Informatik

Hans Heering

Objektorientierte Middleware-Konzepte und Infrastrukturen in verteilten Anwendungen

Diplomarbeit

Diplom.de

Bibliografische Information der Deutschen Nationalbibliothek:

Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de/ abrufbar.

Dieses Werk sowie alle darin enthaltenen einzelnen Beiträge und Abbildungen sind urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsschutz zugelassen ist, bedarf der vorherigen Zustimmung des Verlages. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen, Auswertungen durch Datenbanken und für die Einspeicherung und Verarbeitung in elektronische Systeme. Alle Rechte, auch die des auszugsweisen Nachdrucks, der fotomechanischen Wiedergabe (einschließlich Mikrokopie) sowie der Auswertung durch Datenbanken oder ähnliche Einrichtungen, vorbehalten.

Copyright © 1998 Diplom.de
ISBN: 9783832412005

https://www.diplom.de/document/217120

Hans Heering

Objektorientierte Middleware-Konzepte und Infrastrukturen in verteilten Anwendungen

Diplom.de

Hans Heering

Objektorientierte Middleware-Konzepte und Infrastrukturen in verteilten Anwendungen

Diplomarbeit
an der AKAD Rendsburg, Hochschule für Berufstätige
Oktober 1998 Abgabe

Diplomarbeiten Agentur
Dipl. Kfm. Dipl. Hdl. Björn Bedey
Dipl. Wi.-Ing. Martin Haschke
und Guido Meyer GbR

Hermannstal 119 k
22119 Hamburg

agentur@diplom.de
www.diplom.de

Heering, Hans: Objektorientierte Middleware-Konzepte und Infrastrukturen in verteilten Anwendungen / Hans Heering - Hamburg: Diplomarbeiten Agentur, 1998
Zugl.: Rendsburg, AKAD - Hochschule für Berufstätige, Diplom, 1998

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtes.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, daß solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Die Informationen in diesem Werk wurden mit Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden, und die Diplomarbeiten Agentur, die Autoren oder Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für evtl. verbliebene fehlerhafte Angaben und deren Folgen.

Dipl. Kfm. Dipl. Hdl. Björn Bedey, Dipl. Wi.-Ing. Martin Haschke & Guido Meyer GbR
Diplomarbeiten Agentur, http://www.diplom.de, Hamburg 1999
Printed in Germany

Diplomarbeiten Agentur

Wissensquellen gewinnbringend nutzen

Qualität, Praxisrelevanz und Aktualität zeichnen unsere Studien aus. Wir bieten Ihnen im Auftrag unserer Autorinnen und Autoren Wirtschaftsstudien und wissenschaftliche Abschlussarbeiten – Dissertationen, Diplomarbeiten, Magisterarbeiten, Staatsexamensarbeiten und Studienarbeiten zum Kauf. Sie wurden an deutschen Universitäten, Fachhochschulen, Akademien oder vergleichbaren Institutionen der Europäischen Union geschrieben. Der Notendurchschnitt liegt bei 1,5.

Wettbewerbsvorteile verschaffen – Vergleichen Sie den Preis unserer Studien mit den Honoraren externer Berater. Um dieses Wissen selbst zusammenzutragen, müssten Sie viel Zeit und Geld aufbringen.

http://www.diplom.de bietet Ihnen unser vollständiges Lieferprogramm mit mehreren tausend Studien im Internet. Neben dem Online-Katalog und der Online-Suchmaschine für Ihre Recherche steht Ihnen auch eine Online-Bestellfunktion zur Verfügung. Inhaltliche Zusammenfassungen und Inhaltsverzeichnisse zu jeder Studie sind im Internet einsehbar.

Individueller Service – Gerne senden wir Ihnen auch unseren Papierkatalog zu. Bitte fordern Sie Ihr individuelles Exemplar bei uns an. Für Fragen, Anregungen und individuelle Anfragen stehen wir Ihnen gerne zur Verfügung. Wir freuen uns auf eine gute Zusammenarbeit

Ihr Team der *Diplomarbeiten* Agentur

Dipl. Kfm. Dipl. Hdl. Björn Bedey
Dipl. Wi.-Ing. Martin Haschke
und Guido Meyer GbR

