Kapitel 1: Einführung und Überblick -
Anwendungen, Systeme, Prinzipien

1.1 Anwendungen

Die Vorlesung vermittelt grundlegende Kenntnisse über die Informatikmethoden zum Entwurf, der Implementierung und dem Betrieb moderner Informationssysteme, insbesondere Konzepte und Schnittstellen von Datenbanksystemen und anderen Arten von Informationsdienstsoftware (z.B. Search-Engines oder Data-Mining-Tools) sowie Einblicke in Anwendungsentwicklungsmethoden. Die folgenden Beispiele nennen einige typische Informationssysteme, die von außerordentlich hohem wirtschaftlichem und gesellschaftlichem Nutzen sind:

- *Flugreservationen und -buchungen einer Fluggesellschaft*. Derartige Systeme verwalten typischerweise einige Millionen Reservationen und Buchungen (bis zu einem Jahr im voraus) und müssen in Spitzenzeiten bis zu 100 Such- und Änderungsoperationen pro Sekunde ausführen können. Darüber hinaus müssen solche Systeme praktisch rund um die Uhr an 365 Tagen im Jahr betriebsbereit sein. Eine Ausfallzeit von mehr als wenigen Minuten im Jahr gilt bereits als unakzeptabel.
- *Verwaltung eines Teilelagers eines Automobilherstellers*. Dies umfaßt u.U. mehrere Millionen Teile für alle Modelle, für die der Hersteller die Lieferung von Ersatzteilen garantiert. Das System dient u.a. zur Buchführung über die Lagerbestände und zum Auffinden der Teile in roboterbedienten Lagerhallen. Zusätzlich gibt das System Auskunft darüber, welche Teile für welche Modelle verwendet werden können, welche Teile untereinander kompatibel sind, usw.
- *Entscheidungsunterstützung für den Wertschriftenhandel einer Bank*. Börsenkurse und damit zusammenhängende Informationen (z.B. Wirtschaftsnachrichten) werden über entsprechende Datenleitungen von verschiedenen Börsen kontinuierlich erfaßt, und es werden komplexe Suchoperationen auf den Daten spezifiziert, die die Händler auf interessante Konstellationen aufmerksam machen sollen. Dies ist ein Beispiel eines Echtzeit-Informationssystems mit harten Anforderungen an die Antwortzeit der Suchoperationen und extrem hohen Änderungsraten der Daten (z.T. mehr als 100 Kurse pro Sekunde).
- *Verwaltung von Patientendaten in einem Krankenhaus*. Dies beinhaltet Informationen über die Krankengeschichte eines Patienten, Untersuchungsergebnisse, Einsatz von Medikamenten, Behandlungen, bis hin zur Kostenabrechnung. Über die für administrative Zwecke notwendigen Such- und Änderungsoperationen hinaus ist u.a. die Unterstützung der Suche nach ähnlichen Krankheitsbildern wünschenswert. Zusätzlich wird die Anbindung von Archivsystemen für Ultraschall- und Röntgenbilder, Computertomographien u.ä. angestrebt.
- *Informationssuche für Zeitungsredakteure*: In der Redaktion einer Zeitung; Zeitschrift oder Nachrichtenagentur werden riesige Mengen an Nachrichten inkl. Photos, Videomaterial, etc. archiviert. Auf diesen Daten suchen Redakteure und Journalisten nach relevantem Hintergrund- oder Vergleichsmaterial zu einer aktuellen Meldung oder Story. Die Suche soll zum einem Ergebnisse liefern, die zur Anfrage textuell ähnlich sind, aber darüber hinaus auch inhaltliche Ähnlichkeit berücksichtigen, die sich auf linguistische Analysen und Begriffsverwandtschaften aus sog. Thesauri oder Ontologien abstützt. Beispielsweise sollte eine Anfrage wie „Sommer 2000 im Saarland“ auch einen Artikel finden, in dem „1. August 2000:

Marienwunder in Marpingen“ steht. Die Daten selbst können dabei reine Textdokumente sein, zunehmend werden aber auch sog. semistrukturierte Datenformate wie XML verwendet, wobei z.B. Zeit- und Ortsangaben mit entsprechenden „Tags“ explizit markiert sind.

- *Verwaltung von Kundenanfragen in einem Customer-Support-System*: Bei Unternehmen mit vielen verschiedenen Kunden und wartungsintensiven Produkten bzw. Dienstleistungen, wie etwas einer großen Softwarefirma, einem Internet-Händler für Unterhaltungselektronik oder einem Internet-Service-Provider, müssen Problemberichte („Bug Reports“) und Anfragen von Kunden verwaltet werden. Eine typische Anfrage, die per Email, Web-Interface oder Call-Center eintrifft, könnte folgendermaßen lauten: “My notebook, which is model … configured with …, has a problem with the driver of its WaveLAN card. I already tried the following fixes … but only received error messages …” Aus einem solchen Text versucht man, möglichst viele strukturierte Attribute wie z.B. NotebookModel zu extrahieren. Auf der Basis dieses strukturierten Teils, aber auch des verbleibenden reinen Textteils versucht man, die Anfrage automatisch zu klassifizieren, z.B. einem Thema wie /notebooks/drivers/network innerhalb einer Taxonomie von Kategorien zuzuordnen. Dies ist entweder die Grundlage für die Zuteilung des Problems an geeignet qualifizierte Techniker oder hilft bei der Suche nach inhaltlich ähnlichen, früheren Problemberichten, für die bereits eine Lösung bekannt ist. Idealerweise könnte man aus einer großen Sammlung völlig unterschiedlich formulierter Kundenanfragen eine Sammlung von FAQs (Frequently Asked Questions) aufbauen und neue Anfragen automatisch darauf abbilden. Schwierige oder völlig neuartige Anfragen, für die es keine sinnvolle Abbildung gibt, erfordern eine tiefergehende Problemanalyse sowie ggf. einen Dialog mit dem Kunden und werden als Workflow im Customer-Support-System verwaltet.

