1 Einleitung

Aufgrund gesellschaftlicher und technologischer Entwicklungen (z.B. Globalisierung oder die rasante Verbreitung von Internet/Intranet-Technologien), sind Unternehmen gezwungen, sich Veränderungen flexibel anzupassen. Insbesondere auf sich durch solche Trends ändernde Marktsituationen muß schnell reagiert werden können. Betrieblichen Anwendungssystemen kommt dabei eine besondere Bedeutung zu.

Hierbei stellt sich die Frage, wie solche Anwendungen implementiert werden sollen, um höchstmögliche Flexibilität, Verlässlichkeit und Anpassbarkeit zu erreichen. Monolithische Anwendungssysteme[1] (1-Ebenen Architekturen vgl. *Kapitel 2.1*) und klassische Client/Server-Architekturen (vgl. *Kapitel 2.2*) sind hier nur noch bedingt geeignet.

Diese Problematik soll zunächst anhand eines Beispiels veranschaulicht werden:

Ein Zulieferunternehmen der Prozeßindustrie erstellt zu jedem gefertigten Produkt ein Qualitätszeugnis. Die Qualitätszeugnisse werden vermittels einer monolithischen Datenbankanwendung verwaltet.

Beim Verkauf eines Produkts wird vermittels dieser Anwendung das zugehörige Qualitätszeugnis gesucht, gedruckt und dem Produkt beigefügt. Der Kunde muß das Qualitätszeugnis aufbewahren und sicherstellen, daß das Zeugnis, falls das gekaufte Produkt einen Fehler aufweist, wiedergefunden wird.

Diese Vorgehensweise verursacht sowohl beim Hersteller als auch bei den Kunden unnötigen Aufwand und damit unnötige Kosten. Angeregt durch die Möglichkeiten der Internet-Technologie wünschten wichtige Kunden, daß die Qualitätszeugnisse beim Hersteller verbleiben und sie im Fehlerfall von Produkten die Möglichkeit erhalten, entsprechende Zeugnisse über das Internet zu selektieren und sich zufaxen zu lassen.

Diese Anforderung erfordert nun ein Redesign der Anwendung. Die Anwendung muß als verteiltes System über das Internet konzipiert werden, da Komponenten der Anwendung zumindestens auf einem Internet-Client (WWW-Browser), einem WWW-Server und einem Datenbank-Server implementiert werden müssen.

Folgender Ablauf wurde realisiert:

- Der Kunde lädt eine html-Seite des Herstellers in einen WWW-Browser (z. B. Netscape). Aus der html-Seite wird ein Java-Applet[2] gestartet. Das Applet blendet eine Anmeldemaske auf, in der die Kunden Benutzernamen und Paßwort eingeben können.

- Danach baut das Applet eine Verbindung zu einer auf dem WWW-Server laufenden Anwendung (im folgenden als Server-Anwendung bezeichnet) auf. Die Server-Anwendung führt eine Datenbankabfrage auf dem Datenbankserver durch und übermittelt, falls der Kunde existiert, kundenabhängige Auswahlkriterien an das Applet.

---

1. Die gesamte Anwendung läuft hier in einem Prozeß auf einem Rechner ab.
2. Java-Applets sind (zumeist kleine) Programme, die nur im Kontext eines Browsers ablauffähig sind.

- Das Applet stellt die Auswahlkriterien in einer Auswahlmaske dar (vgl. *Abbildung 1*). Das Ergebnis der Kundenauswahl wird zur Server-Anwendung übertragen.

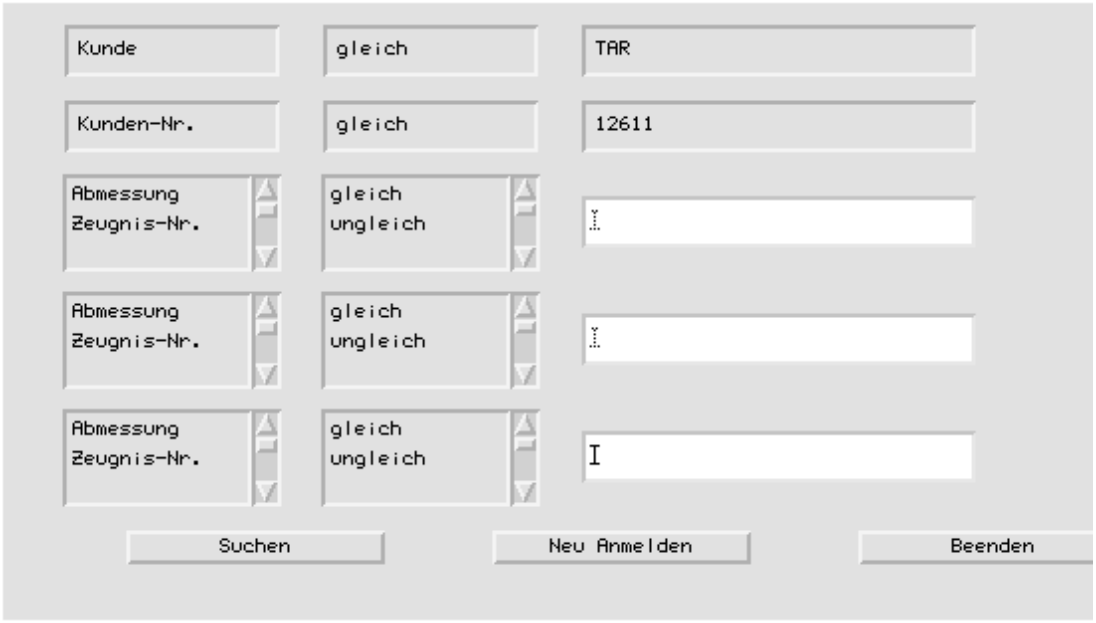

Abbildung 1 Auswahlmaske der Beispielanwendung

- Über eine Abfrage beim Datenbankserver ermittelt die Server-Anwendung die den ausgewählten Kriterien genügenden Zeugnisse.
- Falls die Auswahl des Kunden eindeutig war, wird das gefundene Zeugnis an eine voreingestellte Nummer gefaxt. Andernfalls werden die gefundenen Zeugnisse an das Applet übermittelt.
- Das Applet stellt Parameter der gefundenen Zeugnisse in einer weiteren Maske dar (vgl. *Abbildung 2*). Der Kunde selektiert das gewünschte Zeugnis. Die Selektion wird zur Server-Anwendung übertragen.
- Die Server-Anwendung faxt das ausgewählte Zeugnis an eine voreingestellte Telefonnummer des Kunden.

Abbildung 3 stellt die Architektur dieses Beispiels insgesamt dar.

Die oben vorgestellte Realisierung ist ein typisches verteiltes Anwendungssystem. Die Anwendung ist auf drei Komponenten verteilt:

- Der Client (vgl. *Abbildung 3*) stellt das Benutzer-Interface zur Verfügung.
- Die Server-Anwendung erzeugt Datenbankabfragen und entscheidet, was mit den Ergebnissen der Abfragen zu geschehen hat. Hier wird die “Logik” der Anwendung implementiert.
- Der Datenbankserver verwaltet die der Anwendung zugrundeliegenden Daten und führt die Datenbankabfragen aus.

Es wurden 11 Zeugnisse von 9708 gefunden.

