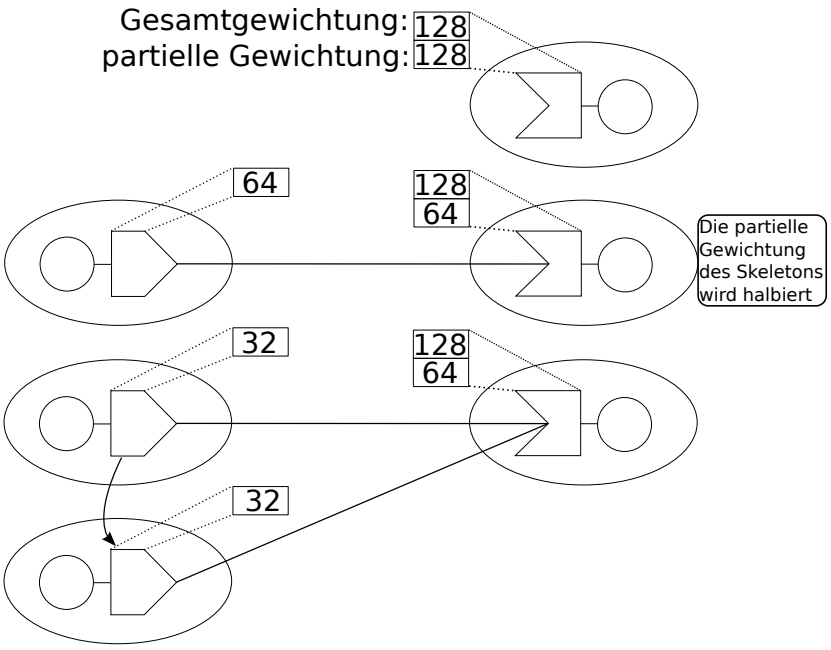


Abbildung 4.24: Gewichtete Referenzzähler

Problematisch ist lediglich, dass nur eine begrenzte Anzahl von Referenzen erzeugt werden kann, sobald die Referenzen nur den ganzzahligen Wertbereich annehmen können.
Eine Lösung dafür wären **indirekte Referenzen**, die bei einer Gewichtung von 1 einen Vorwärtszeiger erzeugen.
Nachteilig dabei ist jedoch, dass lange Ketten die Leistung wesentlich beeinträchtigen.
Schließlich gibt es die Möglichkeit, innerhalb eines Skeletons **Referenzlisten** zu importieren. Diese sind von Speicherplatz und Kommunikation her aufwendig und schlecht skalierbar, dafür jedoch eine idempotente Variante, um Verweise zu speichern.
Insbesondere können sie auch bei Prozessausfällen korrekt gehalten werden, indem das Skeleton des Objekts regelmäßig ping- Nachrichten sendet und damit überprüft, ob die Prozesse noch leben. Vom Prozess wird erwartet, dass er sofort auf diese Nachricht reagiert. Wird keine Nachricht empfangen - auch nach wiederholten ping- Versuch - so löscht das Skeleton den Prozess aus seiner Liste.
Alternativ kann eine Referenz auch für eine bestimmte Zeit als gültig angesehen werden und wird dann erneuert oder gelöscht (**Leasing**).
Referenzlisten werden in Java RMI basierend auf einer Methode von Birrell (1993) verwendet. Wenn ein Prozess P eine Referenz auf ein Objekt erzeugt, sendet er seine ID an das Skeleton des Objekts und dieses trägt P in seine Referenzliste ein. Es erfolgt eine Bestätigung. Dann erzeugt P einen Proxy für das Objekt in seinem Adreßraum.

Tracing-basierte Speicherbereinigung Auch bei Verwaltung von Referenzen mittels Zähler oder Listen kann es vorkommen, dass Einheiten sich gegenseitig referenzieren, jedoch nicht von der Wurzelmenge aus erreichbar sind.
Zu diesem Zweck überprüft man, welche Einheiten von der Wurzelmenge aus erreichbar sind und löscht alle anderen. Im Gegensatz zu den verteilten Referenzen weist diese Tracing-

basierte Speicherbereinigung jedoch Skalierbarkeitsprobleme auf.
Das Naive Tracing basiert auf sog. Mark&Sweep- Kollektoren, d.h. Kollektoren zum Markieren und Säubern. Dabei gibt es zwei Varianten:

- **2-Phasen-Ansatz:** In der Markierungsphase werden Verweisketten von der Wurzel aus ausgewertet. Jede Einheit, die erreicht werden kann, wird in einer Tabelle gespeichert. In der Säuberungsphase wird der Speicher ausgewertet, um zu erkennen, welche Einheiten nicht markiert sind. Diese werden darauf hin gelöscht.
- **Dreifache Markierung:** Anfänglich wird jede zu untersuchende Einheit weiß gefärbt. Eine Einheit wird grau gefärbt, wenn festgestellt wurde, dass sie erreichbar ist, jedoch noch Referenzen der Einheit untersucht werden müssen. Wurden alle ihre Verweise grau oder schwarz markiert oder hat die Einheit selbst keine Verweise, so wird sie schwarz markiert.

Nachteilig bei Mark&Sweep-Kollektoren ist, dass der Erreichbarkeitsgraph über die Phasen hinweg gleich bleiben muss.
Im verteilten Fall bedeutet dies, dass alle Prozesse zunächst synchronisiert und am Ende der Speicherbereinigung wieder frei gegeben werden müssen. Dies wird auch als Stop-the-World-Synchronisation bezeichnet und ist häufig nicht akzeptabel.

4.7 Session Initiation Protocol

Das Session Initiation Protocol (SIP) dient gemäß [8] dem Aufbau, der Veränderung oder dem Abbau von Verbindungen mit einem oder mehreren Partnern. Diese Verbindungen zwischen zwei oder mehreren Teilnehmern werden als Sessions bezeichnet. Bei der Spezifikation dieses Protokolls war es das Ziel, ein umfassendes Kommunikationssystem zu definieren, das alle nur möglichen traditionellen Systeme einschließt:

- das klassische Fernsprechnetz,
- die Mobilfunknetze,
- klassische Internetanwendungen wie E-Mail und WWW,
- Whiteboardanwendungen sowie den Abruf und die Steuerung von Audio- und Videodaten.

Insbesondere ist es das gewählte Protokoll für die Mobilkommunikation der 3. Generation. Zur Historie: Aus der Arbeitsgruppe Multipasty Multimedia Session Control (MMU-SIC) wurde 1999 das SIP zur Steuerung von Verbindungen definiert. Das SIP entstand aus den beiden Vorschlägen von Mark Handley und Henning Schulzrinne. Es ist ein textbasiertes Client/Server-Protokoll, das vom Ablauf her etwas dem http ähnelt.
Für das SIP sind folgende Komponenten von Bedeutung:

- SIP-Terminal: es besteht aus einem so genannten User-Agent-Client (UAC), der Methoden (Anfragen) an einen User-Agent-Server (UAS) richten möchte, der selbst wieder mit Responses darauf reagiert. D.h. der UAC initiiert eine Session und der UAS als Gegenstück zum UAC beantwortet die Anfragen des Clients. UAC und UAS werden dabei auch als SIP-Systeme bezeichnet.

– SIP-Proxy-Server: leiten Steuernachrichten im Netz weiter

– SIP-Redirect-Server zur Ermittlung der aktuellen Zieladresse das gewählten Nutzers

– Gateways für den Übergang zwischen Netzen

Das SIP ist ein Peer-to-Peer-Protokoll, mit dem einzelne Komponenten im einfachsten Fall direkt miteinander verbunden werden können, ohne auf eine zentrale Einheit zuzugreifen.
Die Instanzen eines Peers innerhalb einer Session werden UserAgents (UA) genannt. Diese UA's können in zwei Rollen auftreten:

– als User-Agent-Client (UAC) oder

– als User-Agent-Server (UAS).

Es gibt aber auch reine UAS, die keine Client-Rolle (als UAC) einnehmen können, d.h. die keine Verbindungen von sich aus initiieren.
Ein SIP-Proxy-Server handelt im Auftrag anderer Endgeräte und beinhaltet neben der Server- auch eine Client-Funktion. Er bearbeitet SIP-Nachrichten und fügt ihnen neue Header zu. Damit werden Antworten auf Anforderungen von einem Proxy auch wieder an den Proxy zurückgesendet. Proxy-Server routen SIP-Nachrichten durch das Netz.

Wir wollen nun verschiedene Basisabläufe betrachten.

1. **Direkte Verbindung**

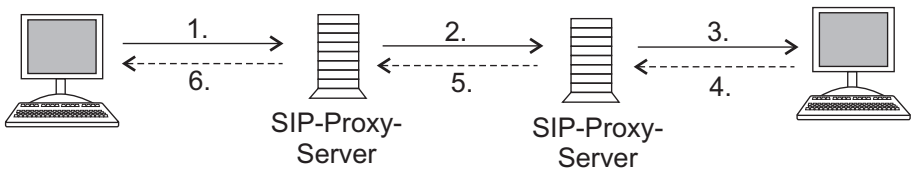


Abbildung 4.25: SIP-Ablauf direkte Verbindung

1. Es wird z.B. eine IP-Zieladresse von SIP-Client, d.h. vom Initiator der Verbindung zum zugehörigen SIP-Proxy-Server gesendet
2. Der Proxy kennt die Adresse und sendet die Anfrage zum Ziel-Proxy-Server
3. Der Ziel-Proxy-Server kennt die aktuelle IP-Adresse des angefragten Empfängers und sendet die Anfrage dort hin.
4. Der Nutzer nimmt die Verbindung an und sendet eine OK-Response-Message schrittweise bis zum Initiator zurück.

2. **Redirect-Server** Sofern der Nutzer Umleitungen eingestellt hat, kann der Redirect-Server diese handhaben.

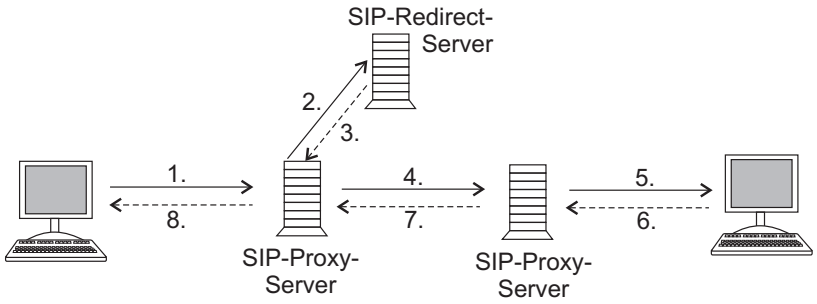


Abbildung 4.26: SIP-Ablauf mit Redirect-Server

2. Der SIP-Proxy-Server kennt die Zieladresse nicht und kontaktiert daher den SIP-Redirect-Server
3. Die ermittelte Zieladresse (Ziel des Proxys) wird zum anfragenden Proxy zurückgesendet.
Der restliche Ablauf ist mit der direkten Verbindung identisch.

3. **Location Server** Ggf. ist es nötig, in Abhängigkeit vieler Bedingungen mittels einer zentralen Datenbank die aktuell gültige Zieladresse zu ermitteln.

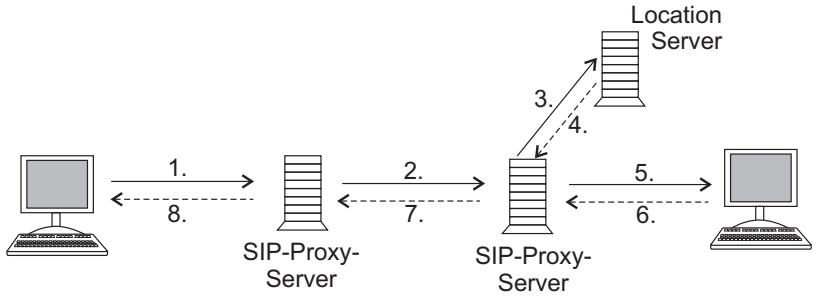


Abbildung 4.27: SIP-Ablauf mit Location-Server

3. Zur Ermittlung einer gültigen IP-Adresse wird ein lokaler Location-Server kontaktiert, z.B. innerhalb der Firma des adressierten Benutzers.
4. Das Ergebnis wird dem Zielproxy übermittelt
Der restliche Ablauf ist mit oben identisch.

4. **Gateways** Im einfachsten Fall kann der gewünschte Partner direkt durch die Angabe einer IP-Adresse erreicht werden. Wird aber z.B. ein Teilnehmer im Telefonnetz adressiert, liegt als Zieladresse nur eine einfache Telefonnummer vor.

   Für den Übergang vom Computer- ins Telefonnetz müssen sowohl die Nutzdaten (z.B. die Sprache) als auch die Signalisierungsinformationen konvertiert werden. Für diese Aufgabe stehen entsprechende Gateways im IP-Netz zur Verfügung.

   Die nächstliegende Frage ist jedoch, wie das richtige Gateway ausgewählt werden kann, d.h. welche Adresse dieses hat. Für dieses Problem wurde das Gateway Location Protocol (GLP) festgelegt.

Mittels dieses Protokolls kann statt der direkten Adressierung der Location-Server kontaktiert werden, der dann das richtige Gateway für die Anfrage ermittelt und die Anfrage dorthin sendet.

Damit ergibt sich folgendes Grundprinzip:

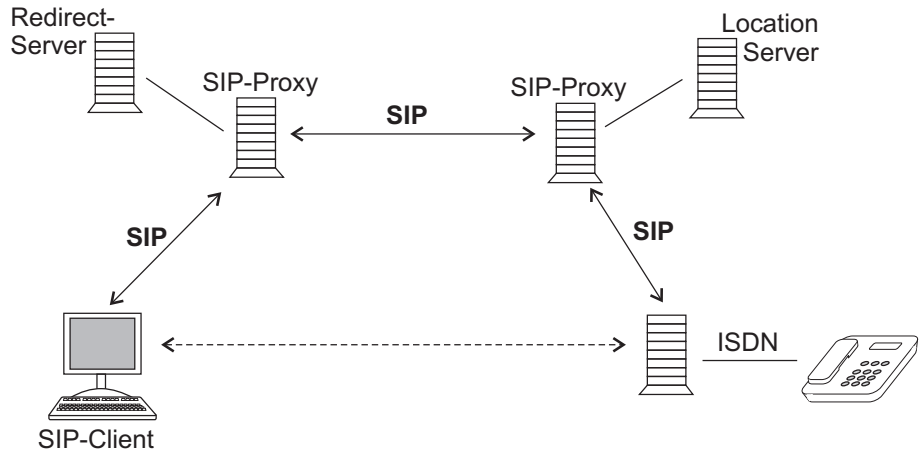


Abbildung 4.28: SIP-Ablauf mit Gateway

Eine Anfrage (Request) mit Antwort (Response) wird als Transaction bezeichnet.

Für die Adressierung eines Endgeräts können beispielsweise sowohl E-Mail-Adressen als auch Telefonnummern mit Zusatzangabe von z.B. einer IP-Adresse oder einem Host angegeben werden, z.B. +49-711-82141454@128.203.65.111 Durch ein Telephone Number Mapping kann auch eine klassische Telefonnummer auf einen DNS-Eintrag abgebildet werden.

Der Transport des SIP erfolgt bevorzugt über UDP, kann aber auch über TCP erfolgen. Dafür ist ein fester Port zugeordnet, der Port 5060. Für UMTS werden damit UMTS-Endgeräte zu Internet-Endgeräten.

SIP unterstützt verschiedene Arten von Mobilität:

- die Terminalmobilität (1.), auch als Sitzungsmobilität bezeichnet, d.h. das übertragen einer laufenden Session netzübergreifend von einem Endgerät auf ein anderes Endgerät,
- Persönliche Mobilität (2.), d.h. der Nutzer ist in verschiedenen Netzen und an verschiedenen Terminals unter ein und derselben Adresse erreichbar und
- Service Mobilität (3.), d.h. die Bereitstellung ganz persönlicher Dienste unabhängig von Netz und Terminal

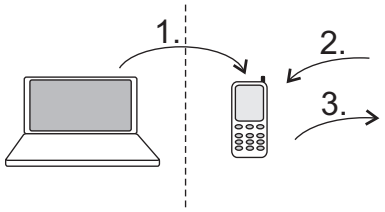


Abbildung 4.29: Mobilitätsformen des SIP

5

Dienste und Dienstvermittlung

►

Inhaltsangabe

Bislang haben wir verschiedene Mechanismen zur Realisierung von Verteilten Systemen betrachtet. Nach dieser „inneren Sichtweise“ soll nun zu einer „Anwendungssicht“ übergegangen werden. Die Realisierung einer Anwendung soll dabei zunächst mit dem Erbringen eines Dienstes gleichgesetzt werden. Dieser Dienst ist Ausgangspunkt der folgenden Betrachtungen.

5.1 Beschreibung von Diensten

Der Begriff des *Dienstes* wird innerhalb der Datenkommunikation sehr häufig benutzt. Seine Verwendung ist jedoch nicht einheitlich, sondern von dem verwendeten Kontext abhängig.
Von der CCITT wird der Begriff *Dienst* in verschiedenen Bedeutungen verwendet. Recommendation E.800/M.60 definiert einen *Dienst* als „eine Menge von Funktionen, die dem Nutzer einer Organisation angeboten werden“, Recommendation Z.100 verknüpft die Bedeutungen von Dienst und Prozess. Darin heißt es: „Ein Dienst ist eine alternative Möglichkeit, einen Prozess zu spezifizieren. Jeder Dienst kann einen anderen Teil des Verhaltens eines Prozesses definieren“.
CCITT-Dienste werden allgemein „in den CCITT-Recommendations definiert, um sie an die Nutzer von Systemen zu vermarkten“. Betrachtet man speziell die Recommendation X.200, so wird der (N)-Dienst einer (N)-Schicht als „eine Leistung der (N)-Schicht und der darunterliegenden Schicht“ definiert, „und an der Schnittstelle zwischen (N)- und (N+1)-Schicht den (N+1)-Instanzen angeboten“.
Dieser Uneinheitlichkeit von Dienstdefinitionen begegnet das ODP-Referenzmodell damit, dass der Dienst an sich gar nicht definiert wird, sondern die Verknüpfung zwischen Dienst und Objekt betrachtet wird. Auf diesen Zusammenhang soll im Folgenden eingegangen werden.

Der Diensttyp
Diensttypen bestehen aus mindestens einem **Rechnerschnittstellentyp** und beliebig vielen **Diensteigenschaftstypen.**

Beispiel:

```
Type COMPILER
Input: C, C++, PASCAL, FORTRAN, MODULA3, ...
Output: 68040_Code, Power_PC_Code, Alpha_Code, ...
Location: String
Identifier: String
Cost_per_1000_lines: Real
```

`Input, Output, Location, Identifier` und `Cost_per_1000_lines` sind hierbei Typen der Diensteigenschaften.

Der Dienst
Ein Dienst ist definiert als Funktionalität, die ein Objekt an einer Schnittstelle (eines Rechners) zur Verfügung gestellt wird.
Der Dienst im objektorientierten Sinne ist die Instanziierung eines Diensttyps, dem ein so genannter Rechnerschnittstellentyp zugeordnet ist (abstraktes Konzept).

Dienste ein- und desselben Diensttyps besitzen die gleiche Funktionalität und haben das gleiche Rechnerverhalten. Sie können jedoch in einigen rechnerunabhängigen und verhaltensunabhängigen Aspekten voneinander abweichen. Diese zusätzlichen Aspekte werden **Diensteigenschaften** genannt. Diensteigenschaften werden durch (Name, Wert)-Paare charakterisiert.

Dienstangebote Wird ein Dienst angeboten, so spricht man von einem Dienstangebot. Es beschreibt einen Dienst, der anderen Objekten an einer Schnittstelle bereitgestellt wird. Ein

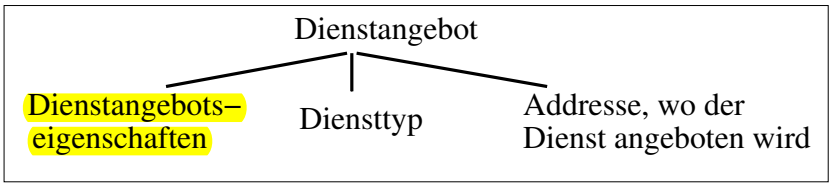


Abbildung 5.1: Darstellung von Dienstangeboten

Dienstangebot besteht aus dem Diensttyp, der Schnittstellenbezeichnung und einer Menge von Dienstangebotseigenschaften, d.h. Werten von Diensteigenschaftstypen, die den Dienst charakterisieren. Dieses Dienstangebot erbt alle Merkmale und Eigenschaften vom Diensttyp und Dienst und es muss zusätzlich durch Angabe einer Adresse spezifiziert werden, wo der Dienst in Anspruch genommen werden kann. Dienstangebote werden vermittelt.

Beispiel:

```
Type COMPILER
Input: C++
Output: 68040_Code
Location: ‘‘Rechnerraum‘‘
Identifier: ‘‘Computer_0815‘‘
Cost_per_1000_lines: 0.12
```

Dienstangebote können auf verschiedene Arten dargestellt werden. Diese sind in Abbildung 5.1 dargestellt.

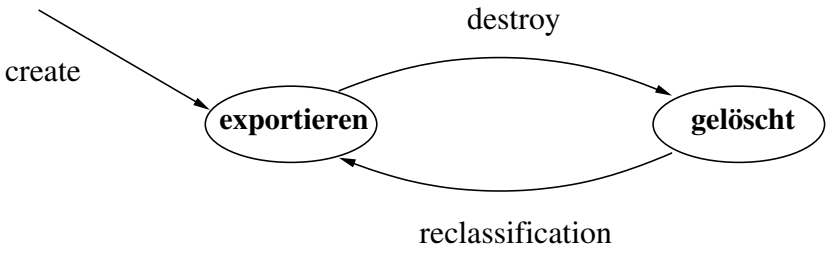


Abbildung 5.2: Zustände und Zustandsübergänge eines Dienstangebotes

Zusammenhang zwischen Dienst, Diensttyp und Dienstangebot Zusammenhang zwischen Dienst, Diensttyp und Dientsangebot ist in der Abbildung 5.3 veranschaulicht. Die Abbildung 5.4 zeigt ein Beispiel für den Diensttyp COMPILER.

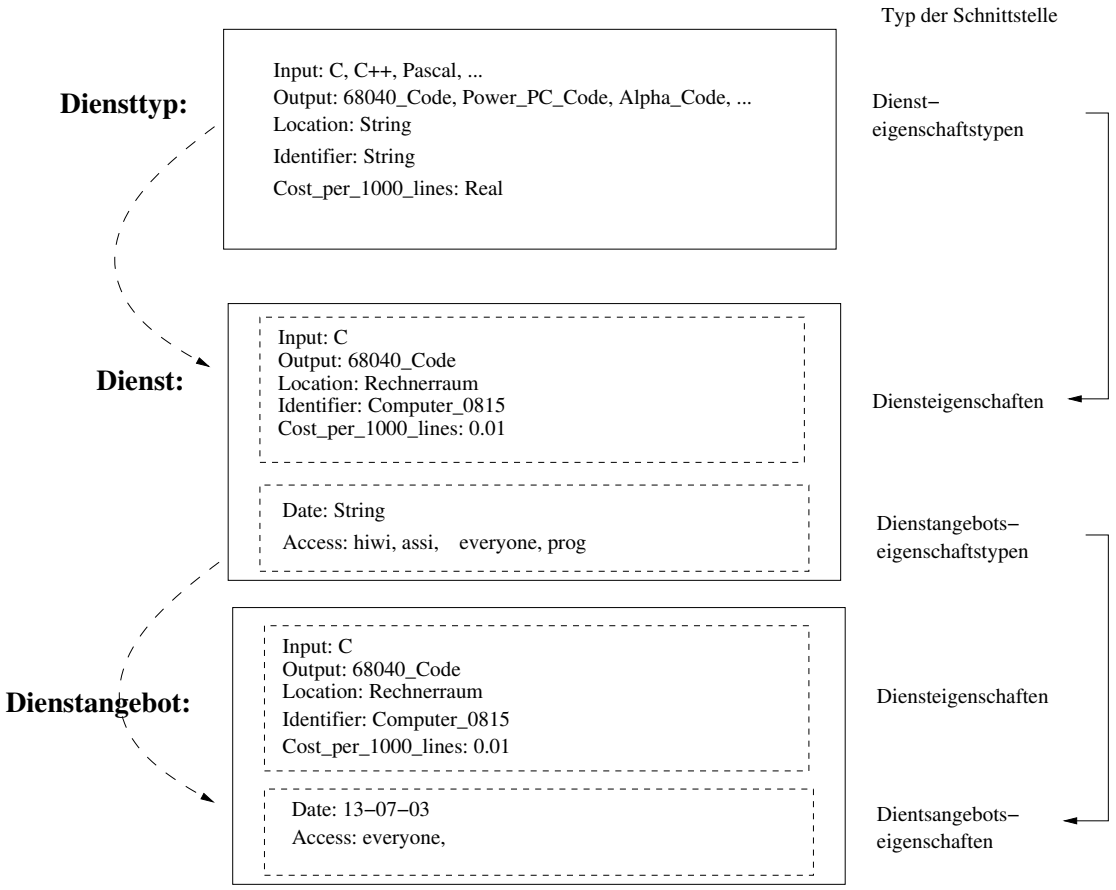


Abbildung 5.3: Zusammenhang zwischen Dienst, Diensttyp und Dienstangebot

5.2 Dienstvermittlung

Idee hinter der Dienstvermittlung Entspricht eines der Angebote dem Dienstgesuch? Wenn ja: Matching(übereinstummung). D.h. es wird eine syntaktische Auswerung durchgeführt, ob zwei Dienstbeschreibungen (eine angebotene und eine gesuchte) übeinstimmen.
⇒Generell: Komponenternweise wird geschaut, ob Strings identisch sind bzw. Werte Elemente einer bestimmten Menge sind.
Probleme:

– Soll ein einzelner Dienst in Schritt 3 zurückgegeben werden oder eine Menge von Diensten?
Je genauer die Spezifikation der Anfrage erfolgt, desto weniger Dienstangebote werden ermittelt.

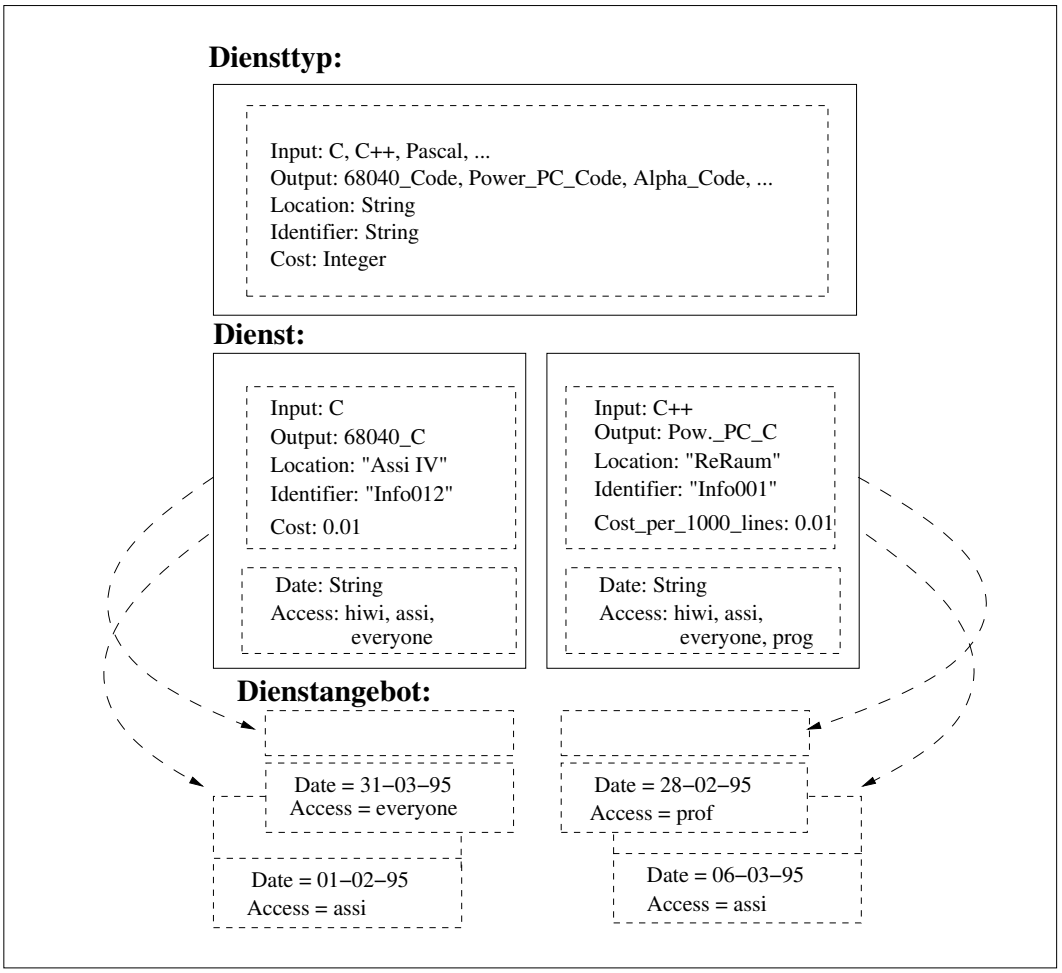


Abbildung 5.4: Dienstangebote für den Diensttyp COMPILER

→Werden Superlativfunktionen(Min, Max, Last, First, ...) spezifizieret, so kann die Menge von Angeboten auf einen Wert reduziert werden
⇒Daher unterscheidet man 2 Klassen von Anfragen:

- Search: gibt Menge aller in Frage kommenden Dienstangebote zurück
- Select: wählt Ein Dienstangebot aus.

– Kann ich alle Diensteigenschaften nummerich durch Mengenbeziehungen auswerten?
Oft lassen sich nicht genau die Dienste finden, die man sucht (beste Qualität zum niedrigsten Preis, technische Parameter). Dann habt man aber Interesse daran, die Dienste auszuwählen, die der Suche am nächsten kommen. Prinzip: Diensttyp muss übereinstimmen. Diensteigenschaften werden bzgl. ihrer Abweichung bewertet und der Dienst bzw. das Dinenstagebot mit der kleinsten Abweichung ausgewählt.
Ganz einfaches, aber verbreitetes Prinzip:

Gegeben: $\begin{pmatrix} \text{Typ1} \\ \text{E10} \\ \text{E20} \\ \vdots \end{pmatrix} \begin{pmatrix} \text{Typ2} \\ \text{E11} \\ \text{E21} \\ \vdots \end{pmatrix} \ldots$ Gesucht: $\begin{pmatrix} \text{Typ} \\ E_{ges1} \\ E_{ges2} \\ \vdots \end{pmatrix}$

Lösung: Minimum über die euklidische Metrik zwischen je einem gegebenem Dienstangebot und dem gesuchtem Dienst: $\min \sqrt{\sum_{i=1}^{n} (E_{Geg_i} - E_{Ges})^2}$.

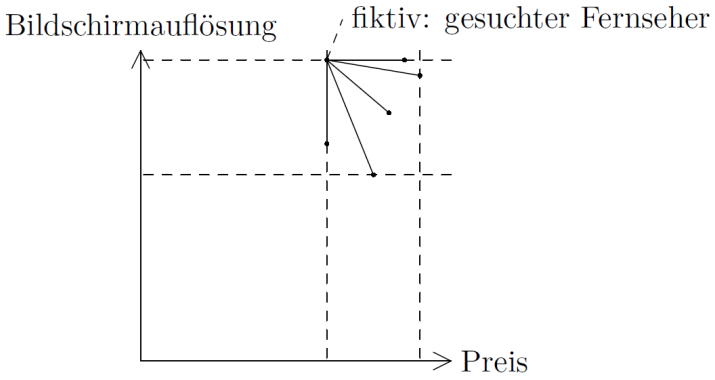


Abbildung 5.5: Beispiel der euklidischen Norm bei der Suche nach einem Fernseher

– Sind alle Diensteigenschaften gleich zu behandeln? Nein. Siehe Unten.

Das Prinzip eines ODP-Traders

Erweitert man ein als Client/Server-Architektur modelliertes Verteiltes System um eine dritte Instanz, welche die Vermittlung von Dienstangeboten übernimmt, so spricht man von einem Importer/Exporter/Trader-System. Ein Trader bietet Dienste in einer verteilten Umgebung an. Man sagt auch, dass diese Dienste exportiert werden. Bei Bedarf können Objekte diese Dienstangebote beim Trader einholen. In diesem Fall sagt man, dass Dienstangebote importiert werden. Das anbietende Objekt teilt dem Trader mit, dass es existiert und liefert Informationen über sich selbst sowie seine Dienstangebote. Dieser Vorgang wird als **Export** bezeichnet, vgl. (1) in Abbildung 5.6; das den Dienst anbietende Objekt wird Exporter genannt. Ein anderes Objekt hat nun die Möglichkeit, mit dem Trader in Kontakt zu treten (2)

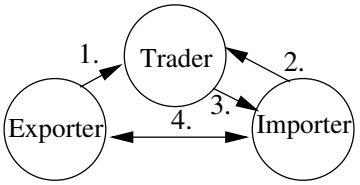


Abbildung 5.6: Wechselwirkungen zwischen Trader, Exporter und Importer

und nach einer Schnittstelle für einen Dienst mit ausgewählten Charakteristiken zu fragen.