Viele Anwendungen haben heute Web-basierte Benutzerschnittstellen und sind auch in ihrer Systemarchitektur stark mit Web-Technologien verknüpft. Ein weiteres zentrales Thema ist die Interoperabilität zwischen verschiedenen, weitgehend unabhängigen Datenbanken und sonstigen Informationsquellen untereinander sowie mit den verschiedenen Anwendungssystemen vor dem Hintergrund einer verteilten, hochgradig heterogenen Informationslandschaft. Ziel ist es, den Anwendungen einen systemübergreifenden Datenzugriff auf möglichst bequeme Art zu ermöglichen. Im folgenden sind drei Beispielszenarios aufgeführt, für die diese Forderung essentiell ist:

- Integrierte Auswertung verschiedenartiger Informationsquellen zur intelligenten Steuerung des Straßenverkehrs. Die Grundlage solcher Leitsysteme bilden einerseits geographische Daten und Daten über das Straßennetz sowie andererseits ständig aktualisierte Daten über die Verkehrsdichte, die Straßenbeschaffenheit, die Wetterlage und -vorhersage, Auslastung von Parkhäusern usw.
- Integrierte und weitestgehend automatisierte Abwicklung von Vorgängen, sogenannten „Workflows“, die eine Vielzahl von Informationssystemen verschiedener Institutionen betreffen. Beispielsweise beinhaltet der Erwerb eines Einfamilienhauses Prüfungen und Eintragungen beim Grundbuchamt, der finanzierenden Bank, dem Notar usw. Gleichzeitig sollten auch alle mit dem Umzug verbundenen Formalitäten wie beispielsweise die Ab- und Anmeldung von Gas/Wasser/Strom, Telefon, Internetanschluß usw. elektronisch abgewickelt werden.
- Prüfung von Firmenkrediten bei einer Bank mit entsprechenden Bonitätsprüfungen und Risikoabschätzungen bezüglich des Kreditnehmers sowie seiner finanziellen Verflechtungen mit anderen nationalen und internationalen Unternehmen. Dies beinhaltet die Unterstützung ver-

teilter Arbeitsabläufe, die sich über verschiedene Geschäftssparten und verschiedene internationale Niederlassungen einer Bank erstrecken und Zugriff auf ein breites Spektrum von Informationssystemen erfordern - von Wirtschaftsnachrichten und Börsenkursen bis zu Informationen über die Aufsichtsratsmitglieder des potentiellen Kreditnehmers.

Ein moderner Trend ist der Einsatz generischer (d.h. für breite Einsatzfelder konzipierte) Anwendungsklassen wie etwa

- Enterprise Resource Planning (z.B. SAP R/3) für die umfassende – operative und strategische – Führung von Unternehmen,
- E-Commerce und E-Business in den Spielarten B2C (Business-to-Consumer, z.B. Handel, Auktionen, Maklerdienste) und B2B (Business-to-Business, z.B. Logistikketten, Service-Outsourcing) oder
- digitale Bibliotheken und multimediale Archive (z.B. Patentsammlungen, Nachrichtenarchive, virtuelle Museen).

Hierfür gibt es hochgradig „parametrisierte“ und flexibel anpaßbare Softwarelösungen, die für den individuellen Einsatzfall konfigurierbar und – im Sinne eines „Customizing“ – spezialisierbar sind sowie durch weitere Software ergänzt werden können.

Einige interessante Web-basierte Anwendungen

http://www.imdb.com - Internet Movie Database für Filmfans

http://www.amazon.com - E-Commerce-System zum Verkauf von Büchern, CDs, etc.,
Textinformationssystem mit Buchrezensionen und Data-Warehouse mit Kundenprofilen

http://www.ebay.com - Auktionssystem für Artikel aller Arten

http://www.terraserver.com bzw. http://www.terraserver.microsoft.com –
Geographisches Informationssystem mit Landkarten und Satellitenbildern

http://skyserver.sdss.org/dr1/en/ - Himmelsatlas mit astronomischen Karten und Daten

http://www.google.com und http://vivisimo.com/ - Suchmaschinen für das Web

http://www-db.stanford.edu/IMAGE und http://129.132.14.36/Chariot - Ähnlichkeitssuche auf Photos

http://www.hermitagemuseum.org - Virtuelles Museum mit Ähnlichkeitssuche auf Bildern

http://www.musicline.de/de/melodiesuche/input und http://www.name-this-tune.com/index.php?id=9&L=2
Ähnlichkeitssuche auf Musikstücken („Query by Humming“, „Query by Whistling“)

http://chembank.med.harvard.edu
Chemische Datensammlung mit Ähnlichkeitssuche auf Molekülen

http://www.eti.uva.nl/Database/WBD.html
Datenbank mit organischen Spezies

http://au.expasy.org/srs5/
Proteindatenbanken

http://www.wikipedia.org
Online-Enzyklopädie, die von Internet-Benutzern der ganzen Welt freiwillig gepflegt wird

Technische Anforderungen an Informationssysteme

- Textsuche mit nach Relevanz geordneten Trefferranglisten
- Automatische Organisation von Dokumenten
- Verwaltung von (Deep-)Web-Daten, strukturierten Datenbanken und semistrukturierten (XML-)Daten
- Komfortable GUIs (Graphical User Interfaces)
- deklarative Anfragesprachen
- Zuverlässige Verwaltung sehr großer, persistenter Datenmengen (> 5 Terabytes)
- Hohe Verfügbarkeit (7 x 24 Stunden in der Woche) und langfristige Archivierung
- Gute - vorhersagbare und garantierbare - Leistung
    - Hoher Durchsatz (Anzahl der bedienbaren Aufträge pro Zeiteinheit, Anzahl der gleichzeitig aktiven Clients)
    - Kurze, benutzerakzeptable Antwortzeiten im Mehrbenutzerbetrieb („Quality of Service“)
- Konsistenz verteilter Daten
- Integrierter Zugriff auf heterogene Daten
- Daten-Mining nach Korrelationen, Regeln, Klassifikationen, etc.
- Aktive Regeln und Prozesskoordination
- Komplexe Datentypen (Landkarten, Satellitenbilder, Himmelskarten, Moleküle, usw.)
- Ähnlichkeitssuche auf Bildern, wissenschaftlichen Daten, etc.
- Integration mit Web-Applikation und Web-Services

1.2 Systemarchitekturen

Drei-Ebenen-Architektur

Die typische Systemarchitektur moderner Informationssysteme ist in Abbildung 1.1 dargestellt. Sie wird in der Regel als Drei-Ebenen-Architektur (engl. „three-tier architecture" oder noch allgemeiner "multi-tier architecture") bezeichnet und enthält eine Zwei-Ebenen-Architektur als Spezialfall. Die Architektur umfaßt *Clients* (PCs, Workstations, Notebooks, Palmtops, Terminals, digitale TV Set-Top-Boxes, Handies und etliche weitere Arten von "Gizmos" und intelligenten Sensoren und Aktoren). Die Clients kommunizieren mit einem *Applikations-Server* (oder mehreren solcher Server), die wiederum mit einem *Daten-Server* (oder mehreren solcher Server) kommunizieren.