Nr.	Zeugnis-Nr.	Auftrag	BA-Nr.	Charge
1	212249	25/20097/06	RA0003	13761
2	212551	15/22062/10	30457/	67431
3	213132	25/20598/04	232178	427560
4	213133	25/20598/01	231009	V 139046
5	214908	25/20097/09	231897	V 147035
6	410308	35/11870/02	331820	330516
7	412953	45/07281/01		34018
8	412991	45/07281/01		34018
9	513280	55/06403/01	196064	600780
10	513289	55/06403/01	196064	600780

Geben Sie bitte die Datensatz-Nr. des gew nschten Zeugnisses ein:

OK

Abbildung 2 Zeugnisselektionsmaske

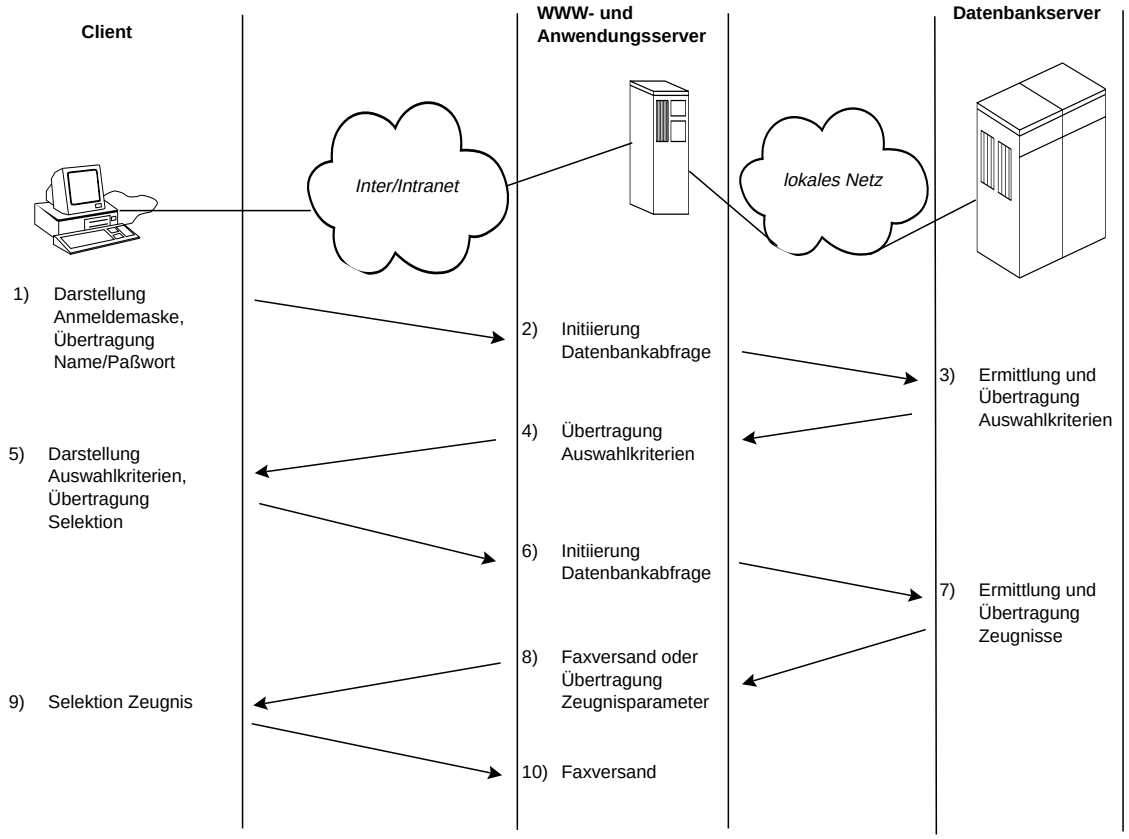

Abbildung 3 Struktur der Beispielanwendung

Der weitere Artikel gliedert sich wie folgt: In *Kapitel 2* werden Architekturen betrieblicher Anwendungen vorgestellt. Ihre Vor- und Nachteile werden dargestellt. Die Vorteile von 3-Ebenen Anwendungen (*Kapitel 2.3*) werden herausgearbeitet. *Kapitel 3* beleuchtet die konzeptionellen Eigenschaften von Mehr-Ebenen Anwendungen, soweit sie für das Verständnis dieses Beitrags erforderlich sind. In *Kapitel 4* wird ein kurzes Resümee gezogen.

2 Architekturmodelle von Anwendungssytemen

2.1 Ein-Ebenen Architekturen (One-Tier Architectures)

Bei der Ein-Ebenen Architektur wird die gesamte Anwendung auf einem Computersystem implementiert. Auf diesem Rechner (hier zumeist Host genannt) erfolgen die Datenhaltung, alle Berechnungen und die Vorbereitung der Darstellung der Ergebnisse.

Ein bekanntes Beispiel für eine Anwendung mit Ein-Ebenen Architektur ist R2 von SAP. Darüberhinaus sind heute noch vielfach Anwendungen mit Ein-Ebenen Architektur im Einsatz, so z.B. bei Banken oder großen Reiseveranstaltern.

2.1.1 Ein-Ebenen Architekturen (One-Tier Architectures) mit Terminals

Abbildung 4 zeigt beispielhaft Ein-Ebenen Architekturen in der Unix- und der IBM-Welt. Die Benutzerinteraktion erfolgt hier jeweils über Terminals. Unterschiedlich ist nur, wie die Terminals an den Rechner angebunden werden. In der Unix-Welt geschieht dies durch Direkt-Anschluß oder über Terminalserver, in der Großrechnerwelt über Kommunikationsrechner.

Terminals sind “dumme” Endgeräte ohne CPU und Speicher. Dies bedeutet, daß die gesamte Benutzerinteraktion vom Host verwaltet wird. Der Host übernimmt also sowohl die Darstellung auf den Terminal-Bildschirmen als auch die Reaktion auf die Eingaben der Benutzer (Benutzerinteraktion).

Ein-Ebenen Architekturen haben viele Vorteile:

- *Hohe Verfügbarkeit:* Host-Rechner sind typischerweise in Rechenzentren installiert. Sie werden professionell gewartet und verfügen über professionelle Betriebssysteme. Verfügbarkeiten von über 99 % sind keine Seltenheit.
- *Zentrale Datensicherung:* Da alle Daten und alle Anwendungen auf einem im Rechenzentrum aufgestelltem Rechner liegen, kann dort eine automatisierte Datensicherung durchgeführt werden.
- *Geringe Kosten im End-User Bereich:* Terminals sind äußerst preiswerte Endgeräte und darüberhinaus fast wartungsfrei.

Diesen Vorteilen stehen natürlich Nachteile gegenüber:

- *“Software-Stau”:* Jede Anwendung und jede Anwendungsänderung muß auf dem zentralen Rechner programmiert werden. Benutzer können kleinere Anwendungen, wie z.B. einfache Datenbankabfragen, nicht eigenständig realisieren. Dies verteuert und verzögert die Anwendungsentwicklung.