Kann diese Anfrage positiv beantwortet werden (3), so spricht man von einem **Import**; das dienstanfragende Objekt wird Importer genannt. Nach Erhalt der Schnittstellenreferenz hat der Importer die Möglichkeit, direkt mit dem Exporter in Kontakt zu treten und den angebotenen Dienst zu nutzen (4). Grundsätzlich kann ein Exporter zu jeder Zeit von seinem Angebot zurücktreten. Dieser Prozess wird als **Löschen** (bzw. Zurückziehen, Withdraw) des Exports bezeichnet.
Der Trader kann zwei Operationen zum Zwecke der eigentlichen Dienstvermittlung ausführen, und zwar das **Suchen (Search)** von geeigneten Diensten und die **Auswahl (Selection)** eines 'optimalen' Dienstes aus der Menge der verfügbaren Dienstangebote. Der ausgewählte Dienst muss dabei den Anforderungen des Dienstnutzers entsprechen, d.h. die Auswahl muss auf der Grundlage der geforderten Charakteristiken erfolgen. Eine erfolgreiche Dienstvermittlung ist nur möglich, wenn die Dienstspezifikation des Importers eine Teilmenge der Dienstspezifikation des Exporters ist, also einen vorhandenen oder weniger komfortablen Dienst im Verhältnis zu den verfügbaren Angeboten anfordert.

Hier sollen nur zwei Beispiele betrachtet werden:

Suche:

```
SEARCH Compiler WITH
Input = C++ AND Output = Power_PC_Code
AND Location = 'Informatik'
ENDSEARCH
```

Diese Suche gibt alle Dienstangebote vom Typ Compiler an, welche die entsprechenden Eigenschaften (`Input = C++ AND Output = Power_PC_Code AND Location = ''Informatik''`) erfüllen.

Auswahl:

```
SELECT Compiler WITH
IF Input = C++ AND Output = Alpha_Code
AND Location ='Informatik'
AND MINIMUM (Cost_per_1001_lines)
THEN TAKE THAT
ELSE Input = C++ AND Output = Alpha_Code
AND FIRST (Cost_per_1000_lines < 0.2)
ENDSELECT
```

Durch die Superlativfunktionen (z.B. `MINIMUM, MAXIMUM, FIRST, LAST, RANDOM`) wird eine Auswahl auf genau ein Dienstangebot vorgenommen.
Diensteigenschaften werden unterteilt in statische Diensteigenschaften und dynamische Diensteigenschaften, die in der Tabelle 5.1 näher erläutert sind.

Statische Diensteigenschaften	**Dynamische Diensteigenschaften**
- beschreiben die Fähigkeit eines Dienstes	- beschreiben die Verfügbarkeit eines Dienstes
- werden innerhalb des Traders gespeichert, können aber jederzeit durch den Exporter modifiziert werden	- sollten beim Exporter verfügbar sein und bei Bedarf abgefragt werden können
- haben selten Wertänderungen	- werden nicht im Trader gespeichert, da häufig Wertänderungen auftreten
- z.B. Identifier Strings	- z.B. Queue Lengths (Integer)

Tabelle 5.1: Charakteristika von statischen und dynamischen Diensteigenschaften

Bei der Beschreibung eines Dienstverhaltens ist die Dienstqualität (Quality of Service) ebenfalls von Bedeutung. Dienstqualitätsmerkmale können in sechs grundlegende Klassen eingeteilt werden.
Nach ihrer Bedeutung sortiert gibt es folgende Merkmale:

- zeitbezogene Charakteristiken (absolute Zeit, Verzögerungen, Schwankungen),
- kapazitätsbezogene Charakteristiken (Kapazität, Durchsatz, Menge pro Zeit),
- vollständigkeitsbezogene Charakteristiken (Genauigkeit, Vollständigkeit, die garantiert werden kann, Adressierungs- und Zustellungsfehler),
- kostenbezogene Charakteristiken,
- sicherheitsbezogene Charakteristiken (Autorisierung, Authentifizierung),
- zuverlässigkeitsbezogene Charakteristiken und ihre Ableitungen (Jitter, Fehlertoleranz, ...).

Die letzten 4 Eigenschaften werden in der Regel nicht im Servicedirectory abgespeichert, sondern ergeben sich (sekundär) aus der Dienstnutzung.
Bei der Dienstanfrage treten auch Probleme auf. Das erste Problem ist die Berücksichtigung von Aktualität. Insbesondere wenn dynamische Dienstattribute Verwendung finden, müssen Aktualitätsaspekte berücksichtigt werden, d.h. dynamische Attribute müssen so spät wie möglich ausgewertet werden.
Zunächst wird der Suchbereich eingeschränkt, indem der Tradingkontext ausgewertet wird. Anschließend werden der Diensttyp und die statischen Diensteigenschaften ohne Superlativfunktionen (nur SEARCH-Ausführung) ausgewertet. Dann werden die dynamischen Diensteigenschaften der in Frage kommenden Angebote betrachtet, und zum Schluss werden die dynamischen Diensteigenschaften ausgewertet bzw. der optimale Dienst ausgewählt.
Als weiteres Problem stellt sich der Zugriff auf dynamische Diensteigenschaften dar. Es sollen zwei verschiedene Verfeinerungen, wie auf dynamische Diensteigenschaften zugegriffen werden kann, betrachtet werden.
Das Grundprinzip des Cachings, siehe Abbildung 5.7, besteht aus einer Zwischenspeicherung von Werten dynamischer Diensteigenschaften im Service Directory des Traders. Treten Wertänderungen bei dynamischen Eigenschaften auf, so wird durch den entsprechenden Exporter über den Trader eine Aktualisierung der Werte vorgenommen. Die Netzlast hängt von

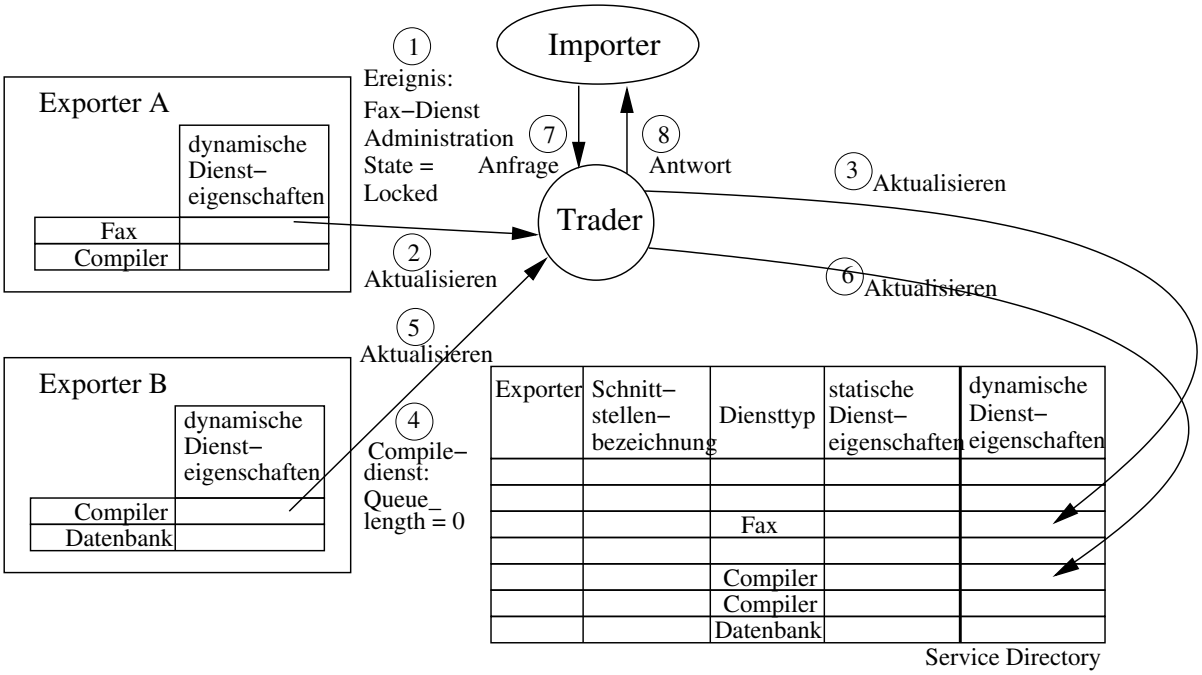


Abbildung 5.7: Prinzip des Cachings bei einer Dienstanfrage

der Anzahl der Aktualisierungen dynamischer Diensteigenschaften seitens der Exporter und der Anzahl der Dienstanfragen seitens der Importer ab. Konkret heißt das: Treten sehr viel häufiger Änderungen in den Werten der dynamischen Diensteigenschaften auf, als diese abgefragt werden, so entsteht eine unnötig hohe Netzlast. Der Grund dafür ist, dass die Werte der dynamischen Diensteigenschaften im Trader aktuell gehalten werden, obwohl sie gar nicht benötigt werden.
Daraus folgt, dass Caching zwar den vom Trading vorgeschlagenen Konzept entspricht, bei geringem Dienstanfrageaufkommen jedoch nicht optimal ist.
Als zweite Verfeinerung soll hier das Polling beschrieben werden (vgl. Abbildung 5.8). Dynamische Diensteigenschaften werden beim Polling nicht mehr im Trader gespeichert, sondern am Exporter abgerufen.

Der Trader übernimmt anhand des Service Directories eine Vorauswahl hinsichtlich des Diensttyps (und ggf. statischer Diensteigenschaften). Die verbleibende Menge der Dienste wird bei den Exporten auf die dynamischen Eigenschaften hin untersucht.

Fazit:
Werden sehr oft dynamische Diensteigenschaften geändert und selten abgefragt, so reduziert sich die Netzlast (im Verhältins zum Caching) erheblich. Erfolgen häufiger Dienstanfragen, so ist jedoch das Caching günstiger.
Das bedeutet, dass man von Fall zu Fall entscheiden sollte, welche Strategie besser geeignet ist. Auch ein dynamisches Entscheiden ist möglich, wobei in Abhängigkeit der aktuellen Netzlast zwischen Polling und Caching hin- und hergeschaltet werden kann.

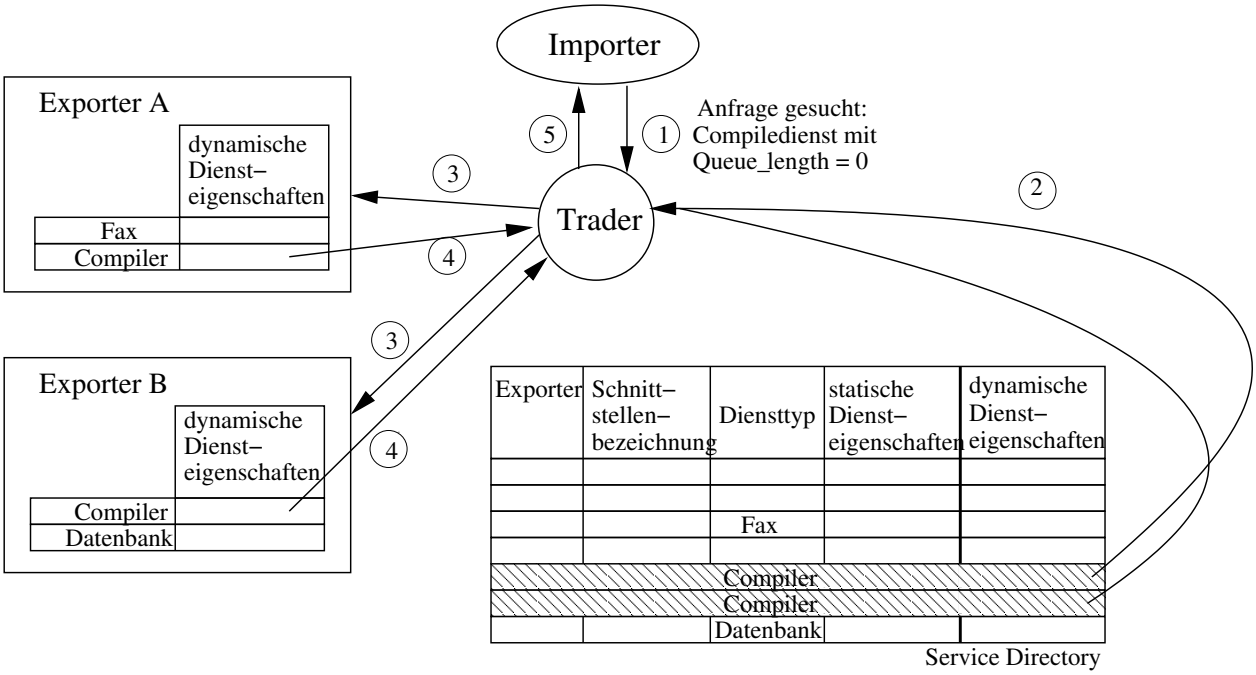


Abbildung 5.8: Prinzip des Pollings bei einer Dienstanfrage

5.3 Praktische Systeme

Im Folgenden sollen einige praktisch relevante Dienstvermittlungssysteme exemplarisch betrachtet werden.

5.3.1 Jini

Nach [9] beschreiben die folgenden Abschnitte die Dienstvermittlung innerhalb von Jini. Jini ist eine Entwicklung der Firma Sun Microsystems und basiert auf Java. Da das ganze System in Java implementiert wurde, ist es erforderlich, dass alle Geräte die Jini einsetzen wollen über eine Java Virtual Machine verfügen. Neben der eigentlichen Dienstvermittlung, bietet Jini noch eine Menge weiterer Funktionalitäten an. Dazu gehören beispielsweise Transaktions- und Persistenzmanagement in Verteilten Systemen. Diese Punkte sollen im Folgenden nicht beachtet werden.

Basiskomponenten

Ein Jini System besteht aus drei Komponententypen: Clients, Dienstinstanzen, und einem Lookup Service (LUS). Der LUS implementiert die Dienstvermittlung auf Anfrage eines Clients, und verwaltet das aktuelle Dienstangebot. Die Jini Spezifikation macht keine expliziten Einschränkungen bzgl. des eingesetzten Netzwerkprotokolls, bisher existieren allerdings nur Implementierungen für das TCP/IP Protokoll. Eine Voraussetzung für den Einsatz von Jini ist ein bereits vollständig konfiguriertes Netzwerk. Damit unterscheidet Jini sich z.B. von Ansätzen wie UPnP (vgl. Kapitel 5.3.2), die auch Aufgaben der Netzwerkkonfiguration, wie etwa

die dynamische Zuordnung von Netzwerkadressen, realisieren.
Die Plattformunabhängigkeit dieser Architektur erlaubt den Einsatz von mobilem Code zwischen den Basiskomponenten eines Jini Systems. Die Kommunikation zwischen verteilten Komponenten in Jini erfolgt durch RMI (vgl. Kapitel 3.6).

Dienstregistrierung

Die Dienstregistrierung erfordert zunächst die Lokalisierung eines zuständigen Lookup Services. Dies erfolgt normalerweise mittels Multicast. Der Client sendet solange Discovery-Nachrichten an eine definierte Multicast-Adresse bis sich ein geeigneter Lookup Service meldet. Ist die Adresse des Lookup Service bekannt, kann die Kommunikation unmittelbar durch einen TCP Unicast erfolgen.
Nach der erfolgreichen Lokalisierung des Lookup Service, sendet dieser eine Objektreferenz an den Client zurück. Dieses Objekt *(ServiceRegistrar)* implementiert eine Managementschnittstelle, welche die Registrierung von Diensten ermöglicht. Der bisher beschriebene Vorgang wird im Rahmen von Jini als Discovery bezeichnet.
Nach erfolgtem Discovery-Prozess wird der Dienst registriert. Dazu erzeugt die Dienstinstanz eine Datenstruktur aus einem Dienst-Proxy-Objekt und einer assoziierten Dienstbeschreibung bestehend aus einer Menge von Dienstattributen, so genannten *Entries*. Der Prozess der Dienstregistrierung wird als *Join* bezeichnet.

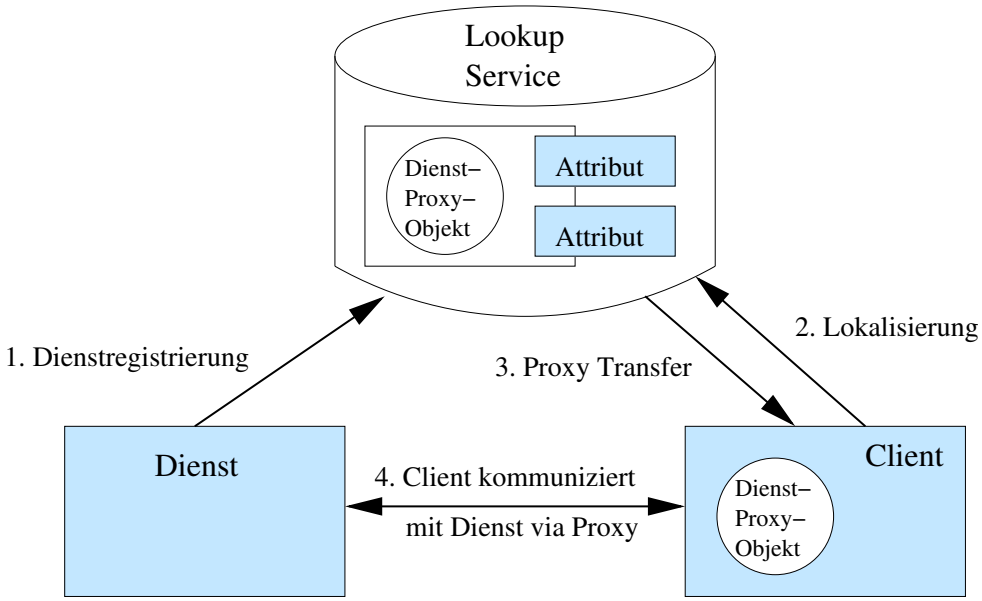


Abbildung 5.9: Dienstvermittlung in Jini

Lokalisierung eines Dienstes

Die Lokalisierung eines Dienstes erfordert - wie bei der Dienstregistrierung - zunächst das Auffinden eines Lookup Services. Dazu wird auch der oben beschriebene Discovery Mechanismus angewandt. Der Client konkretisiert seine Anforderungen an den gesuchten Dienst

mittels eines so genannten *Service Templates*. Ein Service Template spezifiziert den Typ eines Dienstes und eine Menge von Attributen. Wird im Lookup Service ein Diensteintrag gefunden, welcher dem eingehenden Service Template genügt, so wird das korrespondierende Dienst-Proxy-Objekt zum Client gesendet. Mittels des Proxies kann der Client mit dem Dienst kommunizieren (vgl. Abbildung 5.9). Der Proxy implementiert eine lokale Dienstschnittstelle für den Client. Das Kommunikationsprotokoll zwischen Proxy und Dienst kann frei definiert werden.

Leasing

Um die Aktualität des vom Lookup Service verwalteten Dienstangebotes zu gewährleisten, wird das Konzept des Leasings angewandt. Leasing basiert auf der Idee, dass Referenzen auf Objekte nur eine vordefinierte Zeitspanne gültig und dann nicht mehr anwendbar sind. Diese Art von Objektreferenzen wird in der Konzeption von Jini als *Leases* bezeichnet. Die Zeitspanne eines Leases kann je nach Bedarf neu verlängert werden. Wird die assoziierte Zeitspanne nicht periodisch erneuert, ist die Referenz auf das Objekt ungültig. Die Referenz kann auch vor Ablauf der definierten Zeitspanne als ungültig markiert werden. Im Allgemeinen realisieren Leases eine Methode zur Verwaltung von verteilten Ressourcen. Im Falle des Lookup-Service kann eine konsistente Datenbasis beibehalten werden, auch dann wenn die zukünftige Nichtverfügbarkeit eines Dienstes nicht angegeben wird, d.h. dass der zugehörige Lease nicht explizit gelöscht wird. In diesem Fall läuft die Gültigkeit des Leases ab, da keine Erneuerung erfolgt. Entsprechend wird der Dienst aus der Datenbasis des Lookup Services entfernt.

5.3.2 Universal Plug and Play (UPnP)

Bei Universal Plug and Play bieten UPnP-Devices wie etwa Drucker oder Kameras jeweils einen oder mehrere Dienste an, z.B. kann ein Drucker einen Druck- oder auch einen Faxdienst anbieten.
Ein Service Announcement besteht im Einsatz eines Discovery Services, um das Dienstangebot im Netzwerk bekannt zu machen. Ein Description Document beschreibt dabei die Funktionalität eines Dienstes.
Zur Lokalisierung eines Dienstes initiiert der Client eine Anfrage mittels des Simple Service Discovery Protocols (SSDP). Dies ist ein HTTP-basiertes Dienstvermittlungsprotokoll, welches das Auffinden von Diensten erlaubt. Geräte, die der Anfrage entsprechen, senden ihre URL zurück.
In Analogie zu Jini unterstützt SSDP ein Leasing-Konzept, das die Verwendbarkeit von Diensten zeitlich beschränkt. Im Gegensatz zu Jini unterstützt UPnP dabei die spontane Vernetzung von Geräten in einem Ad-Hoc-Netzwerk.

5.3.3 Service Location Protocol (SLP)

Das Service Location Protocol (SLP) ist eine Entwicklung der Internet Engineering Task Force (IETF). SLP definiert sowohl die Spezifikation zur Beschreibung von Diensten als auch Protokolle zwischen Systemkomponenten. In Analogie zu UPnP und im Gegensatz zu Jini ist SLP nicht an einen bestimmten Sprachstandard wie z.B. Java gebunden.

SLP basiert auf unterschiedlichen Agenten:

– **User Agents**
  realisieren die Dienstsuche im Auftrag eines Clients über einen auf Multicast basierenden Discovery-Prozess. Dazu lokalisieren sie einen so genannten Directory Agent.

– **Service Agents**
  verbreiten die Nachricht über das Vorhandensein eines Dienstes im Netzwerk und vermitteln eine Dienstbeschreibung, also eine Menge von Attributen.

– **Directory Agent**
  hat die Funktion eines zentralen Dienstverzeichnisses und bearbeitet eingehende Dienstanfragen (Lookup-Prozess)

In SLP ist auch der Einsatz von mehreren Directory Agents möglich, wobei dann eine Replikation von Dienstbeschreibungen sinnvoll ist. Dies ist z.B. zwecks Optimierung der Systemleistung ratsam.
Im Ergebnis der Arbeit eines Directory Agents wird eine Liste mit Dienstreferenzen in Form von URLs generiert, über die die Dienste direkt angesprochen werden können.
Vorraussetzung ist ein beliebiges Netzwerkprotokoll und ein vollständig konfiguriertes Netz.

6

Kontextsensitive Dienstnutzung

►

Inhaltsangabe

6.1 Kontextadaptivität

Nach [9] erlaubt das Konzept der Kontextadaptivität die Entwicklung von Anwendungen, die sich implizit der Situation eines Benutzers anpassen. Damit lassen sich nicht nur neue Anwendungstypen entwickeln, sondern auch bereits bestehende Anwendungen lassen sich um diesen Aspekt erweitern.
Im Folgendem wird zunächst das Konzept der Kontextinformation erläutert. Der Begriff der Kontextadaptivität und Kontextinformation wurde zum ersten Mal in einer Arbeit von Schilit und Theimer (vgl. [11]) erwähnt. Im Folgenden wird jedoch die Definition nach Dey und Abowd [6] angewandt:

> *Als* **Kontextinformation** *(oder Kontext) wird jegliche Information bezeichnet, welche benutzt werden kann, um die Situation einer Entität zu charakterisieren. Eine Entität ist eine Person, ein Lebewesen, ein Ort oder ein Objekt, welches bzgl. der Interaktion zwischen einem Benutzer und einer Anwendung als relevant einzuordnen ist. Dabei schließt man auch den Benutzer und die Anwendung mit ein.*

Beispiele:

- Location based Services: Person $\overset{\text{Kontext}}{\rightarrow}$ Ort. Dies bedeutet die Person X ist an Ort Y.
- Ort $\overset{\text{Kontext}}{\rightarrow}$ Person. Dies bedeutet am Ort Y sind die Personen X, X_1,... . Dies ist Zonenorientiert.

Die Nutzung von Kontextinformationen ermöglicht Anwendungen, die sich der jeweils gegenwärtigen Situation anpassen. In diesem Zusammenhang ist zu beachten, dass Kontextinformationen nur Hinweise auf eine bestimmte Situation geben, und die jeweilige Anwendung muss diese Information erst interpretieren. Die Interpretation von Kontextinformationen ist immer abhängig vom Typ der Applikation.
Die vorgestellte Definition für Kontextinformationen berücksichtigt neben impliziten Eingaben auch explizite Eingaben. Es existieren einige Kontexttypen, welche im praktischen Einsatz

Kontexttyp	Beispiele
Identität	Benutzername und Passwort
Ortsinformationen	Ort, Orientierung, Geschwindigkeit, Beschleunigung
Aktivität	Sprechend, lesen, laufend, sitzend
Zeitinformationen	Uhrzeit, Datum, Jahreszeit
Umgebungsvariablen	Temperatur, Luftqualität, Geräuschpegel
Ressourcen in der Nähe des Benutzers	Erreichbare Server und Endgeräte

Tabelle 6.1: Kontexttypen

eine größere Relevanz haben als andere, dazu gehören *Identität, Ortsinformation und Aktivität*. Mittels dieser Kontexttypen lassen sich in einigen Szenarien bereits viele Aussagen bzgl. der Situation einer Entität schließen. Zudem können von diesen Kontextinformationen andere abgeleitet werden. Aus der Identität eines Benutzers kann man z.B. dessen E-Mail-Adresse

oder Telefonnummer ableiten (vgl. [6]). In diesem Zusammenhang spricht man auch von *primären* und *sekundären* (also abgeleiteten) Kontexttypen.
Kontextadaptivität zeichnet ein System aus, welches Kontextinformation einsetzt, um eine bestimmte Funktionalität zu erfüllen. Ein kontextadaptives System interpretiert diese Informationen, um sich einer Situation anzupassen. Die zentralen Herausforderungen eines solchen Systems bestehen in der *Konstruktion, Repräsentation* und *Verarbeitung* von Kontextinformationen.
Entität und Kontext sind Rollen.
Vorgehen:

1. 1. Welcher Dienst soll erbracht werden? Z.B. Child-Tracking, Friend-Finder, PoI-Discovery (Point of Interest-Dicovery).

2. 2. Was ist die Entität(zentral für Dienst)? Z.B. entlaufendes Kind.

3. 3. Welcher Kontext ist für die Entität relevant?

In der Regel erfordern kontextsensitive Dienste eine Personalisierung zu Beginn des Dienstes (z.B. welche Zone ist für das Kind erlaubt [ist zeitunabhägig], welcher Radius definiert den Begriff „meine Nähe“).
Beispiel: Primärer Kontext Temperatur: z.B. $5°C$: wird gemessen mittels Sensor.
Mapping: $\begin{cases} \leq 22°C \rightarrow kalt \\ > 22°C \rightarrow warm \end{cases}$
Ergebins des Dienstes ist ein skundärer Kontext: kalt.

Lokalitätsprinzip Kontextinformationen sind in ihrer Bedeutung örtlich eingeschränkt. Sie beziehen sich auf einen bestimmten Ort oder auch ein Raumsegment. Insbesondere im Bereich mobiler Systeme bzw. eingebetteter Systeme sind der Ort der Nutzung und der Ort, an dem die Informationen entstehen, wesentlich. Der Ort ist immer ein kennzeichnender Parameter einer kontextadaptiven Anwendung. Diese Begebenheit ist offensichtlich bei einer ortsbezogenen Anwendung, wo der Ort einen direkten Einfluß auf das Verhalten eines Systems hat (z.B. Navigationssystem).
Die Gewinnung von Kontextinformationen ist immer ortsgebunden. Am Ort der Gewinnung ist die Relevanz von Kontextinformationen am höchsten und mit zunehmendem Abstand vom Ursprungsort nimmt die Relevanz im Allgemeinen ab. Es ist also eine hohe Sensordichte gefordert.

Prinzip der Zeitnähe Komplementär zum Lokalitätsprinzip ist das Prinzip der Zeitnähe zu verstehen. Die Erfassung einer Situation erfordert die Berücksichtigung zeitlicher Abläufe. Insbesondere die Gleichzeitigkeit ist bei der Bewertung und Auswertung von Messdaten von vorrangiger Relevanz. Das Prinzip der Zeitnähe ist ein essentieller Aspekt bei der Konstruktion von Kontextinformationen.

Welcher Dienst?	Child-Tracking	Friend-Finder	PoI-Discovery
Was ist Entität?	entferntes Kind	ich	ich/PoI
Welcher Kontext?	Ort	Peronen in der Umgebung	PoIs in der Umgebung

Tabelle 6.2: Vorgehen

Analog zum Lokalitätsprinzip nimmt die Relevanz von erfassten Daten mit zunehmender zeitlicher Verzögerung ab. In Konsequenz wird das Signal eines Sensors, welches bzgl. eines Ereignisses mit geringster zeitlicher Verzögerung gemessen wird, vorrangig bei der Gewinnung von Kontextinformationen berücksichtigt. Die Gültigkeit einer Kontextinformation ist i.d.R. begrenzt auf einen beschränkten Zeitraum.
Aus Sicht der Softwaretechnik erfordert das Prinzip der Zeitnähe Echtzeiteigenschaften des kontextadaptiven Systems. So könnte beispielsweise der Einsatz echtzeitfähiger Betriebs- und Kommunikationssysteme erforderlich sein.

6.2 Prinzip der kontextadaptiven Dienstnutzung

Die kontextadaptive Dienstnutzung wird hier als Oberbegriff für die kontextadaptive **Selektion** und **Ausführung** von Diensten verstanden. Die kontextadaptive Selektion erweitert die grundlegenden Techniken der Dienstvermittlung, wie sie in Kapitel 5.2 erläutert wurden, insofern, dass die Vermittlung nicht ausschließlich durch die Spezifikation von gewünschten Dienstattributen erfolgt, sondern auch kontextuelle Einschränkungen Berücksichtigung finden. Auch die Ausführung bzw. Nutzung eines Dienstes kann innerhalb des hier erläuterten Verfahrens an kontextuelle Bedingungen geknüpft werden. Die Kombination von kontextadaptiver Selektion und Ausführung ermöglicht eine personalisierte und situationsbezogene Bereitstellung von Diensten. Durch die Durch diekontextadaptive Selektion wird die Frage beantwortet, welcher Dienst gewählt werden soll. Die kontextadaptive Ausführung beantwortet, wie der Dienst ausgeführt wird, also entweder personalisiert, also an den Nutzer angepasst, was relativ statisch ist, oder situationsbezogen, also kontexabhängig, was hochdynamisch ist.

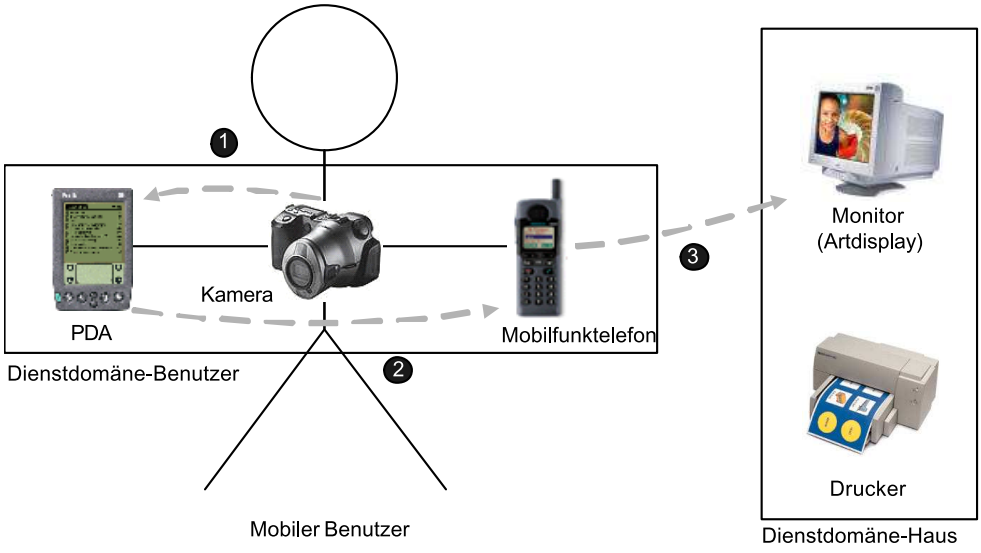


Abbildung 6.1: Szenario

Zur Motivation soll das in Abbildung 6.1 dargestellte Szenario betrachtet werden. Ein Benutzer trägt drei Endgeräte bei sich: eine digitale Kamera, einen PDA und ein Mobiltelefon. Die

einzelnen Geräte sind innerhalb eines **Personal Area Network (PAN)**, wie etwa Bluetooth organisiert. Das PAN bildet eine abgeschlossene Dienstdomäne (vgl. Abbildung 6.3), in der die angebotenen Dienste der eingeschlossenen Geräte wechselseitig genutzt werden können. Es existieren zwei weitere Dienstdomänen. Es gibt zum einen eine entfernte Dienstdomäne (nicht erreichbar durch das PAN), die ein so genanntes **Artdisplay** zur dauerhaften Anzeige von digitalem Bildmaterial bereitstellt und zum anderen eine Dienstdomäne, die einen Drucker zur Verfügung stellt. Bei beiden Dienstentitäten handelt es sich um eingebettete Geräte, die womöglich Teil eines Gebäudes sind.
Nachfolgend werden Wechselwirkungen zwischen den einzelnen Dienstentitäten anhand von Aktionen des Benutzers beschrieben.

– Zunächst schießt der Benutzer Bilder mit seiner Digitalkamera. Dann veranlasst er, dass die gemachten Bilder auch auf seinem PDA anzuschauen sind.

– Neben dem Vorteil des größeren Displays, bietet das PDA auch Möglichkeiten der Weiterverarbeitung von eingehenden Bildern. Zu diesem Zweck bietet das PDA dem Nutzer weiterführende Dienstoptionen an. Die Selektion der Dienstoptionen erfolgt kontextadaptiv (z.B. abhängig vom Dienstangebot der Umgebung, Datentyp, etc.)

– Im hier betrachteten Szenario entscheidet der Benutzer sich dafür, ein gemachtes Bild zu Hause auf einem Artdisplay anzubringen. Da sich das Artdisplay in einer entfernten Dienstdomäne befindet, wird es zunächst zum Mobiltelefon übertragen.