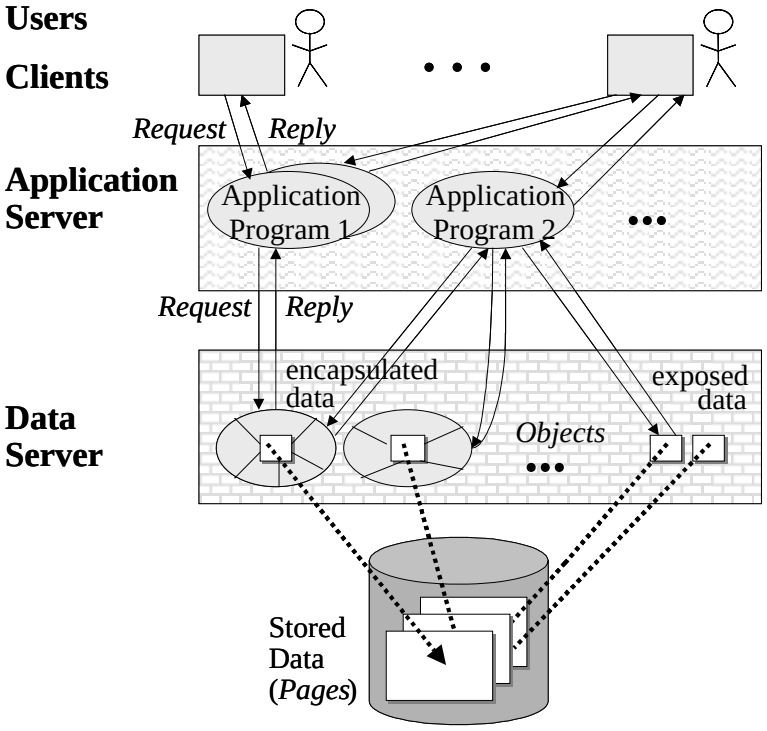

Abbildung 1.1: Drei-Ebenen-Architektur eines Informationssystems

Clients schicken anwendungsorientierte Aufträge (zur Erfüllung eines Geschäftsziels) an den Applikations-Server. Dies kann z.B. der Auftrag zu einer Bankkontogutschrift oder –lastschrift im Home-Banking sein, das Starten einer E-Commerce-Sitzung zum Füllen eines elektronischen Warenkorbs oder das Initiieren eines Workflows zur Planung und Buchung einer individualisierten Reise. In modernen Anwendungen werden diese Aufträge typischerweise mittels eines GUI (Graphical User Interface) generiert; analog wird die Antwort des Applikations-Servers häufig ebenfalls in graphischer Form präsentiert mittels Formularen, Diagrammen oder sogar Animationen in Virtual-Reality-Szenarien. Die Verarbeitung der Präsentation findet sowohl bei der Eingabe als auch bei der Ausgabe (primär) im Client statt. HTML (Hypertext Markup Language), das meistverbreitete Datenformat

des World-Wide-Webs, ist besonders attraktiv als Präsentationsgrundlage, weil zur Verarbeitung auf Client-Seite in der Regel ein Web-Browser genügt, so daß eine riesige Anzahl potentieller Clients ohne besonderen Installationsaufwand mit dem Informationssystem arbeiten kann.

Der Applikations-Server verwaltet einen Pool von Anwendungsprogrammen in ausführbarer (also compilierter oder interpretierbarer) Form und ruft das für einen Client-Auftrag zuständige Programm in seinem Adressraum auf. Programme können in einer konventionellen Programmiersprache wie z.B. C oder Cobol geschrieben sein oder in einer Skriptsprache wie z.B. PHP, mit der sehr einfach dynamische HTML-Seiten erzeugt werden können.

Die Anzahl der in einem Applikations-Server verwalteten Programme und zu unterstützenden Typen von Client-Aufträgen kann sehr groß sein, so daß es sich anbietet, mittels des objektorientierten Paradigmas Struktur in den Gesamtpool der Programme zu bringen. Dies führt auf sog. *Komponenten* oder *Business-Objekte* als Verwaltungseinheiten im Applikations-Server. Solche Objekte sind z.B. Konten, Kunden, Warenkataloge u.ä., die nach dem Prinzip Abstrakter Datentypen (ADTs) gekapselt sind, so daß ihre Implementierung verborgen bleibt und die Objekte nur über eine möglichst schmale Schnittstelle spezifischer Methoden (z.B. Gutschrift, Lastschrift, Kontoauszug, etc.) manipuliert werden können. Für ihre Implementierung dürfen Business-Objekte selbst wieder Methoden anderer Objekte aufrufen, so daß Komponenten-Code möglichst wiederverwendet wird. Die Kapselung wiederum vermeidet Abhängigkeiten von den Interna anderer Objekte, so daß der langfristige Wartungsaufwand für die Software geringer wird. Die Implementierung der Methoden eines Business-Objekts beinhaltet typischerweise einen oder mehrere Aufrufe an einen oder mehrere Daten-Server. Ein Beispiel eines Informationssystems, dessen Applikations-Server-Programme in Business-Objekten organisiert sind, ist das ERP-System (Enterprise Resource Planning) SAP R/3, eine umfassende Softwaresuite für die betriebswirtschaftlichen Belange eines Unternehmens.