Hermannstal 119 k
22119 Hamburg

Fon: 040 / 655 99 20
Fax: 040 / 655 99 222

agentur@diplom.de
www.diplom.de

Inhaltsverzeichnis

1. Einleitung 6

1.1 Aufbau der Arbeit 6

1.2 Abgrenzung und Überblick über Client-/Server-Modelle und Middleware-Konzepte 6

1.2.1 Klassische Aufgaben von Middleware 8

1.2.2 Kommunikations-Middleware 9

1.2.3 SQL-Middleware zum Zugriff auf verteilte Daten 9

1.2.4 Aktuelle und neue Middleware-Entwicklungen 11

1.2.5 Zusammenfassung 15

2. Common Object Request Broker Architecture 17

2.1 Historie, Entwicklung und Ziele 17

2.2 Die Object Management Architecture 18

2.2.1 Der ORB 19

2.2.2 Die Object Services 19

2.2.3 Die Common Facilities 21

2.2.4 Die Domain Services 21

2.2.5 Die Application Objects 22

2.3 CORBA 2.0 22

2.3.1 Ein Überblick 22

2.3.2 Exkurs: Broker-Pattern 25

2.4 Der Object Adapter (OA) 26

2.5 Das Implementation Repository 26

2.6 Möglichkeiten des Aufbaus für den ORB 27

2.7 Verbindungsmöglichkeit eines Klienten 28

2.8 Aufbau einer Objekt-Implementation 29

2.9 Interoperabilität 30

2.9.1 Interoperabilität zwischen verschiedenen ORBs 31

2.9.2 Interoperable Object Reference (IOR) 35

2.10 Interface Definition Language (IDL) 35

2.10.1 Selbstbeschreibung durch den IDL-Datentyp any und TypeCodes 36

2.10.2 Problemfälle 38

2.11 Interfaces und Objekt-Referenzen 39

2.11.1 Typkonvertierungen für Objektreferenzen 40

2.11.2 Abstraktionen und Polymorphie: Vererbung in IDL 41

2.11.3 Problemfall 42

2.12 Das Interface-Repository (IR) 42

2.12.1 Sinn und Zweck des IR 42

2.12.2 Informationen im IR 43

2.12.3 Identifizierung der IR-Einträge 44

2.12.4 Informationen zur Objekt-Referenz 45

2.12.5 Informationen zum RepositoryId 46

2.12.6 Zusammenarbeit mit mehreren IRs 47

2.13 Das Dynamic Invocation Interface (DII) 47

2.13.1 NamedValue 49

2.13.2 NVList ... 50
2.13.3 Erzeugung von Requests ... 51
2.13.4 Synchrone Ausführung von Requests ... 54
2.13.5 Asynchrone Ausführung von Requests ... 54
2.13.6 Bewertung ... 55
2.14 Das Dynamic Skeleton Interface (DSI) ... 56
2.14.1 Überblick und Einsatzmöglichkeiten ... 56
2.14.2 Repräsentation eines Requests im Server ... 56
2.14.3 Die Request-Behandlungsroutine ... 57
2.15 Aktivierungsmodi für Server ... 57
2.15.1 Multi-threaded Server ... 59
2.15.2 Aktivierungsmodus: "Shared" ... 59
2.15.3 Aktivierungsmodus: "Unshared" ... 60
2.15.4 Aktivierungsmodus: "per method" ... 61
2.15.5 Aktivierungsmodus: "persistent" ... 61
2.15.6 Problemfälle ... 62
2.16 Bewertung ... 63
2.16.1 Objektmetamodell ... 63
2.16.2 Softwarearchitektur ... 64
2.16.3 Komponenten ... 64
2.16.4 Der Standard ... 65
3. Beispiel: ein Taschenrechner ... 67
3.1 IDL-Spezifikationen des calc-Objekts ... 67
3.2 Zusammensetzung von Datentypen und Übertragungssmodi ... 68
3.3 Server-Implementierung ... 69
3.4 Klient-Implementierung ... 71
3.5 Stubs und Header-Dateien ... 72
3.6 Objekt-Referenzen ... 73
3.7 Nichtblockierende Operationen ... 74
3.8 Erweiterungen von Interfaces ... 75
3.9 CORBA Exceptions ... 76
3.9.1 CORBA-System-Exceptions ... 77
3.9.2 CORBA-Nutzer-Exceptions ... 77
3.9.3 Problemfälle ... 81
3.10 Wrapper ... 81
4. Namensdienst: ein COSS Naming ... 86
4.1 Das COSS-Naming im Überblick ... 87
4.2 Beispielimplementation für das COSS-Naming ... 89
4.2.1 IDL-Definition ... 90
4.2.2 Die Methoden des Interfaces NamingContext: ... 91
4.2.3 Die Klassen naming_i und iterator_i ... 93
4.2.4 Persistente Objekte und das Loader-Konzept ... 94
4.2.5 Die Main-Funktion des Servers ... 95
4.2.6 Verbesserter Taschenrechner ... 97
4.3 Bewertung ... 98
5. Alternative Verteilungstechnologien im Vergleich ... 99
5.1 Microsoft: DCOM im Überblick ... 99