– Das Mobiltelefon agiert hier als ein allgemeiner Kommunikationsserver. Es empfängt das Bild, welches für das Artdisplay bestimmt ist. Zu einem früheren Zeitpunkt hat der Benutzer entschieden, dass Daten, die ein bestimmtes Volumen überschreiten, aus Kostengründen nicht mittels Weitverkehrsnetzen (z.B. GPRS) versendet werden dürfen. Da es sich nicht um zeitkritische Daten handelt, werden diese erst dann übertragen, wenn sich der Benutzer in Reichweite seines Heimnetzwerkes befindet.

– Schließlich wird das Bild auf dem Artdisplay angezeigt.

Alternativ wäre folgender Ablauf denkbar (vgl. Abbildung 6.1):

– Statt der Anzeige des Bildes auf dem Artdisplay entscheidet sich der Benutzer für Dienstoption „Drucken auf lokal verfügbarem Drucker“.

– Da sich gerade ein oder mehrere Drucker in direkter Nähe des Benutzers befinden, wählt das System entsprechend seines Kontextes (hier Ort, Benutzerprofil) einen Drucker aus.

– Der resultierende Druckauftrag wird ausgeführt.

Das Szenario zeigt zum einen Anwendungen für das Konzept der kontextadaptiven Selektion, so etwa am Beispiel der Auswahl eines Druckers bzgl. kontextueller Einschränkungen. Zum anderen wird auch die Notwendigkeit einer kontextadaptiven Ausführung von Diensten demonstriert. So erfolgt die Ausführung des Anzeigedienstes (Artdisplay) erst dann, wenn die kontextuellen Einschränkungen bzgl. dieser Dienstnutzung erfüllt sind. In diesem Fall bezogen sich die Einschränkungen auf die Minimierung der anfallenden Kosten und die Erreichbarkeit eines lokalen Netzwerkes. Zudem motiviert das angeführte Szenario eine kontextadaptive Weiterleitung von Datenfragmenten.

Die Benutzung von Diensten soll aus Sicht des Nutzers implizit erfolgen. Das entsprechende System unterstützt den Nutzer bei der Auswahl und Ausführung von Diensten.
Das System verkörpert ein **Peer-to-Peer-Netzwerk** und zeichnet sich somit nicht durch eine Netzwerktopologie mit dedizierter Client/Server-Architektur aus. Jeder Knoten in einem Peer-to-Peer-Netzwerk kann sowohl Client- als auch Server-Funktionen übernehmen.
Ein wesentlicher Charakterzug von kontextadaptiven Systemen ist der Ortsbezug. In der Systemarchitektur sind Systemressourcen an einen bestimmten Ort gebunden. Deshalb ist die Angabe von Ortsinformationen zur Identifikation einer Ressource notwendig. Dieses Modell widerspricht den Architekturprinzipien von vielen anderen Verteilten Systemen, wo von dem Ort einer Ressource gewöhnlich abstrahiert wird. Analog sind Informationen an die Zeit gebunden (Zeitprinzip).

Dienstmodell Vor der Behandlung des hier zugrundeliegenden Dienstmodells, sollen noch einige Begriffe zusammengefasst werden:

- *Dienst:*
  Im Kontext dieser Arbeit bezeichnet ein Dienst ein Objekt, welches mittels eines Dienstvermittlungsprotokolls (z.B. Jini, SLP) lokalisiert werden kann. Dies ist eine Betrachtungsweise aus Sicht der Infrasturktur.
- *Diensterbringer:*
  Der Diensterbringer ist die Ressource (oder eine Menge von Ressourcen) auf der ein Dienst ausgeführt wird.
- *Dienstendpunkt:*
  Der Dienstendpunkt definiert den Kommunikationsendpunkt eines Dienstes, also einen Punkt, wo der Dienst von Außen in Anspruch genommen werden kann. Dieser kann z.B. durch eine IP-Adresse oder eine Objektreferenz repräsentiert werden.
- *Dienstnutzung:*
  Die Dienstnutzung bezieht sich auf den Vorgang der Inanspruchnahme eines Dienstes. Es wird beispielsweise eine Methode aufgerufen.

Abbildung 6.2 veranschaulicht die Architektur des Dienstmodells. **Domänen** (oder auch **Dienstdomänen**) werden durch einen zugehörigen **Domänenkontroller** verwaltet. Eine Domäne ist eine Menge von Geräten, von denen jedes mit jedem kommunizieren kann. Der Domänenkontroller verwaltet die Dienste und Sensoren in seiner Domäne. Von diesen kann jeder mit jedem kommunizieren. Die Domänenkontroller sind über ein Backbone Netzwerk miteinander verbunden, so dass es möglich ist, die Ressourcen mehrer Domänen in einen individuellen Verarbeitungsprozess mit einzubeziehen. Allerdings ist eine direkte Kommunikation zwischen Diensten unterschiedlicher Domänen nicht möglich. Die Kommunikation erfolgt immer indirekt über die entsprechenden Domänenkontroller.
Einer der Vorzüge dieses Prinzips, ist die Möglichkeit, dass man die Kommunikation zwischen Domänenkontrollern standardisieren kann. Folglich lässt sich beispielsweise auch in einer heterogenen Dienst- bzw. Sensorumgebung ein einheitliches Datenformat zur Kommunikation einsetzen.
Das hier betrachtete Dienstmodell schließt auch *abgeschlossene* Dienstdomänen (siehe Abbildung 6.3) mit ein. In diesem Fall sind die Interaktionen zwischen Diensten auf eine individuelle Domäne beschränkt. Die Konzeption der abgeschlossenen Domäne wurde bereits anhand

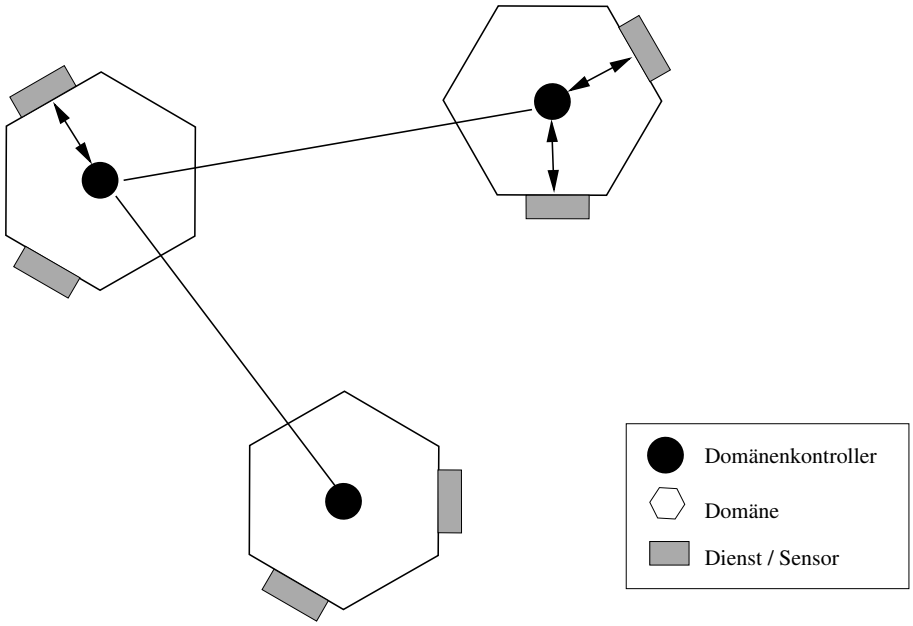


Abbildung 6.2: Dienstmodell

des einführenden Szenarios (siehe Abbildung 6.1) erläutert. In abgeschlossenen Domänen existiert kein expliziter Domänenkontroller, dennoch ist eine Anbindung an andere Domänen mittels eines Gateway-Dienstes möglich, im Szenario mittels des Mobilfunktelefons.
Abgeschlossene Domänen werden insbesondere im Fall von Personal Area Networks eingesetzt. Hier beschränken sich die Interaktionen i.d.R. auf Dienstendpunkte der jeweiligen Domäne. Zudem können unterliegende Dienste der eingesetzten Netzwerktechnologie direkt genutzt werden, so etwa das lokale Dienstvermittlungsprotokoll von Bluetooth.

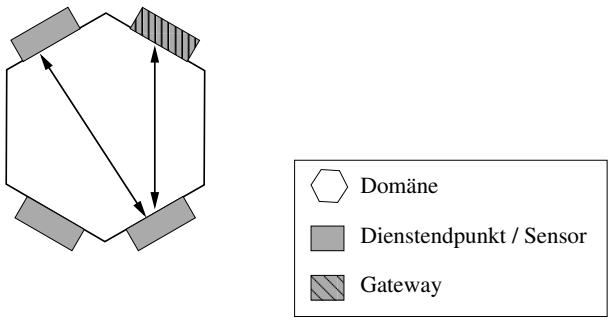


Abbildung 6.3: Abgeschlossene Domäne

Wir wollen das Prinzip des Regelkreises veranschaulichen.

Dieses Prinzip kann zur Ausführung von kontextabhängigen Diensten angewandt werden.
Als **Sollwert** dient dabei ein sog. **Context Constraint**, das definiert, unter welcher Kontextbedingung ein Dienst ausgeführt werden soll, z.B. wenn der Nutzer sich in der Nähe eines öffentlichen Druckers befindet oder Zugang zu seinem Heimatnetz hat.

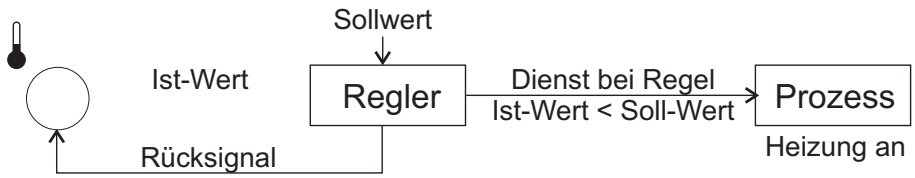


Abbildung 6.4: Schematische Darstellung des Regelkreises

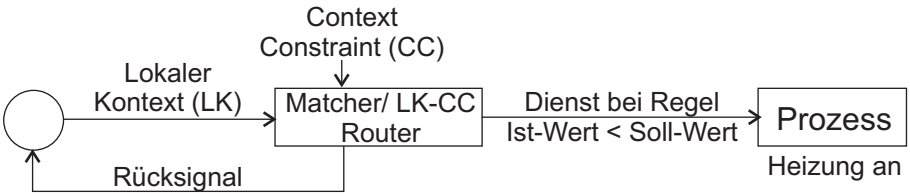


Abbildung 6.5: Regelkreis mit Context Constraint

Context Constraints werden innerhalb einer Dienstdomäne mit aktuell vorliegendem lokalen Kontext verglichen. Dieser lokale Kontext entspricht dem Ist-Wert des Systems.

Kommt es zu einer Übereinstimmung von lokalem Kontext und Context Constraints, so wird ein Dienst ausgeführt.

Ein solcher Dienst ist ein Push-Dienst, da er durch ein Ereignis getriggert wird. Er bedingt, dass das System ständig busy ist, um die zugrunde liegenden Auswertungen vorzunehmen.

Im Fall der **Nicht-Übereinstimmung** kann entweder auf eine passende Situation gewartet werden oder es wird festgestellt, dass die Bedingungen der Context Constraints in der aktuellen Domäne überhaupt nicht erfüllt werden können. Dann kann der Dienst innerhalb eines Routing-Prozesses zu einer weiteren Domäne versendet werden oder explizit abgebrochen werden.

Aus Sicht der Abarbeitung passiert folgendes:

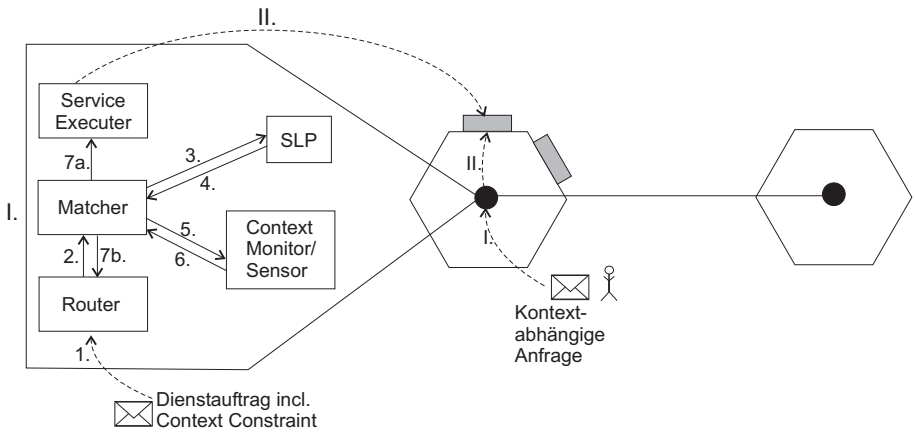


Abbildung 6.6: Abarbeitung von kontextsensitiven Diensten

Ein Benutzer tritt in Interaktion mit seiner Umgebung über einen Agenten, der als Interaction Organizer bezeichnet werden kann und die Organisation eines Auftrags/Dienstes übernimmt. Benutzer- und aufgabenbezogen erzeugt dieser Agent eine kontextabhängige Anfrage, die an die Infrastruktur gesendet wird. Diese Infrastruktur liegt in Form vernetzter Dienstdomänen vor, die durch jeweils einen Domänenkontroller mit Funktion eines Service Interaction Proxies kontrolliert werden.
Diese Komponente kann fünf Teilaufgaben erfüllen: Routen, Matchen ((ist = soll)und(SLP=true[Dienst liegt vor])), Context monitoren sowie ein Service Location Protocol (SLP) ausführen und die Ausführung eines Dienstes initiieren. Der Matcher realisiert die Übereinstimmung von gegebenem und gesuchtem Kontext. Der Router ist der Netzzugangspunkt.
Das Context Information System (CIS) beinhaltet typischerweise einen Sensor, wie z.B. den Ortungssensor. Das Context Aware System (CAS) beinhaltet dagegen mehr, nämlich z.B. einen Dienst, wie z.B. „Das Kind ist außerhalb des Bereichs“.

6.3 Das Kontextmodell

Wie bereits erwähnt sind Kontextinformationen zur Selektion und Ausführung von kontextadaptiven Diensten nötig. Da nicht jede Entität alle nötigen Kontextinformationen korrekt und zu jeder Zeit liefern kann, wird hierfür ein **Context Provider** eingesetzt. Der Context Provider stellt Kontextinformationen über Sensoren oder mit Hilfe anderer Context Provider zur Verfügung. Die Kontextinformationen können daraufhin vom Nutzer oder der Applikation angefordert und verarbeitet werden. Dafür sind die in Abbildung 6.7 dargestellten Schritte notwendig:

1. **Kontextmessung (Context Sensing)**
   Bei der Kontextmessung werden - meist mit Hilfe von Sensoren - Daten generiert, aus denen dann im nächsten Schritt Kontextinformationen erzeugt werden. Ein Sensor für örtliche Daten kann zum Beispiel ein GPS-Empfänger sein, der die aktuelle Position einer Entität im WGS84-Format ermittelt.

2. **Kontexterstellung (Context Construction)**
   Im folgenden Schritt, der Kontexterstellung, werden aus den Sensordaten Kontextinformationen erstellt oder aus bereits erstellten Kontextinformationen durch Transformation neue Kontextinformationen konstruiert. Ein Beispiel für den letzteren Fall ist ein kontextadaptiver Dienst, der temperaturabhängig agiert. Diese Abhängigkeit bezieht sich beispielsweise auf zwei disjunkte Zustände {warm, kalt}. Stehen Kontextinformationen eines Thermometers, welches nur Werte zwischen 0 und 50°C liefert, zur Verfügung, muss aus diesen Werten eine Abbildung auf die Skala {warm, kalt} erfolgen:

   $$[0-23) \quad \rightarrow \quad kalt$$
   $$[23-50] \quad \rightarrow \quad warm$$

   Ein anderes Beispiel ist die Aggregation von Kontextinformationen. Die bereits genannte Kontextinformation Temperatur könnte so mittels einer Kontextinformation Niederschlag und einer Kontextinformation Luftdruck zu einer Kontextinformation Wetter „veredelt“ werden.

Letztendlich müssen bei der Kontexterstellung Kontextinformationen in einer standardisierten Form generiert und mit einer standardisierten Schnittstelle zur Verfügung gestellt werden.

3. **Kontextverteilung (Context Dissemination)**
   Die Verteilung von Kontextinformationen nennt man auch Context Dissemination. Im Wesentlichen unterscheidet man zwei verschiedene Verfahren: **Polling** (reaktiv) und **Pushing** (proaktiv). Beim Polling fordert eine Entität eine Kontextinformation bezüglich eines relevanten Aspekts (z.B. Temperatur) bei einem Context Provider an. Der Transfer von Kontextinformationen wird also vom Nutzer angestoßen. Beim Pushing hingegen registriert sich der Nutzer beim Context Provider bezüglich eines relevanten Aspekts und erhält dadurch bei jeder Veränderung die aktuelle Kontextinformation dieses Aspektes vom Context Provider. In diesem Fall wird der Transfer von Kontextinformationen vom Context Provider angestoßen.

4. **Kontextinterpretation (Context Interpretation)**
   Im letzten Schritt, der Kontextinterpretation, müssen die erhaltenen Kontextinformationen vom Nutzer oder der Applikation interpretiert werden.

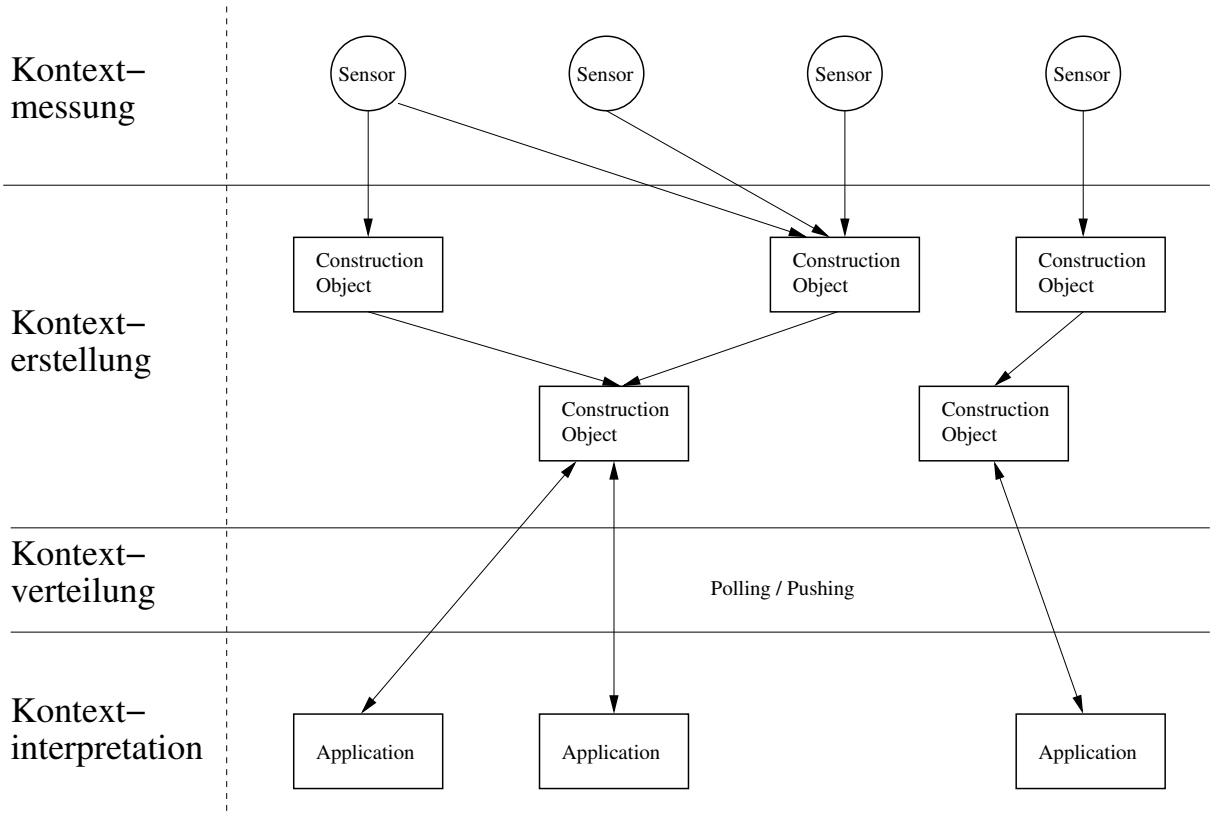


Abbildung 6.7: Kontextverarbeitung

Um die Kontextinformationen zwischen den beteiligten Entitäten auszutauschen, ist bei der Kontexterstellung eine einheitliche Form zur Beschreibung von Kontextinformationen nötig.

Im folgenden Kapitel soll nun ein formales Modell zur Beschreibung und Bearbeitung von Kontextinformationen vorgestellt werden.

Modellierung von Kontextinformationen

Im Allgemeinen bezieht sich eine Kontextinformation auf eine Entität. Ist diese Entität zum Beispiel ein Zimmer, können Kontextinformationen die Menge der Personen in diesem Zimmer oder die Temperatur des Zimmers sein.

Eine konkrete Kontextinformation $c \in C$ sieht dann wie folgt aus:

$$c_e := (e, attr_1, \ldots, attr_N) \qquad \text{mit Entität } e \in E, attr_1 \in A_1, \ldots, attr_N \in A_N$$

Beispiel:

Ist eine Art von Kontextinformationen wie folgt aufgebaut:

$$C = (Room, Temperature, PersonsPresent)$$

wäre eine konkrete Kontextinformation dieser Art beispielsweise:

$$C \ni c = (Room.45, 23°, \{Birgit, Max\})$$

Um nun mit den Kontextinformationen arbeiten zu können, sind einige Funktionen auf Kontextinformationen nötig:

Aggregation 2 Kontexte, die sich auf dieselbe Entität beziehen, werden vereinigt.

$$agg : C \times \cdots \times C \to C$$

Beispiel:

$$agg((Room.45, 23°), (Room.45, \{Max, Birgit\})) = (Room.45, 23°, \{Max, Birgit\})$$

Transformation Ableitung von schwächeren Kontexten

$$trans : C \times \cdots \times C \to C$$

Beispiel:

$$trans(([Room], [T])) = \begin{cases} ([Room], cold), & if([T] < 23°) \\ ([Room], warm), & else \end{cases}$$

wenn $c_1 := (Room.45, 23°)$ dann:
$trans(c_1) = (Room45, warm)$

Selection Bei der Selektion können aus der Menge aller konkreten Kontextinformationen die Kontextinformationen ausgewählt werden, die eine bestimmte Bedingung erfüllen. Oft interessiert nur eine Teilmenge aller Entitäten mit einer bestimmten Bedingung.

C_{sys} :=Menge aller Kontexte
$select : C_{sys} \times cond \rightarrow \mathcal{P}(C_{sys})$

Beispiel:

Gegeben sei folgender Bestand von Kontextinformationen im System (C_{sys}):

$$(Room.45, 23°, \{Max, Birgit\})(Room.23, 21°, \{Claudia, Thomas\})$$

Wäre die Bedingung $cond$, alle Räume mit einer Temperatur von größer als 22° zu finden, so ergäbe sich:
$\mathcal{P}(C_{sys}) = (Room.45, 23°, \{Max, Birgit\})$

Projektion Mit der Projektion können Kontextinformationen mit M Attributen auf Kontextinformationen mit $N < M$ Attributen projeziert werden.

$project : C \times \Pi \rightarrow C^*$
Hierbei beschreibt Π die Projektionsvorschrift.

Beispiel:

Gegeben sei wieder C_{sys} wie bei der Selektion. C_{sys} soll auf Kontextinformationen projeziert werden, die nur den Raum und die Temperatur beinhalten.

$$C_{sys} \times \Pi_{(Room,Temperature)} \rightarrow \begin{pmatrix} (Room.45, 23°) \\ (Room.23, 21°) \end{pmatrix}$$

Weiterführendes Beispiel:

$trans_i := trans$

$$trans_j(([Room], [T])) = \begin{cases} ([Room], cold), & if([T] < 28°) \\ ([Room], warm), & else \end{cases}$$

$trans_i(c_1) = (Room.45, warm)$
$trans_j(c_1) = (Room.45, cold)$

7

Synchronisation

►

Inhaltsangabe

Bislang haben wir uns mit Prozessen, der Kommunikation zwischen Prozessen sowie deren Unterstützung durch das Naming beschäftigt. Nun soll eine andere Thematik anhand konkreter Szenarien betrachtet werden:

- Falls mehrere Prozesse jeweils ein Dokument ausdrucken sollen, so darf auf die gemeinsam genutzte Ressource des Druckers nicht simultan zugegriffen werden, sondern es muss eine Kooperation in dem Sinne erfolgen, dass jeder Prozess temporär exklusiven Zugriff hat.

- In bestimmten Szenarien ist es notwendig, dass sich mehrere Prozesse auf eine zeitliche Ordnung von Ereignissen einigen, z.B. dass die Nachricht *m1* von Prozess *P* vor der Nachricht *m2* von Prozess *Q* gesendet wurde.

In einem Verteilten System ist eine solche Synchronisation wesentlich schwieriger zu realisieren als in einer Ein-Prozessor-Umgebung. Die Realisierung einer zeitlichen Ordnung ist dabei nicht über die Ausführungsreihenfolge von Befehlen in einem Prozessor zu realisieren, und im Falle eines Zeitstempels sind die Uhrzeiten der einzelnen Prozessoren aufeinander abzustimmen. Diesem Problem wollen wir uns im Folgenden widmen. Zunächst soll die Komplexität dieses Problems am Beispiel der Navigationssysteme erläutert werden.

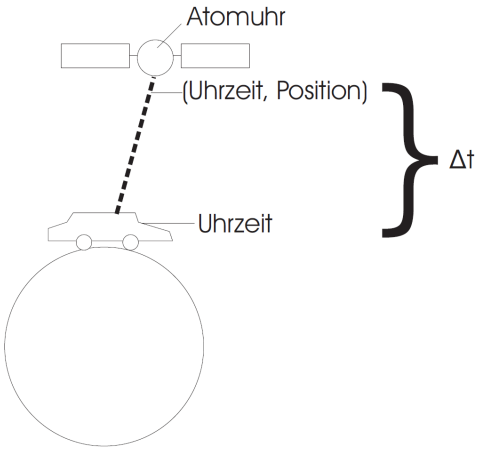


Abbildung 7.1: GPS

Global Positioning System (GPS) Δt :=Sendezeit−Empfangszeit=Laufzeit des Signals
$v = \frac{s}{t}; s = v \cdot t$ =Abstand des Empfängers vom Satelliten
$v \approx 300000 \frac{km}{s}$

Historische Bemerkungen Aufgabe des Global Positioning System (GPS) ist nach [4] die satellitengestützte Ortung von Objekten auf oder in der Nähe der Erde. Bei bewegten Objekten läßt sich darüber hinaus auch ihre Geschwindigkeit und Bewegungsrichtung feststellen.
Das Global Positioning System, auch NAVSTAR (Navigation System with Time and Ranging) genannt, begann zunächst als militärisches Projekt, wird heute aber auch in zivilen Bereichen

eingesetzt, z.B. bei der Navigation von Schiffen, Flugzeugen und Autos. Es ist die Weiterführung des Systems Transit, das die USA in den 60er Jahren als Reaktion auf die sowjetischen Erfolge in der Weltraumfahrt als erstes Navigationssystem auf Basis von Satellitenortung entwickelten. Allerdings hätten unerwartete technische und damit auch finanzielle Probleme zu Beginn der 90er Jahre fast zum Abbruch des Projekts geführt. Seit 1995 ist das System aber voll einsatzfähig.
Als allein von den USA finanziertes und betriebenes System lässt sich GPS jederzeit für zivile Anwendungen manipulieren und unbrauchbar machen. Tatsächlich wurden die Satellitensignale bis Mai 2000 für nichtmilitärische Anwendungen künstlich verfälscht, so dass nur eine eingeschränkte Genauigkeit zur Verfügung stand (Selective Availibility). Dem konnte aber auch im zivilen Bereich durch den Einsatz von DGPS (differenziellem GPS) begegnet werden.

Aufbau des Systems GPS besteht aus einem Verbund von zunächst 24, später 36, Satelliten, die die Erde auf elliptischen (nahezu kreisförmigen) Bahnen in ca. 20200 km Höhe umrunden. Je höher der Satellit kreist, desto länger ist die Umlaufzeit (z.B. 36.000 km $\triangleq$ 24 Std., 200 km $\triangleq$ 86 Min.). Dabei bewegen sich je vier Satelliten auf sechs unterschiedlichen Bahnebenen, die um 55Â° gegen die Äquatorebene geneigt und gegeneinander um 60Â° versetzt sind. Nach dem 3. Keplerschen Gesetz (die Quadrate der Umlaufzeiten verhalten sich wie die Kuben der mittleren Entfernungen) ergibt sich - z.B. mit dem Erdmond als Referenzsatellit - für die GPS-Satelliten eine Umlaufzeit von nahezu 12 Stunden, genauer 11h 57' 59,3" $\triangleq \frac{1}{2}$ Sternentag. Die GPS-Satelliten sind nicht geostationär! Aus dieser Anordnung folgt einerseits, dass sich zu jedem Zeitpunkt an jedem Ort der Erde mindestens vier Satelliten in brauchbarer Höhe über dem Horizont befinden (dies ist für das Funktionieren des Systems notwendig) und dass andererseits die Satelliten in unseren Breiten vorwiegend in südlicher Richtung stehen.

Funktionsprinzip Das Prinzip der Satellitennavigation ist recht einfach: Jeder Satellit sendet laufend mit 50 Bit pro Sekunde ein Datenpaket aus, das u.a. die Sendezeit als genaue Uhrzeit seiner Atomuhr („Zeitstempel") und die augenblickliche Position des Satelliten enthält. Der Empfänger auf der Erde bestimmt die Ankunftszeit des Signals und erkennt mittels CDMA (Code Division Multiple Access), von welchem Satelliten er soeben Signale empfängt. Aus der Laufzeit (zwischen 0,067 s und 0.086 s) ergibt sich dann die Entfernung zum Satelliten. Genauer: Die Übertragungszeit vom Satelliten zum Empfänger (bereinigt um Fehler, wie z.B. Zwillingsparadoxon der Relativitätstheorie) multipliziert mit der Lichtgeschwindigkeit ergibt die Entfernung zwischen Satellit und Empfänger. Mit drei solcher Messungen zu verschiedenen Satelliten kann man die Position des Empfängers im Raum bestimmen. Vom jeweiligen Satelliten aus gesehen befindet sich der Empfänger nämlich auf der Oberfläche einer Kugel, deren Radius gerade über die Signallaufzeit bestimmt wurde. Zwei solcher Kugeloberflächen schneiden sich in einem Kreis, der wiederum die dritte Kugeloberfläche in zwei Punkten schneidet, von denen einer meist sofort ausgeschlossen werden kann, da er irgendwo im Weltall liegt.
So einfach liegen die Verhältnisse in Wirklichkeit jedoch nicht. Da die Signale mit Lichtgeschwindigkeit (im Vakuum 299792,5 km/s) übertragen werden, sind die Anforderungen an die Genauigkeit enorm: Ein Laufzeitfehler von einer zehntausendstel Sekunde würde einen Distanzfehler von 30 km bewirken und damit das System unbrauchbar machen. Die erforderliche Präzision lässt sich nur mit Atomuhren erreichen. An Bord der Satelliten werden daher

Cäsium- und Rubidium-Atomuhren verwendet, die regelmäßig von fünf Bodenstationen kontrolliert und nachgeregelt werden. Die Abweichung zwischen den Satellitenuhren und der Uhr in der terrestrischen Station betragen nur 5 Milliardstel Sekunden. Das entspricht etwa 3 Metern.
Mit 3 Satelliten hat man 3 Gleichungen mit 3 Unbekannten zur Ortung. Wenn man noch einen 4. Satelliten dazunimmt, kann man als 4. Unbekannte die Abweichung der Uhren wählen.

7.1 Uhrensynchronisation

In Synchronisationsverfahren spielt die Zeit i.d.R. die Hauptrolle. In Verteilten Systemen steht aber keine allgemeine Uhr, d.h. keine globale Zeitquelle, zur Verfügung, da die Gesamtheit relevanter Informationen zum Zwecke einer Auswertung nicht an einer Stelle gesammelt wird. Dies ergibt sich aus der Forderung, dass Verteilte Systeme fehlertoleranter, zuverlässiger und ausfallsicherer als zentrale Systeme sein sollen.