Der Applikations-Server bildet die Laufzeitumgebung der Anwendungsprogramme: er erzeugt entsprechende Threads oder Prozesse, überwacht die Programmausführungen und behandelt bestimmte Fehler- und Ausnahmesituationen. Clients kommunizieren mit dem Applikations-Server im Web-Zeitalter in der Regel mittels HTTP (Hypertext Transport Protocol), das wiederum auf TCP/IP-Verbindungen aufsetzt. Da HTTP ein zustandsloses Protokoll ist (der Empfänger eines Auftrags vergisst den Auftrag nach der Ausführung und dem Absenden der Antwort), Clients aber häufig ganze Interaktionsfolgen als eine Einheit betrachten, ist der Applikations-Server auch für die Realisierung von Sitzungen (engl. sessions) zuständig: er muß sich entsprechende Zustandsinformationen über die Sitzungen aktiver Clients merken, Sitzungen überwachen und – nach Timeouts - ggf. zwangsweise terminieren. Insgesamt kann man die Rolle des Applikations-Servers als die eines Auftragsmaklers (engl. request broker) bezeichnen.

Solche Auftragsmakler und Laufzeitsysteme für Anwendungsprogramme wurden früher als *TP-Monitore* (engl. transaction processing monitors) bezeichnet (z.B. CICS, Tuxedo, etc.), weil sie primär für sog. OLTP-Anwendungen (online transaction processing) wie Reservations- und Buchungssysteme konzipiert waren. Eine modernere Variante davon sind die sog. "*Object Request Broker*" (ORBs), die die Infrastruktur für allgemeine verteilte Anwendungen bereitstellen und auf Komponentenmodellen wie CORBA (Common Object Request Broker Architecture), COM+ (Component Object Model) oder EJB (Enterprise Java Beans) basieren (z.B. VisiBroker, Orbix, etc.). Eine letzte Kategorie von Applikations-Servern sind *Web-Server* (wie z.B. Apache oder Internet Information Server), also im wesentlichen eine Kombination aus HTTP-Server und Servlet-Engine, wobei Servlets die Web-Variante von Business-Objekt-Methoden sind, die unter der Kontrolle des Applikations-Servers laufen. Wenn man schließlich den Kontrollfluß der Objektmethodenaufrufe auf langlebige (Geschäfts-) Prozesse wie z.B. den gesamten Zyklus eines Warenkaufs von der Bestellung bis

zur Mahnung ausdehnt und die Koordination dieser Zusammenhänge im Applikations-Server realisiert, spricht man auch von einer *Workflow-Engine* oder einem *Workflow-Management-System* (z.B. MQ Series Workflow, Staffware, etc.). Alle diese Varianten faßt man auch unter dem Begriff *Middleware* zusammen. Die Unterschiede zwischen den verschiedenen Kategorien von Infrastruktursoftware sind konzeptionell eher nebensächlich und nur hinsichtlich der Terminologie und Features von Produkten signifikant.

Wie erwähnt beinhaltet die Implementierung von Business-Objekten in der Regel Aufrufe eines Daten-Servers oder mehrerer solcher Server (die man im Kontext einer Drei-Ebenen-Architektur auch als "Backends" bezeichnet). Typischerweise werden alle persistenten, d.h. die Dauer einer Sitzung überlebenden, Zustandsinformationen im Daten-Server verwaltet. Datenbank-Server (z.B. Oracle, IBM DB2 oder MS SQL Server) sind mit Abstand die wichtigste Daten-Server-Kategorie, aber auch spezialisierte Textdokument- oder Multimedia-Systeme (z.B. Google oder Verity) oder Mail-Server (z.B. MS Exchange oder Lotus Domino) können von Business-Objekten aufgerufen werden. Zur Kommunikation mit einem Datenbank-Server wird die standardisierte Sprache SQL (Structured Query Language) verwendet, wobei SQL-Kommandos und -Resultate oft in ein Datenverbindungsprotokoll wie ODBC (Open Database Connectivity) oder JDBC (Java Database Connectivity) eingebettet sind. Viele Datenbanksysteme unterstützen auch sog. Stored Procedures, mit denen sich Abstrakte Datentypen bzw. Business-Objekte im Datenbank-Server selbst realisieren lassen; damit hat man Flexibilität, inwieweit man die Kapselung von Business-Objekten im Applikations-Server und/oder im Daten-Server realisiert.

Zwei-Ebenen-Architektur

Durch Weglassen des Applikations-Servers, also der mittleren Ebene einer Drei-Ebenen-Architektur erhält man eine Spezialisierung, die als *Client-Server-Architektur* bezeichnet wird und historisch vor der mit der Web-Revolution einhergehenden Verbreitung der Drei-Ebenen-Architektur die häufigste Informationssystemarchitektur war. Wenn man dabei die Business-Objekte der Anwendung im Client realisiert, hat man sozusagen "fette Clients" (engl. "fat clients"); wenn man sie im Datenbank-Server mittels Stored Procedures o.ä. realisiert, hat man "dünne Clients" (engl. "thin clients"), unter die der historisch noch ältere Ansatz fällt, als Clients einfache Terminals zu verwenden. Der Hauptgrund, der zu der heute dominierenden Drei-Ebenen-Architektur geführt hat, liegt in der Anforderung der Skalierbarkeit, also der Möglichkeit, das System auf einfache Weise für eine größer werdende Anzahl von Clients aufzurüsten. Bei einer Zwei-Ebenen-Architektur kann insbesondere die große Anzahl von Client-Server-Sitzungen leicht zu Engpässen im Datenbank-Server führen. Bei einer Drei-Ebenen-Architektur dagegen erreicht man die Skalierbarkeit einfach durch Replikation des Applikations-Servers, wobei im Internet-Einsatz alle Applikations-Server durch entsprechende Router unter derselben IP-Nummer bzw. URL erreichbar sind.