5.1.1 Historie ... 99
5.1.2 Ziele ... 100
5.1.3 COM-Schnittstellen ... 101
5.1.4 Binäre Interoperabilität ... 103
5.1.5 Microsofts IDL ... 104
5.1.6 Zugriff auf COM-Objekte ... 105
5.1.7 Aggregation und Wiederverwendung ... 107
5.1.8 Automatisierung oder OLE Automation ... 108
5.1.9 Verteilung im Netz ... 109
5.1.10 Interoperabilität zu CORBA ... 112
5.1.11 Zusatzdienste ... 112
5.1.12 Zukünftige Entwicklungen ... 113
5.1.13 Fazit ... 113
5.2 Sun: Java RMI im Überblick ... 115
5.2.1 Historie ... 115
5.2.2 Die Java Remote Method Invocation ... 115
5.2.3 Invocation Layer ... 117
5.2.4 Transport Layer ... 117
5.2.5 Spezifikation eines Interfaces ... 117
5.2.6 Server-Implementierung, Stubs und Skeletons ... 118
5.2.7 Namensdienst und Objekt-Referenzen ... 119
5.2.8 Persistenz ... 120
5.2.9 Ein Klienten-Applet ... 120
5.2.10 Java-RMI im Web ... 120
5.2.11 Interoperabilität ... 121
5.2.12 Fazit ... 122
5.2.13 Zukünftige Entwicklungen ... 123
5.3 Alternative Verteilungstechnologien im Überblick ... 124
6. Schlußbemerkungen ... 126
6.1 Bewertung ... 126
6.2 Ausblick auf zukünftige Entwicklungen (CORBA 3.0) ... 128
7. Anhang ... 130
7.1 Standardisierte Schnittstellen (IDL, CORBA2.0) ... 130
7.1.1 IDL-Definitionen für Object, ORB und BOA ... 130
7.1.2 IDL-Definitionen für das Interface Repository ... 131
7.1.3 IDL-Definitionen für das Dynamic Invocation Interface ... 139
7.1.4 IDL-Definitionen für das Dynamic Skeleton Interface ... 142
7.2 Ableitungen der SystemException-Klasse ... 142
7.3 ORB-Core Feature Matrix ... 143
7.4 CORBA Services Feature Matrix ... 144
7.5 CORBA OS-Plattform Matrix ... 145
8. Quellenverzeichnis ... 146
8.1 Literatur ... 146
8.2 World Wide Web ... 147
8.3 News Groups ... 148
9. Abbildungsverzeichnis ... 149
10. Index ... 150

Vorwort

Diese Ausarbeitung erfolgt im Rahmen der Diplom-Arbeit des Studiengangs Wirtschafts-Informatik der Fachhochschule Rendsburg (Schleswig-Holstein).
Die Auswahl des Themas ist einerseits beruflich bedingt: Aus der deutschen Zentrale eines internationalen Textil-Konzerns heraus sollte eine Anwendung den einzelnen Filialen im gesamten Bundesgebiet einschließlich Österreichs zur Verfügung gestellt werden. Die Umsetzung erfolgte auf einfache Weise über ISDN (Remote Access) und Terminal-Emulationen auf PC's.
Diese verbesserungswürdige Situation und andererseits der Wunsch, einen *allgemeinen* und *zukunftsgerichteten* Weg zur Anwendungsverteilung zu beschreiben, ist die Motivation der nun folgenden Arbeit. Sie richtet sich vorwiegend an Entwickler und Entscheider, die sich einen Überblick über Verteilungstechnologien und deren Funktionsweise auf OO-Basis verschaffen wollen.
Danken möchte ich meiner Schwester Katja für die stetige Ermunterung und das sorgfältige Lesen dieser Arbeit.