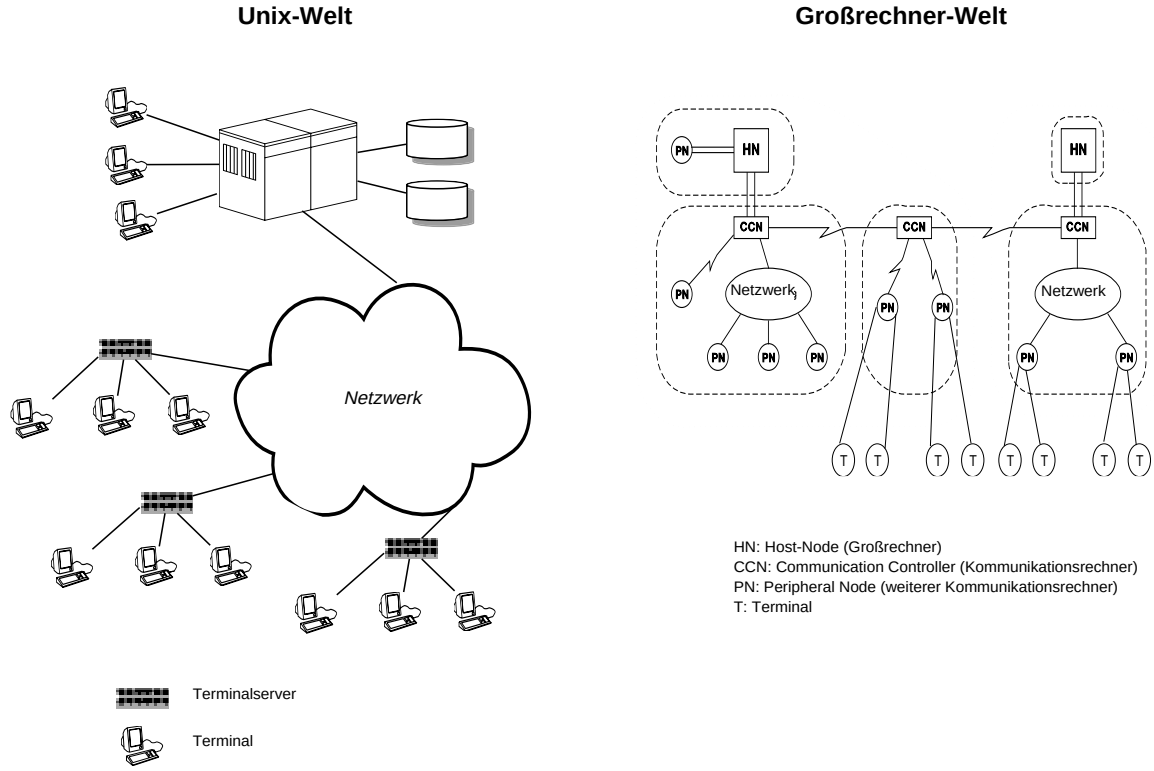

Abbildung 4 Ein-Ebenen Architektur mit Terminals

- *Keine Anwendungen mit grafischer Benutzerschnittstelle:* Terminals sind zeichenorientierte Endgeräte.
- *Hohe Kosten im Rechenzentrum:* Die Kosten für Großrechner sind im Vergleich zu PC's beispielsweise hoch.
- *Mangelnde Eignung für "Büroanwendungen":* Manche Anwendungen, wie z.B. Textverarbeitung, sind auf Großrechnern nicht optimal implementierbar.

2.1.2 Ein-Ebenen Architektur (One-Tier Architecture) mit PC's

Seit dem Aufkommen der Personal-Computer werden die Terminals zunehmend durch PC's ersetzt (vgl. *Abbildung 5*). Hierbei werden die PC's entweder direkt an den Rechner angeschlossen oder über ein lokales Netzwerk (vgl. Beitrag über Kommunikationstechnik) an den zentralen Rechner herangeführt.

An der Struktur der Anwendungen ändert dies allerdings nichts. Datenhaltung, Anwendungslogik und Darstellung verbleiben auf dem Großrechner. Um die für Terminals bestimmte Benutzerinteraktion durchführen zu können, emulieren[3] die PC's Terminals.

Allerdings werden durch den Einsatz von PC's einige Nachteile aus *Kapitel 2.1.1* revidiert:

3.Dies bedeutet: "verhalten sich wie"

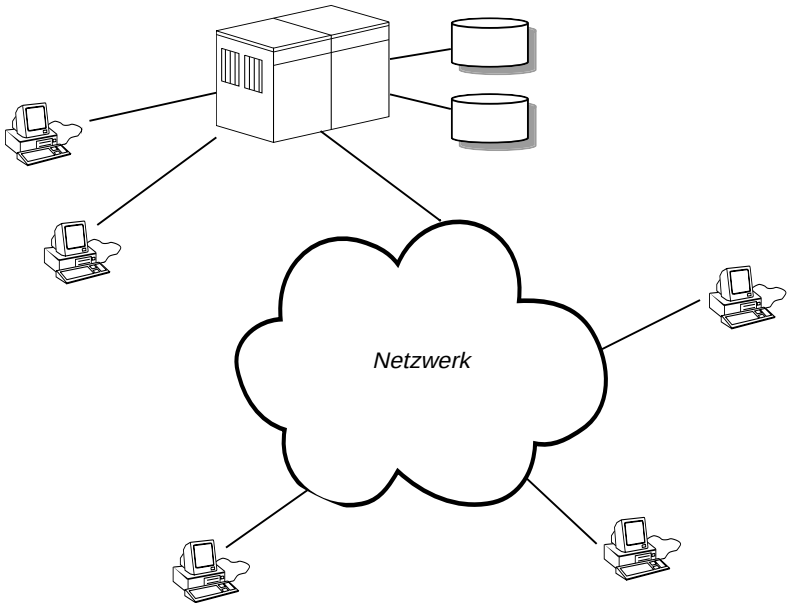

Abbildung 5 Ein-Ebenen Architektur mit PC's

- Büroanwendungen, wie Textverarbeitung, können lokal auf den PC's stattfinden.
- "Kleinere" Anwendungen können für PC's entwickelt werden. Die dazu benötigten Daten können auf dem Großrechner zur Verfügung gestellt und dann über das Netzwerk auf PC's kopiert werden. Dies hilft, dem Software-Stau zu begegnen.

Allerdings ergeben sich neue Nachteile:

- Die Kosten im End-User Bereich steigen. PC's sind teurer als Terminals und darüberhinaus nicht wartungsfrei.
- Für die PC-Anwendungen müssen geringere Verfügbarkeiten eingeplant werden.
- Konfigurationsänderungen, Betriebssystem- und Software-Updates bei den PC's sind aufwendiger und damit teurer.
- Versionskontrolle bei PC-Anwendungen ist schwieriger.
- Durch die Nutzung von PC's ergeben sich neue Problematiken, wie
  - o Virenbefall oder
  - o die Möglichkeit, firmeninterne Daten über Wechselspeichermedien zu "stehlen".

  Zur Eindämmung dieser Problematiken müssen geeignete Maßnahmen ergriffen werden.
- Die Datensicherung wird durch lokale Platten an den PC's schwieriger.

2.2 Zwei-Ebenen Architekturen (Two-Tier Architectures)

Bei der Zwei-Ebenen Architektur wird die Anwendung auf zwei Komponenten aufgeteilt. Hier wird bereits von verteilten Anwendungen oder auch von Client/Server-Systemen gesprochen. *Abbildung 6* zeigt eine typische Anwendung mit Zwei-Ebenen Architektur.