Föderative Systemstrukturen

Anspruchsvolle Informationssysteme können Daten aus verschiedenen Informationsquellen integrieren, so daß ihre Business-Objekte mit mehr als einem Daten-Server kommunizieren. Außerdem ist es möglich, daß der oder die Applikations-Server ihrerseits die Dienste eines anderen Applikations-Servers oder mehrerer Applikations- und Daten-Server in Anspruch nehmen. Alle diese Möglichkeiten können frei kombiniert werden. Dabei sind die beteiligten Server häufig heterogen und autonom in dem Sinne, daß sie nicht exklusiv für eine Anwendung arbeiten, sondern vielmehr sowohl "lokale" Aufträge bedienen als auch ihre Dienste in einem Systemverbund anbieten. Solche

verteilten Systemstrukturen mit lose gekoppelten Server-Systmen nennt man daher auch föderative Informationssysteme. Abbildung 1.2 illustriert diesen Fall. Ein Praxisbeispiel eines solchen Dienstes ist das "virtuelle" Reisebüro Expedia, das intern sowohl "lokale" Daten-Server hat als auch mit autonom betriebenen anderen Diensten wie Amadeus, Sabre und weiteren Großhändlern der Reisebranche kommuniziert

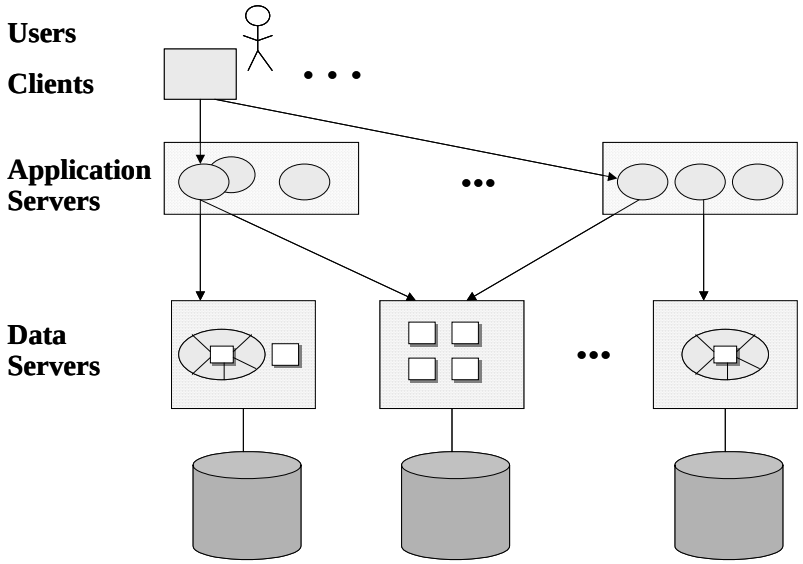

Abbildung 1.2: Föderative Systemarchitektur

Föderative Informationssysteme benötigen offensichtlich eine besonders zuverlässige und zugleich leicht wartbare - "offene" - Kommunikationsinfrastruktur, die über die einfachen Protokolle des Internet wie TCP/IP oder HTTP hinausgeht. Dafür sind die bereits erwähnten Varianten komponentenorienter Middleware geeignet, wie z.B. CORBA, EJB, .Net und sogenannten Web Services. Die Middleware verbirgt weitgehend die Heterogenität der Systemlandschaft und dient als Auftragsmakler für verteilte Programmaufrufe (engl. remote method invocation).

1.3 Grundprinzipien von Datenbanksystemen

Datenbanksysteme bilden das Softwarerückgrat der meisten computergestützten Informationssysteme. Sie stellen dem Anwendungsentwickler mächtige, generische Operationen zum Suchen in sehr großen Datenbeständen sowie zum Einfügen, Ändern und Löschen von Daten zur Verfügung. Operationen werden in einer deklarativen Datenmanipulationssprache spezifiziert, meist in der standardisierten Sprache SQL (Structured Query Language), und die Operationsaufrufe werden in Anwendungsprogramme eingebettet, die in einer konventionellen Programmiersprache (z.B. C, C++, Java, Cobol) oder einer Skriptsprache (z.B. Perl, PHP, Visual Basic, PL/SQL) geschrieben sind.

Aus den aufgeführten Anwendungen geht klar hervor, daß Datenbanksysteme u.U. riesige Datenmengen zu verwalten haben. Typische Datenbankgrößen in kommerziellen Anwendungen liegen im Bereich zwischen 10 und einigen 100 Gigabytes, sehr große Datenbanken umfassen mehr als 1 Terabyte und die allergrößten Datenbanken werden bald an die Petabyte-Grenze stoßen. Daraus folgt zwangsläufig, daß Datenbanken in erster Linie auf Sekundärspeichermedien liegen, in aller Regel auf Magnetplatten. Zwischen Hauptspeicher und Magnetplatten aber klafft ein riesiger Geschwindigkeitsunterschied von 5 bis 6 Zehnerpotenzen. Wenn wir uns beispielsweise die Geschwindigkeit verschiedener Speicherhierarchiestufen als eine räumliche Entfernung der Daten vom Prozessor vorstellen und annehmen, daß der CPU-Cache im selben Raum wie der Prozessor, also vielleicht 1 Meter entfernt, ist, dann befindet sich der Hauptspeicher im Nebenraum (ca. 10 Meter entfernt), die Magnetplatten aber stehen 1000 bis 10000 Kilometer entfernt in Moskau oder gar in Tokio. Aus diesem gewaltigen Geschwindigkeitsunterschied sowie aus der Nebenbedingung, daß Daten auf Magnetplatten nur blockweise adressiert werden können, ergibt sich die Notwendigkeit, daß Datenbanken anders als mit klassischen, hauptspeicherorientierten Datenstrukturen organisiert werden. Diese Überlegung führt auf das folgende Grundprinzip.

> *Grundprinzip 1:*
> Um große Datenmengen effizient verwalten zu können, sind Datenbanksysteme im Hinblick auf *Sekundärspeicherzugriffe* optimiert.

Insbesondere sind auch die klassischen, RAM-orientierten Modelle der Komplexitätstheorie für Datenbanksysteme nur von untergeordneter Bedeutung; die entscheidende Effizienzmetrik ist vielmehr die Anzahl der Sekundärspeicherzugriffe.

Da Datenbanksysteme generische Systeme sind, also gleichermaßen in Bankkonten wie auch in Patientendaten suchen und ändern können, sind universelle Optimierungen der Systemeffizienz nur eingeschränkt möglich. Vielmehr hängt der Nutzen einer Optimierungsmaßnahme häufig von den spezifischen Lastmerkmalen der jeweiligen Anwendung ab. In einer Anwendung mit geringer Änderungsrate kann man Daten - ggf. unter Einführung von Redundanz - so speichern, daß verschiedenartige Suchoperationen extrem effizient ausgeführt werden können; in einer Anwendung mit hoher Änderungsrate hingegen kann dies unakzeptabel sein, wenn die Aktualisierung der entsprechenden Datenstrukturen zu aufwendig ist. Aus diesem Grund unterstützt ein Datenbanksystem in der Regel ein breites Spektrum verschiedener Speicherungs- und Zugriffsstrukturen und überläßt es einem Tuning-Spezialisten, diese Strukturen für eine Anwendung festzulegen. Da sich aber die Lastmerkmale einer Anwendung im Laufe der Zeit ändern (z.B. durch Erweiterung der Anwendungsfunktionalität), müssen diese Tuning-Entscheidungen hin und wieder revidiert werden.