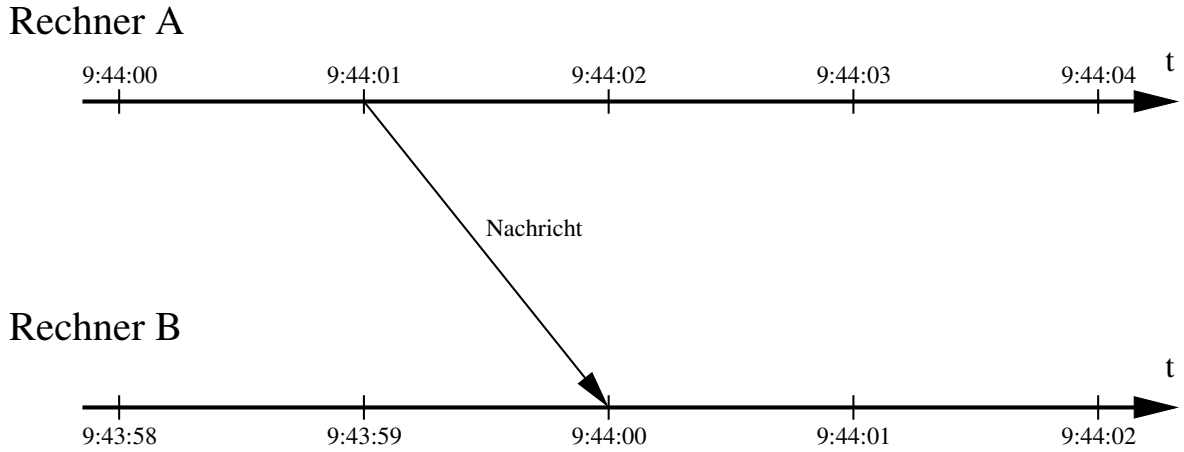


Abbildung 7.2: Verschiedene Systemzeiten

Die Abbildung 7.2 zeigt zwei Rechner, die jeweils unterschiedliche Systemzeiten besitzen. Da Rechner B nachgeht, hat das Ereignis der Ankunft einer Nachricht (9:44:00) einen früheren Zeitpunkt als das Abschicken auf Rechner A (9:44:01).
Um dieses Problem zu lösen, soll zunächst die Frage betrachtet werden, wie Uhren in Rechnern eigentlich funktionieren. Ein Rechner verfügt über eine Stoppuhr (Timer), d.h. der Timer enthält einen Quarzkristall, der unter Spannung mit einer definierten Frequenz schwingt. Außerdem besitzt der Rechner ein Zähl- und ein Laderegister. Das Zählregister wird bei einer Schwingung um eins vermindert. Hat der Zähler den Wert 0 erreicht, so wird eine Unterbrechung ausgelöst, die man Uhrtick nennt. Anschließend wird der Zähler mit dem Wert des Laderegisters neu geladen. Beim Systemstart gibt der Operator Datum und Uhrzeit an. Die Werte werden von diesem Zeitpunkt an dann vom Uhrtick aktualisiert (z.B. 60 mal pro Sekunde).
Das Problem der Uhrensynchronisation kann also auf die Initialisierung der Uhren zurückgeführt werden. D.h. wie werden Datum und Uhrzeit angegeben? In diesem Zusammenhang sollen die Grundlagen einer exakten Zeitmessung im Folgenden betrachtet werden.

7.1.1 Physikalische Uhren

Zunächst soll ein Blick in die Historie der Zeitmessung geworfen werden. Die Zeit wurde bis zum 17. Jahrhundert astronomisch gemessen. Jeden Tag geht die Sonne im Osten auf und im Westen unter. Wenn die Sonne am Mittag ihren höchsten Stand erreicht hat, so nennt man dieses Ereignis zu diesem Zeitpunkt *Sonnendurchlauf*. Die Distanz zwischen zwei Sonnendurchläufen wird Sonnentag gennant. Da jeder Tag aus 24 Stunden zu je 3600 Sekunden besteht, ist die *Sonnensekunde* als der 86.400te Teil eines Sonnentages definiert. Mit der Einführung mechanischer Uhren hatte man erstmals die Möglichkeit, die Zeit unabhängig von der Sonne zu messen. Da sich die Erde jedoch nicht gleichmäßig dreht, ist auch der auf der Sonnensekunde beruhende Sekundenbegriff ungenau (Geologen wiesen durch Untersuchungen an 300 Millionen Jahren alten Korallen nach, dass das Jahr damals aus 400 Tagen bestand). D.h. die Erde dreht sich zunehmend langsamer um ihre eigene Achse. Als 1948 die Atomuhr eingeführt wurde, konnte man den Begriff der Sekunde erheblich präzisieren. Eine Sekunde ist die Zeit, die ein Cäsium-133-Atom für 9.192.631.770 Zustandsübergänge benötigt. Zur Zeit gibt es etwa 50 Einrichtungen auf der Welt, die über eine Cäsium-Uhr verfügen. Diese Einrichtungen teilen dem *Bureau International de l'Heure (BIN)* in Paris regelmäßig mit, wie ihre Uhren gehen. Aus dem Mittelwert errechnet sich die **Internationale Atomzeit (TAI)**. Sie wird seit dem 1.1.1958 berechnet (ganz präzise ausgedrückt ist die TAI also die mittlere Anzahl von Zustandsübergängen der Cäsium-133-Uhren seit 0.00 Uhr am 1.1.1958 dividiert durch 9.192.631.770).

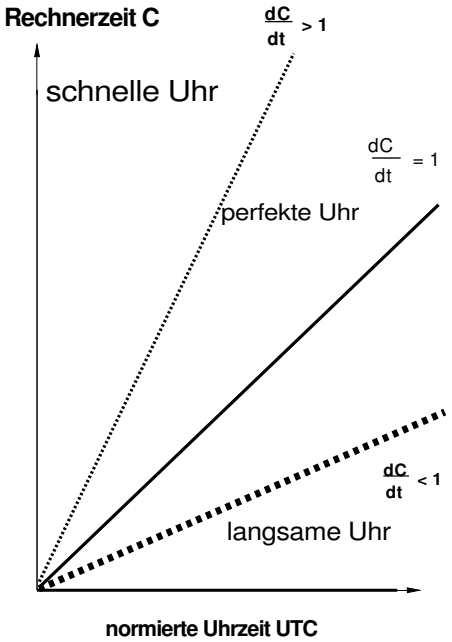


Abbildung 7.3: Langsame, perfekte und schnelle Uhren

Leider erweist sich aber sogar die TAI als ungenau. Dies hängt damit zusammen, dass 86.400 TAI-Sekunden genau 3-Millisekunden kürzer sind als ein normaler Sonnentag. Die Rotation der Erde verlangsamt sich nämlich durch die Gezeitenreibung und die Trägheit der Atmo-

sphäre. Das heißt: Die Anzahl der Tage pro Jahr wird bei konstanter Jahreslänge kleiner, bzw. die Tage werden immer länger. Um diesen Effekt zu kompensieren, wurden vom BIH Schaltsekunden eingeführt, die man dann einfügt, wenn der Unterschied zwischen TAI und Sonnenzeit auf 800 Millisekunden angewachsen ist (dies war seit 1958 schon über 30 Mal der Fall). Die Zeit, die man aus der TAI durch diese Korrektur erhält, heißt **Universal Coordinated Time (UTC)**.
Die UTC ist die Basis der modernen Zeitrechnung, die die Greenwich Mean Time abgelöst hat. Wenn eine Schaltsekunde einzufügen ist, so erhöhen die meisten Stromversorgungsunternehmen ihre Frequenz 60 bzw. 50 mal von 60 bzw. 50 Hz auf 61 bzw. 51 Hz, um alle Uhren in ihrem Versorgungsgebiet um 1 Sekunde vorzustellen. Ein Betriebssystem, das die genaue Zeit nachbilden möchte, müsste über spezielle Software verfügen, um den Einfügeprozess nachzuvollziehen, da die (meist sehr ungenaue) Stromfrequenz hier nicht zur Zeitmessung verwendet wird. Eine Möglichkeit wäre eine Schnittstelle für eine Funkuhr. Mit der UTC ergeben sich nun folgende Grundlagen. Sei $C_p(t)$ die Zeit zur UTC-Zeit t auf einem Rechner p. In einem idealen Verteilten System gilt dann $C_p(t) = t$, d.h. $\frac{dC}{dt} = 1$. Man spricht davon, dass eine Uhr innerhalb ihrer Spezifikation arbeitet, wenn es eine Konstante p gibt, so dass $1 - p \leq \frac{dC}{dt} \leq 1 + p$. Die Konstante p wird von Hersteller angegeben und heißt maximale Abweichungen (vgl. Abbildung 7.3).

7.1.2 Algorithmen zur Uhrensynchronisation

Falls ein Rechner über Kurzwelle oder Satellitenempfang die UTC erhält, besteht das Ziel, alle anderen Rechner mit dieser Uhrzeit zu synchronisieren. Dazu sollen folgende Algorithmen betrachtet werden.

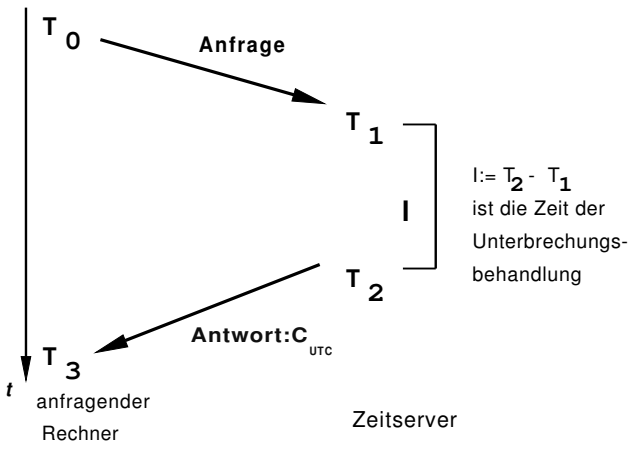


Abbildung 7.4: Zeitabfrage beim Zeitserver

Der Algorithmus von Christian zur physikalischen Uhrensynchronisation Die Voraussetzung für diesen Algorithmus ist ein Rechner, der eine Zeitbasis (z.B. UTC) liefert, ein sog. **Zeitserver**. Ziel ist es, andere Rechner mit ihm zu synchronisieren. Synchronisieren ist hier ein Verfahren, das auf einem anderen Endgerät auch eine Uhr nach UTC-Zeit stellt. Dazu

sendet jeder Rechner regelmäßig eine Anfrage nach der aktuellen Uhrzeit an den Zeitserver (vgl. Abbildung 7.4).
Der Server setzt seine Uhr auf $t = C_{UTC} + \frac{T_3 - T_0 - I}{2}$, d.h. er zählt zur UTC-Zeit noch die Dauer einer Nachrichtenübertragung hinzu. Ist I nicht bekannt, so setzt man $I = 0$.
Problematisch ist, dass die Zeit nie rückwärts laufen darf. Dies wäre nämlich dann der Fall, wenn die Uhr des aufrufenden Rechners zu schnell läuft und dann die lokale Zeit später als die übermittelte C_{UTC} ist. Somit kann das einfache Übernehmen der Informationen des Zeitservers Probleme bereiten. Eine Lösung wäre z.B. das Aufaddieren einer definierten Anzahl Millisekunden bei jeder Unterbrechung der Stoppuhr.

Der Berkeley-Algorithmus zur logischen Uhrensynchronisation Hier wird genau der umgekehrte Ansatz zum Christian Algorithmus gewählt. Der Zeitserver wird Zeitdämon genannt und ist eine aktive Komponente mit dem Ziel, die Zeit logisch zu synchronisieren. Das Verfahren eignet sich für Systeme, in denen keine exakte Zeit zur Verfügung steht.

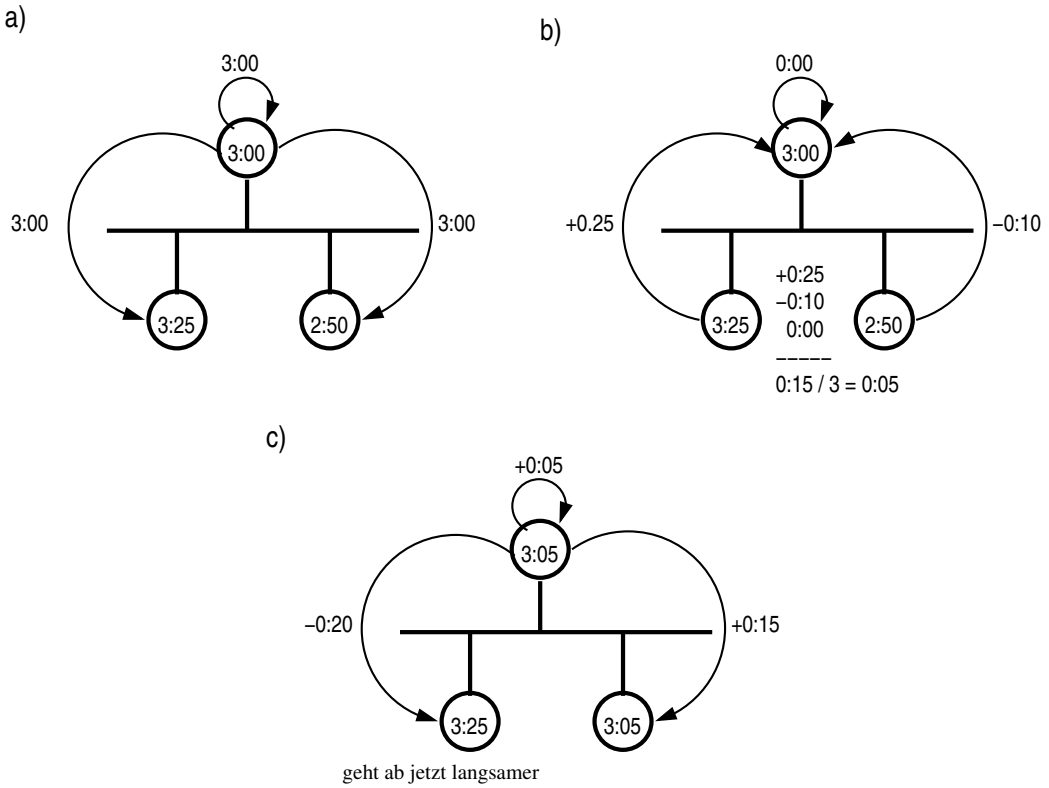


Abbildung 7.5: Synchronisation durch einen Zeitdämon

In regelmäßigen Abständen wird jeder Rechner nach seiner Zeit gefragt. Aus den erhaltenen Antworten wird der Durchschnitt berechnet und dieser den Rechnern mitgeteilt. Diese erhöhen die Uhrzeit auf den aktuellen Wert bzw. verlangsamen ihre Uhr dann (vgl. Abbildung 7.5). In der Abbildung 7.5 a) sendet der Dämon seine aktuelle Zeit (3:00) an die angeschlossenen Rechner. Die jeweilige Zeitdifferenz wird in Teil 7.5 b) an den Dämon zurückgesendet. Dieser ermittelt die relative Abweichung, aus der dann die neue Zeit (3:05) berechnet und

abgeschickt wird. Übertragungswege werden durch diesen Algorithmus automatisch herausgerechnet.

Mittelwert-Algorithmen Beide bislang beschriebenen Algorithmen verwenden zentrale Komponenten und haben alle damit verbundenen Nachteile (Bottleneck, schlechtes Fehlertoleranzverhalten, ...). Im Gegensatz dazu gibt es jedoch auch verteilte Algorithmen. Im Folgenden soll eine Art dieser verteilten Algorithmen betrachtet werden, die auf der Basis von Fixlängen-Resynchronisationsintervallen arbeiten (vgl. Abbildung 7.6).

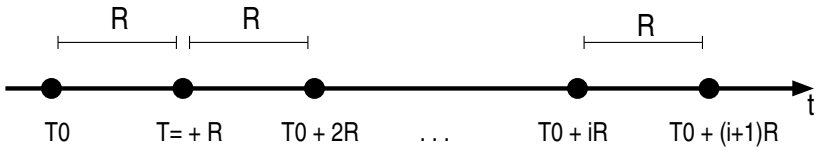


Abbildung 7.6: Fixlängen-Resynchronisationsintervalle

T_0 ist dabei ein vereinbarter Zeitpunkt der Vergangenheit und R ein Systemparameter, der die Länge eines Intervalls charakterisiert.
Zu Beginn jedes Intervalls broadcastet jeder Rechner die aktuelle Uhrzeit gemäß seiner Uhr. Allerdings geschieht dies in der Praxis doch nicht ganz simultan, da die Uhren in der Regel nicht mit exakt derselben Geschwindigkeit laufen.
Nach dem Abschicken (Broadcasten) der eigenen Uhrzeit startet der jeweilige Rechner einen lokalen Timer, um die Broadcast-Nachrichten aller anderen Rechner zu sammeln, die während eines Zeitintervalls S ankommen. Dann wird ein Algorithmus ausgeführt, der aus den gesammelten Werten eine neue Zeit berechnet. Als Algorithmus gibt es folgende Alternativen:

1. Der einfachste Algorithmus besteht in der Mittelwertbildung der von den anderen Rechnern erhaltenen Werte.

2. Eine leichte Variation geht davon aus, die m höchsten und m tiefsten Werte zu streichen und vom Rest den Mittelwert zu bilden.

3. Eine weitere Variation besteht darin, jede Nachricht durch einen Wert zu korrigieren. Dieser Wert ist der Erwartungswert, der für das Weiterleiten einer Zeit von der Quelle zur Senke besteht. Dieser Wert kann aus der Netztopologie gefolgert oder durch das Echo einer Probenachricht berechnet werden.

Einer der am meisten genutzten Algorithmen im Internet ist das **Network Time Protocol (NTP)**. Mit diesem Protokoll erreicht man eine Genauigkeit im Bereich von 1 bis 50 msek auf der Basis fortgeschrittener Uhrensynchronisationsalgorithmen nach Mills (1992 bis 1995).

Das Network Time Protocol (NTP) Dem NTP liegen folgende Ziele zugrunde:

– Internetweit sollen Clients genau gemäß UTC synchronisierbar sein.

– Der Dienst soll zuverlässig sein, auch wenn einzelne Komponenten, Verbindungen ausfallen.

– Der Dienst soll skalierbar sein.

– Ggf. soll durch Authentifizierungsmechanismen ein Schutz vorgenommen werden, d.h. vertrauenswürdige Sourcen sollen von manipulierten Daten unterscheidbar sein.

Die Realisierung des NTP-Dienstes wird durch ein Netz von Diensten realisiert, die über das Internet verteilt sind. Dabei unterscheidet man verschiedene Arten von Servern:
Primäre Server sind direkt mit einer Zeitquelle verbunden, die die UTC erhält. Sekundäre Server synchronisieren sich mit primären Servern, u.s.w. Dadurch entsteht eine logische Hierarchie von Servern, die Synchronisation Subnet (Synchronisationsteilnetz) genannt wird. Die einzelnen Level werden Strata genannt (Plural von Stratum, Schichten). Die Server auf der

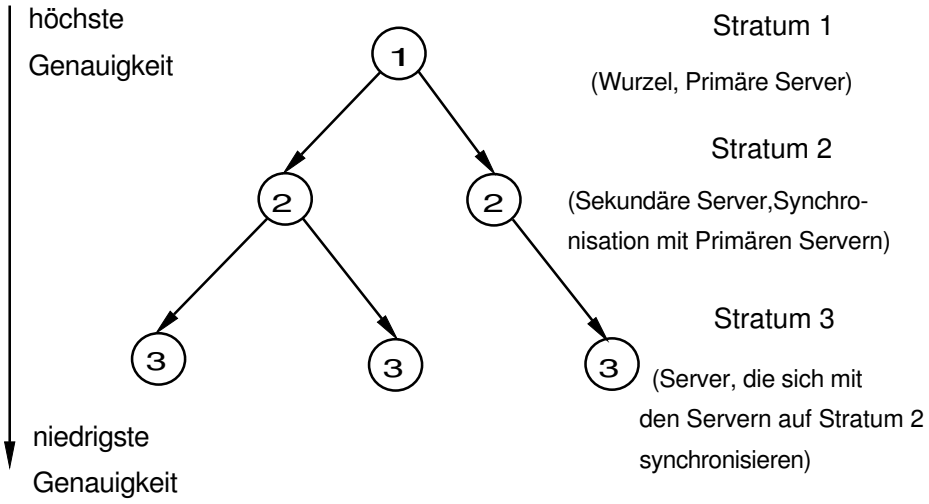


Abbildung 7.7: Schichtenstruktur des NTP

niedrigsten Schicht (Blätter) werden auf den Workstations der Nutzer ausgeführt. Sie sind die ungenauesten.
Fällt eine UTC-Zeitquelle für einen Stratum-1-Server aus, so kann dieser Server sich mit einem anderen Stratum-1-Server synchronisieren und wird selbst zu einem Stratum-2-Server.

Für die Synchronisation gibt es drei Modi:

– **Multicast Modus:**
geeignet für High Speed LAN: ein oder mehrere Server multicasten periodisch die Zeit an andere Rechner, wobei eine nur kleine Verzögerung angenommen wird.

– **Procedure-Call-Modus:**
ähnlich zu Christians Algorithmus, d.h. ein Server akzeptiert Anfragen anderer Rechner und antwortet mit der Rücksendung von Zeitstempeln. Dieser Modus wird angewandt, falls Multicasten systemseitig nicht unterstützt wird oder aber eine höhere Genauigkeit erreicht werden soll.

– **Symmetrischer-Modus:**
Server im selben, niedrigsten Stratum, d.h. mit höherer Genauigkeit, tauschen Zeitinformationen aus, um ihre Genauigkeit zu erhöhen.

Unabhängig vom konkreten Modus erfolgt der Nachrichtenaustausch über UDP. Der Procedure-Call-Modus und Symmetrische Modus basieren auf dem paarweisen Austausch von Nachrichten. Dabei trägt jede Nachricht die Zeitstempel der unmittelbar interessanten Ereignisse:
Die lokale Zeit, zu der die vorangegangene NTP-Nachricht gesendet und empfangen wurde, sowie die lokale Zeit, zu der die aktuelle Nachricht abgeschickt wurde. Zusätzlich notiert der Empfänger einer NTP-Nachricht die Empfangszeit. Damit ergeben sich die 4 Zeiten $T_i, T_{i+1}, T_{i+2}, T_{i+3}$ (vgl. mit Algorithmus von Christian für i=0). Dann gilt: $T_{i+1} = T_i + t + \sigma$

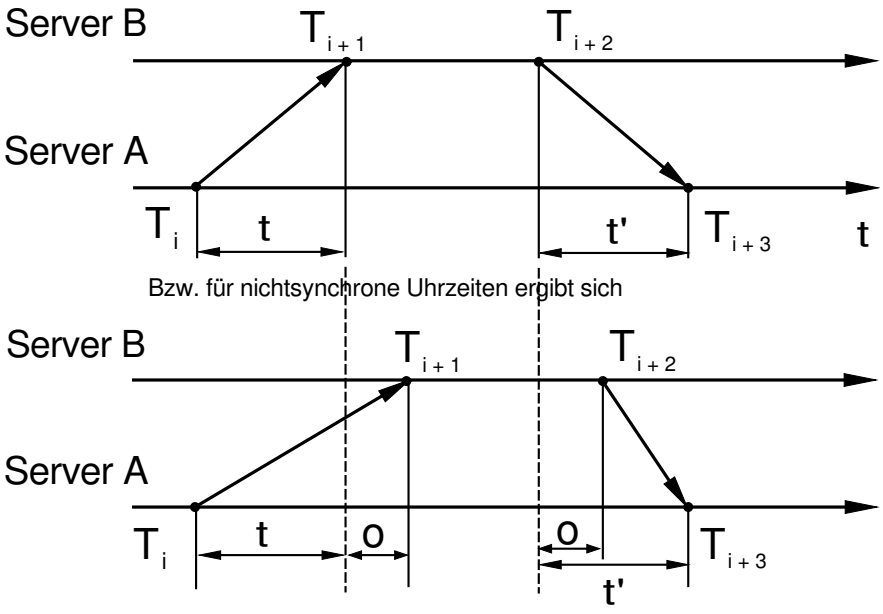


Abbildung 7.8: Synchronisierte versus nichtsynchronisierte Uhren

und $T_{i+3} = T_{i+2} + t' - \sigma$
Damit ergibt sich für die kommunikationsbedingte Verzögerung (Delay d_i):
$d_i = t+t' = (T_{i+1}-T_i-\sigma)+(T_{i+3}-T_{i+2}+\sigma) = T_{i+1}-T_i+T_{i+3}-T_{i+2} = T_{i+3}-T_i-(T_{i+2}-Ti+1)$.
Da wir ein Gleichungssystem aus 2 Gleichungen mit den 3 Unbekannten t, t' und σ haben, läßt sich σ nicht eindeutig berechnen, sondern lediglich abschätzen, z.B. $\sigma < T_{i+1} - T_i$ oder $\sigma < t'$. Mit der Annahme $t = t'$ würde sich ergeben: $\sigma = \frac{T_{i+1}-T_i+T_{i+2}-T_{i+3}}{2} = \frac{\sigma+t+\sigma-t}{2}$
(vergleiche auch Formel im Algorithmus von Christian).

7.1.3 Logische Uhren

Für viele Zwecke genügt es, wenn Rechner sich untereinander auf eine einheitliche Zeit verständigen. Es ist jedoch nicht unbedingt erforderlich, dass diese Zeit mit der realen, physikalischen Zeit übereinstimmt. Zeitdifferenzen spielen also keine Rolle, sondern nur die Reihenfolge.
Im Zusammenhang mit einer solchen internen Konsistenz entstand das Konzept der logischen Uhren.
Logische Uhren ermöglichen ein „temporal ordering“ d.h. eine zeitliche Reihenfolge von Ereignissen, zu bestimmen. Dazu muss die Uhrenabweichung im Verteilten System 0 bzw. mi-

nimal sein. Den Grundstein hierfür legte Lamport 1978 in seiner klassischen Arbeit. In dieser Arbeit definierte Lamport für 2 Ereignisse a und b eine Relation „vor“, in Zeichen $a \rightarrow b$ („a tritt vor b ein“). Dies bedeutet, dass alle Prozesse darin übereinstimmen, dass zunächst Ereignis a und danach Ereignis b eintritt, wenn

1. a und b Ereignisse desselben Prozesses sind, und a tritt vor b ein, oder

2. a ist ein Ereignis, das dem Senden einer Nachricht durch einen Prozess entspricht, und b ein Ereignis, das dem Empfangen einer Nachricht durch einen anderen Prozess entspricht. (Eine Nachrich kann nicht empfangen werden, bevor sie gesendet wurde.)
   Für die Relation „vor“ gilt:
   Transitivität: $a \rightarrow b$ und $b \rightarrow c$ impliziert $a \rightarrow c$.

   Treten zwei Ereignisse x und y in verschiedenen Prozessen ein, die keine Nachrichten austauschen, so gilt weder $x \rightarrow y$ noch $y \rightarrow x$ und die Prozesse werden als nebenläufig bezeichnet. Man kann dann keine Aussage über deren Reihenfolge machen.
   Es wird nun für ein Ereignis eine Funktion C eingeführt, die dieses Ereignis auf die Zeit des Eintreffens dieses Ereignisses abbildet. Dabei wird die Forderung gestellt, dass wenn $a \rightarrow b$ gilt, auch $C(a) \leq C(b)$ gelten muss.
   Lamports Algorithmus basiert nun auf den folgenden Punkten:

   – Das Empfangen einer Nachricht passiert nach dem Senden,
   – eine Uhrzeit C schreitet stets positiv fort und
   – eine Angleichung von Uhrzeiten geschieht durch das Addieren positiver Werte.

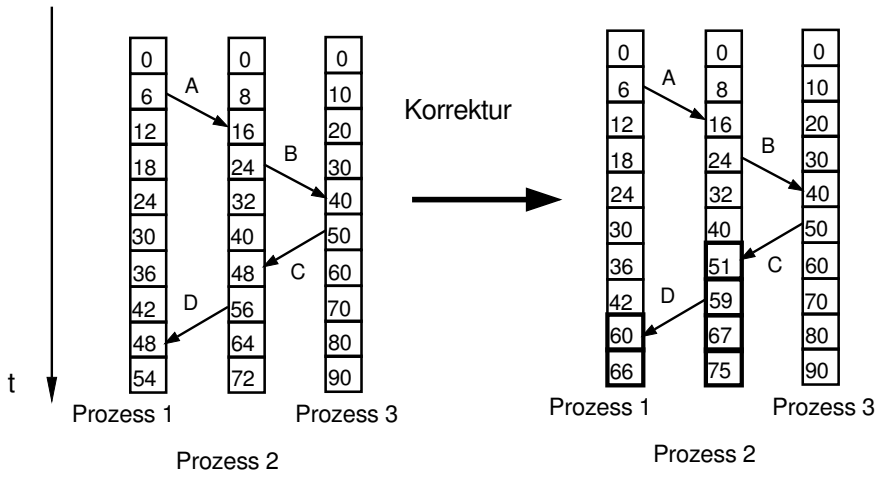


Abbildung 7.9: Uhrenkorrektur nach Lamport

Abbildung 7.9 zeigt in einem Beispiel drei Prozesse, die jeweils eine eigene Uhr besitzen. Die Uhren laufen zudem mit unterschiedlichen Geschwindigkeiten. Man erkennt, dass wenn die Uhr von Prozess 1 den Wert 6 hat, die Uhr von Prozess 2 den Wert 8 und die Uhr von Prozess 3 den Wert 10 hat. Prozess 1 schickt nun eine Nachricht A zum (lokalen) Zeitpunkt 6 an Prozess 2. Diese Nachricht erreicht Prozess 2 zum (lokalen)

Zeitpunkt 16. Man beachte, dass dies dem Zeitpunkt 12 auf dem Rechner von Prozess 1 entspricht. Hier sind also bis zum Eintreffen 6 Zeitintervalle vergangen, während für Prozess 2 schon 8 Zeiteinheiten seit dem Absenden vergangen sind.
Bei näherer Betrachtung von Nachricht C wird das eigentliche Problem offensichtlich: Nachricht C wird zum Zeitpunkt 50 gesendet und kommt zum Zeitpunkt 48 bei Prozess 2 an. Auch Nachricht D kommt bei Prozess 1 früher an, als sie abgesendet wurde. Dies ist jedoch zu vermeiden. Lamports Lösung ist im Prinzip sehr einfach: Wenn die Nachricht C zum Zeitpunkt 50 abgesendet wurde, so darf sie frühestens zum Zeitpunkt 51 ankommen. Trifft nun eine Nachricht bei einem Empfänger ein, dessen Uhr einen kleineren Wert zeigt als der Sendezeitpunkt der Nachricht, so setzt der Empfänger seine Uhr auf den Wert, der um 1 größer als der Sendezeitpunkt ist. Ab diesem Wert läuft die lokale Uhr dann wie vorher weiter.
Wenn dann noch die Bedingung hinzugenommen wird, dass zwischen zwei Ereignissen die Uhr mindestens einen Tick weitergeht, dann ist eine globale Zeit realisierbar.
Mit einer Erweiterung erfüllt der Algorithmus auch ferner Anforderungen an eine total geordnete Zeit.

3. Zwei Ereignisse treten nie zum gleichen Zeitpunkt auf.
   Dies lässt sich zum Beispiel dadurch erreichen, dass man die Prozessnummer des entsprechenden Prozesses an den Eintrittszeitpunkt durch ein Komma getrennt anhängt. Es wird eine künstliche Ordnung eingeführt. Man spricht dann von einer totalen Ordnung, wenn es kein „=“ gibt.

Mittels 1.-3. erhält man eine totale Ordnung aller Ereignisse in einem Verteilten System. Damit sollen die Ausführungen der Uhrensynchronisation abgeschlossen werden. Es stellt sich die Frage, in welchen Anwendungsszenarien eine logisch oder physikalisch synchronisierte Uhrzeit nützlich ist.
Derartige Anwendungsgebiete sollen im Folgenden betrachtet werden.

7.2 Wechselseitiger Ausschluss

Im Folgenden soll zur Frage zurückgekehrt werden, wie Prozesse miteinander kooperieren und wie sie sich synchronisieren lassen. Mehrprozessorsysteme werden häufig mit Hilfe von kritischen Bereichen programmiert. Ein kritischer Bereich ist ein Teil eines Programms, in dem auf Ressourcen (z.B. Speicher) zugegriffen wird, die von mehreren Rechnern oder Prozessen benutzt werden. Ist zu jedem Zeitpunkt gewährleistet, dass nicht mehr als ein Prozess auf dieselben gemeinsam benutzten Ressourcen lesend oder schreibend zugreift, so können insbesondere keine Inkonsistenzen auftreten. Man spricht dann von wechselseitigem Ausschluss. In Einprozessorsystemen werden kritische Bereiche z.B. durch Semaphore und Monitore geschützt. Diese Methoden sind für Verteilte Systeme nicht angemessen. Im Folgenden werden drei Algorithmen angegeben, mit denen man in Verteilten Systemen kritische Bereiche und wechselseitigen Ausschluss realisieren kann.

7.2.1 Der zentrale Algorithmus

Beim zentralen Algorithmus wird ein Einprozessorsystem simuliert. Ein Prozess wird hier zum Koordinator ernannt (z.B. der mit der höchsten Netzwerkadresse). Der Koordinator gewähr-

leistet, dass sich zu jedem Zeitpunkt höchstens ein Prozess in einem bestimmten kritischen Bereich befindet.