1. Einleitung

1.1 Aufbau der Arbeit

Die Arbeit beginnt mit einem Überblick über Verteilungsmodelle und Middleware-Konzepte der letzten Jahre. Das Kapitel 2 wird sich mit der Beschreibung und dem Einsatz objektorientierter Verteilungsmechanismen der Common Object Request Broker Architecture (CORBA) befassen. In diesem Rahmen muß auch näher auf die Paradigmen und Konzepte der Objektorientierung eingegangen werden. Danach werden die historischen Entwicklungen, Grundlagen, Dienste (fortan Services und Facilities genannt) und Architekturkonzepte vorgestellt. Im Anschluß wird zur Veranschaulichung eine kleine Beispielanwendung implementiert (Kapitel 3) und zum Beweis der Integrationsfähigkeit ein auf Methodenebene standardgerechter Namensdienst und sein Gebrauch (Kapitel 4) vorgestellt. Daran schließen sich die Beschreibung und der Vergleich konkurrierender Verteilungstechnologien an (Kapitel 5). In den Schlußbemerkungen (Kapitel 6) kommt es zur einer kritischen Würdigung, in der Aspekte wie fehlende Standardisierungsdetails, mangelnde Unterstützung von Fehlerbedingungen (Exceptions) und Wertesemantiken (Objekt-Referenzen) behandelt werden, und einem Ausblick auf zukünftige Entwicklungen (CORBA 3.0).

Aufgrund der zahlreichen internen und externen (und noch aktuellen) Hyper-Links möchte ich noch darauf hinweisen, daß zum Studium dieser Arbeit ein PDF-Reader (Portable Document Format) am besten geeignet ist[1]. Ist ihm ein mit dem Internet verbundener Browser vorgeschaltet, können während des Lesens sämtliche Referenzen auf das World Wide Web (im folgenden mit "[www]" bezeichnet) direkt eingesehen werden. Sie stellen eine thematische Erweiterung dar und dienen der Abrundung des jeweiligen Themengebietes. Deshalb liegt den ausgedruckten Exemplaren auch jeweils eine Diskette in der PDF-Version dieser Arbeit bei.

1.2 Abgrenzung und Überblick über Client-/Server-Modelle und Middleware-Konzepte

Seit mehreren Jahren entwickeln sich zentrale Mainframe-basierte Anwendungssysteme hin zu dezentralen Applikationen auf Client-/Server-Systemen. Wie die zentralen Anwendungssysteme auf Hostrechnern ablaufen und Benutzer mit einfachen Terminals auf sie zugreifen, verteilen sich dezentrale Anwendungen auf spezialisierte Server und leistungsfähige Clients meistens in Form von PC's. Die *Middleware* als vermittelnde Software-Schicht zwischen Clients und Servern nimmt vielfältige Aufgaben in verteilten Informationssystemen wahr, was folgende Abbildung nochmals grafisch illustriert.

---

[1] Er kann z.B. als "Acrobat Reader 3.0" der Firma Adobe in http://www.adobe.com/acrobat bezogen werden.

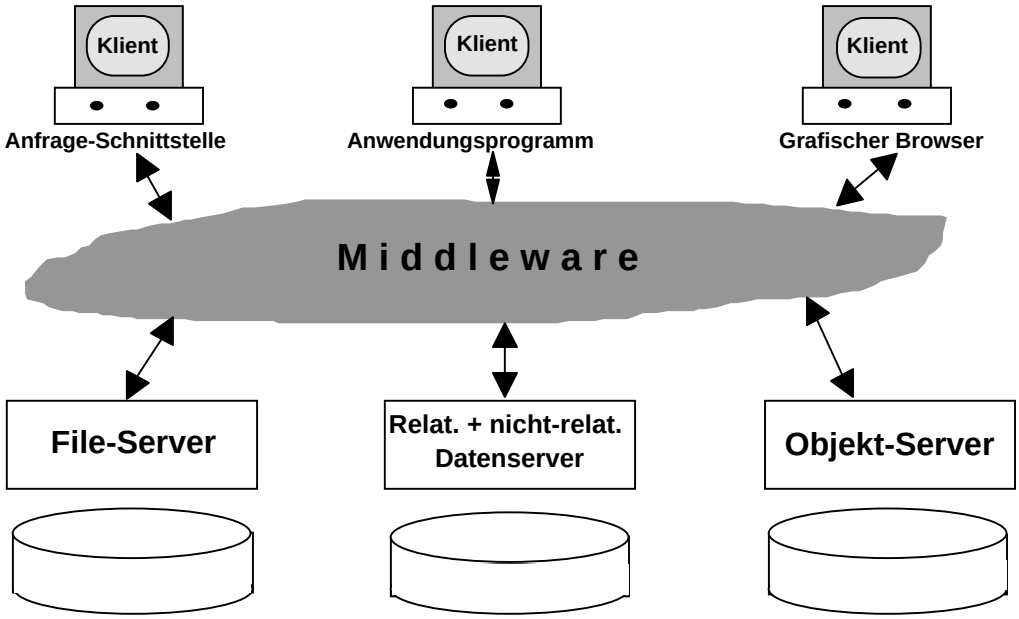

Abbildung 1: Middleware-Architektur mit verschiedenen Client-/Server-Typen

Zur Charakterisierung verteilter Anwendungssysteme können folgende *Vorteile* genannt werden:

- der gute Wirkungsgrad durch fachliche (inhaltliche) und technische Arbeitsteilung zwischen Klient und Server,
- die hohe Anpassungsfähigkeit und gute Skalierbarkeit,
- die Effizienz und Wirtschaftlichkeit gemeinsamer Nutzung von Ressourcen (z.B. Peripheriegeräte),
- die guten Einbindungsmöglichkeiten zusätzlicher Ressourcen,
- die Vermeidung zentraler Fehlerorte durch Datenkopien (Replikation) und Spiegelung (Shadowing) von Servern,
- das breite Marktangebot und der Einsatz bekannter Werkzeuge auf Klienten- und Server-Seite (z.B. zur Implementierung von Benutzerschnittstellen)
- das gute Preis- Leistungsverhältnis und das große Angebot standardisierter Hard- und Software.

Diesen Vorteilen verteilter Anwendungssysteme stehen folgende bekannte *Nachteile*:

- die aus einer inhärenten Komplexität der Entwicklung verteilter Anwendungen entstehenden hohen Entwicklungskosten,
- der höhere Aufwand für Wartung und Administration,
- die im Vergleich zu Mainframes geringere und zu Engpässen führende Leistung von Servern,
- die aus erhöhter Client-Server- und Server-Server-Kommunikation resultierenden Sicherheitsprobleme und
- die z.T. mangelhafte Verfügbarkeit von Werkzeugen zur Entwicklung und Wartung

gegenüber.

So führt die oft unterschätzte Entwicklungskomplexität zu schwerwiegenden Problemen in verteilten Client-/Server-Anwendungssystemen. Eine große Anzahl von Entwurfsalternativen, und eine verwirrende Vielfalt verschiedenster Middleware-Komponenten erschweren zudem den Einsatz und die Übersicht über diese Technologien, denen sich ein Entwickler

solcher Systeme gegenüber sieht. In diesem Abschnitt wird somit auf eine zentrale Schlüsseltechnologie der *Middleware* eingegangen, die überwiegend zur erfolgreichen Entwicklung und Verbreitung verteilter Client-/Server-Informationssysteme beigetragen hat. Zum Begriff Middleware ist in [NEXTGEN][1] zu lesen: "....Middleware is system integration software for distributed processing and for database and user interfaces. Distributed processing middleware is rapidly migrating toward supporting the Object Management Group's CORBA. Other distributed processing standards, such as Microsoft's Object Linking and Embedding (OLE), are developing CORBA support.".

1.2.1 Klassische Aufgaben von Middleware

Zu den Bausteinen verteilter Anwendungssysteme gehören der Klient, Server und die Middleware. Unter *Klienten* können eine große Anzahl konkreter Systeme, z.B. PC's, Workstations, Laptops, Mobil-Telefone etc. verstanden werden. Auch bei *Servern* kann von physischen und logischen Einheiten ausgegangen werden wie z.B. Workstations, Satelliten oder Informationsdienste (Börsenticker, Newsnet etc.).
In der Regel sind die Vorstellungen über *Middleware* aber wesentlich weniger präzise. Sie besteht meistens aus einer hoch entwickelten Kommunikationskomponente, die Klientenanwendungen mit Servern verbindet. Jedoch weit darüber hinausgehend ist die Middleware in der Lage, auf verschiedenen Transparenz-Ebenen die unterschiedlichsten Dienste und Funktionalitäten zu realisieren. Zu den typischen Middleware-Diensten können gezählt werden:

Middleware-Dienste					
Kommunikation	**System**	**Information**	**Ablaufkontrolle**	**Präsentation**	**Berechnung**
- Peer-to-peer Messaging - Remote Procedure Call - Message Queuing - elektr. Post - elektr. Datenaustausch	- Event-Notification - Konfigurationsmanagement - Software-Installation - Fehlererkennung - Recovery-Koordination - Authentifizirung -Verschlüsselung	- Directory Server - Log Manager - File- + Record-Manager - Relationale + objektorientierte DB - Repository-Verwaltung	- Thread Manager - Transaktionsverarbeitung - Resource-Broker - Request-Broker - Job Scheduler	- Masken- + Grafikverarbeitung - Drucker-Verwaltung - Hypermedia Verbindungen - Multimedia-Aufbereitung	- Sortieren - mathem. Berechnungen - Internationalisierung - Datenkonversion - Zeit-Management

Wir wollen uns hier auf die klassischen Aufgaben der Middleware, der Kommunikation zwischen Klienten und Servern sowie der Zugriff auf verteilte Datenbanken, konzentrieren. In diesem Rahmen werden die zwei grundsätzlichen Mechanismen der Client-/Server-Kommunikation, *Remote Procedure Call* und *Message Queuing*, diskutiert. Anschließend werden die auf diese Mechanismen aufbauende SQL-Middleware und ihre Verbreitungsformen erörtert.