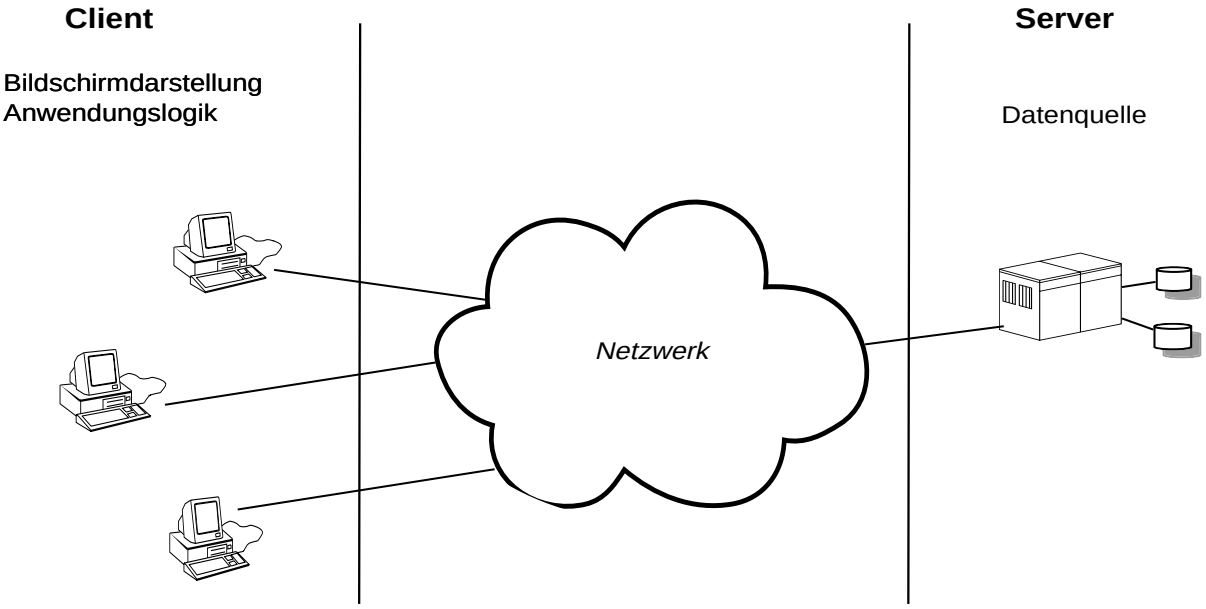

Abbildung 6 Zwei-Ebenen Architektur

Die Benutzerschnittstelle bilden PC's. Hier wird die Bildschirmdarstellung durchgeführt. Darüberhinaus ist im PC-Programm die Anwendungslogik implementiert. Die Datenhaltung findet zentral auf einem Datenbankserver statt.

Zwei-Ebenen Architekturen kommen aus der klassischen Client/Server-Datenbankentwicklung. Die Client-Anwendung wird typischerweise in einer prozeduralen Sprache, wie C oder Pascal entwickelt. Heute kommen allerdings zunehmend auch objektorientierte Sprachen wie C++, Java oder Delphi zum Einsatz.

Der Zugriff auf die Datenbank erfolgt über das Netzwerk z.B. mit ODBC[4], JDBC[5] oder SQL*Net[6]. Als Kommunikationsprotokolle (vgl. Beitrag über Kommunikationssysteme) auf dem lokalen Netz werden TCP/IP oder proprietäre Protokolle wie z.B. die nativen Netware-Protokolle eingesetzt.

Als Betriebssysteme kommen clientseitig typischerweise die Microsoft-Windows Produkte (Windows 95/98 oder NT) zum Einsatz. Serverseitig werden UNIX-Systeme, Großrechner mit proprietären Betriebssystemen oder Windows NT-Rechner genutzt.

4.Dies ist eine von Microsoft entwickelte offene Schnittstelle für den Datenbankzugriff.
5.Dies ist die von Sun entwickelte offene Schnittstelle für den Datenbankzugriff.
6.Dies ist eine von Oracle entwickelte proprietäre Schnittstelle zur Oracle-Datenbank.

Client/Server-Architekturen können Hardware-Kosten senken, da Teile der Anwendung im Gegensatz zu der in *Kapitel 2.2* dargestellte 1-Ebenen Architektur auf die PC's ausgelagert sind. Dadurch wird der Server entlastet.

Dieser Vorteil wird durch erhöhte Aufwendungen bei der Wartung der Systeme erkauft. Die Client-Programme müssen auf die PCs verteilt und im Regelfall dort installiert werden (Rollout). Jede Anwendungsänderung führt somit zu einem Aufwand, der durch die Anzahl der Clients beeinflußt wird. In zentral organisierten Anwendungen, wie in *Kapitel 2.2* beschrieben, werden Installationen nur auf einem System durchgeführt.

Auch die Fehlerfindung und Fehlerbehebung gestalten sich schwieriger. Ein Anwendungsfehler auf einem Client kann durch die Serverkomponente oder durch eine Clientkomponente verursacht sein[7]. Clientseitige Fehler können darüberhinaus durch

- die Betriebssystemversion des Client,
- Hardwarekomponenten des PC's oder
- die Interaktion mit anderen auf dem jeweiligen Client laufenden Anwendungen

entstehen.

Zudem sinkt die Verfügbarkeit solcher Systeme durch die Verteilung von Teilen der Anwendung auf PC's im Vergleich zu der in *Kapitel 2.1* dargestellten Architektur.

Diese Nachteile werden heute häufig unter dem Schlagwort "Fat-Client-Problematik" thematisiert.

2.3 Drei-Ebenen Architekturen (Three-Tier Architectures)

Bei der Drei-Ebenen Architektur wird die Anwendung nach funktionalen Gesichtspunkten auf drei Komponenten aufgeteilt. Die hierbei berücksichtigten Funktionen sind:

- Bildschirmdarstellung.
- Anwendungslogik.
- Datenverwaltung.

Abbildung 7 zeigt eine typische Anwendung mit Drei-Ebenen Architektur.

Auf dem Client findet sich nur noch die Benutzerschnittstelle. Hier können Benutzer mit den Elementen des grafischen Benutzer-Interfaces interagieren. Der Client führt z.B. die Event-Behandlung[8] und Eingabeprüfung durch. Darüberhinaus ist der Client für die Darstellung der ihm vom Anwendungsserver übermittelten Informationen verantwortlich.

Auf dem Anwendungsserver ist die "Logik" der Anwendung implementiert. Dies kann von der Berechnung betriebswirtschaftlicher Kennzahlen bis hin zur Simulation komplexer Systeme reichen.

7.Fehler durch das Netzwerk sollen hier nicht betrachtet werden.
8.Dies ist die z.B. Reaktion auf "Maus-Klicks" oder das Betätigen der Eingabetaste in Textfeldern.

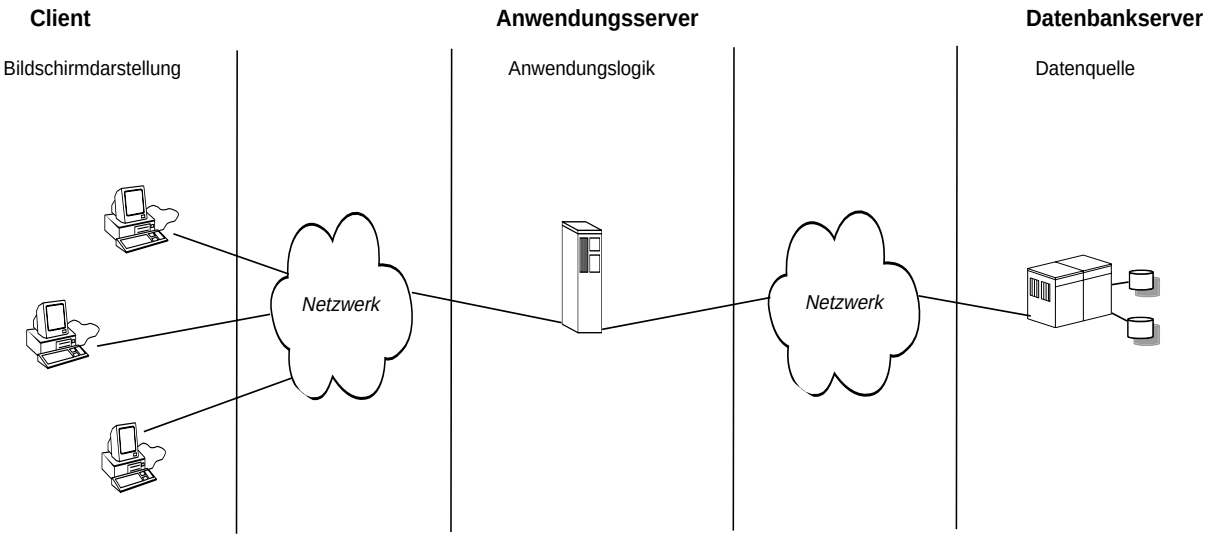

Abbildung 7 Drei-Ebenen Architektur

Die für die Berechnung notwendigen Daten werden vom Datenbankserver angefordert und Ergebnisse, wenn notwendig, dem Datenbankserver zur Abspeicherung übergeben.