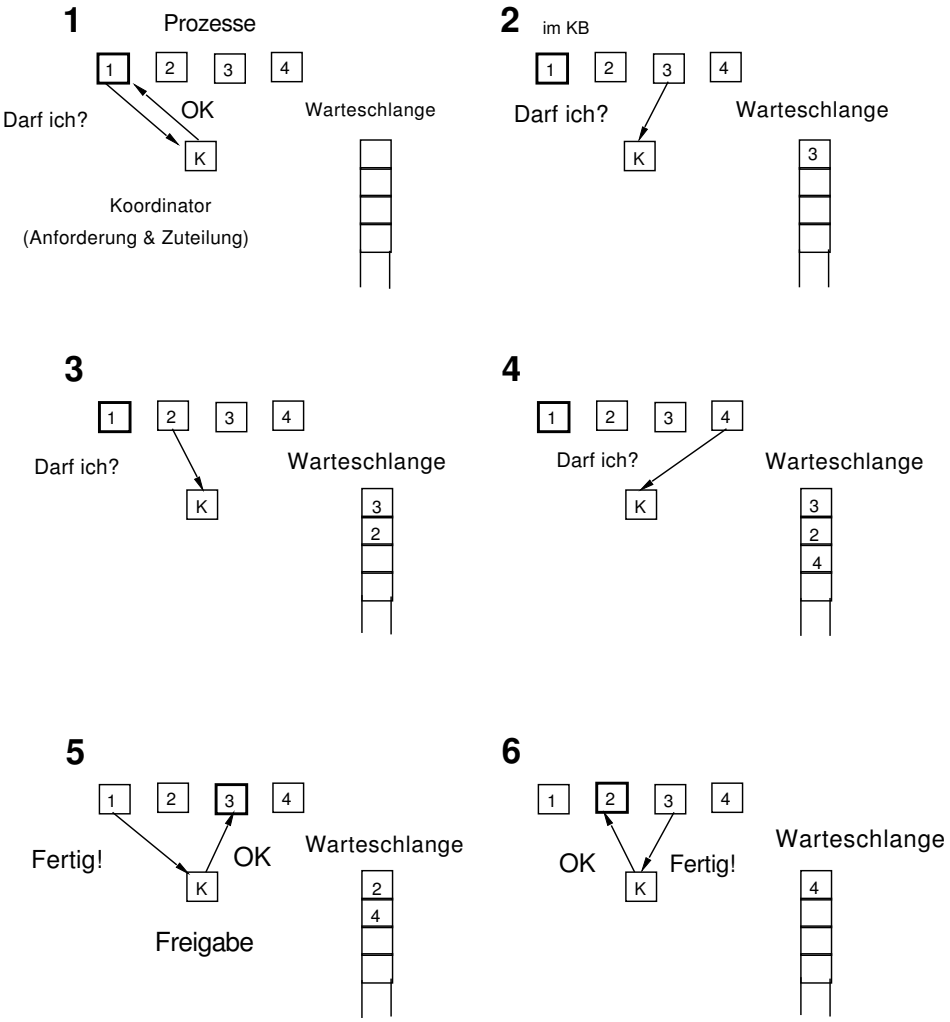


Abbildung 7.10: Zentraler Algorithmus

Der Algorithmus arbeitet wie in Abbildung 7.10 dargestellt. Möchte ein Prozess (Nr. 1) in einen kritischen Bereich eintreten, so stellt er eine Anfrage an den Koordinator. Dieser hat Informationen darüber, ob sich bereits andere Prozesse im kritischen Bereich befinden oder der kritische Bereich - wie im Beispiel - zum Anfragezeitpunkt von keinem anderen Prozess benutzt wird. In letzterem Fall wird vom Koordinator ein OK an die anfragende Einheit gesendet.

Erfolgen weitere Anfragen von Prozessen, die ebenfalls in den kritischen Bereich eintreten wollen (Nr. 3,2,4), so werden die zugehörigen Prozessnummern in der Reihenfolge ihres Anfragens in einer Warteschlange des Koordinators gespeichert. Gibt der benutzende Prozess (Nr. 1) den kritischen Bereich mit einer Fertigmeldung frei, so erhält der am längsten wartende Prozess ein OK und seine Prozessnummer wird aus der Warteschlange entfernt.

Die Vorteile des zentralen Algorithmus sind:

- Fairness: Die Reihenfolge der Anfrage bleibt erhalten, und kein Prozess wartet ewig.
- Die relativ einfache Implementierung der drei Operationen: Anforderung (= darf ich?), Zuteilung (= OK) und Freigabe (= fertig!).

Dem gegenüber gibt es aber auch Nachteile:

- Der Koordinator bildet einen Engpass (keine Skalierbarkeit). (Abhilfe: pro kritischem Bereich ein eigener Koordinator)
- Der Koordinator darf nicht ausfallen.
- Nachricht geht verloren → Algorithmus arbeitet nicht mehr (Prozess verhungert).

Insbesondere die Ausfallsicherheit stellt einen wesentlichen Aspekt dar. Daher hat man nach Alternativen zum zentralen Ansatz gesucht. Weitere Motivation: Wie viele Nachrichten werden über das Netz versendet, wenn jeder Prozess einmal in den kritischen Bereich will? Annahme: n Prozesse. Antwort: lineare Konplexität: 3/cdotn

7.2.2 Der verteilte Algorithmus von Ricart und Agrawala

Voraussetzung hierfür ist eine totale Ordnung der Ereignisse im System, d.h. für jedes Paar von Ereignissen ist die Reihenfolge ihres Auftretens eindeutig bestimmt (dies lässt sich z.B. mit dem Algorithmus von Lamport erreichen).
Das Prinzip: Wenn ein Prozess in den kritischen Bereich eintreten will, so sendet er eine Anfragenachricht mit dem Namen des kritischen Bereichs, seiner Prozessnummer und aktuellen Zeit (total geordnet) an alle Prozesse des verteilen Systems (alle anderen und sich selbst). Das Senden soll zuverlässig sein, d.h. jede Nachricht wird bestätigt. Was passiert nun, wenn eine Komponente des verteilten Systems eine solche Anfrage erhält? Dann sind drei Fälle zu unterscheiden:

1. Der Empfänger befindet sich nicht im kritischen Bereich und will auch nicht in diesen eintreten → sendet OK an den Sender.
2. Der Empfänger befindet sich im kritischen Bereich → sendet kein OK, sondern merkt sich die Anfrage in einer Warteschlange vor.
3. Der Empfänger will selbst in den kritischen Bereich, hat „fast“ Zeitgleich eine eigene Anfrage abgesendet → er vergleicht die Zeitstempel der Nachrichten; der frühere gewinnt: Zeitstempel des anfragenden Prozesses ist kleiner als der eigene: OK senden, sonst: Wartet der Prozess / die Anfrage in einer Queue, bis der entsprechende Prozess mit dem kritischen Brereich fertig ist und in der Queue keine anderen Prozesse vor ihm dran sind.

Der Sender wartet auf alle OKs, dann tritt er in den kritischen Bereich (Kb) ein. Beim Verlassen sendet er OKs an alle Prozesse in seiner Warteschlange und entfernt die Prozesse hieraus, vgl. Abbildung 7.11.

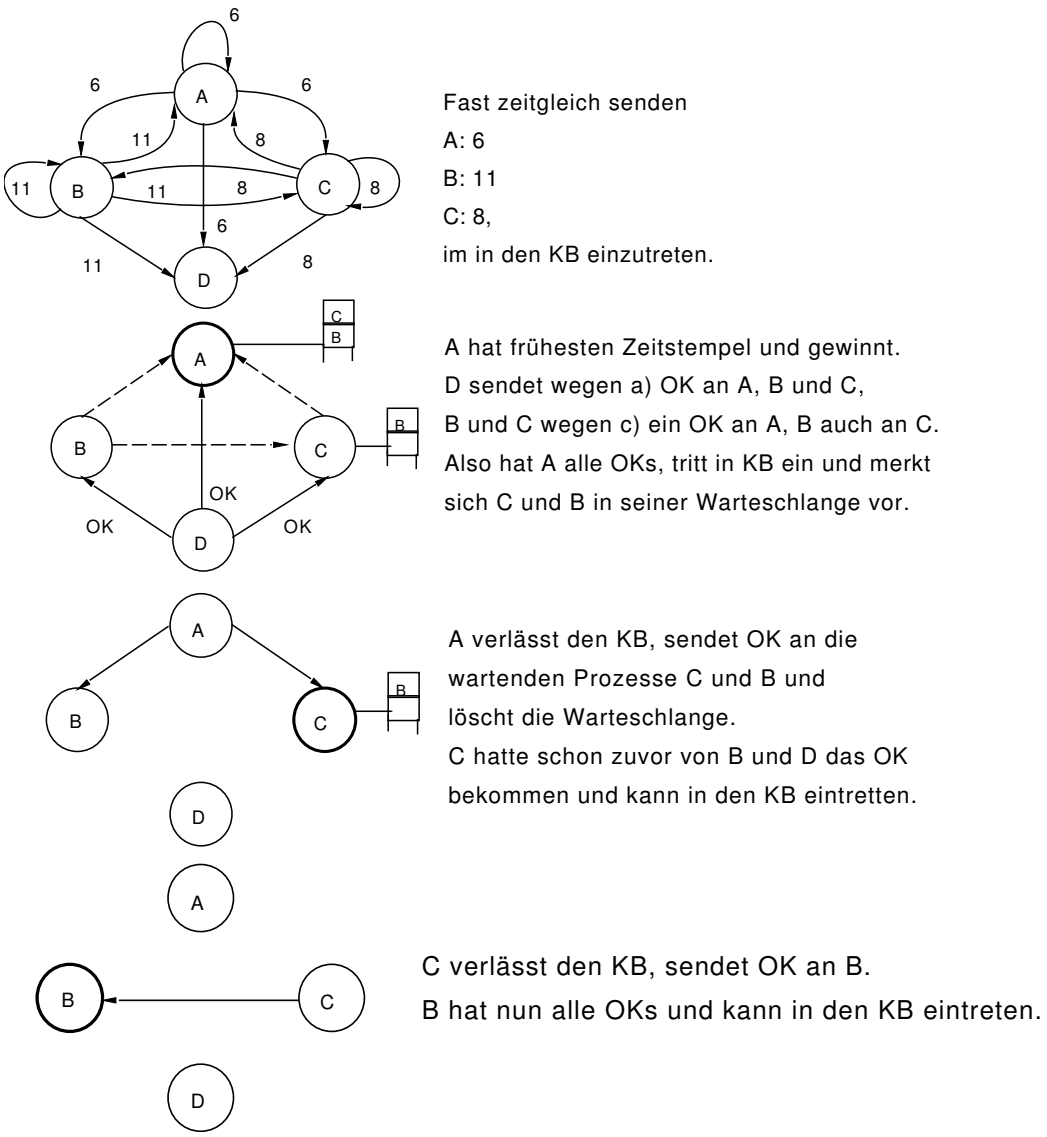


Abbildung 7.11: Das Prinzip des Verteilten Algorithmus von Ricart und Agrawala

Vorteile dieses Algorithmus sind Fairness und Deadlockfreiheit. Allerdings ist die Netzwerkbelastung durch $2(n-1)$ Nachrichten für eine Anfrage bei n-Komponenten im System recht hoch. Problematisch ist hier auch, dass beim Ausfall einer Komponente die Antwort auf eine Anfrage nicht möglich ist, was ggf. als Ablehnung interpretiert wird. Somit werden alle folgenden Versuche davon abgehalten, in den kritischen Bereich einzutreten. Ein Ausweg wäre, dass, falls nach einem Timeout noch keine Antwort eingetroffen ist, erneut eine Anfrage gestellt wird und dann ggf. davon ausgegangen wird, dass der Empfänger ausgefallen ist.
Der Algorithmus ist also zwar verteilt, aber auch langsamer, komplizierter und weniger robust als der zentrale. Robust heißt hier: Wenn eine Komponente ausfällt.
Komplexität: $2 \cdot n \cdot (n-1)$, wenn man nicht an sich selbst Nachrichten schickt. D.h. die KomÂ´plexität ist höher (quadratisch) und damit schlechter als beim zentralen Ansatz.

7.2.3 Ein Token-Ring-Algorithmus

Gegeben seien *n* Prozesse. Auf diesen wird ein logischer Ring aufgebaut, in dem jedem Prozess (z.B. gemäß aufsteigender Netzwerkadresse) eine Position zugewiesen wird. Damit hat jeder Prozess einen Nachfolger (der letzte Prozess hat wieder den ersten als Nachfolger), vgl. Abbildung 7.12.

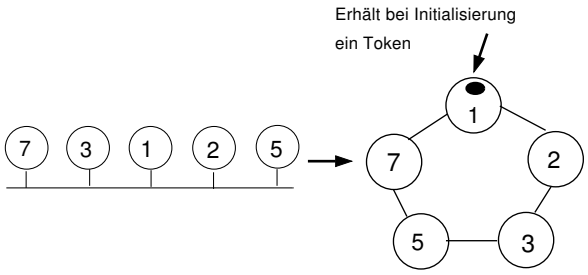


Abbildung 7.12: Token-Ring-Algorithmus

Bei der Initialisierung erhält ein Prozess ein Token. Will dieser Prozess in einen kritischen Bereich eintreten, so hat er (durch das Token) die Erlaubnis dazu. Verlässt er den kritischen Bereich, so gibt er das Token an seinen Nachfolger. Will er nicht eintreten, so gibt er es gleich weiter. Will kein Prozess in einen kritischen Bereich eintreten, so zirkuliert das Token mit hoher Geschwindigkeit im Ring.
Der Algorithmus ist korrekt, da zu jedem Zeitpunkt nur ein Prozess das Token besitzt und somit auch nur dieser Prozess in den kritischen Bereich eintreten kann. Durch die Zirkulation ist gewährleistet, dass jeder Prozess nur endlich lange auf den Eintritt in einen kritischen Bereich warten muss (bei unendlicher Wartezeit spricht man auch vom *starvation Problem)*.
Probleme bei diesem Algorithmus sind der Verlust des Tokens und der Ausfall eines Prozesses. Insbesondere das Entdecken des Tokenverlustes ist schwierig, da die Zeit zwischen zwei Umläufen des Tokens auf dem Netzwerk nicht begrenzt ist. Auch wenn kein Prozess in den kritischen Bereich möchte, entsteht Netzlast. Komplexität: n
Beim Vergleich der drei Algorithmen erweist sich der zentrale Algorithmus als am günstigsten. Dieser erfordet jedoch die Benennung eines Koordinators. Diese Auswahl wird durch bestimmte Regeln (z.B. höchste Netzwerkadresse) oder so genannte Wahlalgorithmen getroffen.

7.3 Wahlalgorithmen (Election Algorithm)

In Verteilten Systemen mit **gleichberechtigten Komponenten** erfordern viele verteilte Algorithmen einen ausgewählten Rechner, der als **Initiator oder Koordinator** arbeitet oder auch **eine spezielle Rolle** übernimmt. Im Prinzip ist es egal, welche Komponente diese ausgezeichnete Rolle übernimmt - wichtig ist nur, dass **genau eine** dies tut.
Das **Grundprinzip** besteht nun darin, jeder Komponente bzw. **jedem Prozess eine eindeutige Nummer** zu geben, **z.B. die Netzadresse** und dann den Prozess oder Rechner mit der höchsten Nummer als Koordinator auszuwählen.

Voraussetzung: Dabei wird **angenommen**, dass jede Komponente **ihre eigene Nummer kennt und die Nummern jeder anderen Komponente.**

Problem: Was die Komponenten jedoch nicht wissen, ist, welche arbeiten und welche nicht erreichbar sind.

Das **Ziel** eines Wahlalgorithmus (Election Algorithm) besteht darin, **die aktuell verfügbare Komponente mit der höchsten Nummer auszuwählen und** dafür zu sorgen, dass alle Komponenten **über die Auswahl dieses Koordinator informiert werden**.

7.3.1 Der Bully-Algorithmus

Dieser Algorithmus (Bully = Schläger, Rüpel, jmd. Einschüchtern) wurde von Garcia-Molina (1982) entwickelt.

Stellt eine Komponente P fest, dass ein benötigter Koordinator nicht verfügbar ist, so initiiert diese eine Wahl wie folgt:

Stufe 1:

1. P sendet eine **ELECTION-Nachricht an alle Prozesse mit höheren Nummern als der eigenen**

2. Antwortet **niemand**, so gewinnt P die Wahl und wird selbst Koordinator

3. Antwortet irgendeine der anderen Komponenten, so ist P's Arbeit erledigt.

Stufe 2: Aktivität geht zum Empfänger über: Zu jedem Zeitpunkt kann ein beliebiger Prozess eine ELECTION-Nachricht von den Komponenten mit niedrigerer Nummer erhalten. Dann sendet die Komponente ein **OK zurück** und zeigt damit an, dass sie arbeitet. **Der Empfänger initiiert dann seinerseits wieder einen Wahl-Algorithmus bis alle Komponenten** bis auf genau eine aufgegeben haben. **Diese eine ist dann der neue Koordinator**. D.h. in diesen zwei Stufen ist der Koordinator gewählt.

Stufe 3: Dieser neue Koordinator teilt seine Wahl dann unmittelbar allen anderen Komponenten mit.

Ggf. später: Falls eine höher nummerierte Komponente wieder ins System zurückkommt, initiiert diese eine Wahl. In der Regel gewinnt diese Komponente die Wahl und schlägt damit den aktuellen Koordinator (daher der Name „Bully").

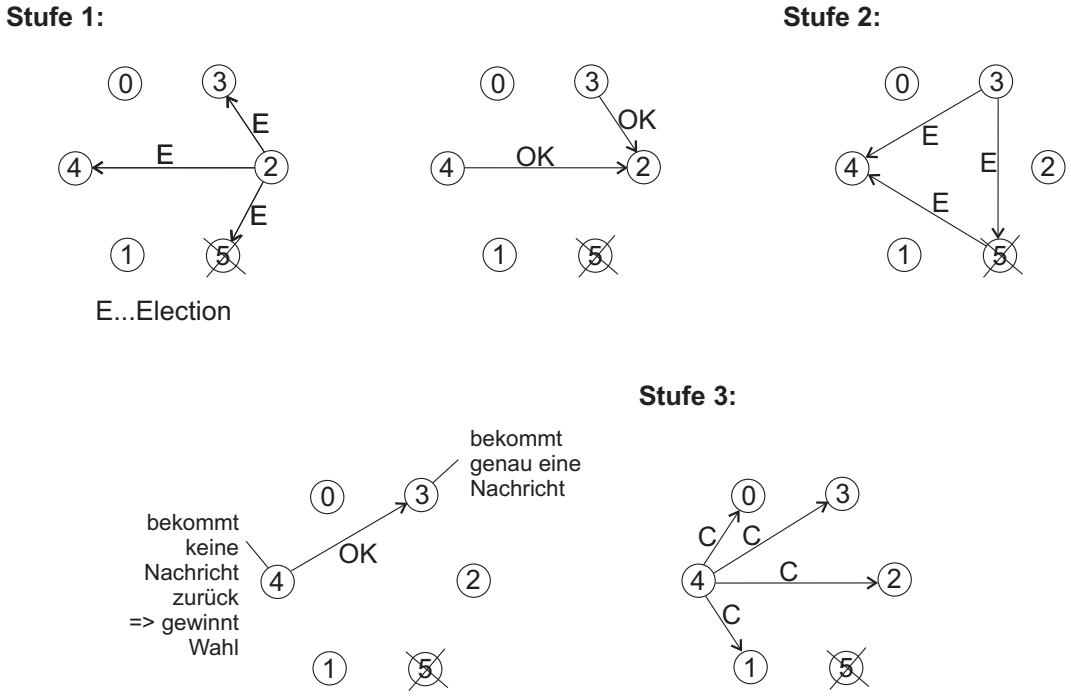


Abbildung 7.13: Ablauf des Bully-Algorithmus

Der Algorithmus ist verteilt, auch wenn er von einer einzelnen Stelle angestoßen werden kann.

7.3.2 Ein Ring-Algorithmus

Es wird **angenommen**, dass **alle Komponenten logisch oder physisch geordnet sind, so dass** jede genau einen Nachfolger hat. Allerdings wird bei diesem Algorithmus **kein Token** verwendet.

Stellt eine Komponente fest, dass der Koordinator nicht funktioniert, so generiert sie eine **ELECTION-Nachricht**, die ihre **eigene Nummer enthält** und **sendet diese zu ihrem Nachfolger**. Sofern diese nicht arbeiten sollte, wird der nächste Nachfolger genommen oder der danach, bis eine laufende Komponente gefunden wurde. **Diese nächste funktionierende Komponente fügt ihre eigene Nummer der Liste hinzu und gibt die Nachricht weiter, bis diese Nachricht die ursprüngliche Komponente wieder erreicht**, indem sie einmal durch den Ring rotiert ist. Dann wird die **ELECTION-NACHRICHT in** eine **COORDINATOR-Nachricht transformiert**, indem sie die höchste Nummer aus der Liste als neuen Koordinator angibt und rotiert nochmals durch den Ring. Nach der erneuten Rotation wird die Nachricht gelöscht.

Sollten **zwei Komponenten quasi gleichzeitig** eine ELECTION-Nachricht generieren, so würde dieser Algorithmus **nicht** zu **Inkonsistenzen** führen. In diesem Fall liegt lediglich eine gewisse **Redundanz** hinsichtlich der Netz- und Arbeitslast der Komponenten vor.

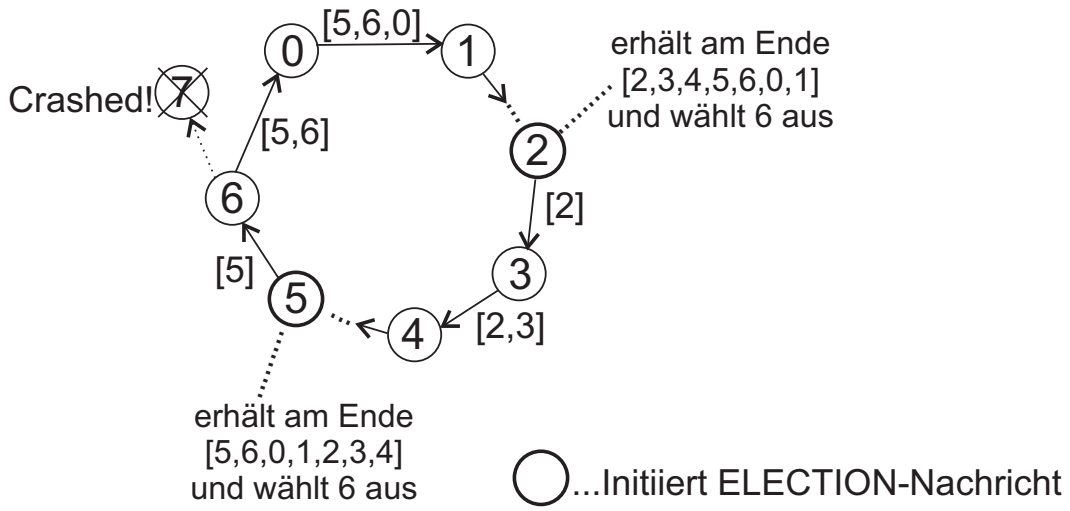


Abbildung 7.14: Ablauf des Ring-Algorithmus

7.4 Deadlocks in Verteilten Systemen

Deadlocks in Verteilten Systemen ähneln denen in Einprozessorsystemen, sind aber schlimmer! Zunächst wird die zentrale Deadlock-Erkennung in Verteilten Systemen behandelt. Wird

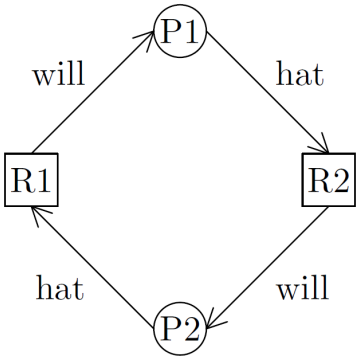


Abbildung 7.15: Wdh.: Deadlock

ein Deadlock in einem Verteilten System erkannt, das auf atomaren Transaktionen aufbaut, so wird der Deadlock durch den Abbruch der Transaktion behoben.

7.4.1 Zentrale Deadlock-Erkennung

Das Prinzip der zentralen Deadlock-Erkennung führt auf die zentrale Methode zurück. Dabei verwaltet jeder Rechner einen Betriebsmittelgraphen für seine Prozesse und Betriebsmittel. Zusätzlich verwaltet ein Koordinator den Betriebsmittelgraphen für das Gesamtsystem (als Vereinfachung aller Einzelgraphen). Entdeckt der Koordinator einen Zyklus, so bricht er zur Auflösung des Zyklus einen Prozess ab. Im Unterschied zu einem zentralen System müssen

hier die einzelnen Komponenten ihre Betriebsmittelgraphen explizit an den Koordinator senden, vgl. Abbildung 7.16.

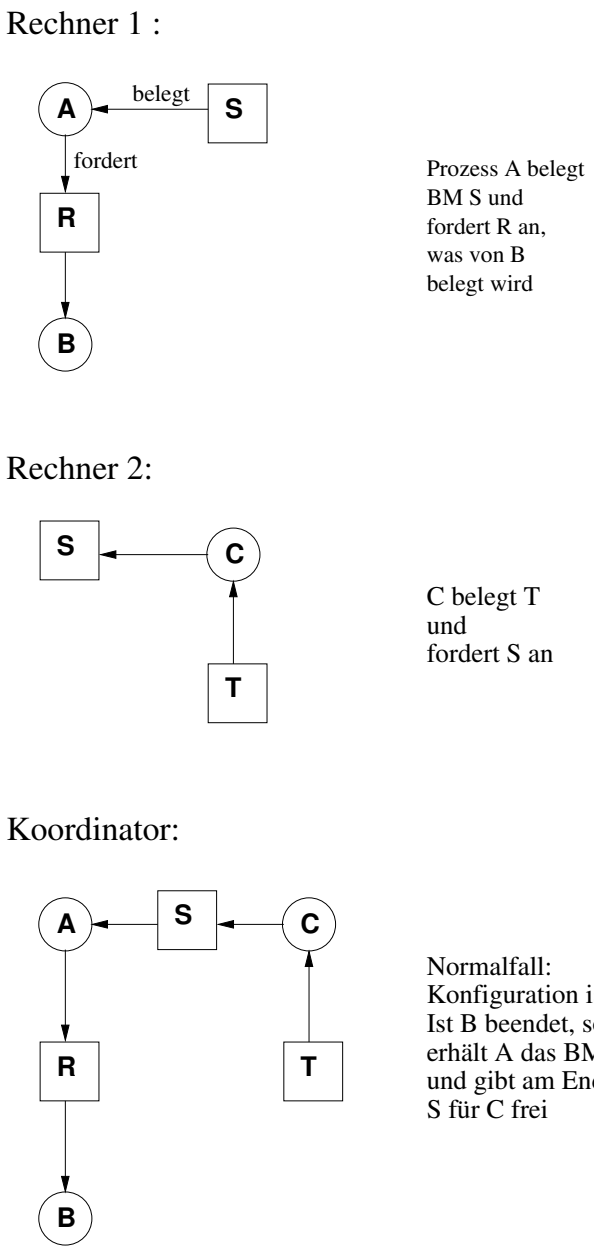


Abbildung 7.16: Zentrale Deadlock-Erkennung

Dabei besteht ein Problem: B gibt R frei und fordert T an, aber beim Koordinator kommen die Nachrichten vom Rechner 1 (Freigabe von BM R) und Rechner 2 (Warten auf BM T) in verkehrter Reihenfolge an, so dass ein **scheinbarer Deadlock** entsteht, vgl. Abbildung 7.17 Genauso kann es passieren, dass ein Deadlock nicht erkannt wird.
Ein möglicher Ausweg besteht in folgendem: Mittels des Algorithmus von Lamport kann man eine globale Zeit bereitstellen und jeder Nachricht einen Zeitstempel geben.

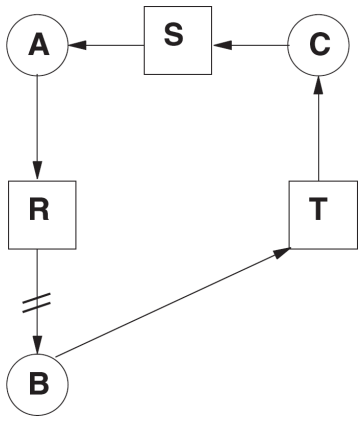


Abbildung 7.17: scheinbarer Deadlock

7.4.2 Verteilte Deadlock-Erkennung

Zur verteilten Deadlock-Erkennung existieren verschiedene Algorithmen. Interessant ist dabei vor allem der Chandy-Misra-Haas-Algorithmus. Dieser Algorithmus hat den Vorteil, dass es den Prozessen gestattet ist, mehrere Betriebsmittel auf einmal statt nacheinander anzufordern. Abbildung 7.18 veranschaulicht beispielhaft das Prinzip.

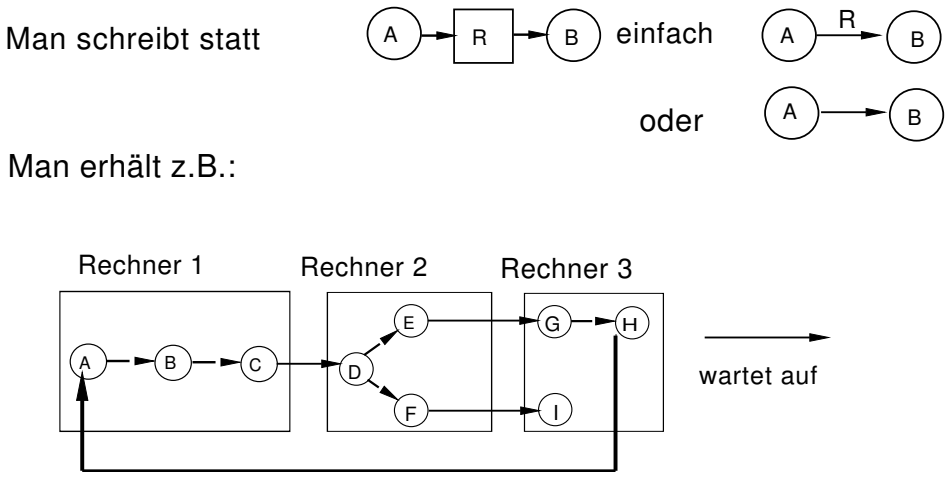


Abbildung 7.18: Verteilte Deadlock - Erkennung

Das Prinzip dieses Algorithmus besteht darin, dass ein Aufruf erfolgt, wenn ein Prozess auf ein BM wartet, das von einem anderen Prozess belegt wird.
Braucht beispielsweise ein Prozess A ein Betriebsmittel von Prozess B, so wird eine so genannte Untersuchungsnachricht erzeugt und an alle Prozesse gesendet, die dieses Betriebsmittel belegen. Diese Nachricht besteht aus:

- dem Bezeichner des wartenden Prozesses (Initiator),
- dem Bezeichner des sendenden Prozesses (Absender),

– dem Bezeichner des Prozesses, an den die Nachricht gesendet wird (Empfänger),

also z.B. (A,A,B), siehe Abbildung 7.19.
Trifft die Nachricht ein, so überprüft der Empfänger, ob er auf das gleiche Betriebsmittel wie der Sender wartet. Ist dies nicht der Fall, so werden Sender- und Empfängerfelder ersetzt, das erste Feld bleibt.
Kommt die Nachricht zum ürsprunglichen Empfänger zurück (wird dadurch erkannt, dass die erste Komponente der Nachricht gleich der letzten Komponente ist), dann existiert ein Zyklus, und das System befindet sich in einer Deadlock-Situation.

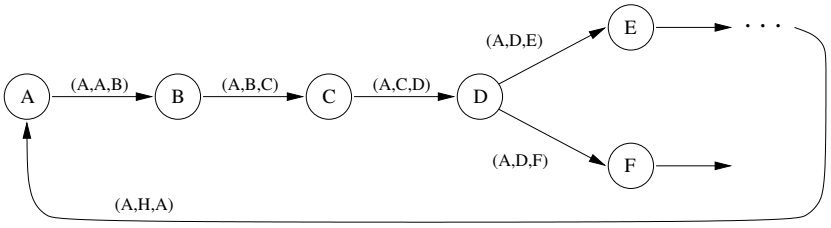


Abbildung 7.19: Prinzip von Chandy et al.

Zur Auflösung gibt es verschiedene Möglichkeiten. Die einfachste ist, dass der initiierende Prozess „Selbstmord“ begeht. Dies kann allerdings Probleme mit sich bringen, falls evtl. zwei Zyklen existieren. Alternativ hierzu könnte jeder Prozess seine eigene Nummer an die Untersuchungsnachricht anhängen, dann könnte der Prozess mit der höchsten Nummer beginnen.
Eine Alternative zur Deadlock-Erkennung ist die Deadlock-Vermeidung. Hier konstruiert man die Systeme von vornherein so, dass Deadlocks nicht möglich sind, z.B. wenn die Zyklusnachricht bei A ankommt, dann wird B dies mitgeteilt und B fordert dann die Resource, die A hat, nicht an.

7.5 At-most-once-Nachrichtenzustellung

Oft ist es nötig, Nachrichten nicht versehentlich mehr als einmal zuzustellen. Diese Gefahr resultiert aus der Behandlung von Fehlern bei der Kommunikation Verteilter Systeme.

7.5.1 Ursachen für Mehrfachzustellung

Im Folgenden sollen Fehlerquellen bei entfernten Prozeduraufrufen betrachtet werden. Dabei unterscheidet man fünf Klassen:

1. Client kann Server nicht lokalisieren
   z.B. weil kein passender Server ausgeführt wird (Versionsproblem o.Ä.).
   Lösungsmöglichkeiten:
   – Fehlertyp anzeigen lassen und reagieren oder
   – Ausnahmebehandlung auslösen
2. Anfragenachricht von Client an Server geht verloren
   Ist einfach zu lösen: Kern wiederholt Anfrage nach einem Timeout.

3. Antwort von Server an Client geht verloren
   Ist schwieriger als Fall 2: Man unterscheidet **idempotente Nachrichten** und nicht idempotente. Eine idempotente Nachricht kann beliebig oft wiederholt werden, ohne dass sich das Ergebnis ändert, etwa das Lesen der ersten 1024 Bytes einer Datei. Das Ergebnis einer nicht idempotenten Nachricht ist von der Anzahl des Eintreffens einer Nachricht abhängig, z.B. das Überweisen von 100 Euro auf ein Konto.
   Es dürfen nur idempotente Nachrichten wiederholt werden, d.h. man muss eine Sequenznummer beim Senden einfügen oder zwischen Original und Wiederholung unterscheiden.

4. Der Server fällt nach Erhalt einer Anfrage aus
   Hierbei müssen drei Fehler unterschieden werden, die auch in Abbildung 7.20 veranschaulicht werden.