[1] [NEXTGEN] Aronica und Rimel: S. 140

1.2.2 Kommunikations-Middleware

Der *Remote Procedure Call* (RPC) stellt eine Möglichkeit dar, mit der eine Prozedur durch einen auf einen entfernten Server laufenden Prozeß ausgeführt werden kann. Durch einfache Erweiterung des lokalen Prozeduraufrufes gelangt man zur Entwicklung verteilter Anwendungen, wie dies seit Mitte der 80er Jahre Grundlage für die meisten Client-/Server-Datenbanksysteme ist. Deshalb führt die Einfachheit des RPC zur schnellen und effizienten Anwendungsentwicklung.
Außer den auf Threads basierenden asynchronen Mechanismen gehören die RPC's zur *synchronen* Kommunikationsart, die den Klienten vor der Fortsetzung der Codeabarbeitung zum Warten auf die Antwort zwingt. Voraussetzung dazu ist eine stetige Netzwerkverbindung und Verfügbarkeit des Servers. In fortgeschrittenen Anwendungen, die beispielsweise das mobile Computing beinhalten, ruft dieses Verfahren jedoch Probleme hervor.
Zum Beispiel in [SLOKRA88][1] kann die Entwicklung mit Remote Procedure Calls nachvollzogen werden. Sie führt zur Erkenntnis, daß diese *low-level*-Programmierung uns mit Recht eine semantisch höhere Abstraktionsebene der verteilten Entwicklung mit reicheren Verteilungsdiensten wünschen läßt. Hier tritt der Standard des *Distributed Computing Environment* (DCE) der Open Software Foundation (OSF) auf, siehe auch Abschnitt Namensdienst: ein COSS Naming. Er stellt nicht nur eine Übereinkunft verschiedener Hersteller für RPC-Implementierungen dar, sondern standardisiert darauf aufbauend auch erweiterte Konzepte wie verteilte Katalog-, Sicherheit-, Zeit- und File System-Dienste [LANG94][2].
Im Gegensatz zum RPC realisiert das *Message Queuing* die *asynchrone* Kommunikation. Zur Verwaltung von Message Queues ist neben einem automatisierten Mechanismus in Form eines Queue Managers (wie z.B. der mail daemon) auch ein Persistenz-Dienst zum Zwischenspeichern der Meldungen notwendig. Als Ein-Weg-Kommunikation verliert der Sender nach Abschicken der Meldung sämtliche Kontrolle über die Weiterleitung und Ankunft beim Empfänger. Wie Nachrichtenwarteschlangen zu implementieren sind und auf welche IPC-Mechanismen sie basieren, kann z.B. in [ROCH88][3] nachgelesen werden.
Deshalb ist das Message Queuing für Echtzeit-Anwendungen unbrauchbar. Jedoch findet es in solchen Anwendungssystemarten seine Verbreitung, in denen Klienten und Server nur teilweise verfügbar sind. Als typisches Einsatzbeispiel sind hier Mail-Systeme zu nennen. Nachrichten-Warteschlangen sind wesentlich flexibler als RPC und ermöglichen Variationen, bei denen Meldungen an mehrere Empfänger gesendet (Broadcasting) oder an andere Empfänger weiter- oder umgeleitet werden. Aktuelle Entwicklungen wie das Produkt "MQ-Series" der IBM [www] stellen außerdem persistente und zuverlässige Queues mit transaktionsorientierten Recovery-Mechanismen bereit. Vereinfachte Integration verschiedener Message Queue-Produkte und die Fähigkeit der Nachrichten, den Zusammenhang über sie auslösende Ereignisse mit sich zu führen, verspricht Hewlett-Packard mit dem Produkt "Broadcast Message Server"[4]. Das Produkt "Tooltalk Service" der Sun Microsystems [www] bietet über ein beschreibendes Message-Protokoll ein Informationsstatus und Lebenszyklus der jeweiligen Nachrichten.