Das bekannteste Beispiel einer Anwendung mit 3-Ebenen-Architektur ist das R3-System von SAP. Auf den Clients wird hier nur die Benutzerschnittstelle (bei R3 SAP-GUI[9] genannt) installiert. Die R3 zugrundeliegende betriebswirtschaftliche Logik ist auf dem Anwendungsserver (bei R3 heißt dieser Application-Server) implementiert. Die Datenhaltung wird vom Datenbank-Server übernommen.

Zur Entwicklung von 3-Ebenen Anwendungen werden zumeist objektorientierte Programmiersprachen (vgl. Beitrag über objektorientierte Entwicklung) genutzt. Die Client- und Anwendungsserver-Ebenen stellen sich dann als interagierende Objekte dar. Objekte sind (stark vereinfacht ausgedrückt) kleine eigenständige Programme, die über in den Objekten festgelegte Schnittstellen miteinander kommunizieren.

Die Kommunikation mit der Datenbank erfolgt entweder mittels eines der Datenbank vorgeschalteten Objektadapters (vgl. *Abbildung 8*) oder vermittels direkter Datenbankaufrufe aus Objekten des Anwendungsservers (z.B. über JDBC, ODBC oder proprietäre Techniken wie SQL*Net, vgl. *Abbildung 9*)

Als Kommunikationsprotokoll im Netzwerk wird im Regelfall die TCP/IP-Protokollarchitektur (vgl. Beitrag über Kommunikationssysteme) eingesetzt. Die Objektinteraktion wird über sogenannte “Middleware” realisiert. Beispiele für Middleware sind CORBA, RMI oder DCOM. Middleware wird in *Kapitel 3* ausführlich behandelt.

Anwendungs- und Datenbankserver müssen nicht notwendigerweise unterschiedliche Computer sein. Wichtig im Sinne einer 3-Schichten (die Begriffe “Ebenen” und “Schichten” werden im folgenden synonym verwendet) Architektur ist nur, daß zwischen Anwendungsobjekten und Datenbankzugriff eine klar definierte Schnittstelle

9.GUI = Graphical User Interface

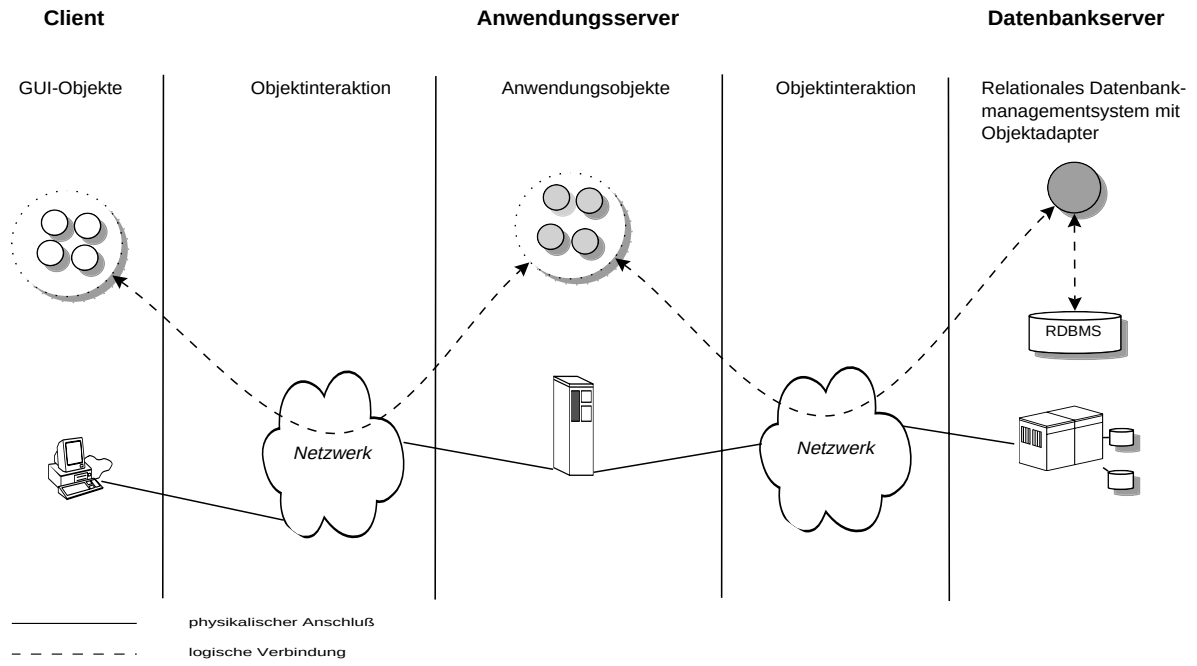

Abbildung 8 Durchgehende Objektkommunikation im 3-Ebenen Modell

besteht. Die Hardware, auf der diese Komponenten ablaufen, ist durch die zur Objektkommunikation eingesetzte Middleware transparent (vgl. *Abbildung 10* und *Kapitel 3*). Viele kleinere R3-Systeme werden auf diese Art installiert.

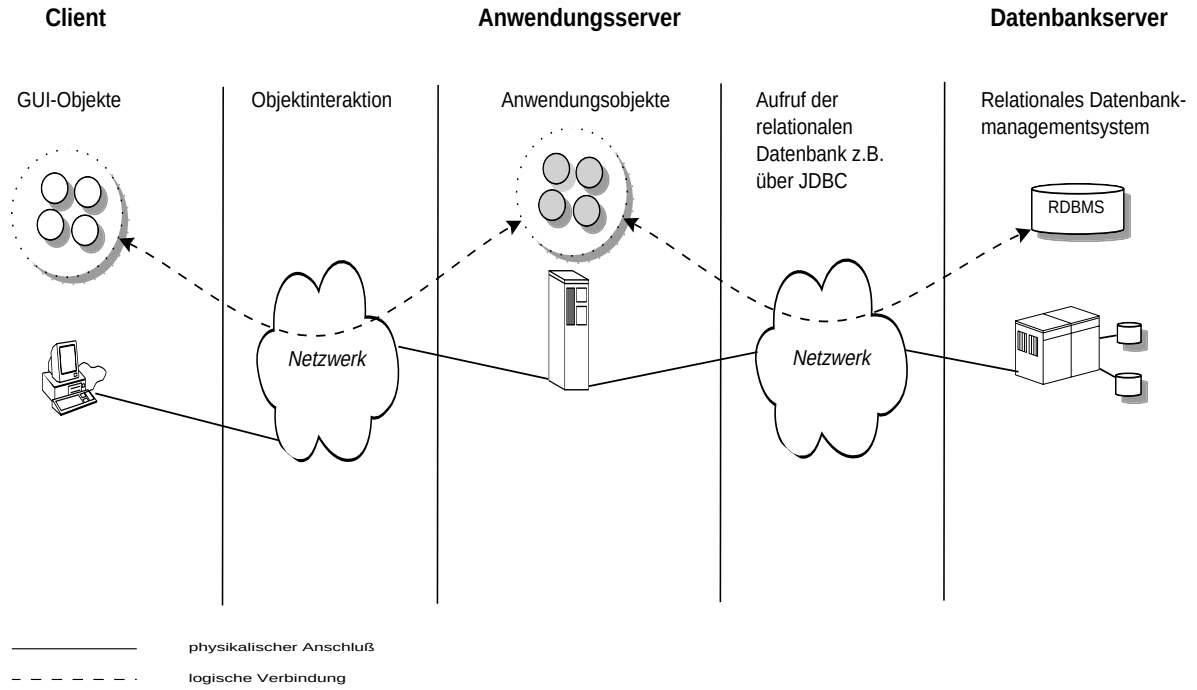

Abbildung 9 Objektkommunikation mit direktem Datenbankaufruf aus der Anwendungsschicht