Solche Änderungen der Speicherungs- und Zugriffsstrukturen sind nicht gerade eine billige Maßnahme, da dazu häufig große Datenmengen vom Datenbanksystem umgespeichert werden müssen. Von fundamentaler Bedeutung ist jedoch, daß bei einer solchen Revision der Datenspeicherung die Anwendungsprogramme auf keinen Fall geändert werden müssen, denn der entsprechende Änderungsaufwand wäre kostenmäßig in den meisten Fällen absolut unakzeptabel. Datenbanksysteme stellen diese sogenannte *Programm-Daten-Unabhängigkeit* (häufig auch nur: Datenunabhängigkeit) sicher, indem sie die eigentlichen Speicherungs- und Zugriffsstrukturen vor dem Anwendungsprogramm verbergen. Mit anderen Worten: Datenbanksysteme wenden das Prinzip der Abstrakten Datentypen auf generische Weise an, indem sie die verwalteten Daten kapseln und nur deskriptive Operationen anbieten, die unabhängig von den zugrundeliegenden konkreten Datenstrukturen sind. In eingeschränkter Form gilt dieses Prinzip sogar auch bei einer anwendungsbedingten Änderung des Datenformats (z.B. der Umstellung der Postleitzahlen in Deutschland) sowie bei der nachträglichen Erweiterung existierender Daten um zusätzliche Informationen (z.B. der Hinzunahme einer Kreditkartennummer für alle Privatpatienten einer Klinik).

> *Grundprinzip 2:*
> Die Speicherung von Daten kann geändert werden, ohne daß Anwendungsprogramme geändert werden müssen (*Programm-Daten-Unabhängigkeit*).

Dieses fundamentale Prinzip hat ganz entscheidend zum kommerziellen Erfolg von Datenbanksystemen beigetragen. Etliche Informationssysteme, die ohne echtes Datenbanksystem entwickelt wurden (also beispielsweise auf der Basis eines File-Systems), sind mittelfristig gescheitert, weil sie aufgrund fehlender Programm-Daten-Unabhängigkeit unakzeptable Programmwartungskosten verursachten.

Das Grundprinzip der Programm-Daten-Unabhängigkeit und die damit einhergehende Konzeption deskriptiver Datenmanipulationssprachen wurde vor allem durch die Entwicklung *relationaler Datenbanksysteme* in den Siebziger und Achtziger Jahren entscheidend vorangebracht. Bei diesen Systemen, die heute den Markt dominieren, werden alle Daten konzeptionell in Form von Tabellen (mathematisch: Relationen), repräsentiert. Einen Schritt weiter geht man seit einigen Jahren mit *objektorientierten Datenbanksystemen (*bzw. entsprechenden „postrelationalen“ Weiterentwicklungen relationaler Systeme). Diese Systemklasse erlaubt es insbesondere, Aggregationsbeziehungen zwischen Datenelementen in Form „komplexer Objekte“ direkt zu repräsentieren; beispielsweise können alle Teile eines Motors zu einem Objekt „Motor“ zusammengefaßt werden -- eine Form der Aggregation, die mit relationalen Systemen nur auf umständliche Art möglich ist. Generische Such- und Änderungsoperationen werden sowohl auf der Ebene der Gesamtobjekte als auch auf der Ebene der Teilobjekte angeboten. Darüber hinaus ist es in objektorientierten Datenbanksystemen auch möglich, anwendungsspezifische Operationen für eine Menge von Objekten zu definieren, und diese können in Ausdrücken der Datenmanipulationssprache verwendet werden. Das Datenbanksystem, welches an sich als generisches Softwarepaket konzipiert ist, wird damit anwendungsspezifisch erweiterbar. Beispielsweise kann man eine Funktion „Volumen“ für geometrische Objekte definieren, die auf Polygonen für die Begrenzungsflächen eines dreidimensionalen Körpers operiert. Wie in objektorientierten Programmiersprachen können diese Funktionen mittels „Vererbung“ auf flexible Art und Weise für speziellere Objekte wiederverwendet werden (z.B. für die speziellen Körper, die beim mechanischen CAD auftreten); dabei können bei Bedarf spezifische Details „überschrieben“, d.h. neu festgelegt werden (z.B. die Interpolationsvorschrift zwischen Stützpunkten eines Polygons).

Das Prinzip der Programm-Daten-Unabhängigkeit sollte auch für objektorientierte Datenbanksysteme unter Einschluß der anwendungsspezifischen Operationen gelten. Allerdings gibt es durchaus noch offene Fragen in diesem Zusammenhang, die Gegenstand aktueller Forschung sind, beispielsweise die Konzeption und effiziente Implementierung objektorientierter Datenmanipulationssprachen und deren Einbettung in objektorientierte Programmiersprachen.

Die Effektivität eines Informationssystems steht und fällt mit der Korrektheit der zugrundeliegenden Daten. Eine Form der Datenkorrektheit, die von Datenbanksystemen sehr weitreichend unterstützt wird, ist die *Konsistenz* der Daten (oder auch: *Integrität* der Daten). Diese fordert, daß bestimmte Zusammenhänge zwischen verschiedenen Datenelementen, die in der realen Welt gelten sollen, auch in der Datenbank gewährleistet werden. Beispielsweise soll das Konto eines Kunden bei einem Versandhaus nur dann belastet werden, wenn eine entsprechende Bestellung ausgeliefert wird. Datenbanksysteme bieten die Möglichkeit an, daß solche anwendungsspezifischen Konsistenzbedingungen spezifiziert werden, und die Systeme überwachen die Einhaltung dieser Bedingungen. Änderungen, die die Konsistenz verletzen würden, werden zurückgewiesen. Damit werden Systeme gewissermaßen „immun“ gegen fehlerhafte bzw. inkonsistente Dateneingaben. Diese Vorgehensweise ist zwar grundsätzlich auf Plausibilitätsprüfungen beschränkt, da auch fehlerhafte Eingaben formal konsistent sein können; in vielen Anwendungen aber können dadurch fast alle Eingabefehler erkannt und eliminiert werden.