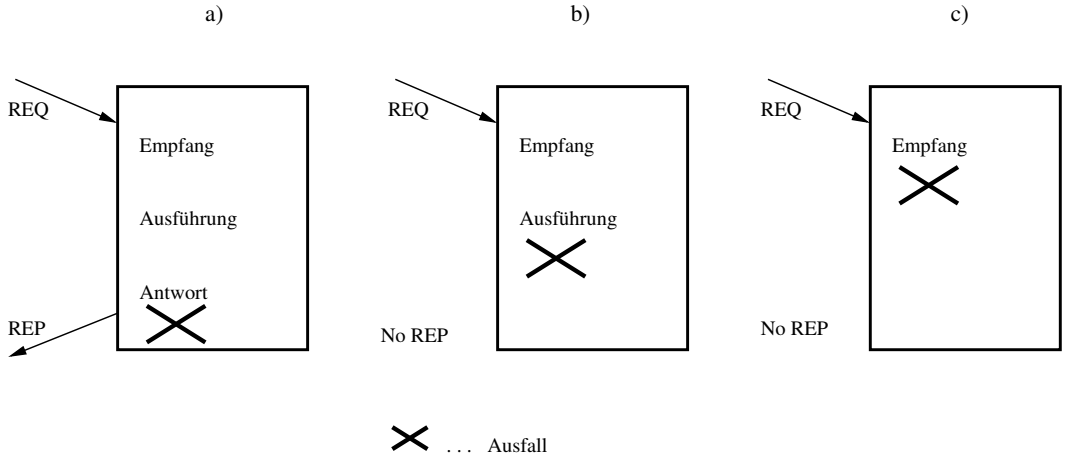


Abbildung 7.20: Serverausfall nach Erhalt einer Anfrage

Im einzelnen beinhalten diese drei Fehler folgende Probleme:

- die Sequenz Empfang, Ausführung, Antwort ist der Normalfall,
- nach der Ausführung folgt ein Ausfall
- nach dem Empfang folgt der Ausfall

ABER: Wie soll der Client b) und c) unterscheiden?

5. Der Client fällt aus, nachdem eine Anfrage gestellt wird
   Dann wird die Berechnung ausgeführt, obwohl niemand auf eine Antwort wartet („verwaiste Berechnung“). Dies verschwendet Rechnerzeit und kann zu Konfusionen führen, wenn der Client danach eine neue Anfrage startet und daraufhin das alte Ergebnis eintrifft.

7.5.2 Lösung basierend auf synchronen Uhren

Im Folgenden soll davon ausgegangen werden, dass in den vergangenen Jahren die für eine Uhrensynchronisation erforderliche Hard- und Software geschaffen wurde.
Mit dieser Technologie ist es möglich, internetweit Millionen von Uhren innerhalb weniger Millisekunden mit der UTC zu synchronisieren.
Der traditionelle Ansatz der At-most-once-Nachrichtenzustellung (At-most-once message delivery) geht wie folgt: Für jede Nachricht wird ein eindeutiger Nachrichtenidentifikator vergeben. Jeder Server, der Nachrichten empfängt, speichert alle Identifikatoren von erhaltenen Nachrichten, so dass neue Nachrichten von Wiederholungen unterschieden werden können.
Stürzt der Server ab und wird wiedergestartet,so geht die Tabelle mit den Nachrichtenidentifikatoren verloren. Andererseits stellt sich die Frage, wie lange solche Daten gespeichert werden sollen.
Basierend auf abrufbaren synchronisierten Uhrzeiten kann dieser Algorithmus wie folgt modifiziert werden. Jede Nachricht erhält vom Sender einen Verbindungsidentifikator und einen Zeitstempel. Der Server speichert in einer Tabelle für jede Verbindung den neuesten Zeitstempel. Falls irgendeine ankommende Nachricht für eine Verbindung einen älteren Zeitstempel als gespeichert aufweist, wird diese Nachricht als Duplikat abgewiesen.

7.6 Atomare Transaktionen

Motivation: Ein Konzept, das sehr eng an den Wechselseitigen Ausschluss angelehnt ist, ist das der Transaktionen.
In beiden Fällen wird eine gemeinsam genutzte Ressource gegen den gleichzeitigen Zugriff durch verschiedene nebenläufige Prozesse geschützt.
Die bisher kennengelernten Synchronisationsmechanismen befinden sich auf einem semantisch niedrigen Niveau, d.h. der Programmierer ist gezwungen, sich mit Details wie wechselseitigem Ausschluss, Deadlockvermeidung u.Ä. auseinandersetzen. Das Ziel ist aber eine höhere Abstraktionsstufe, die technische Details verbirgt. Eine solche Abstraktion wird in Verteilten Systemen häufig eingesetzt und als (atomare) Transaktion bezeichnet.
Das Prinzip: Zwischen zwei Parteien wird - wie bei einem Vertrag - eine Vereinbarung ausgehandelt, wer welche Leistungen zu erbringen hat. Diese Vereinbarung ist atomar und kann nur ganz oder gar nicht ausgeführt werden.

Beispiel: Ein Kaufhaus hat auf einem Magnetband den Bestand vom Vortag und auf einem zweiten Magnetband die Ein- und Verkäufe vom Tage gespeichert. Ein Computer soll auf einem dritten Band den neuen Bestand berechnen. Falls ein Fehler auftritt, so werden alle Bänder zurückgespult und alles wiederholt.
Der Vorteil dieser Vorgehensweise liegt darin, dass beispielsweise bei einem Bankprogramm, welches Geld überweist (d.h. erst einen Betrag von einem Konto A abhebt und danach den gleichen Betrag auf Konto B gutschreibt), ein Systemabsturz dazu führen kann, dass sich das Geld nicht „in Luft auflöst“.
Für Transaktionen verwendet man folgendes Modell: Das Verteilte System bestehe aus n unabhängigen Prozessen, die zufällig ausfallen können. Unsichere Kommunikation werde von der Software auf unteren Schichten behoben.
Ferner wird vorausgesetzt:

– **Stabiler Speicher**: Da ein RAM-Inhalt bei Stromausfall verloren gehen und ein Plattenspeicher bei einem Hardwaredefekt zu Problemen führen kann, erhält man einen stabilen Speicher z.B. durch redundante Abspeicherung auf Platten (z.B. doppelt mit Prüfsumme; bei unterschiedlichen Platteninformationen wird die Platte mit korrekter Prüfsumme als richtig angenommen und die andere mit diesen Informationen überschrieben).

– **Transaktionsprimitive** müssen entweder vom Betriebssystem oder vom Laufzeitsystem einer Sprache zur Verfügung gestellt werden, z.B.:

  - BEGIN_TRANSAKTION
  - END_TRANSACTION
  - ABORT_TRANSACTION
  - READ, WRITE,...

  Beispiel: Flugbuchung

  1. BEGIN_TRANSACTION {Flugbuchung Düsseldorf-St.Louis}
  2. reserve Düsseldorf-Frankfurt
  3. reserve Frankfurt-Chicago
  4. reserve Chicago-St.Louis
  5. END_TRANSACTION

  Ist der Flug Chicago-St.Louis ausgebucht, so folgt ABORT_TRANSACTION und alle Reservierungen werden storniert.

Transaktionen besitzen vier Eigenschaften. Sie sind

1. **Atomar**: Nach außen scheinen Transaktionen unteilbar.
2. **Consistent**: Transaktionen verletzen keine Systeminvarianten.
3. **Isolated**: Nebenläufige Transaktionen beeinflussen sich nicht gegenseitig.
4. **Durable**: Falls eine Transaktion ausgeführt wird, sind alle Änderungen dauerhaft.

Diese Eigenschaften werden auch als **ACID-Eigenschaften** zusammengefasst.
Die im Beispiel betrachtete Transaktion war eine Folge von Operationen, welche die ACID-Eigenschaften besaßen. Vom Typ her handelte es sich um den einfachsten Typ, die sogenannte **flache Transaktion (flat transaction)**.
Nachteilig ist bei flachen Transaktionen, dass dabei keine Teilergebnisse bestätigt oder wieder rückgängig gemacht werden können. Damit kann in bestimmt Anwendungen die Atomarität auch wieder ein Nachteil sein.
Aus diesem Grund wurden so genannte **geschachtelte Transaktionen (nested transactions) eingeführt**.
Von einer top-level-Transaktion kann dabei eine Verzweigung zu Kind-Transaktionen erfolgen, die parallel zueinander ggf. auch auf verschiedenen Rechnern abgearbeitet werden können.

Diese können rekursiv wiederum Kind-Transaktionen enthalten. Wichtig aus Sicht der Implementierung ist hierbei, dass im Fall des Abbruchs einer top-level-Transaktion auch alle Kind-Transaktionen abgebrochen werden.
Ferner gibt es **verteilte Transaktionen (distributed transactions)**, die sich uns in einer Feinheit von den geschachtelten Transaktionen unterscheiden. Sie weisen nicht notwendigerweise Hierarchien auf und bestehen aus einem gemeinsamen aber verteilten Zugriff auf eine Datenbank.

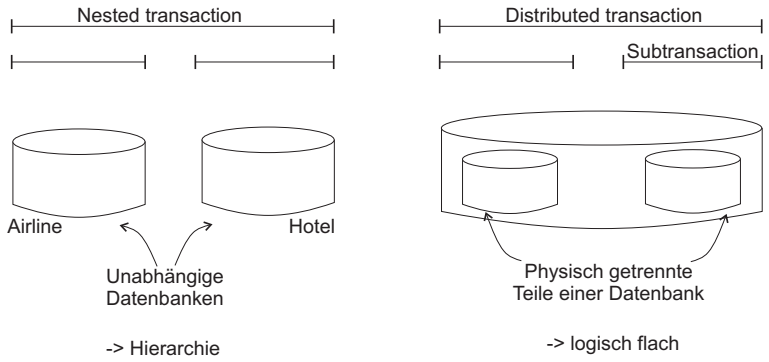


Abbildung 7.21: Vergleich verschachtelte und verteilte Transaktionen

8

Sicherheit in Verteilten Systemen

►

Inhaltsangabe

Es gibt verschiedene Varianten, sich der Sicherheitsproblematik zu nähern. Wir wollen Sicherheit zunächst in ihrer breitesten Form als Vertrauenswürdigkeit und Schutz der Interessen eines Nutzers betrachten. Die verschiedenen Aspekte der Sicherheit in Verteilten Systemen sind in Abbildung 8.1 dargestellt.
„Etwas ist sicher, wenn der Aufwand, es zu stehlen/manipulieren, größer als der Wert ist“
Sicherheit in Verteilten Systemen ist eines der schwierigsten Themen, da entspechende Mechanismen sich lückenlos durch das gesamte System hindurchziehen müssen.

Sicherheit in Verteilten Systemen kann grob in drei Bereiche eingeteilt werden:

1. **Kommunikation zwischen Nutzern oder Prozessen (Secure Channels)**, die sich ggf. auf unterschiedlichen Rechnern befinden und

2. **Zugriffskontrolle (Access Control)**, d.h. Sicherstellung, dass ein Prozess nur die Zugriffsrechte für eine Ressource im Verteilten System bekommt, die ihm zustehen. Insbesondere bei mobilem Code kommt dieser Thematik eine große Bedeutung zu.

   Ferner ist von Bedeutung:

3. **Sicherheitsmanagement (Security Management)**, d.h. Mechanismen für die Handhabung kryptographischer Schlüssel, Mechanismen für das Hinzufügen und Entfernen von Nutzern eines Systems, etc.

Sicherheit Verteilter Systeme ist stets verbunden mit dem Begriff der **Zuverlässigkeit** (Dependability). Ein zuverlässiges Computersystem ist ein System, dem man zu Recht vertrauen kann, einen Dienst auszuführen.
Zuverlässigkeit beinhaltet dabei:

– **Verfügbarkeit** (Availability)
  Das System ist bereit, um sofort genutzt zu werden. Ein hochverfügbares System ist ein System, das quasi zu jedem Zeitpunkt seine Funktionalität ausführen kann.

– **Verlässlichkeit** (Reliability)
  Das System kann kontinuierlich ohne Fehler laufen. Im Gegensatz zur Verfügbarkeit wird hierbei zeitlich gesehen kein Augenblick, sondern ein Intervall betrachtet. Ein hochzuverlässiges System kann über eine recht lange Zeitdauer ohne Unterbrechungen arbeiten.
  Beispiel: Ein System, das im Schnitt eine Millisekunde pro Stunde ausfällt, hat eine Verfügbarkeit von 99,9999% und ist nicht hochzuverlässig.
  Im Gegensatz dazu hätte ein nie ausfallendes System, das jedes Jahr im August 2 Wochen gewartet wird, nur eine Verfügbarkeit von 96%, kann aber als hochzuverlässig bezeichnet werden.

– **Sicherheit** (Safety)
  Ein System heißt sicher, wenn bei einem temporären Ausfall kein katastrophaler Fehler eintreten kann. So erfordert bspw. Software von Atomkraftwerken oder Flugzeugen hohe Sicherheitsmechanismen.

– **Instandhaltung** (Maintainability)
  Diese Anforderung bedingt, dass ein fehlerhaftes System einfach repariert werden kann.

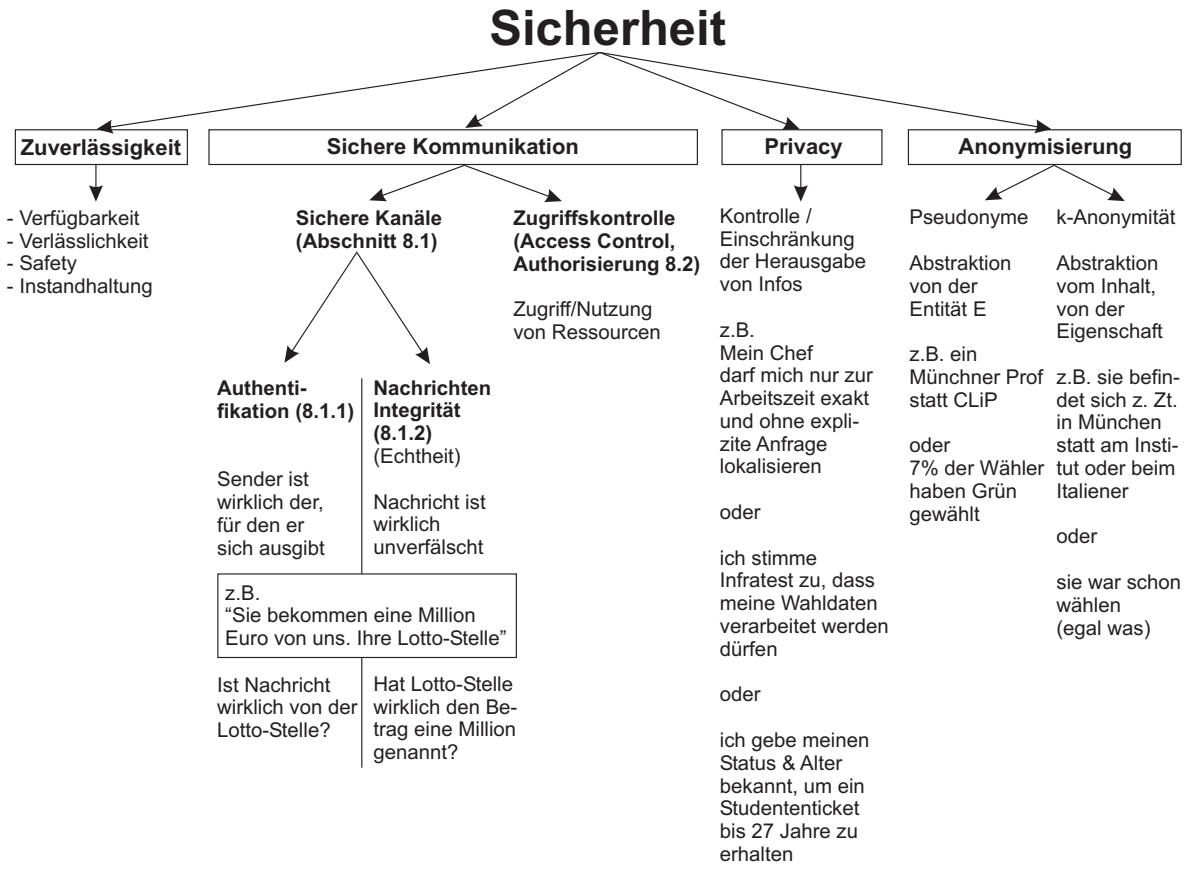


Abbildung 8.1: Aspekte von Sicherheit

Ein hoch instandhaltbares System impliziert in der Regel auch eine recht hohe Verfügbarkeit, da Fehler oft automatisch erkannt und repariert werden können. Aber: Dies ist leichter gesagt als getan!
Hinsichtlich der Vertrauenswürdigkeit in ein Computersystem (Trust in a Computer System) sind noch zwei weitere Eigenschaften von Bedeutung:

– **Vertraulichkeit** (Confidentiality)
Informationen sind nur autorisierten Parteien zugänglich. D.h. es liegt eine Liste vor, worauf steht, für wen die Informationen zugänglich sind.

– **Echtheit** (Integrity)
Systemänderungen können nur durch erlaubte Vorgehensweisen erfolgen, d.h. unerwünschte Modifikationen sind erkennbar und können rückgängig gemacht werden (Unverfälschtheit). Dies betrifft sowohl Hard- und Software als auch Daten.

Zu diesem Zweck kann die Authentifizierung genutz werden: Dabei wird überprüft, ob der Ausweis mit der Person übereinstimmt.
Eine andere Herangehensweise, die Sicherheit eines Verteilten Systems zu gewährleisten, besteht darin, dieses gegen Sicherheitsbedrohungen zu schützen. Dabei gibt es 4 Typen:

1. Abhören (Interception)
2. Außerbetriebsetzen (Interruption)
3. Verfälschen (Modification)
4. Einschleusen (Fabrication)

Weitere wichtige Aspekte sind die Folgenden:
Anonymisierung: Meine Handlungen sollen nach außen nicht nachvollziehbar sein. Entitäten werden beim Dienstaufruf (z.B. Wahl, LBS) mit einem Inhalt/Kontext korrelliert.
Privacy: Schutz der Privatsphäre. Kontrolle bei der Herausgabe von Informationen $\rightarrow$ bewußte Entscheidung dafür/dagegen, z.B. mit Policys. Wir wollen nun zu den beiden Hauptbereichen bei der Sicherheit zurückkommen.

8.1 Secure Channels

An dieser Stelle soll die klassische Client/Server-Architektur betrachtet werden und ein sicherer Kanal zwischen den Kommunikationspartnern angestrebt werden, der Sender und Empfänger gegen das Abhören, Verfälschen und Einschleusen von Nachrichten schützt.

8.1.1 Authentifizierung (Authentication)

Authentifizierung, d.h. das Absichern der Echtheit einer Kommunikationspartei steht immer in direktem Zusammenhang mit der Message Integrity, d.h. der Nachrichtenechtheit.

Beispiel:
Auch wenn Bob sich sicher ist, dass Alice der Sender einer Nachricht m ist, so ist es dennoch nötig, sicher zu sein, dass m nicht während der Übertragung manipuliert wurde. Denn was nützt sonst die Kenntnis, dass Alice das Original der Nachricht m gesendet hat.

Umgekehrt steht die Nachrichtenintegrität auch immer im Zusammenhang mit der Authentifizierung.

Beispiel:
Angenommen, Bob erhält eine Nachricht, dass er 1 Million Euro im Lotto gewonnen hat, so nützt ihm diese Nachricht nicht, wenn er nicht verifizieren kann, dass diese von den Betreibern der Lotterie abgeschickt wurde.

Prinzip der Authentifikation: Alice und Bob wollen kommunizieren, wobei angenommen werde, dass Alice den Aufbau eines Kanals initiiert. Alice beginnt mit dem Senden einer Nachricht an Bob oder an eine vertrauenswürdige dritte Partei. Ist der Kanal aufgebaut, so kann Alice sicher sein, dass sie mit Bob kommuniziert und Bob ist sicher, dass er mit Alice kommuniziert. Dann kann der eigentliche Nachrichtenaustausch erfolgen.
Dabei wird in der Regel eine Secret-Key-Kryptographie mittels Session Keys durchgeführt, d.h. es werden gemeinsam genutzte Schlüssel verwendet, die in der Regel so lange verwendet werden können, wie der Kanal existiert.
Es gibt zwei Stufen der Authentifizierung:

- Stufe I Schlüssel werden für Echgtheitsprüfung des Absenders eingesetzt.
- Stife II Schlüssel werden für Verschlüsselung von Nachrichten verwendet: 2 Grundprinzipien:
  - jeder hat mit jedem einen Schlüssel: $\frac{n\cdot(n-1)}{2}$ Schlüssel.
  - jder hat mit „KDC“ einen Schlüssel: n Schlüssel

Prämisse: Je öfter und je „länger“ Schlüssel verwenden werden, desto einfacher lassen sie sich entschlüsseln.
Im Folgenden sollen konkrete Protokolle betrachtet werden:

Authentifizierung basierend auf einem Shared Secret Key Es soll angenommen werden, dass sowohl Alice als auch Bob auf einem sicheren Weg einen gemeinsam zu nutzenden Schlüssel $K_{A,B}$ erhalten haben. Darauf basierende Ansätze werden auch als **Challenge-Response-Protocols** (Herausforderungs-Antwort-Protokolle) bezeichnet.
Das Protokoll arbeitet wie in Abbildung 8.2 dargestellt. Zunächst sendet Alice ihre Identität an Bob (1), womit der Wunsch nach Aufbau eines Kommunikationskanals angezeigt wird. Daraufhin sendet Bob eine Herausforderung R_B an Alice, z.B. eine zufällige Zahl (2). Alice verschlüsselt diese Zahl und schickt sie zurück (3). Bob kann das Ergebnis entschlüsseln und mit R_B vergleichen. Im positiven Fall weiß er, dass Alice auf der anderen Seite ist. Alice will nun aber auch sichergehen, dass Bob auf der anderen Seite ist. Dazu sendet sie eine Herausforderung (4), die Bob mit der entsprechenden Verschlüsselung beantwortet (5), die Alice wiederum entschlüsselt und vergleicht.
Eine „Optimierung“ dieses Protokolls könnte wie in Abbildung 8.3 dargestellt aussehen.
Dabei werden nur drei Nachrichten übermittelt. Dieses Protokoll funktioniert jedoch nicht, da so genannte **Reflection Attacks** auftreten können.
Zwecks Veranschaulichung gehen wir von einem Eindringling Chuck aus, der Bob vortäuschen will, er sei Alice. Über zwei parallele Sitzungen erfolgt dabei der in Abbildung 8.4 dargestellte Nachrichtenaustausch.

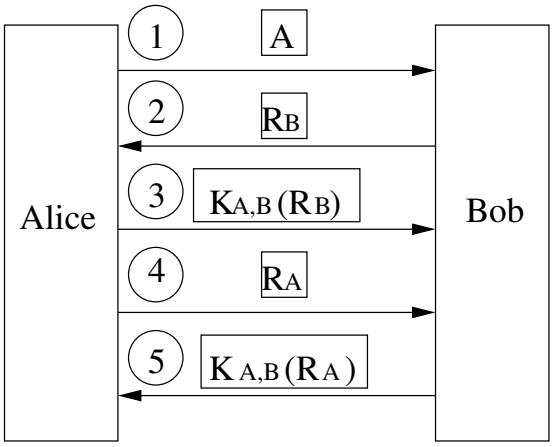


Abbildung 8.2: Challenge-Response-Protocol

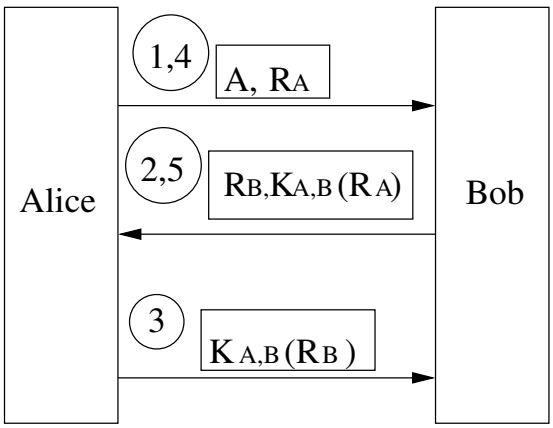


Abbildung 8.3: optimiertes Challenge-Response-Protocol

Fatal ist hierbei, dass in zwei verschiedenen Sitzungen und bei Herausforderungen in gegenläufigen Richtungen mit der gleichen Zahl R_b gearbeitet wird!
Abhilfe würde schaffen, wenn ein Initiator und ein Responder immer unterschiedliche Herausforderungen nutzen würden, z.B. gerade bzw. ungerade Zahlen für A und
Ein weiteres Problem besteht darin, dass Bob in seiner ersten Antwortnachricht wertvolle Informationen in Form der Verschlüsselung $K_{A,B}(R_x)$ herausgibt, ohne zu wissen, an wen diese gehen.
Unabhängig von diesen konkreten Problemen hat das vorgestellte Verfahren noch den Nachteil einer schlechten Skalierbarkeit. Bei N Hosts in einem Verteilten System würde jeder Host mit $N-1$ anderen Hosts einen Secret Key gemeinsam haben. Im System würde dies zu $N*(N-1)/2$ Schlüsseln führen, wobei jeder Host selbst $N-1$ Schlüssel verwalten müsste.
Lösungen:

– Repository von Zufallszahlen auslesen und ausstreichen, was bereits verwendet wurde.

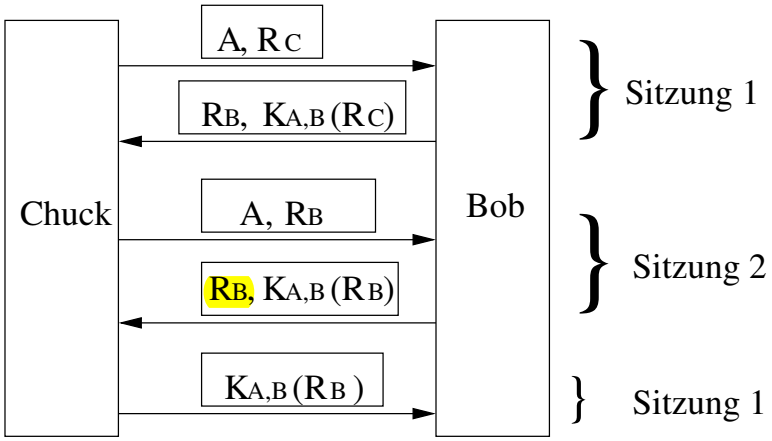


Abbildung 8.4: Reflection Attack

- Für Erstanfragen gerade Zahlen für R verwenden, für Replies ungerade.
- (Für Erstanfragen das Original verschlüsseln, für Replys R-1 verschlüsseln): Nur sicherer, aber nicht sicher (Chuck Sendet im 3. Schritt A, $R_B - 1$ und erhällt im 4. Schritt R_B zurück).

Eine Alternative besteht in einem zentralen Ansatz.

Authentifizierung mittels einem Key Distribution Center (KDC) Das Key Distribution Center besitzt mit jedem Host einen geheimen Schlüssel. Untereinander brauchen die Hosts paarweise keinen gemeinsamen Schlüssel besitzen. Damit existieren N Schlüssel anstelle der bislang $N * (N - 1)/2$.
Das Initiieren eines sicheren Kanals kann Alice nun über ein vertrauenswürdiges KDC wie in Abbildung 8.5 realisieren.

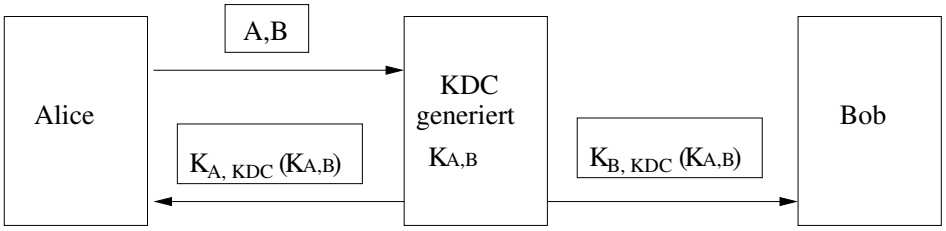


Abbildung 8.5: Initiierung eines sicheren Kanals mittels einem Key Distribution Center

Nach Entschlüsselung der erhaltenen Nachrichten können Alice und Bob direkt miteinander kommunizieren.
Ein Problem kann darin bestehen, dass Alice schon einen Kanal zu Bob aufbauen will, bevor Bob den verschlüsselten Schlüssel vom KDC erhalten hat. Dies kann durch den in Abbildung 8.6 dargestellten modifizierten Ansatz behoben werden.
Die Nachricht $K_{B,KDC}(K_{A,B})$ wird dabei auch als Ticket bezeichnet. Dabei ist es Aufgabe von Alice, dieses Ticket an Bob weiterzuleiten. Bob ist der einzige Empfänger, der die Möglichkeit

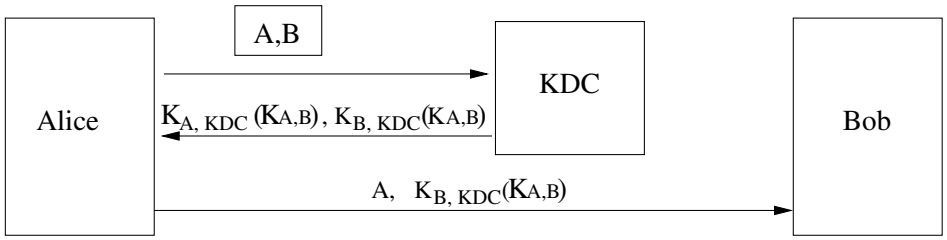


Abbildung 8.6: Modifizierter Ansatz

hat, mittels $K_{B,KDC}$ dieses Ticket zu entschlüsseln und so an $K_{A,B}$ heranzukommen.

Dieses Protokoll soll nun verallgemeinert werden, zu dem bekannten **Needham-Schroeder-Authentifizierungsprotokoll** (1978), das auch im Kerberos-System verwendet wird.
Das Prinzip soll in Abbildung 8.7 vorgestellt werden.
Vor desen Schritten wird als Prämisse angenommen: KDC hat $K_{A,KDC}$ und $K_{B,KDC}$ schon sicher verschickt (z.B. per Post).
Alice sendet zwecks Initierung eines sicheren Kanals zunächst eine Anfrage an das KDC, die neben A und B noch eine Herausforderung R_{A1} enthält. R_{A1} ist dabei ein so genanntes **Nonce**, d.h. eine von sehr vielen zur Verfügung stehenden, zufällig gewählten Zahlen, die nur einmal verwendet werden. Ziel ist dabei, zwei Nachrichten zueinander in Beziehung setzen zu können, hier Nachricht 1 und 2.
Diese Verwendung eines Nonces soll das Stehlen alter Schlüssel durch einen Angreifer sinnlos machen.
Ebenso gilt die Verwendung des Empfängers B in Nachricht 2 als Schutz. Nachdem das KDC das Ticket an Alice zugestellt hat, kann Alice dieses in Nachricht 3 an Bob weiterleiten und eine weitere Herausforderung R_{A2} in kodierter Form übermitteln.
Die jeweiligen Herausforderungen werden dekodiert und um eins vermindert in ebenfalls kodierter Form zurückgesendet. Beachte: Nur durch die Verminderung um eins besteht die Möglichkeit, eine Verschlüsselung in beiden Richtungen vorzunehmen.
Damit ergibt sich in vollständiger Form das in Abbildung 8.7 dargestellte Protokoll.

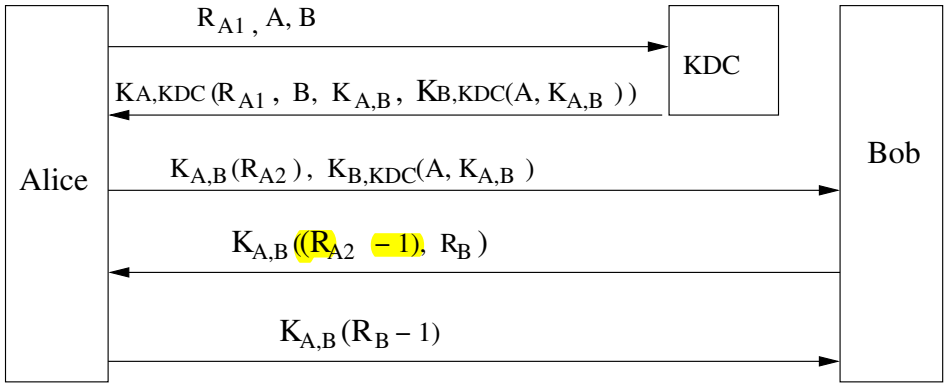


Abbildung 8.7: Needham-Schroeder-Authentifizierungsprotokoll

Eine Variante dieses Protokolls besteht darin, die Antworten der Herausforderungen nicht um

eins zu verringern. Dies ist insbesondere dann zulässig, wenn bei den verwendeten Nonces davon ausgegangen werden kann, dass diese nur einmalig verwendet werden. Dann wäre das Abfangen von Nachrichten zwecks späteren Missbrauchs ohne Relevanz.
Allgemein kann man sagen, dass Schlüssel möglichst oft geändert werden sollen, da die Kommunikation besonders unsicher ist, wenn:

- Schlüssel oft verwendet werden und
- Schlüssel für große Datenmengen angewandt werden.