1.2.3 SQL-Middleware zum Zugriff auf verteilte Daten

Obwohl die Structured Query Language (SQL) als Standard große Akzeptanz beim Aufbau und der Anwendung von Informationssystemen gefunden hat, ist es beinahe unmög-

[1] siehe dazu auch [SLOKRA88] S. 271 - 276
[2] siehe dazu auch [LANG94] S. 308 - 311
[3] siehe dazu auch [ROCH88] S. 202 ff.
[4] siehe dazu auch [NEXTGEN] Narain, Rimel und Fingar: S. 97

lich, *transparent* auf relationale Datenbanksysteme (RDBMS) *unterschiedlicher* Hersteller zugreifen zu können. So bedarf es eines großen Aufwandes, um die Unterschiede der SQL-Dialekte der Datendefinitions-, Datenmanipulations- und Datenabfragesprachen jeweiliger Hersteller zu egalisieren. In Abhängigkeit der Sprach-"Standards" kommen differierende Netzwerkprotokolle, Hardware-Plattformen und Programmierschnittstellen hinzu.

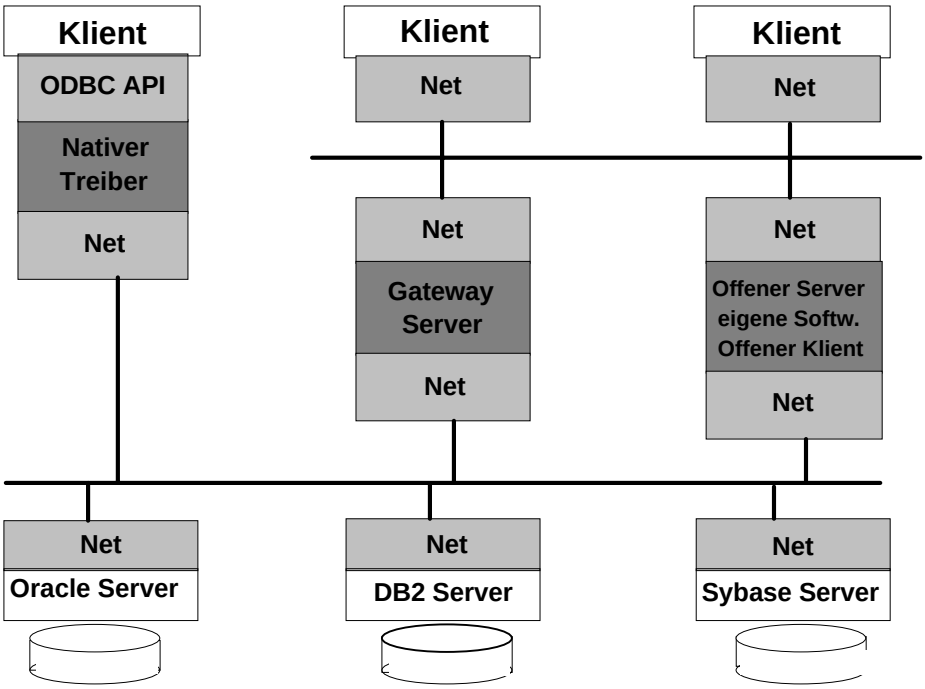

Abbildung 2: Möglichkeiten der Klienten-Anbindungen an Datenbank-Server

Drei Grundformen von SQL-Middleware sollen hier detaillierter beschrieben werden, die die o.g. Unterschiede für Klienten unsichtbar machen: SQL-Treiber, SQL-Gateway Server und offene SQL-Server, wie auch in obigem Schema demonstriert wird.

Native SQL-Treiber stellen die einfachste Form von SQL-Middleware dar, um herstellerunabhängig den Zugriff auf relationale Datenbanken zu bewerkstelligen. Z.B. Microsofts ODBC (Open Database Connectivity) bietet für den Datenbankzugriff eine Programmierschnittstelle an, mit deren Hilfe Aufrufe in Abhängigkeit der zugrunde liegenden Datenbank von einem entsprechenden Treiber übersetzt und ausgeführt werden. Native SQL-Treiber können auf der Klientenseite sowohl *statisch* zur Anwendung gebunden oder vom Klienten über eine *dynamische* Aufrufschnittstelle (Dynamic Invocation Interface) aufgerufen werden.