Einige Klassen von Konsistenzbedingungen können leichter spezifiziert und vom System überwacht werden, wenn die Datenbank selbst daraufhin entworfen wurde. In diese Klasse fallen beispielsweise Bedingungen vom Typ „keine zwei Konten dürfen dieselbe Kontonummer haben“ oder „die bei einer Buchung angegebene Kontonummer muß in der Datenbank existieren“. Die Datenbankforschung hat verschiedene Methoden des sogenannten konzeptionellen (oder auch: logischen) Datenbankentwurfs hervorgebracht, die in diesem Sinne auf eine hohe Qualitätssicherung für die Daten abzielen. Diese Methoden, z.B. die relationale Entwurfstheorie und vor allem der Entwurf mit „Entity-Relationship-Modellen“, haben in der Praxis weite Verbreitung gefunden. Moderne Datenbanksysteme gehen noch einen Schritt weiter und erlauben es, anwendungsspezifische Konsistenzbedingungen mit den Sprachmitteln der Datenmanipulationssprache deskriptiv zu spezifizieren. Mit Hilfe sogenannter *aktiver Datenbankmechanismen (*z.B. „Trigger“ und weitergehende Konzepte) ist es ferner möglich, spezifische Reaktionen beim Erkennen einer potentiellen Konsistenzverletzung festzulegen; beispielsweise können anstatt einer Zurückweisung der Änderung bestimmte Folgeänderungen automatisch ausgelöst werden. Diese Delegation der Verantwortung für die Datenkonsistenz an das Datenbanksystem stellt eine fundamentale Vereinfachung für die Anwendungsentwicklung dar.

> *Grundprinzip 3:*
> Die *Konsistenz* der Daten wird vom Datenbanksystem gewährleistet.

Die Konsistenz der Daten ist nicht nur durch Eingabefehler gefährdet, sondern auch durch „Interferenzeffekte“ paralleler Änderungsoperationen im Mehrbenutzerbetrieb sowie durch Fehler in Anwendungsprogrammen und Ausfälle des Datenbanksystems selbst. Beispielsweise ist es möglich, daß ein Bankkonto „Geld verliert“, wenn etwa zwei Operationen zur Einzahlung von jeweils 100 DM zunächst beide den Kontostand lesen, jeweils 100 DM dazu addieren und dann beide denselben Wert in die Datenbank zurückschreiben; die zuerst schreibende Einzahlung wird von der späteren fälschlicherweise überschrieben und geht praktisch verloren. Vermeiden ließe sich dieser Fehler na-

türlich durch entsprechende Synchronisationsmaßnahmen (z.B. unter Verwendung von Semaphoren) von Seiten der Anwendungsprogramme; dies aber wäre eine erhebliche Komplikation der Anwendungsentwicklung für Mehrbenutzersysteme mit einer entsprechenden Einbuße bei der Entwicklungsproduktivität. Ähnliche Fehlereffekte wie der eben beschriebene ergeben sich potentiell auch beim unkontrollierten Abbruch eines Anwendungsprogramms oder einem Ausfall des Datenbanksystems. Beispielsweise würde eine überweisung, die nach dem Belasten des ersten Kontos, aber vor der Gutschrift auf das zweite Konto durch einen Fehler abgebrochen wird, die Datenbank inkonsistent hinterlassen und wie zuvor „Geld verlieren".

Datenbanksysteme stellen alle Mechanismen bereit, um Fehlereffekte dieser Art garantiert zu verhindern. Dazu wird vom Anwendungsentwickler verlangt, daß Folgen von Datenbankoperationen innerhalb eines Anwendungsprogramms zu konsistenzerhaltenden Einheiten, sogenannten *Transaktionen*, zusammengefaßt und explizit spezifiziert werden. Beispielsweise bilden bei einer überweisung die Änderungsoperationen beider Konten zusammen eine Transaktion. Das Datenbanksystem gewährleistet dann die „(virtuelle) Isolation" von Transaktionen im Mehrbenutzerbetrieb, indem es (z.B. für das erste Beispiel) entsprechende Synchronisationsmaßnahmen für parallele Transaktionen automatisch generiert, und die „Atomarität" von Transaktionen, indem die Änderungen unvollständig ausgeführter Transaktionen (wie im zweiten Beispiel) mit Hilfe eines entsprechenden Logbuchs automatisch rückgängig gemacht werden. Mit Hilfe des Logbuchs wird darüber hinaus sichergestellt, daß Änderungen abgeschlossener Transaktionen wirklich „dauerhaft" sind, also weder durch Ausfälle des Datenbanksystems noch durch sonstige Fehler (z.B. Plattenfehler) verloren gehen. Diese systemgarantierten Eigenschaften von Transaktionen werden auch als *ACID-Prinzip* bezeichnet.

> *Grundprinzip 4:*
> Datenbankänderungen werden innerhalb von *Transaktionen* durchgeführt, die atomar (*atomic*), konsistenzerhaltend (*consistency-preserving*), isoliert (*isolated*) und dauerhaft (*durable*) sind.

Die Unterstützung von Transaktionen ist ein Grundpfeiler aller modernen Datenbanksysteme und ein Schlüsselkonzept zur Sicherstellung der Datenkonsistenz. Die dazu notwendigen Synchronisations- und Fehlertoleranzmaßnahmen werden vom Datenbanksystem sowohl für zentralisierte als auch für verteilte Datenbanken realisiert. Wegen seiner Allgemeinheit und Leistungsfähigkeit finden das Transaktionskonzept und die entsprechenden Implementierungstechniken der Transaktionsverwaltung auch in Betriebssystemen sowie in persistenten Programmiersprachen zunehmende Beachtung. Beispielsweise sehen Industriestandards wie X/Open, OMG OTS (Object Transaction Service der Object Management Group), oder EJB (Enterprise Java Beans) Transaktionen als universelles Konzept vor, um atomare Prozesse mit Aufrufen beliebiger verteilter Ressourcenmanager zu realisieren.