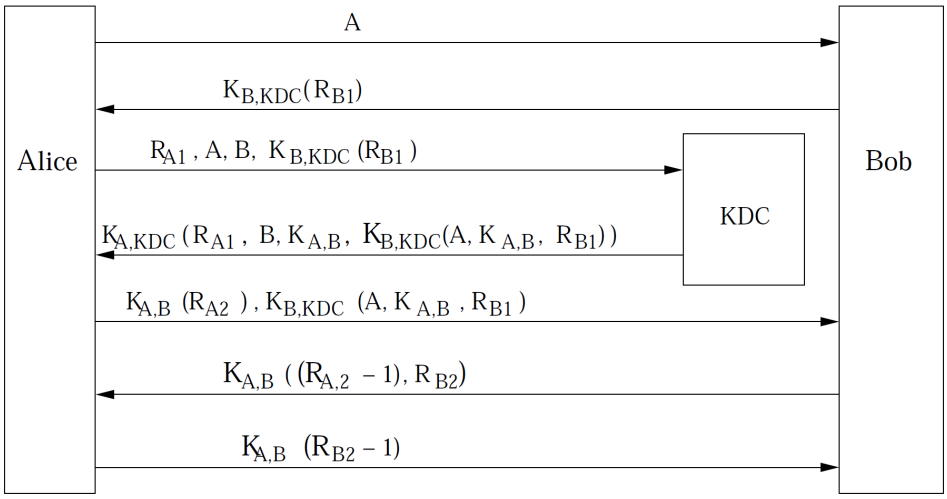


Abbildung 8.8: Nochmalige Verwendung von „alten“ Zufallszahlen

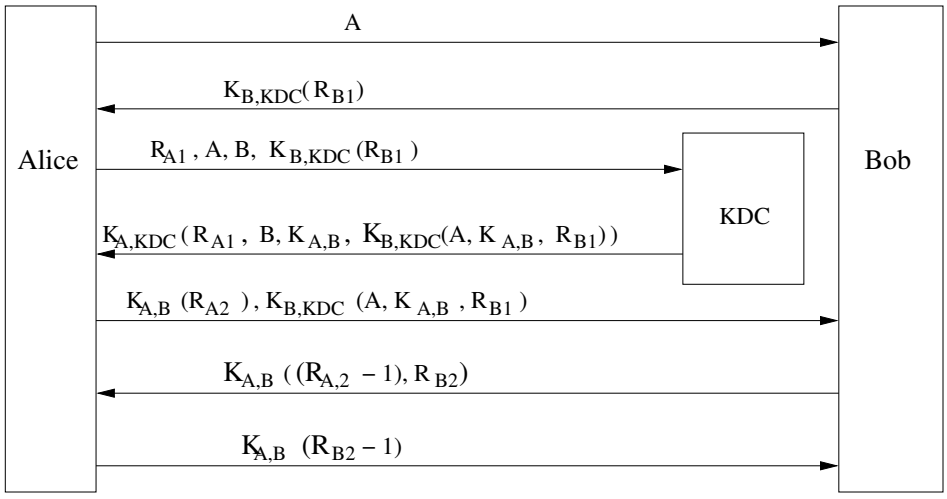


Abbildung 8.9: Modifiziertes Needham-Schroeder-Authentifizierungsprotokoll

Daraus resultiert für das Needham-Schroeder-Protokoll folgende Schwachstelle: Wiederverwendung alter Schlüssel, die gar nicht mehr in Gebrauch sind ($K_{A,B}$) bietet Angreifern poten-

tielle Möglichkeiten, hier einen Missbrauch zu begehen (vgl. 8.8: Chuck hat vorher die Kommunikation zwischen Alice und Bob mitgehört und gespeichert). Das Kritische sind Schlüssel, die oft und die früher für sehr große Datenmengen verwendet wurden: $K_{A,B}$ ist mir der Zeit ein immer unsicherer werdender Schlüssel. Dagegen wird $K_{X,KDC}$ sehr wenig verwendet: relativ selten und für sehr kleine Datenmangen $\Rightarrow$ Daher ist Needham-Schroeder mit Schwachstelle gegenüber geschwächtem $K_{A,B}$ ausgerüstet, die Modifikation nicht: Bei letzterer müssen $K_{A,B}$ und $K_{B,KDC}$ geknackt werden. Abhilfe können prinzipiell zwei Wege schaffen:

1. Einführung eines Timeouts und das Sperren „alter“ Schlüssel.
   Dies führt zu dem Problem, dass Manipulationen über Uhrensynchronisation / falsche Zeitstempel möglich sind.

2. Falls ein Angreifer aus irgendeiner Quelle über den Schlüssel $K_{A,B}$ verfügt und Nachricht 3 abhört und wiederverwendet, so kann dieser Angreifer die Nachricht 4 entschlüsseln und korrekt mit einer Nachricht 5 beantworten. Dies führt zu der bekannten Situation, dass Bob denkt, er kommuniziert mit Alice. In Wirklichkeit wäre Chuck jedoch am anderen Ende des Kommunikationskanals.
   Abhilfe besteht darin, dass das von Alice an das KDC übermittelte Nonce nicht von ihr beliebig generiert wird, sondern von Bob vorgegeben wird. Dazu muss dieser Wert noch zuvor von Alice bei Bob angefragt werden. Somit ergibt sich das in Abbildung 8.9 dargestellte Protokoll.
   Damit wird ein Zeitstempelersatz in Form einer Zufallszahl (Nonce durch Bob generiert) verwendet, die eine bestimmte Zeitdauer ab Generierung der Zahl gültig ist.
   Bemerkung: In der Regel wird davon ausgegangen, dass die Schlüssel, die zwischen den Kommunikationspartnern und dem KDC bestehen, d.h. $K_{X,KDC}$, „sicherer“ sind als die Schlüssel zwischen den Partnern untereinander, z.B. $K_{A,B}$, weil sie:

   - seltener benutzt werden und
   - nur für kleinere Datenmengen relevant sind.

Authentifizierung mittels Public-Key-Kryptographie Ein öffentlicher Schlüssel (Public Key) soll im Folgenden durch K^+ bezeichnet werden. Das geheime, d.h. nicht öffentliche oder private Gegenstück dazu ist K^- (Private Key). Ein typisches Authentifizierungsprotokoll kommt ohne ein KDC aus und basiert auf dem in Abbildung 8.10 dargestellte Grundprinzip.
Alice sendet dabei eine verschlüsselte Anfrage mit einer Herausforderung R_A an Bob, die dieser entschlüsselt und mit einer weiteren Herausforderung beantwortet. Da Bob die einzige Partei ist, die die von Alice gestellte Anfrage entschlüsseln kann, weiß Alice nun, dass Bob am anderen Ende ist. Nun muss Alice ihrerseits noch den Beweis erbringen, dass sie Bobs Nachricht entschlüsseln kann.

Beachte: Der für die Kommunikation auf dem sicheren Kanal zu verwendende Schlüssel $K_{A,B}$ wird hierbei von Bob generiert und dann in Nachricht 2 Alice mitgeteilt.
Wichtig ist ferner, dass die privaten Schlüsseln K_A^- und K_B^- wirklich nur bei Alice bzw. Bob vorhanden sind.
D.h. statt Schlüssel mit KDC (oder einem paarweise symmetischen Schlüssel) werden hier Schlüsselpaare generiert, die aus dem privaten und öffentlichen Schlüssel bestehen.

- Vorteil: KDC entfällt bzw. trotzdem lineare Komplexität.
- Nachteil: sicher?, langsam

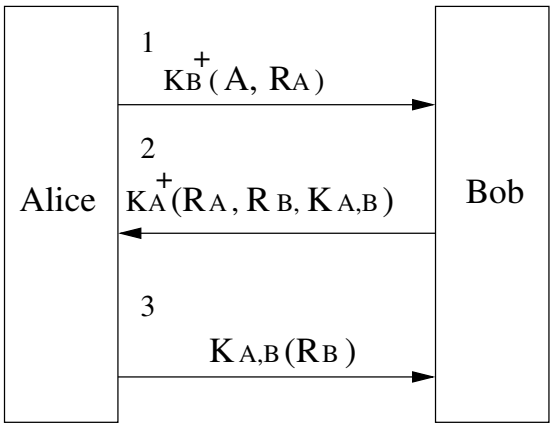


Abbildung 8.10: Public-Key-Kryptographie

8.1.2 Nachrichtenintegrität und Vertraulichkeit

Definition zu den Nachrichten: $A^+(A^-(m)) = m$; $A^-(A^+(m)) = M$
Nachrichtenintegrität (Message Integrity), auch Nachrichtenechtheit genannt, bezeichnet den Sachverhalt, dass Nachrichten gegen Modifikationen geschützt werden. Dies ist recht komplex.
Vertraulichkeit (Confidentiality) dagegen sichert ab, dass Nachrichten nicht durch Lauscher abgehört werden können. Dies kann relativ einfach erreicht werden, indem die Nachricht vor dem Abschicken verschlüsselt wird.
Verschlüsselung kann dabei entweder mittels eines geheimen Schlüssels erfolgen, der von Sender und Empfänger gemeinsam genutzt wird - oder aber mittels eines öffentlichen Schlüssels des Empfängers.

Digitale Signaturen Nachrichtenintegrität geht oft über die Verwendung eines sicheren Kanals hinaus.

Beispiel:
Alice und Bob einigen sich elektronisch, dass Alice für 500 Euro eine Schallplatte von Bob abkauft. Aus Sicht der Nachrichtenintegrität resultieren zumindest zwei Probleme:

- Bob soll später nicht behaupten können, dass als Kaufpreis 5000 Euro ausgehandelt wurden, und
- Alice soll später nicht bestreiten, dass sie einem derartigen Vertrag zugestimmt und ihre Kaufabsicht geäußert hat.

Eine Lösung dieser Probleme kann dadurch erreicht werden, dass Alice den Kaufvertrag so digital unterschreibt, dass eine eindeutige Assoziation zwischen der Signatur und dem Nachrichteninhalt hergestellt wird. Es gibt verschiedene Formen der Realisierung solcher digitaler Unterschriften.

- Digitale Signaturen basierend auf Public-Key-Kryptographie:
  Ein populäres Public-Key-Kryptosystem ist das RSA-Verfahren. Wenn Alice eine Nachricht m an Bob senden will, so verschlüsselt sie diese mit ihrem privaten Schlüssel K_A^-

und sendet sie an Bob. Soll darüber hinaus der Inhalt geheim bleiben, so kann sie ferner Bobs öffentlichen Schlüssel nutzen. Wird parallel dazu die eigentliche Nachricht m gesendet, so kann Bob die entschlüsselte Nachricht mit dem Original vergleichen und bei Übereinstimmung sicher sein, dass diese Nachricht von Alice stammt.
Alice schützt sich gleichzeitig vor Manipulationen durch Bob, da Bob beweisen müsste, dass die veränderte Nachricht ebenfalls von Alice mit dem privaten Schlüssel verschlüsselt wurde. Obwohl das Protokoll an sich korrekt ist, resultiert eine Reihe von Proble-

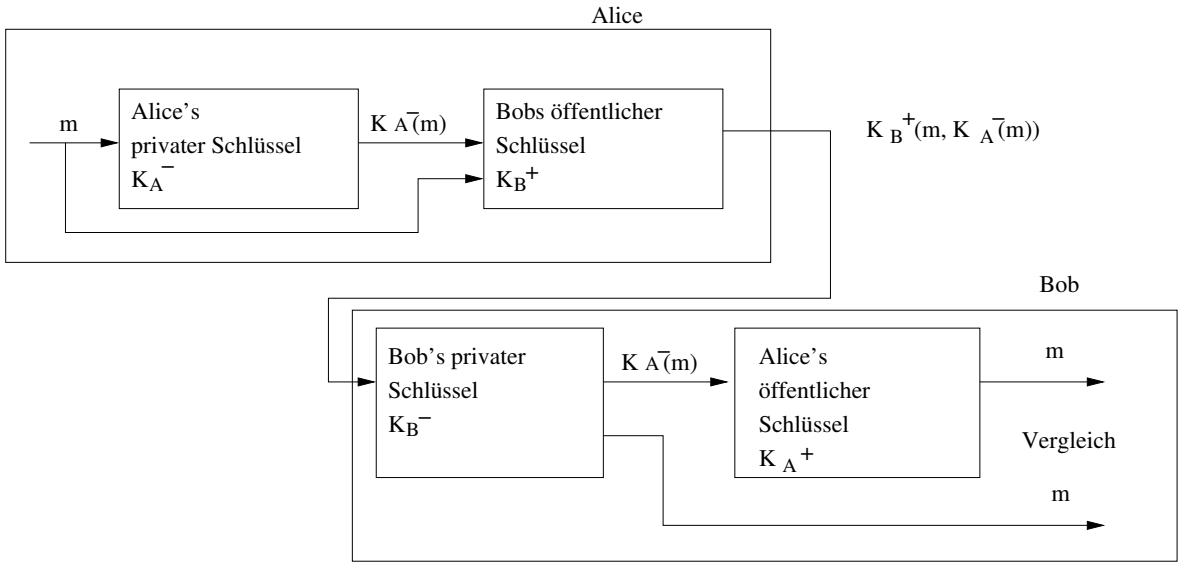


Abbildung 8.11: Digitale Signaturen basierend auf Public-Key-Kryptographie

men:

- Alles funktioniert nur so lange, wie Alices privater Schlüssel auch geheim bleibt.
- Auch geheime Schlüssel sollten von Zeit zu Zeit geändert werden. Damit ihre vertragliche Zusage dennoch gültig bleibt, müsste es eine unabhängige Autorität geben, die solche Schlüssel archiviert und verwaltet.
- Zusätzlich kann die Verschlüsselung mit dem privaten Schlüssel sehr rechenintensiv sein, da hierbei in der Regel komplexe mathematische Algorithmen zugrundeliegen.

– Digitale Signaturen basierend auf Message Digests:
Ein Message Digest (Nachrichtenzusammenfassung) ist ein Bitstring h fester Länge, der aus einer Nachricht m mit willkürlicher Länge mittels einer kryptographischen Hashfunktion H generiert wurde.
Im Folgenden sollen die Grundprinzipien einer Hashfunktion $h = H(m)$ betrachtet werden, wobei m eine beliebige und h eine feste Länge hat.
h entspicht extra Bits, die ähnlich den Paritybits bei der Fehlererkennung und -korrektur zu einer Nachricht hinzugefügt werden. Hashfunktionen haben folgende Eigenschaften:

- Hashfunktionen sind One-way-Funktionen, d.h. es ist rechentechnisch nahezu unmöglich, einen Input m zu finden, zu dem ein bekannter Output h gehört.
- Die Berechnung $h = H(m)$ zu einem gegebenen m ist einfach.
- Bei gegebenen m und $h = H(m)$ ist es rechentechnisch kaum möglich, eine andere Nachricht $m' \neq m$ zu finden, so dass $H(m) = H(m')$. Ferner lassen sich aus H keine zwei verschiedenen Nachrichten m und m' ableiten, für die $H(m) = H(m')$ gilt.

Eine derartige Funktion kann nun wie in Abbildung 8.12 genutzt werden.

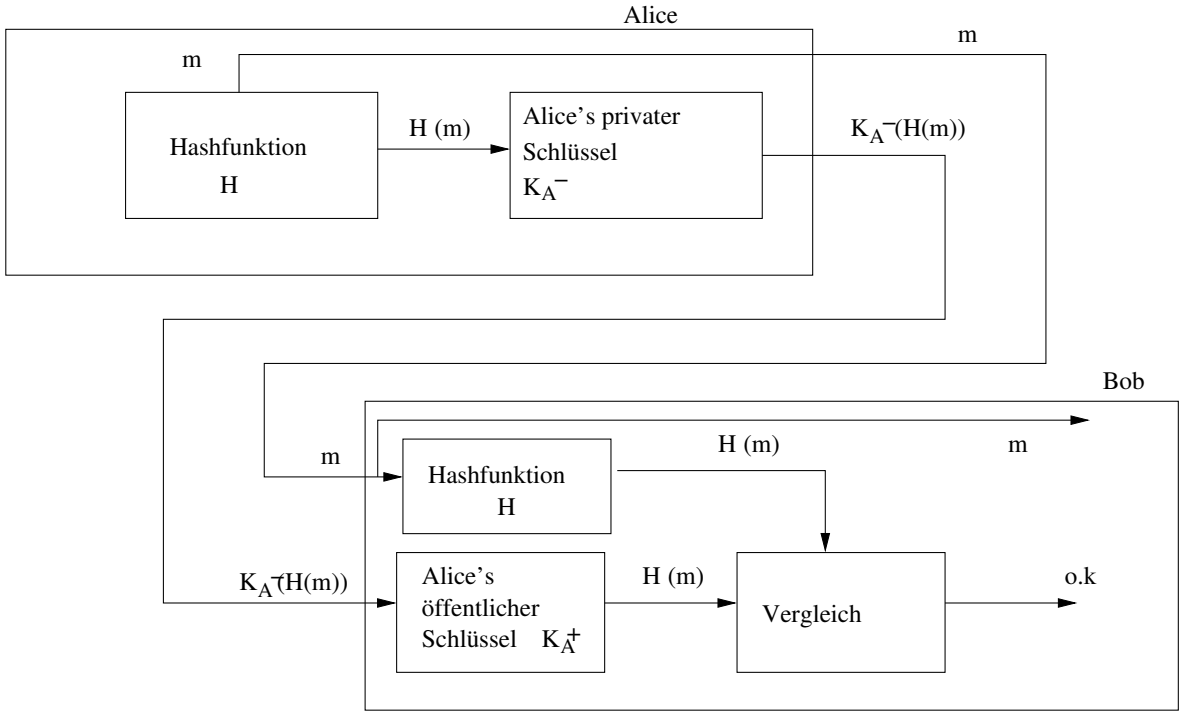


Abbildung 8.12: Digitale Signaturen basierend auf Message Digest

Soll in Ergänzung zu diesem leicht nachvollziehbaren Verfahren die Nachricht m nicht öffentlich verschickt werden, so müsste bei Alice noch eine Verschlüsselung mit Bobs öffentlichen Schlüssel erfolgen. In diesem Fall würde Bob zunächst eine Entschlüsselung mit seinem privaten Schlüssel durchführen. In jedem Fall ist bei korrektem Vergleich der versendeten und der berechneten Hashfunktion $H(m)$ sicher, dass die Nachricht von Alice stammt.

8.2 Zugriffskontrolle

Eine Clientanfrage kann nur ausgeführt werden, wenn der Client auch entsprechende Zugriffsrechte (Access Rights) hat. Die Kontrolle der Zugriffsrechte wird auch als Zugriffskontrolle (Access Control) bezeichnet. Der Vorgang der Erfüllung von Zugriffsrechten ist die Autorisierung (Authorization). Im Folgenden soll das zugrundeliegende Modell betrachtet wer-

den:
Ein Subjekt möchte auf ein Objekt zugreifen. Objekte haben dabei einen Zustand. Sie sind gekapselt, und auf ihnen können Operationen ausgeführt werden. Dabei greifen Subjekte über Schnittstellen auf das Objekt zu. In diesem Sinne sind Subjekte Prozesse, die im Auftrag eines Nutzers ausgeführt werden oder aber selbst Objekte, die einen Dienst eines anderen Objekts benötigen. Zugriffskontrolle auf einem Objekt heißt, das Objekt gegen unerlaubten Zugriff zu schützen. Ferner umfasst die Zugriffskontrolle ein Objektmanagement wie Schaffung, Umbenennung oder Löschung von Objekten. Dieser Schutz wird durch einen so genannten **Reference Monitor** ausgeführt. Dieser Monitor wird bei jedem Objektaufruf aufgerufen.

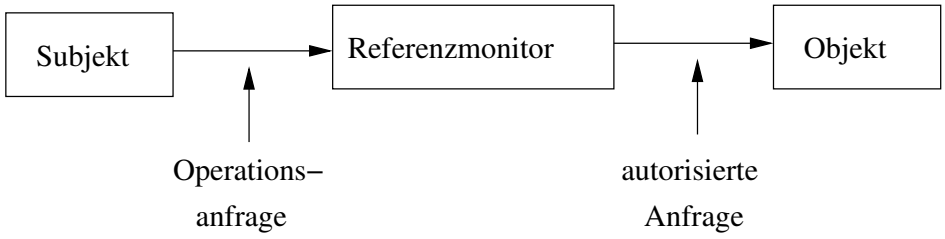


Abbildung 8.13: Referenzmonitor

Ein geläufiger Ansatz für die Modellierung von Zugriffsrechten besteht in einer **Access Control Matrix (Zugriffskontrollmatrix)**.
Jedes Subjekt wird dabei durch eine Zeile repräsentiert, jedes Objekt durch eine Spalte. Die Matrixelemente werden mit $M(S, O)$ bezeichnet und beschreiben, welche Operationen ein Subjekt S auf einem Objekt O ausführen kann. D.h. bei Anforderung einer Methode m prüft der Referenzmonitor, ob m in $M(S, O)$ aufgelistet ist. Andernfalls schlägt die Operation fehl.
Z.B.:

	O_1	$\dots$	O_m
S_1			
$\vdots$		$\ddots$	
S_n			

Problematisch ist hierbei die Skalierbarkeit, wenn man von Tausenden von Nutzern und Millionen von Objekten ausgeht. Deshalb wählt man als Alternative eine spaltenweise Verteilung über alle Objekte in einer **Access Control List (ACL)**. Das bedeutet, dass jedes Objekt eine Liste von Zugriffsrechten der Subjekte verwaltet, die auf dieses Objekt zugreifen dürfen. Leere Einträge der Matrix werden in dieser Liste weggelassen. Ein Nebeneffekt davon ist die Vermeidung von Bottlenecks.
Z.B.:

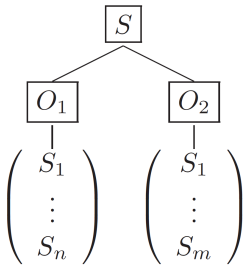


Abbildung 8.14: Access Control List (ACL)

Ein weiterer Ansatz besteht in einer zeilenweisen Verteilung, in der jedem Subjekt eine Liste von Capabilities (Möglichkeiten) gegeben wird, die es für jedes Objekt hat. Ist für ein Objekt in dieser Zeile kein Eintrag vorhanden, so darf das Subjekt nicht zugreifen.
Eine Capability kann dabei mit einem Ticket verglichen werden, das gewisse Rechte assoziiert und entsprechend geschützt werden muss.
Um die Komplexität dieser Thematik weiter zu reduzieren, können Nutzer zu Gruppen zusammengefasst werden, deren Mitglieder alle dieselben Rechte haben. Und ganze Domains können mit gleichen Zugriffsrechten für Subjekte ausgestattet werden.
In der Praxis wird externer Zugriff auf Teile eines Verteilten Systems über eine spezielle Art von Referenzmonitoren kontrolliert, die so genannten **Firewalls**. Eine Firewall trennt - von Prinzip her - einen Teil eines Verteilten Systems von der Außenwelt. Alle ein- und ausgehenden Pakete werden durch einen speziellen Computer geroutet und untersucht, bevor sie durchgelassen werden. Insbesondere werden Quell- und Zieladresse untersucht. Nichtautorisierter Datenverkehr darf dabei nicht passieren. Dabei muss die Firewall selbst geschützt werden, insbesondere darf sie nicht ausfallen.

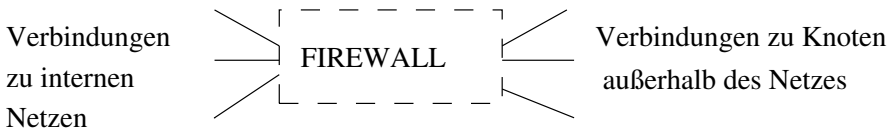


Abbildung 8.15: Firewall

Diese Thematik soll an dieser Stelle nicht weiter vertieft werden.

Weitere Probleme bzgl. der Sicherheit von Verteilten Systemen betreffen das Management. Dabei unterscheidet man:

- Schlüsselmanagement, das die Verteilung kryptographischer Schlüssel beinhaltet und
- Autorisierungsmanagement einschließlich Zertifizierung und Delegation von Zugriffsrechten.

9

Skalierbarkeit mittels Replikation, Caching und Verteilung

►

Inhaltsangabe

Replikation kann sich gleichermaßen auf Daten und Objekte beziehen. Der Einfachheit halber soll im Folgenden primär von den grundlegenden Prinzipen der Datenreplikation ausgegangen werden. Neben Replikation gibt es auch noch Migration (= Verteilung: Kommunikationswege kürzer), Caching, Verteilung (= Dieste in Subnetze anbieten).

9.1 Vorteile der Replikation

Replikation hat im Wesentlichen zwei Vorteile:

- **Zuverlässigkeit / Verfügbarkeit / Fehlertoleranz**
  Nutzer fordern stets hochverfügbare Daten und Dienste. Diese Qualität wird auch in der Zeit bzw. Wahrscheinlichkeit gemessen, in der ein Dienst nutzbar ist.
  Sei p die Wahrscheinlichkeit für den Ausfall eines Servers, P die Wahrscheinlichkeit, dass ein Zugriff auf einen Dienst möglich ist.
  Folglich gilt: $P = 1 - p$
  Nun gehen wir von einer Replikation der Daten aus, wobei *n* Server unabhängig voneinander zur Verfügung stehen. Der Dienst steht nun immer dann zur Verfügung, wenn nicht gerade alle Server gleichzeitig ausfallen, also
  P_n = 1-p^n

  Beispiel:
  Die Ausfallwahrscheinlichkeit eines Servers sei $p = 5\%$. Dann ergibt sich für *n* Replikate folgende Zugriffswahrscheinlichkeit P_n:
  P_1 = 1-$0,05^1$ = 95%
  P_2 = 1-$0,05^2$ = 1-0,0025 = 99,75%
  P_3 = 1-$0,05^3$ = 1-0,000125 = 99,9875%
  ⋮ P_n = 1-$0,05^n$ Unabhängig von diesen quantitativen Ausfallwahrscheinlichkeiten ist eine ganz andere Qualität der Erreichbarkeit bei Netzteilungen (Network Partitions) und fehlendem Zugang zu Operationen (Disconnected Operations, z.B. Wireless Networking) vorhanden.

- **Leistungsfähigkeit / Performance**
  Performance ist insbesondere dann von Bedeutung, wenn die Nutzer eines Dienstes im Verteilten System in ihrer Anzahl und geographischen Verbreitung skalieren. Ein bereits diskutiertes Beispiel war das Entstehen von Clustern replizierter Webserver.

- Bei Netzpartitionierung ist der Dienst, Server etc. noch erreichbar.

Allerdings bringt Replikation auch einige Probleme/ Anforderungen mit sich:

- Replikationstransparenz
  Dem Client sollte verborgen bleiben, ob er mit Originaldaten arbeitet, oder aber physikalische Kopien in Form von Replikaten vorliegen.

- Konsistenz
  Der hauptsächliche Preis, der bei der Replikation bezahlt werden muss, liegt in der Konsistenz. D.h. gibt es Modifikationen des Datenbestandes, so müssen diese Änderungen auf allen Replikaten vorgenommen werden.

Das Wann und Wie der Ausführung von Modifikationen kann hierbei zu unterschiedlichen Kosten führen.

– Wahrscheinlichkeit, dass eine Komponente nicht geht, ist größer.

Veranschaulichung dieses Problems: Der Zugriff auf Webseiten soll optimiert werden. Das Laden einer entfernten Webseite kann zeitweise schon mal einige Sekunden dauern. Um die Leistungsfähigkeit zu erhöhen, speichern Webbrowser oft Kopien der zuletzt aufgerufenen Seiten ab, d.h. sie cachen die Webseiten. Falls der Nutzer eine solche Seite noch einmal anfordert, kann der Browser sehr schnell eine lokale Kopie bereit stellen. Nun ist die benötigte Zeit minimal. Problematisch ist jedoch, dass die Webseite ggf. in der Zwischenzeit modifiziert wurde und die Kopie nicht mehr aktuell ist.

Im Fall eines Zugriffs auf unbedingt aktuelle Kopien resultiert die Frage:
Ist es günstiger, lokale Kopien stets up-to-date zu halten, oder wird Netzbandbreite gespart, wenn gar keine Replikate angelegt werden?

Für eine Trade-off-Betrachtung soll von folgenden Größen ausgegangen werden:

N ist die Anzahl (pro Sekunde) der Zugriffe eines Prozesses auf ein lokales Replikat und

M ist die Anzahl (pro Sekunde) der Modifikationen des Datenbestandes.

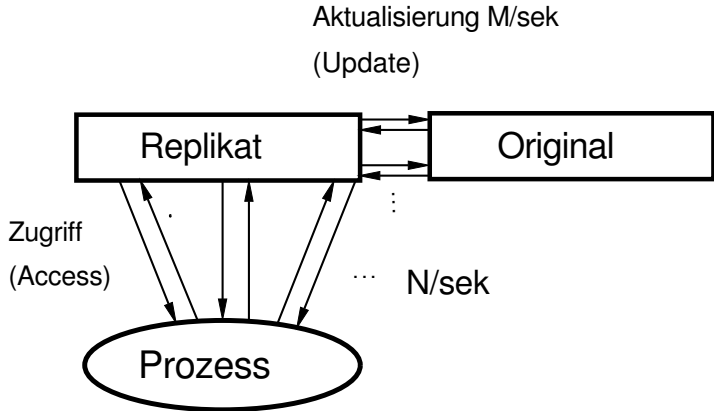


Abbildung 9.1: Zugriffe auf Replikat

Nun sollen zwei Fälle betrachtet werden:

N $\ll$ M, d.h. die Access-to-Update-Rate ist sehr gering. Auf viele Updates wird niemals zugegriffen. In diesem Fall ist die Verwaltung eines Replikats unwirtschaftlich.

N $\gg$ M, d.h. die Access-to-Update-Rate ist sehr hoch, das Anlegen und Aktualisieren des Replikats lohnt sich.

9.2 Eine Systemarchitektur für Replikationen

Um möglichst allgemein zu bleiben, sollen Architektur-Komponenten im Folgenden durch ihre Rollen beschrieben werden. Dies impliziert nicht automatisch, dass jeder Komponente ein separater Prozess oder eine eigene Hardwareeinheit entspricht.
Ausgangspunkt unserer Betrachtungen sind Replikate. Diese werden von so genannten **Replica Managern (RM)** gehalten und verwaltet. In diesem Sinne ist ein RM eine Komponente (bzw. ein Prozess), die Replikate auf einem bestimmten Computer enthält und Operationen darauf direkt ausführt.
Angewandt auf eine Client/Server-Umgebung ist ein RM ein Server. Eine Anwendung kann gleichermaßen als Client und als RM arbeiten. Beispiel: Ein Laptop in einem fahrenden Zug kann gleichzeitig eine Anwendung und einen RM für einen Terminkalender enthalten.
I.d.R. enthält jeder Replica Manager eine physikalische Kopie aller Daten, aber es ist auch denkbar, dass diese Kopien von verschiedenen Replica Managern gehalten werden. Das allgemeine Modell für das Management replizierter Daten beinhaltet außerdem noch Front Ends, siehe Abb. 9.2. Die Aufgaben der **Front Ends (FE)** liegen darin, die Anfragen zu bearbeiten und die Kommunikation mit den Replica Managern zu erledigen. Der Vorteil darin liegt in der so erreichten Transparenz.

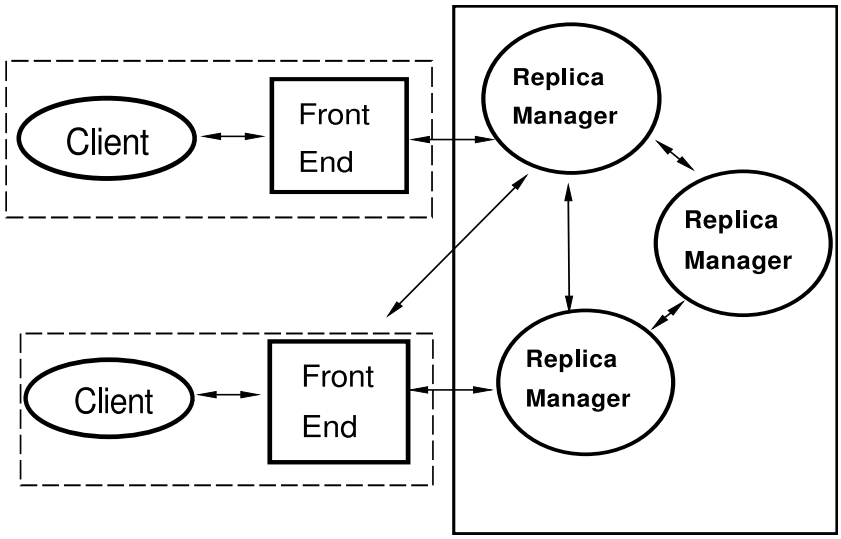


Abbildung 9.2: Das allgemeine Modell für das Management replizierter Daten

Ein Front Ends kann mit einem oder mehreren Replica Managern kommunizieren. Dies geschieht durch das Message Passing. Bei mehreren Replica Managern sammelt das Front End die Antworten und übermittelt dem Client genau eine Antwort. Der Client kennt also das Front End, muss aber nichts von den Replica Managern wissen.