Im Unterschied zu nativen SQL-Treibern läuft ein **SQL-Gateway Server** in einem eigenen Prozeß und kommuniziert mit Klienten über eine unveränderte SQL-Schnittstelle, wie dies in den Produkten "EDA/SQL" der Firma Information Builders [www] oder "OmniCONNECT" [www] von Sybase zu beobachten ist. Die Strategie dieses Konzeptes besteht in dem gleichen Erscheinungsbild, in dem der SQL-Gateway Server dem Klienten gegenüber auftritt. Jedoch verwaltet der Gateway-Server keine eigene Daten, sondern übersetzt das SQL-Statement in die Anweisung des jeweiligen DB-Herstellers. Alle Gateway-Funktionen des Servers sind aus der Klientensicht vollständig transparent.

Das Hauptmerkmal **offener SQL-Server** besteht in der für den Klienten transparenten Erweiterung des Servers durch benutzerspezifische Programme, wie dies "OpenClient/Server"

von Sybase ermöglicht. Auch hier tritt der Server dem Klienten gegenüber als normale SQL-Schnittstelle auf, um die jeweilige Erweiterung für ihn transparent zu halten. Die den Open Server erweiternden benutzerspezifischen Programme werden ausgeführt, bevor das SQL-Statement weitergeleitet wird. Neben der Möglichkeit, sämtliche proprietären Eigenarten des RDBMS vor dem Klienten zu verbergen, können z.B. auch Änderungsoperationen auf einer Datenbank protokolliert oder globale Transaktionen koordiniert werden.
Durch den Umstand, daß Gateway- und Offene Server nicht Bestandteil lokaler Klientenanwendungen sind und in eigenen Prozessen ablaufen, können sie auch im Netz je nach der Auslastungssituation verteilt werden. Dies führt einerseits zu höherer Ausführungssicherheit und andererseits zu mehr Flexibilität, da darunterliegende Datenbankserver transparent werden und teilweise auch zur Laufzeit ausgewechselt werden können.

1.2.4 Aktuelle und neue Middleware-Entwicklungen

Stellen die vorab beschriebenen Middleware-Typen eher Basisfunktionalitäten als Infrastruktur dar, haben sich nun seit einiger Zeit neue Formen entwickelt, die Kommunikationsdienste auf einer höheren semantischen und objektorientierten Ebene realisieren und dabei große Datenmengen verwalten oder aufwendige Ablaufaufgaben kontrollieren können.

1. Die Object Management Group (OMG) entstand 1989 aus dem Notstand, daß es kein standardisiertes, kommerziell verfügbares und weit verbreitetes Rahmensystem für die Entwicklung *verteilter, objektorientierter* Informationssysteme gab.
   Da sie die bestehenden Mechanismen zur Entwicklung verteilter Anwendungen für unzureichend hielt, verfolgt sie einen auf Objekttechnologie basierenden Ansatz, der die Grundsätze der Objektorientierung umfaßt, um eine einheitliche Sicht auf verteilte und heterogene Systeme zu erlangen: *Datenkapselung*, *Polymorphie* und *Vererbung*. Es entstand der **CORBA**-Standard (Common Object Request Broker Architecture), der zur Interoperabilität und Portabilität auf der Basis einer objektorientierten Spezifikation befähigt. Zu den Bestandteilen der CORBA gehören eine einheitliche Terminologie der Objektorientierung, ein abstraktes Objektmodell, einer Referenzarchitektur und ein gemeinsames Protokoll. CORBA selbst ist in die Object Management Architecture (OMA) eingebettet. Diese dient einer Referenzarchitektur, die sich aus standardisierten Spezifikationen folgender Komponenten zusammensetzt:
   - ein Object Request Broker (ORB) umfaßt den Vermittler und Kommunikationskern der CORBA,
   - die Common Object Services stellen elementare und anwendungsunabhängige Basisoperationen zur Verwaltung (Modellierung und Speicherung) von Objekten zur Verfügung, wie z.B. Naming, Relationships, Persistence oder Transactions,
   - die Common Facilities definieren Sammlungen spezialisierter und anwendungsspezifischer Komponenten, die in vielen Applikationen wiederverwendet werden können, z.B. User Interfaces (Navigieren, Abfragen, Drucken) oder Compound Documents,
   - unter den Application Objects werden die eigentlichen, konkreten Anwendungsobjekte verstanden, die natürlich keiner Spezifikation unterliegen.

   Der zentrale Mechanismus des Object Request Brokers läßt lokale Anwendungen mit entfernten Objekten über Netzwerk- und Betriebssystemgrenzen hinweg miteinander kommunizieren („Object Bus“). Zu den Grundsätzen der OMA gehört die Annahme, daß die Klientenanwendungen nur Objekt-Identifikatoren oder -referenzen enthalten und weder Aufenthaltsort noch Sprache der Objekt-Implementierung kennen müssen.