Andererseits muß die Anwendungsschicht nicht zwingend nur aus einem Rechner bestehen. Insbesondere bei verteilten Internet-Anwendungen ist vor den Anwendungs-

server oft ein WWW-Server geschaltet. Dies wird häufig als Mehr-Ebenen Architektur (Multi-Tier Architecture) bezeichnet.

3-Schichten Architekturen bieten vielfältige Vorteile. Da die drei Schichten über festgelegte Objektschnittstellen kommunizieren, bleibt der Austausch oder die Änderung einer Schicht ohne Auswirkungen auf die anderen Ebenen der Anwendung. Wird z.B. ein Fehler im Anwendungsserver festgestellt, kann er zentral auf dem Anwendungsserver behoben werden. Die Clients sind nicht betroffen.

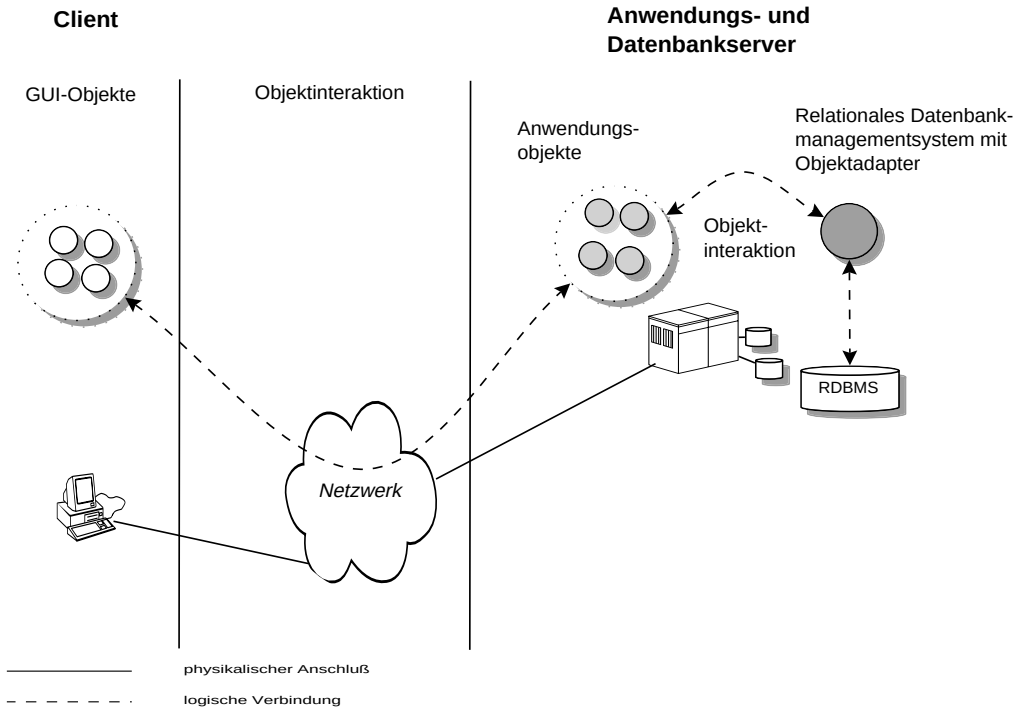

Abbildung 10 Anwendungs- und Datenbankserver auf gleicher Hardware

Änderungen der Client-Komponenten betreffen auch nur die Clients. Soll z.B. eine Anwendung, die bislang nur Windows-Clients unterstützt, auch über das Internet oder ein Intranet verfügbar gemacht werden, muß “nur” eine weitere Client-Implementierung hinzugefügt werden. Die beiden anderen Ebenen bleiben unberührt. Darüberhinaus führt die Reduzierung der Client-Komponente auf die Benutzerinteraktion zu “kleinen” Programmen. Im Idealfall können diese auf einem Server vorgehalten und beim Start der Anwendung auf den Client kopiert und dort gestartet werden. So entfällt jeglicher Installationsaufwand auf den Client-Rechnern. Dies ist übrigens die übliche Vorgehensweise bei Internet/Intranet-Anwendungen.

3-Ebenen Architekturen erlauben darüberhinaus flexible Ausfallsicherheits- und Lastverteilungskonzepte. So kann die Anwendungslast sowohl auf der Applikations- als auch auf der Datenbankschicht auf mehrere Rechner verteilt werden. Darüberhinaus kann die Anwendungsschicht auf mehrere Datenquellen zugreifen. *Abbildung 11* faßt das bisher Gesagte zusammen.

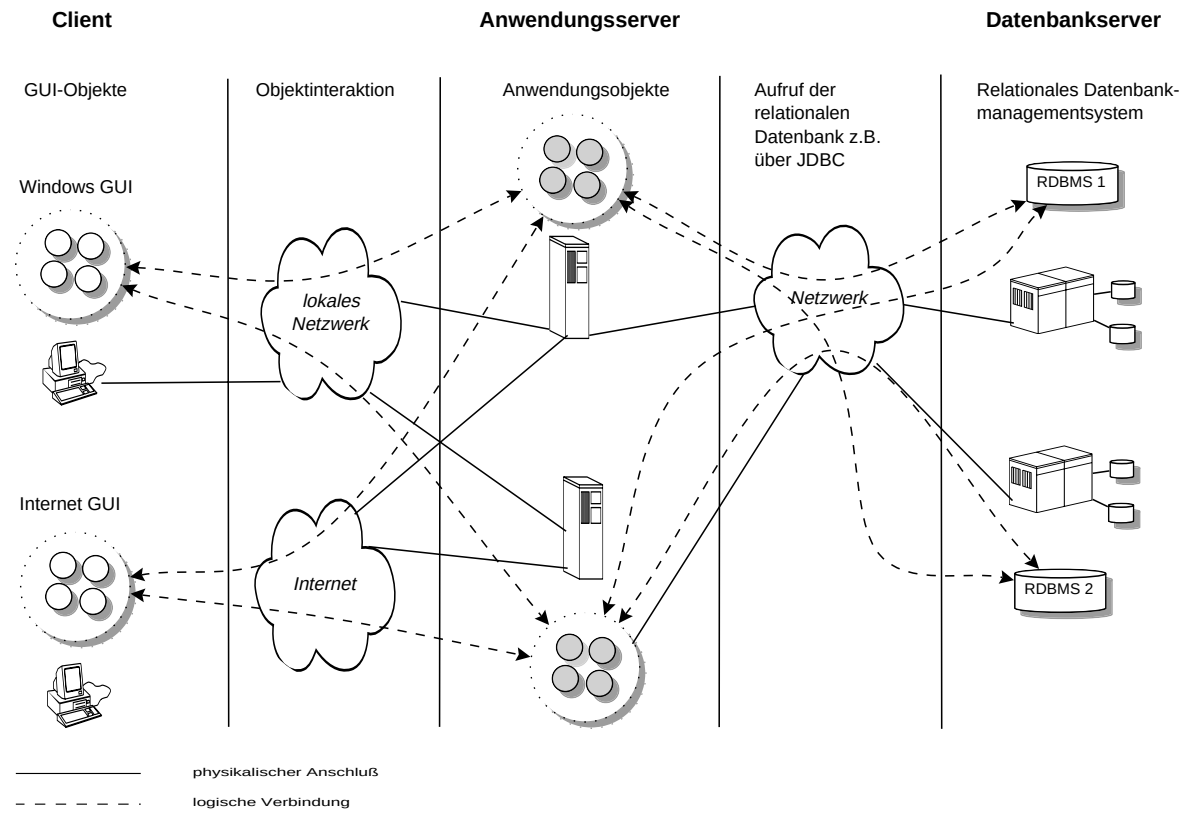

Abbildung 11 3-Ebenen Architektur mit Windows und Internet GUI, Ausfallsicherheit und Lastverteilung auf der Anwendungsebene sowie Zugriff auf 2 Datenquellen (Name: vorteile.eps)