Betrachten wir die Implementierung von Front Ends:
Front Ends können entweder ein Bestandteil eines Clients oder ein eigenständiger Prozess sein. So verwendet beispielsweise die **Gossip Architecture** Front Ends als eigenständigen Prozess, vgl. Abbildung 9.3. Charakteristisch für die Gossip Architekture ist weiterhin, dass ein Front End im Allgemeinen mit einem individuellen Replica Manager für jede Operation kommuniziert. Ein anderes Modell beschreibt das in der Abbildung 9.4 dargestellte **Primary**

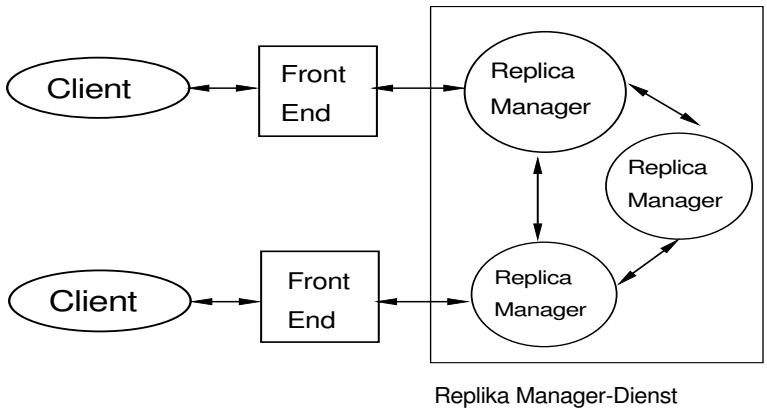


Abbildung 9.3: Die Gossip Architecture

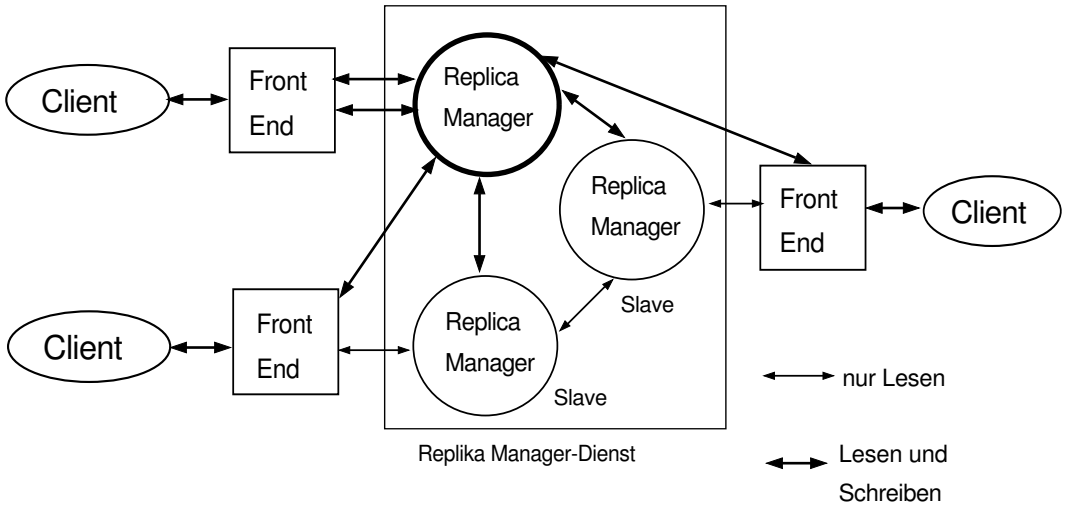


Abbildung 9.4: Das Primary Copy Model der Datenreplikation

Copy Model der Datenreplikation.
Hier ist charakteristisch, dass alle Front Ends mit dem gleichen Primary Server kommunizieren, wenn eine spezielle Dateneinheit aktualisiert wird und der Primary Server die Updates den Slaves bekannt macht (also die Front Ends Daten von den Slaves lesen können). Falls der Primary Server ausfällt, kann einer der Slaves als primär ernannt werden (vgl. Abbildung 9.5). Ein Beispiel für diese Architektur ist der SUN Network Information Service (NIS). Passwort-Einträge, die i.d.R. relativ selten geändert werden, werden hier auf einem Masterserver aktualisiert, und von dort werden die Änderungen den Slaves mitgeteilt. Die Front Ends können sowohl an den Masterserver als auch an einen Slave eine Anfrage stellen.
Ein weiteres Modell ist die Architektur der Shared Editors, siehe Abbildung 9.6. Nutzer können bei diesem an verschiedenen Rechnern einen Editor nutzen, um z.B. ein Dokument zu entwerfen. Somit stellt diese Architektur Möglichkeiten für Gruppenarbeit bereit. Jede ausführende Instanz eines Editor-Programms besitzt eine Kopie des Dokuments in seinem aktuel-

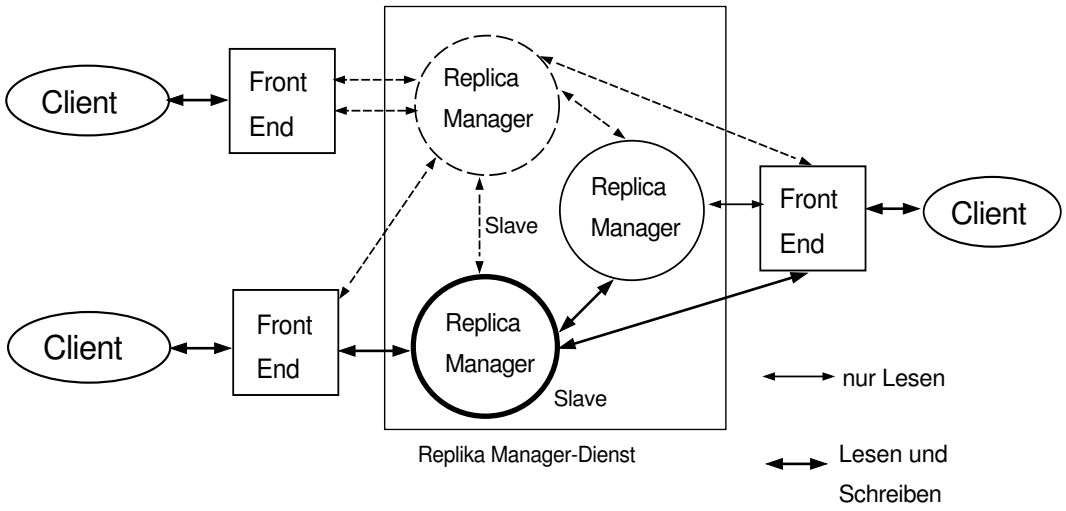


Abbildung 9.5: Das Primary Copy Model der Datenreplikation nach Ausfall des bisherigen Primary Servers

len Zustand. Außerdem existiert nur eine Klasse von Prozessen, die sowohl die Funktionalität der Clients als auch der Replica Manager umfasst.

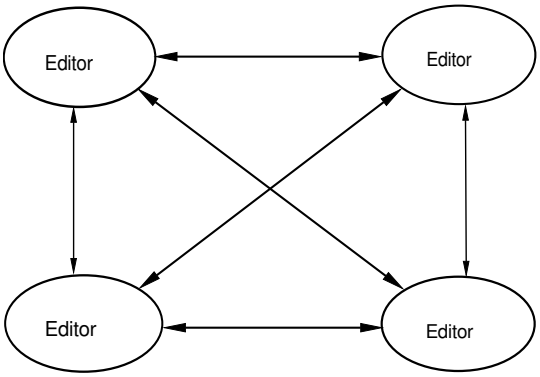


Abbildung 9.6: Die Architektur der Shared Editors

9.3 Anfragebearbeitung bei Replikaten

Bei der Ausführung einer einzelnen Anfrage auf replizierten Objekten unterteilt man 5 Phasen nach [12].

9.3.1 Die Anfrage

Das FE stellt eine Anfrage an einen oder mehrere RM. Dabei gibt es zwei Möglichkeiten: Das FE kann eine einzelne Anfrage an einen RM stellen, der mit anderen RM kommunizieren

kann. Alternativ besteht die Möglichkeit, dass das FE ein Multicast der Anfrage an eine Menge von RM stellt.

9.3.2 Koordination und Replikationskonsistenz

Seitens des RM erfolgt eine Konsistenzkoordination in Vorbereitung auf die Ausführung der Anfrage. In Abhängigkeit einer gewünschten Ordnung wird dabei entschieden, ob eine gewünschte Anfrage ausgeführt werden kann. Dabei sind die folgenden Ordnungen von Interesse.

1. **FIFO-Ordnung**
   Wir gehen zunächst von einer Multicast-Operation *multicast(g,m)* aus. Diese Operation sendet eine Nachricht *m* an alle Mitglieder der Gruppe *g*.
   Eine FIFO-Ordnung für eine Multicast-Operation wird wie folgt definiert:
   Falls **ein** korrekter Prozess erst ein *multicast(g,m)* und dann ein *multicast(g,m')* aufruft, dann erhält jeder korrekte Prozess, der *m'* erhält, zuvor erst *m*.
   Beispiel:

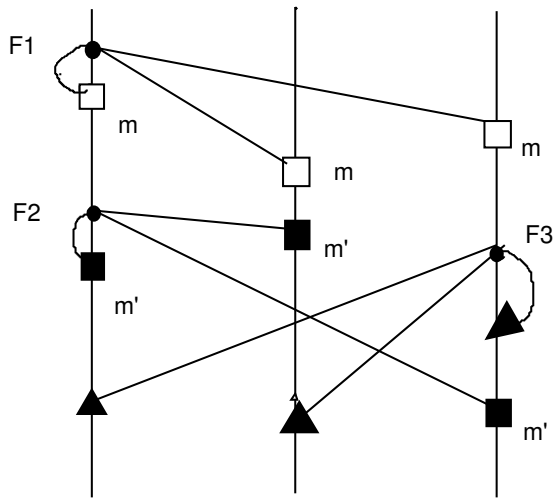


Abbildung 9.7: FIFO-Ordnung

oder auch:

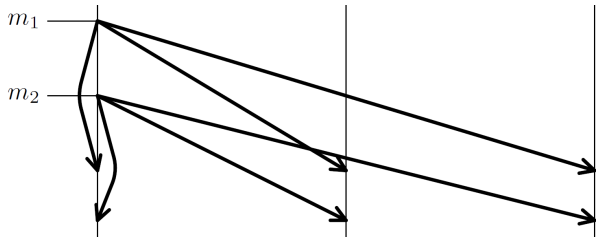


Abbildung 9.8: FIFO-Ordnung

Angewendet auf die Replikation besagt die FIFO-Ordnung:
Falls **ein** FE die Anfrage *r* vor der Anfrage *r'* stellt, dann bearbeitet jeder korrekte RM, der *r'* bearbeitet, die Anfrage *r* noch zuvor.

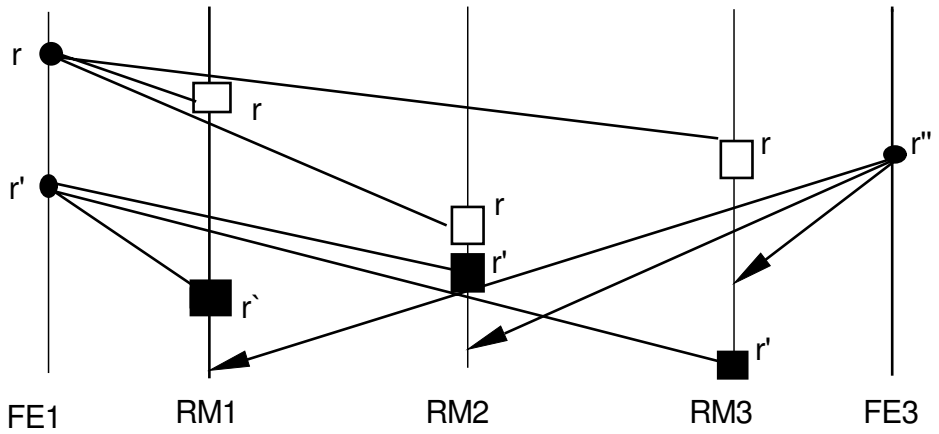


Abbildung 9.9: FIFO-Ordnung bei Replikation

2. **Kausale Ordnung**
Informelle Idee: Eine Nachricht, die vor einer zweiten gesendet wird, kommt auch bei allen Replikaten vor der zweiten an.
Eine kausale Ordnung für eine Multicast-Operation wird wie folgt definiert:
Falls *multicast(g,m)* $\rightarrow$ *multicast(g,m')* (wobei $\rightarrow$ die Lamportsche „vor"-Relation ist, die sich hier ausschließlich auf Nachrichten bezieht, die zwischen den Mitgliedern von *g* gesendet wurden), so gilt für jeden korrekten Prozess, der *m'* zustellt, dass er *m* vor *m'* zustellt.
C_1 und C_3 stehen in kausaler Ordnung. Angewandt auf die Replikation bedeutet dies:
Falls die Anfrage *r* vor der Anfrage *r'* gestellt wird, dann behandelt jeder korrekte RM, der *r'* behandelt, *r* noch vor *r'*.
Die kausale Ordnung impliziert die FIFO-Ordnung, wenn zwei Anfragen vom selben FE oder Prozess gestellt werden und in der „vor"-Relation stehen.

3. **Totale Ordnung**
Eine totale Ordnung für eine Multicast-Operation wird wie folgt definiert:

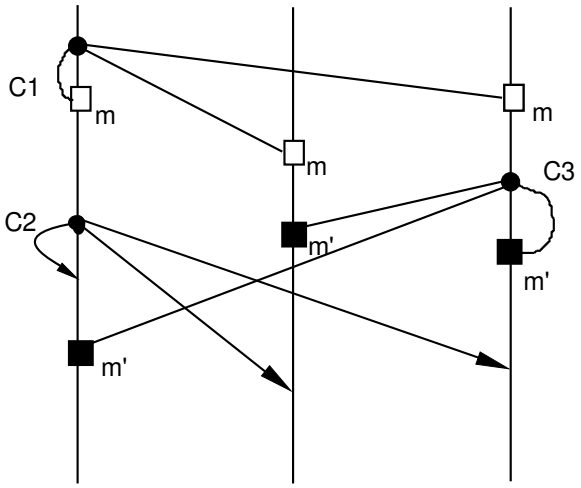


Abbildung 9.10: Kausale Ordnung

Falls ein korrekter Prozess die Nachricht *m* vor der Nachricht *m'* erhält, dann gilt für jeden anderen korrekten Prozess, der *m'* zustellt, dass *m* vor *m'* zugestellt wird. T_1 und T_2

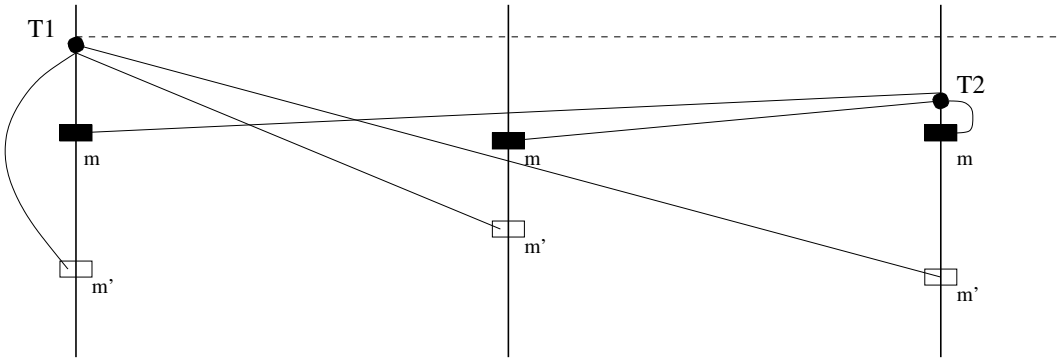


Abbildung 9.11: Totale Ordnung

stehen in totaler Ordnung, auch wenn die Nachrichten in der physikalisch umgekehrten Reihenfolge überall ankommen als sie abgeschickt wurden. D.h. Nachrichten können willkürlich geordnet werden, so lange die Ordnung in allen Prozessen dieselbe ist.
Angewandt auf die Replikation bedeutet dies:
Falls ein korrekter RM *r* vor *r'* handhabt, dann handhabt jeder RM, der *r'* handhabt, *r* noch bevor, unabhängig davon, wann das FE die Nachrichten verschickt hat bzw. in welcher Reihenfolge. Die totale Ordung ist die schwächste aller Ordnungen $\rightarrow$ ist automatisch erfüllt, wenn die anderen erfüllt sind.

Vergleicht man diese Ordnungen und betrachtet sie im Kontext der Replikationskonsistenz, so erfordern die meisten Anwendungen eine FIFO-Ordnung.

9.3.3 Vorläufige Ausführung

Die RM führen Anfragen i.d.R. vorläufig aus, um ggf. unerwünschte Effekte rückgängig zu machen.

9.3.4 Zustimmung

Falls die RM im Ergebnis einer Anfrageausführung zu einem Konsens kommen, so wird das Ergebnis festgeschrieben. Insbesondere erfolgt bei einer Transaktion entweder ein *Abort* oder ein *Commit*.

9.3.5 Response

Ein oder mehrere RM antworten dem FE. In einigen Systemen sendet genau ein RM die Antwort, in anderen erhält ein FE Antworten von einer Menge RMs und generiert dann eine Antwort für den Client.

Diese fünf Phasen werden in unterschiedlichen Systemen ggf. auch unterschiedlich gehandhabt. Die Variation hängt dabei von verschiedenen Zielvorgaben ab, z.B. ob der Nutzer des Laptops im fahrenden Zug vielleicht froh ist, überhaupt noch schnell Zugang zu Daten zu haben, oder ob bei Banktransaktionen die Aktualität der Daten von höchster Priorität ist.

9.4 Implementierung von Replikationstechniken

Implementierung von Anfrageordnungen Im Folgenden betrachten wir die grundlegenden Techniken der Implementierung von Anfrageordnungen.
Bei der **Hold-back-Technik** wird eine eintreffende Anfrage von den Replica Managern nicht verarbeitet, solange keine Ordnungsbedingungen erfüllt sind, d.h. die Anfrage wird zurückgehalten. Z.B. wird in einem E-mail-System „Re:Subject“ immer erst nach „Subject“ weitergeleitet.
Als Variante dazu betrachten wir die **Processing-Queue-Technik** (vgl. Abbildung 9.12). Eine Anfrage heißt **stabil** bei einem Replica Manager, wenn alle vorher abzuarbeitenden Anfragen (bzgl. einer Ordnungsrelation) bereits ausgeführt sind, und die ankommende Anfrage die nächste ist.
Nach dem Verfahren von Schneider gilt: Ankommende Anfragen werden zunächst in der **Hold-Back-Queue** platziert. Dort verbleiben sie, bis ihre Ordnung bestimmt wurde. Ist eine Anfrage stabil, so wird sie in eine **FIFO-Processing-Queue** eingeordnet.
Für die Implementierung von Ordnungsrelationen sollen stets zwei Eigenschaften untersucht werden:
Die **Safety-Eigenschaft** besagt, dass keine Anfrage, welche die Hold-Back-Queue verlässt, außerhalb irgendwelcher Ordnungsanforderungen ausgeführt wird. D.h. die Implementierung muss garantieren, dass nach der Ausführung einer Anfrage keine Anfrage mehr kommt, die eigentlich früher hätte abgearbeitet werden müssen.
Außerdem wird die **Liveness-Eigenschaft** untersucht, die beschreibt, dass keine Anfrage unendlich lange zurückgestellt werden darf. Es darf also nicht der Fall auftreten, dass auf eine Anfrage gewartet wird, die in Wirklichkeit gar nicht mehr kommt.

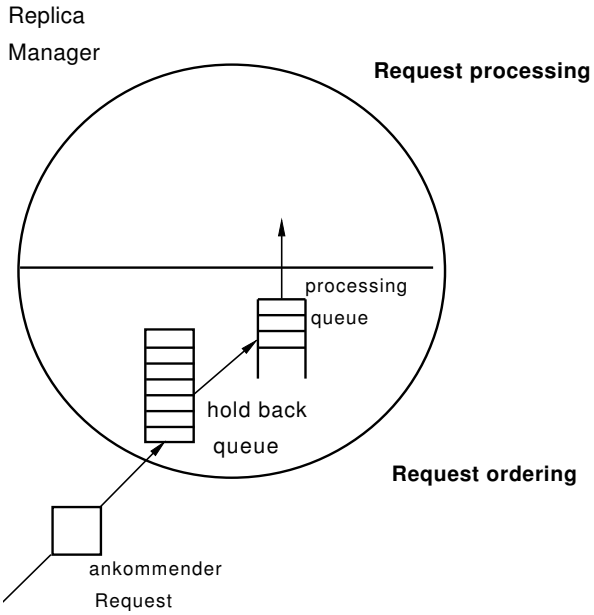


Abbildung 9.12: Die Processing-Queue-Technik

Die Implementierung totaler Ordnungen: Der grundsätzliche Ansatz für die Implementierung totaler Ordnungen ist, dass den Anfragen total geordnete Bezeichner zugeordnet werden, so dass sich jeder Replica Manager basierend auf diesen IDs für die gleiche Ordnung entscheidet.

– **Die Methode des Sequencers**
  Alle Anfragen werden sowohl zum Sequencer als auch zu den Replica Managern gesendet. Der Sequencer weist den Anfragen aufsteigende Bezeichner (in der Reihenfoge des Eintreffens) zu und leitet die IDs zu den Replica Managern weiter, siehe Abbildung 9.13. Kommt bei den Replica Managern eine Anfrage an, so wird diese in der Hold-Back-Queue zurückgehalten, bis sie aufgrund ihrer IDs an der Reihe ist.
  Bei der gestrichelten Alternative in Abbildung 9.13 entsteht weniger Netzlast.
  Es gibt jedoch Probleme, die bei der Methode des Sequencers möglicherweise auftreten können. Es könnte ein Sequencer ausfallen, oder ein Sequencer könnte zum Bottleneck werden, also überlastet sein.

– **Die Verteilte Zuweisung von IDs**
  Im Folgenden soll ein Algorithmus vorgestellt werden, der in den Abbildungen 9.14, 9.15, 9.16 verdeutlicht wird. Dabei sollen 10 Replica Manager gegeben sein, die eine verteilte Zuweisung von IDs ausführen wollen. Wir gehen im Beispiel davon aus, dass eine quasi gleichzeitige Verschickung von Anfragen erfolgt. Angenommen werde, dass erst die Anfragen $\rightarrow$ und dann die Anfragen $\dashrightarrow$ zugestellt werden.
  Der Algorithmus besteht aus drei Schritten:

  Schritt 1 Zunächst sendet das Front End eine Anfrage an seine Replica Manager. Enthalten ist dabei ein temporärer Bezeichner F_{max}, der größer als alle vorangegangenen IDs ist (vgl. Abbildung 9.14).

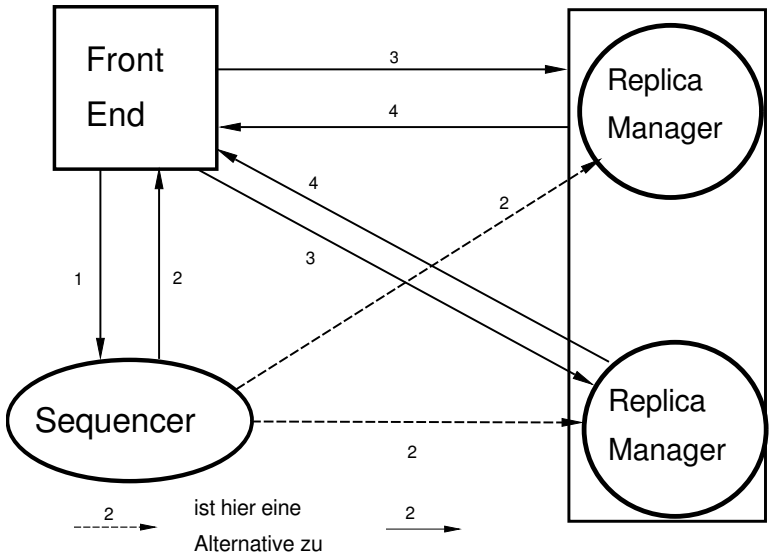


Abbildung 9.13: Die Methode des Sequencers

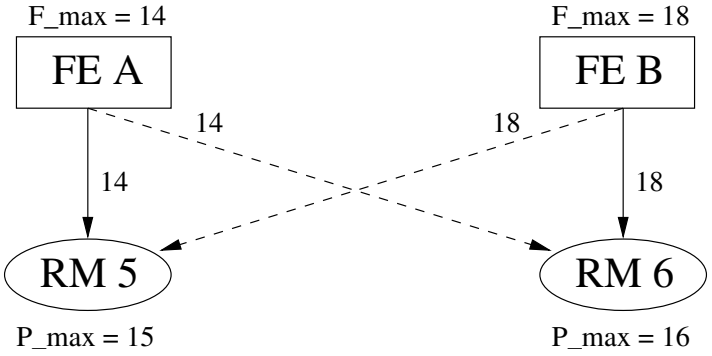
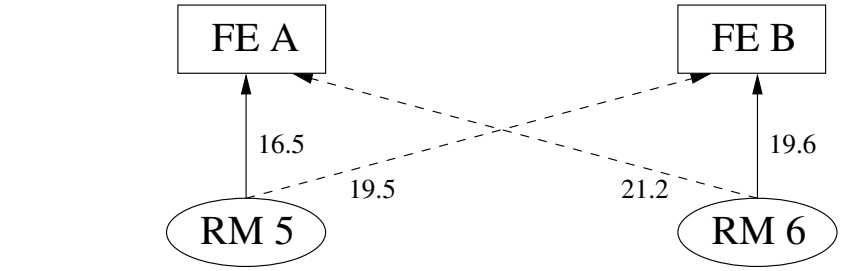


Abbildung 9.14: Die verteilte Zuweisung von IDs - Schritt 1

Schritt 2 Darauf sendet jeder Replica Manager eine Antwort an das Front End mit einer Rückgabe eines neuen Vorschlags ID_{rm} für die ID, wobei
$ID_{rm} = Max(F_{max}, P_{max}) + 1 + \frac{i}{N}$
P_{max} ist dabei der vom Replica Manager selbst vergebene oder gespeicherte eigene größte Bezeichner. N ist die Anzahl von Replica Managern, und i ist die Nummer des jeweiligen Replica Managers. (Mit $\frac{i}{N}$ somit wird eine „Adresse“ des Replica Managers gebildet). Die zu addierende 1 sorgt dafür, dass die vorgeschlagene neue ID auf jeden Fall größer als das bisherige P_{max} ist.
Gemäß der IDs wird die Anfrage in der Hold-Back-Queue gespeichert, wobei die kleinste ID in der Queue vorne steht (vgl. Abbildung 9.15).

Schritt 3 Das Front End sammelt alle ihm vorgeschlagenen IDs und maximiert die Werte, d.h. nimmt den größten der vorgeschlagenen Bezeichner als final ID_{FE} und gibt diese final ID_{FE} an alle Replica Manager zurück. In den Replica Managern wird nun eine Neuordnung gemäß der vorliegenden IDs vorgenommen.



ID_RM5 = Max(14,15) + 1 + 0.5 = 16.5 ID_RM6 = Max(18,16) + 1 + 0.6 = 19.6
ID_RM5 = Max(18,16. 5) + 1 + 0.5 = 19.5 ID_RM6 = Max(14,19.6) + 1 + 0.6 = 21.2

Abbildung 9.15: Die verteilte Zuweisung von IDs - Schritt 2

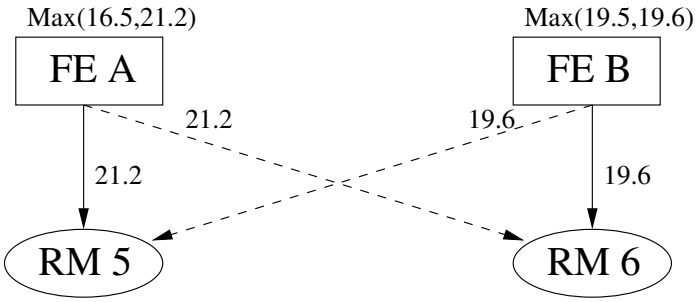


Abbildung 9.16: Die verteilte Zuweisung von IDs - Schritt 3

Bemerkung: Die Liveness- und die Safety-Property sind erfüllt. Der Algorithmus läßt sich geradlinig implementieren, und es gibt keinen Bottleneck oder einen punktuellen Schwachpunkt. Fällt ein Front End aus, so kann ein Replica Manager dessen Funktionalität ggf. übernehmen.
Allerdings ist der Algorithmus teuer (da drei Nachrichten je Replica Manager benötigt werden), insbesondere im Vergleich zum Sequencer-Algorithmus realisiert er eine totale Ordnung, aber nicht notwendigerweise eine kausale Ordnung, da die Anfragen zwar „synchron“, aber in willkürlicher Reihenfolge eintreffen (vgl. Abbildung 9.16).

Die Implementierung kausaler Ordnungen: Im Folgenden soll das Prinzip der Vektoruhren (Vector Clocks) betrachtet werden. Für ein System von *N* Prozessen ist eine Vektoruhr ein Feld von *N* ganzen Zahlen. Jeder Prozess hat dabei seine eigene Vektoruhr, die genutzt wird, um die Zeit lokaler Ereignisse zu stempeln. Wenn Prozesse Nachrichten verschicken, so können Vektorzeitstempel mitgesendet werden.
Für das Aktualisieren der Uhren gibt es folgende Regel:

- Eine Initialisierung erfolgt durch (0,0,..,0)
- Erfolgt im Prozess P_i ein Zeitstempel, so wird die i-te Komponente der Zeitmarke, d.h. des Zeitvektors, um den Wert 1 erhöht.
- Mit jeder verschickten Nachricht wird der Zeitvektor des sendenden Prozesses mitgesendet.

– Beim Empfangen einer Nachricht wird eine Merge-Operation durchgeführt, d.h. das komponentenweise Maximum der Komponenten gebildet.

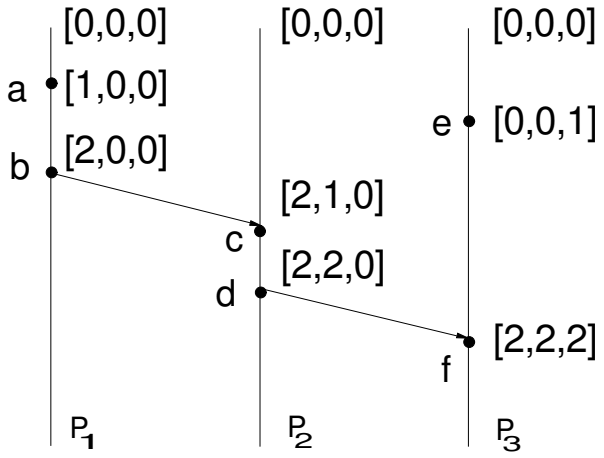


Abbildung 9.17: Vektoruhren

Hierbei gilt:
$V(a) \leq V(b)$, d.h. $a \rightarrow b$, ferner $V(b) \leq V(c), V(c) \leq V(d), V(d) \leq V(f), V(e) \leq f(f)$
Damit folgt: $V(a) \leq V(f)$.
Für $V(c)$ und $V(e)$ kann keine Ordnung angegeben werden, c und e sind damit nebenläufig.

– **Die Methode der Vektor-Zeitmarken (Vector Time Stamps)** Ein Front End darf von der Version eines Replica Managers nur lesen, wenn die Version mindestens so aktuell wie die Version ist, die zuletzt von irgendeinem Replica Manager gelesen und beim Front End gespeichert wurde (also mindestens so aktuell wie die beim Front End gespeicherte Vektor-Zeitmarke).
Ein Front End darf auf einen Replica Manager nur schreiben, wenn der Replica Manager bereits alle (bzgl. der kausalen Ordnung) vorangegangenen Updates berücksichtigt hat. Am Anfang sind alle Vektor-Zeitmarken = $\boxed{0}\boxed{0}\boxed{0}$. Bei einem schreibenden Zugriff wird die entsprechende Komponente um 1 erhöht, also z.B. $\boxed{1}\boxed{0}\boxed{0}$ beim Schreiben auf Replica Manager 1. Nach mehreren Zugriffen sei die Situation z.B. wie in Abbildung 9.18.
Das Front End mit der Zeitmarke $\boxed{2}\boxed{3}\boxed{4}$ liest vom Replica Manager 1.
Dies ist erlaubt, da $\boxed{3}\boxed{4}\boxed{4}$ aktueller ist als $\boxed{2}\boxed{3}\boxed{4}$.
Es findet ein Merging (Auswahl der komponentenweise größten Werte) der Vector Time Stamps statt, d.h. merge (u,o)[k]:=Max(u[k], o[k]) für alle k = 1,...,n, wobei die Vektor-Zeitmarke des Front Ends aktualisiert wird.
In der dadurch entstehenden Situation sollte das Front End keine Daten vom Replica Manager 2 lesen, da die Vektor-Zeitmarke des Front Ends nicht kleiner als die Vektor-Zeitmarke des Replica Manager 2 ist.
Es gibt zwei Probleme zu betrachten: Die Netzteilung und parallele Transaktionen.

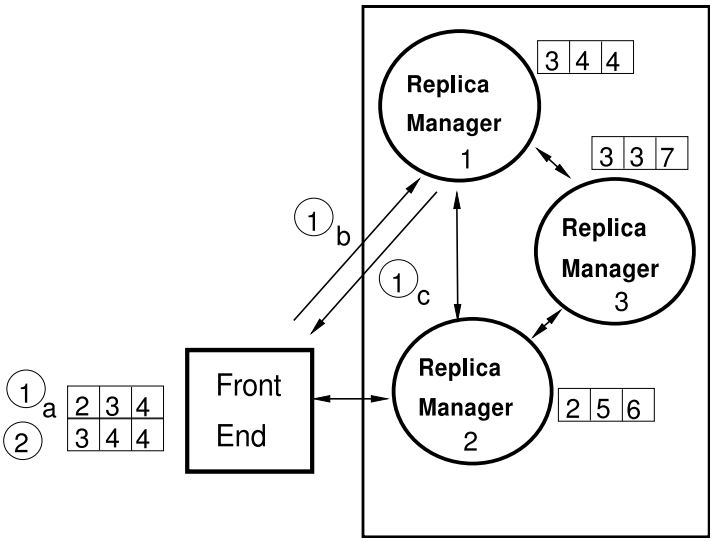


Abbildung 9.18: Die Methode der Vektor-Zeitmarken

Die Netzteilung Eine Netzteilung entsteht durch das Zusammenbrechen der Kommunikation zwischen den Rechnern eines Verteilten Systems, so dass Gruppen entstehen, zwischen denen keine Verbindungen mehr besteht. Diese Gefahr besteht insbesondere bei WANs und beim Mobile Computing, denn je größer bzw. unsicherer das Netz ist, desto mehr Ausfälle sind zu erwarten.
Das Ziel ist es, auch bei einer Netzteilung ein funktionierendes System zu erhalten, d.h. die Korrektheit der Daten der eigenen Gruppe muss vorhanden sein, und keine Operationen dürfen ausgeführt werden, die zu Konflikten mit Operationen der anderen Gruppe führen.
Hierfür gibt es jedoch bislang keine Lösungen, höchstens einige Lösungsansätze.

Parallele Transaktionen Dieses Problem tritt z.B. auf, wenn zwei Kunden unterschiedlicher Reisebüros jeweils den letzten Sitzplatz eines Flugzeuges reservieren lassen wollen.
Lösungen dieses Problems haben gleichermaßen mit der Datenreplikation und der Concurrency Control zu tun.