Der Stand der Technik auf dem Datenbankgebiet läßt sich folgendermaßen zusammenfassen. Datenbanksysteme sind ausgereifte Systeme, ohne die zahlreiche computergestützte Informationssysteme undenkbar wären. Der Markt wird heute von relationalen Systemen dominiert; objektorientierte Systeme haben einen kleinen Marktanteil, aber viele Anbieter relationaler Systeme haben objektorientierte Konzepte in ihre Produkte integriert und nennen diese dann „objekt-relational". Aktuell sind viele Hersteller im Begriff, Unterstützung für den vom W3C (World Wide Web Consortium) zum universellen Datenaustauschformat erklärten XML-Standard (Extensible Markup Language) in ihre Produkte einzubauen.

1.4 Grundprinzipien von Suchmaschinen

Suchmaschinen für das Web, Intranets oder digitale Bibliotheken basieren auf *Information-Retrieval*-Technologie (IR), insbesondere dem *Vektorraummodell*, bei dem Text- oder HTML-Dokumente als Feature-Vektoren repräsentiert sind. Dieser hochdimensionale Vektorraum wird aufgespannt von den durch Stammformreduktion auf sog. Terme normalisierten Wörtern, die im Korpus (d.h. dem Web oder Intranet) vorkommen; u.U. können auch Ankertexte von Hyperlinks oder weitergehende Aspekte der Hyperlinkstruktur als zusätzliche Features verwendet werden. Die Komponenten eines Dokumentvektors sind Feature-Gewichte, die die Signifikanz eines Features im Dokument ausdrücken und aufgrund der Häufigkeit im Dokument sowie im gesamten Korpus berechnet werden. Anfragen von Benutzern werden dann intern ebenfalls auf Vektoren abgebildet, und es werden die zur Anfrage ähnlichsten Dokumentvektoren ermittelt und zu einer nach Relevanz absteigend sortierten Rangliste zusammengesetzt, wobei verschiedene Ähnlichkeitsmetriken Verwendung finden. Dieses Grundprinzip kann auch auf multimediale Daten, z.B. Bilder oder Musikstücke, verallgemeinert werden, wobei dann natürlich andere Features von Bedeutung sind, die z.B. aus Transformationen von Signalen abgeleitet werden. Die Güte eines Suchresultats wird a posteriori empirisch bewertet: die *Präzision* der Anfrageauswertung ist der Anteil der wirklich relevanten Resultdokumente unter den ersten N (z.B. 10) Treffern der Rangliste, die *Ausbeute* ist der Anteil der überhaupt im Korpus vorhandenen relevanten Dokumente, den die Suche gefunden hat.

> *Grundprinzip 1:*
> Dokumente jeder Art werden als *Feature-Vektoren* in einem hochdimensionalen Datenraum repräsentiert.

> *Grundprinzip 2:*
> Eine Anfrage an eine Suchmaschine liefert als Ergebnis (einen Präfix) eine(r) Rangliste, die die Trefferdokumente nach Relevanz bzw. Ähnlichkeit zur Anfrage absteigend sortiert (*Relevanz-Ranking*).

Web-Suchmaschinen sind ihrer Funktionalität durch den Zwang zur Skalierbarkeit, der Unterstützung von Millionen von Benutzern bei der Suche auf Milliarden von Dokumenten, stark eingeschränkt. In anderen Umgebungen, in denen ebenfalls Information-Retrieval-Technologie verwendet wird, etwa in digitalen Bibliotheken oder Intranets, kommen u.U. ausdrucksstärkere und mathematisch anspruchsvollere Techniken zum Einsatz. Dazu zählen insbesondere sog. Spektralanalysen, die auf linearer Algebra beruhen, sowie statistische Lernverfahren. Mit solchen Techniken können Dokumentensammlung tiefergehender analysiert und besser organisiert werden, beispielsweise durch eine automatische Klassifikation von Dokumenten in eine gegebene Themenhierarchie.

> *Grundprinzip 3:*
> Durch *algebraische Analysen* und *statistiche Lernverfahren* können Dokumentsammlungen besser organisiert und bewertet werden..

Suchmaschinen werden nicht nur für die Internet-Suche durch Endbenutzer verwendet, sondern haben auch Programmierschnittstellen, um als Komponenten zu umfassendere Lösungen etwas bei der Generierung von *Portalen* oder der Suche in *Intranets* beizutragen. Dabei müssen sie in vielen Fällen

mit strukturierten Datenquellen interagieren, z.B. mit ERP-Datenbanken (Enterprise Resource Planning, z.B. SAP R/3) in Intranets oder mit Produktkatalogen oder Gen- und Proteindatenbanken im Internet. Im letzeren Fall, der Einbeziehung von Datenquellen, die selbst eine Suchschnittstelle in Form eines HTML-Formulars oder eines sog. Web Services anbieten und Ergebnisse als dynamisch generierte HTML- (oder XML-) Seiten liefern, spricht man auch von der Erschließung des *Deep Web* (Hidden Web). Es wird geschätzt, dass das Deep Web um den Faktor 100 größer ist als das durch normale Suchmaschinen (wie Google) abgedeckte Oberflächen-Web und dass die wirtschaftlich und wissenschaftlich wichtigsten Daten Teil des Deep Web sind. Ein Trend ist, dass Deep-Web-Daten nicht nur im HTML-Format (Hypertext Markup Language) präsentiert, sondern auch im expliziter strukturierten und reicher annotierten *XML-Format* (Extensible Markup Langauge) exportiert werden. Bei der Suche auf solchen Daten ist es oft nötig, Suchverfahren auf strukturierten Daten (z.B. technischen Eigenschaften von Produkten) und Textdaten (z.B. Produktbeschreibungen in natürlicher Sprache) zu integrieren.

Ein langfristiges Ziel ist das *Semantic Web*, bei dem angestrebt wird, alle Informationen im Internet in einer einheitlichen logikbasierten Wissensrepräsentationssprache zu beschreiben. Damit würden alle Arten von digitalen Bibliotheken, wissenschaftlichen Datenarchiven, aktuellen Nachrichten und Unterhaltungsmedien global und präzise suchbar, doch bis dahin ist es wohl noch ein langer und schwieriger Weg, auf dem sich etliche wissenschaftliche Herausforderungen und spannende Forschungsfragen stellen.