Die Vorteile der 1-Ebenen Architektur, nämlich hohe Verfügbarkeit und zentrale Datensicherung sind hier ebenfalls gegeben, da sowohl Anwendungs- als auch Datenbankserver im Regelfall in Rechenzentren aufgestellt werden.

Den Nachteilen der 2-Ebenen Architektur, insbesondere dem hohen Installations- und Änderungsaufwand, wird durch die 3-Schichten Architektur ebenfalls entgegengewirkt.

Als Nachteile verbleiben die bereits in *Kapitel 2.1.2* dargestellten Nachteile des PC-Einsatzes an sich. Zudem wird die Leistungsfähigkeit eines PC bei der in diesem Kapitel vorgestellten Architektur kaum genutzt, da der Rechner nur für die Benutzerinteraktion zuständig ist. Um diese Nachteile abzumildern sind derzeit 2 Ansätze in der Diskussion:

- *Network Computer (NC):* Dies ist ein Vorschlag von Sun, Oracle und IBM. Unter einem NC versteht man einen speziellen Client-Computer, der auf Benutzerinteraktion spezialisiert und optimiert ist. Ein NC besitzt keinerlei Permanentspeicher. Beim Einschalten lädt er ein Spezialbetriebssystem und einen Web-Browser von einem Server herunter. Die GUI-Objekte einer 3-Ebenen Anwendung werden vom Anwendungsserver[10] geladen und im Browser zur Ausführung ge-

10.oder auch einem anderen Serversystem

bracht. Die persönlichen Daten der Benutzer werden auf dem für den NC zuständigen Server abgelegt (vgl. *Abbildung 12*).

Dies impliziert natürlich, daß die Benutzerschnittstelle auch von einem Browser dargestellt werden kann. Damit muß sie im Regelfall mit der Programmiersprache Java entwickelt worden sein. So stellt auch SAP für das R3-System eine Java-GUI zur Verfügung.

Büroanwendungen sollen ebenfalls als Java-Programme auf dem NC-Server vorgehalten und bei Bedarf auf den NC geladen und dort ausgeführt werden.

Um Kompatibilität mit bestehenden Anwendungen zu gewährleisten, werden NC's (zumindest von IBM und Oracle) mit zusätzlichen, vom NC-Server ladbaren, Softwaremodulen angeboten, so u.a.:

- *X-Windows Emulation:* Dies ist der Windows-Standard in der UNIX-Welt. Hierdurch können auf einem Unix-Rechner laufende Anwendungen auf einem NC dargestellt werden.
- *NT-Terminal:* Durch diese Software können auf einem Microsoft Windows NT-Server laufende Anwendungen, wie z.B. Microsoft Word oder Excel, auf dem NC dargestellt werden.

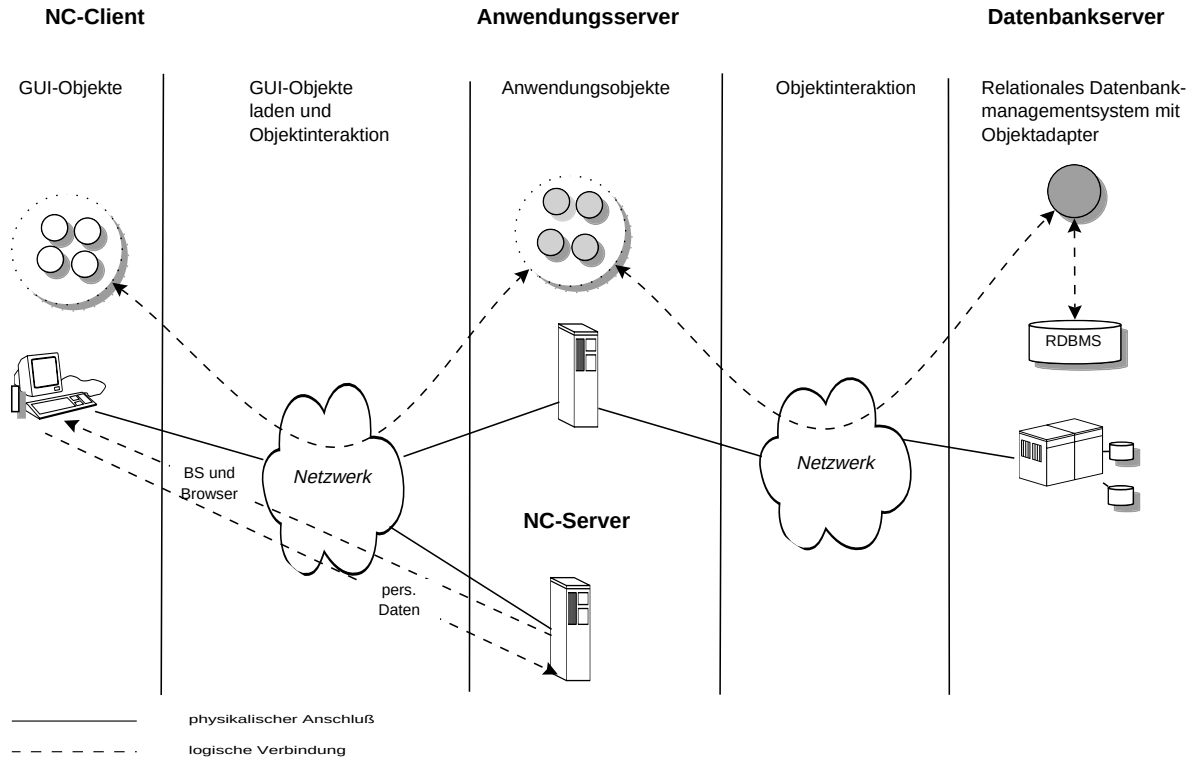

Abbildung 12 3-Ebenen Anwendung mit NC-Client

- *Network PC's (Net-PC):* Dies ist eine Initiative u.a. von Microsoft, Intel, Compaq und Siemens. Hierbei handelt es sich um Standard-PC's mit Windows-NT-Betriebssystem, deren Funktionalität aber stark eingeschränkt werden kann. Dies reicht von "normal" betriebsfähigen PC's bis hin zu PC's, auf denen nur ein

Web-Browser ausgeführt werden kann. Solche Net-PC's entsprechen dann im wesentlichen NC's. Zwischen diesen Extremen ist jeder Zustand möglich. So können z.B. Net-PC's existieren, die so konfiguriert sind, daß auf ihnen zusätzlich zum Browser Microsoft Word und Excel ablauffähig sind. Die Konfiguration eines Net-PC's ist vom Benutzer nicht änderbar.

3 Middleware

Um verteilte Anwendungen zu realisieren, muß Kommunikation zwischen den Komponenten der Anwendung ermöglicht werden. Als Kommunikationsprotokoll im Internet/ Intranet wird die TCP/IP-Protokollarchitektur (vgl. Beitrag über Kommunikationssysteme) genutzt. Diese kommt heute im Regelfall auch in herkömmlichen lokalen Netzen zum Einsatz.

TCP/IP stellt eine einfache Schnittstelle zur Anwendungskommunikation, die "socket library" zur Verfügung. Auf diese Schnittstelle kann von Sprachen wie C++ oder Java zugegriffen werden. Die Programmierung von Netzwerkanwendungen über die "sockket library" ist jedoch sehr aufwendig, da hier im wesentlichen nur Funktionen zur Verfügung gestellt werden, um Bytes über das Netzwerk zu übertragen. Dies entspricht jedoch in keiner Weise dem Abstraktionsgrad, der bei der objektorientierten Programmierung erreicht wird.

Daher wird eine Schicht zwischen den Anwendungsobjekten und der Netzwerksoftware implementiert, die die Netzwerkroutinen kapselt. Den Anwendungsobjekten soll so eine "einfache" Kommunikationsmöglichkeit über das Netz ermöglicht werden. Diese Schicht wird Middleware genannt. Im folgenden sollen zwei Middleware-Spezifikationen kurz vorgestellt werden.

3.1 CORBA

CORBA (Common Object Request Broker Architecture) ist die Spezifikation eines Standards für verteilte objektorientierte Anwendungen, der von der OMG (Object Management Group) verabschiedet wurde. Der OMG gehören z.Zt. über 900 Unternehmen an.

Schnittstellen zwischen Objekten werden in der programmiersprachenneutralen Corba IDL (Interface Defionition Language) implementiert. Um in einer konkreten Programmiersprache entwickelte Objekte anzusprechen, existieren Abbildungen (Mappings) zwischen der jeweiligen Programmiersprache und IDL (z.B. C++ IDL oder Java IDL).

Client-Objekte rufen Methoden von Server-Objekte im CORBA-Ansatz nicht direkt auf. Sie richten ihre Anfrage an einen sogenannten ORB (Object Request Broker). Der ORB ermittelt über ein Implementation Repository einen geeigneten Server und leitet die Anfrage des Clients an den Server weiter. Dabei spielt es keine Rolle, auf welchem Rechner oder in welcher Programmiersprache der Server implementiert wurde. Er muß nur vom ORB erreichbar sein.

Der Server arbeitet die Anfrage des Client ab. Die Resultate oder evtl. auftretende Fehlermeldungen werden vom Server über den ORB an den Client zurückgegeben (vgl. *Kapitel Abbildung 13*).

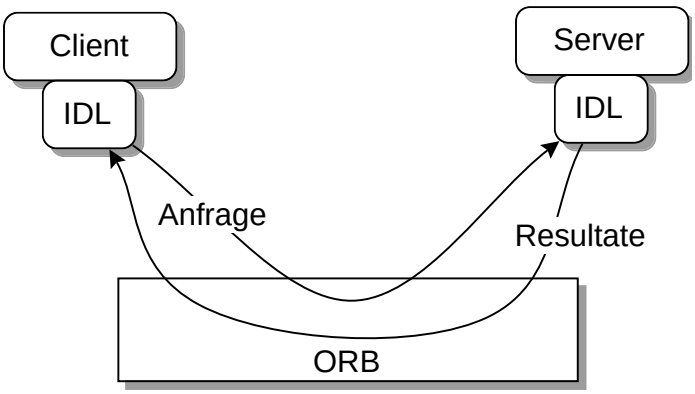

Abbildung 13 Funktionsweise von CORBA (stark vereinfacht)

Ausführliche Darstellungen der CORBA-Technologie finden sich z.B. in OMG 1997, OMG 1995 oder Siegel, J 1996.

3.2 RMI

RMI (Remote Method Invocation) ist seit der Version 1.1 Teil des Java Development Kits und damit Teil der Java Sprachdefinition. Es ist eine der einfachsten Arten, verteilte Anwendungen zu implementieren. Voraussetzung ist hier allerdings, daß Client- und Server-Objekte in Java realisiert werden. Dies ist ein wesentlicher Unterschied zum programmiersprachenneutralen CORBA-Ansatz.

Java RMI besitzt einen ähnlichen Mechanismus wie den CORBA-ORB, die RMI-Registry. Dies ist eine im JDK mitgelieferte Anwendung, die auf Server-Rechnern gestartet werden muß. Objekte, die Server-Dienste zur Verfügung stellen wollen, registrieren sich bei der Registry.

Wollen Clients ein Server-Objekt benutzen, müssen sie den Namen des Objektes und den Server auf dem das Objekt läuft kennen. Sie können dann eine Verbindung zum Server-Objekt aufbauen. Danach ist das Server-Objekt wie ein lokales Objekt nutzbar.

Der Verbindungsaufbau eines Java-Clients findet wie bei CORBA statt. Der Java-Client sendet seinen Verbindungswunsch an die RMI-Registry des Servers. Die Registry checkt dann die bei ihr registrierten Objekte und leitet den Methodenaufruf an das gewünschte Objekt weiter. Die Rückgabewerte werden von der Registry an das Client-Objekt zurückgegeben.

4 Zusammenfassung

Noch vor wenigen Jahren wurden Rechner-Grenzen nach dem klassischen Client/Server-Prinzip mit Datenbankmitteln (z.B. mit ODBC) überwunden (*Kapitel 2.2*). Diese Architektur stößt aber insbesondere durch die Internet-Technologien an ihre Grenzen, da es hier nicht möglich ist, komplexe Anwendungen auf Client-Computer zu verteilen. Auch monolithische Anwendungen sind in vielen Fällen nicht adäquat einsetzbar (*Kapitel 2.1*).

Heute werden betriebliche Anwendungssysteme in Objekte aufgeteilt und nach verschiedenen Gesichtspunkten im Netz verteilt. In solchen mehrstufigen (typischerweise dreistufigen) Architekturen kommunizieren die einzelnen Komponenten über Middleware miteinander (*Kapitel 2.3*).

Middleware kapselt die schwierig zu programmierenden Schnittstellen der den Netzen zugrundeliegenden Kommunikationprotokolle in einer Objektschicht. Als Beispiele werden CORBA (*Kapitel 3.1*) und RMI (*Kapitel 3.2*) kurz vorgestellt.

5 Literaturverzeichnis

Harmon, Trevor: Verteilte Java-Anwendungen mit RMI, Java Magazin 5/98, S. 34- 38

Klimetzky, Hansjörg: Java und Corba, OO @ Sun, in: Arbeitsplatz-Rechensysteme: Anwendungen, Architekturen, Betriebsysteme und Netzwerke, Hrsg.: Steigner Ch.

Mathy, Frank: Verteilte Objekte mit CORBA, Java Magazin 1/99, S. 64-67

OMG, Java, RMI and CORBA, 1997, www.omg.org

OMG, CORBAServices: Common Object Services Implementation, Juli 1997, www.omg.org

OMG, The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995, www.omg.org

Siegel, J. Corba - Fundamentals and Programming, John Wiley, 1996

Wend, H. und T. Salzsieder: Vergleich Java-basierter Architekturen zum Zugriff auf relationale Datenbanken, SUG info 2/98, S. 5-13

Weske, M Business-Objekte: Konzepte, Architekturen, Standards, Wirtschaftsinformatik 1/99, S.4-11