Einsatz von Applikationsservern

Untersucht am Beispiel des Sybase „Enterprise Application Server“

Andreas Schwarzbauer

Geb. am 7. April 1975 in Berlin

Diplomarbeit

Zur Erlangung des akademischen Grades

Diplom-Ingenieur (FH)

eingereicht an der

Fachhochschule Brandenburg

Fachbereich Informatik und Medien

Betreuer:

Prof. Dr. habil. Manfred Günther,

FB Informatik und Medien, FH Brandenburg

Dipl.-Inform. Ingo Boersch,

FB Informatik und Medien, FH Brandenburg

Einleitung und Motivation

Was ist ein Applikationsserver?

Eine Applikation ist eine ausführbare Anwendung. Ein Server stellt in einem Netzwerk eine Anzahl von Diensten bereit. Folglich stellt ein Applikationsserver Anwendungen in einem Netzwerk zur Verfügung.

In dieser Diplomarbeit werden die Fragen nach dem Was, Wie und Warum beantwortet. In den *Kapiteln 1 und 2* werden die Grundlagen verteilter Datenbank-Anwendungen und den dabei verwendeten Techniken erläutert. In *Kapitel 3* bekommt man einen Überblick über die größten, momentan verfügbaren Applikationsserver-Lösungen. Es werden auch einige technische und konzeptionelle Unterschiede der verschiedenen Lösungen aufgezeigt. Am Beispiel des „Enterprise Application Server“ von Sybase wird dargestellt, wie ein Applikationsserver eingerichtet, programmiert und verwendet wird. In *Kapitel 4* beginnt der praktische Teil der Diplomarbeit. Hier wird auf der Basis des Sybase-Systems eine Funktion eines fiktiven, größeren Projektes umgesetzt. Es geht dabei um die Registrierung einer Teilnahme an einem Kurs, der aus einem Kursangebot ausgewählt wird. Um die Unterschiede zwischen der klassischen Programmierung und der eines Applikations-servers kennenzulernen, wird die Aufgabe in mehreren Schritten gelöst. Vom Client-Server Programm über eine Variante, bei der der Datenbank-Server einige Aufgaben übernimmt, bis hin zur reinen Verwendung des Applikationsservers gibt es verschiedene Lösungsansätze.

Hat sich der Mehraufwand gelohnt? Wann lohnt er sich und für wen?

Diese Fragen werden in *Kapitel 5* beantwortet.

Inhaltsverzeichnis

Kapitel 1 Datenbank-Architekturen

In diesem Kapitel werden verschiedene Architekturen von Datenbank-Systemen vorgestellt. Durch die Betrachtung der Unterschiede vom Client-Server-System zu einem mehrschichtigen Server-System wird die Entwicklung in diesem Bereich deutlich.

1.1 Einsatzgebiete von Datenbanken

Heutzutage zählen Datenbanksysteme zu den wichtigsten Software-komponenten. Immer wenn es eine große Menge an Daten zu verarbeiten gilt, werden Datenbanken eingesetzt. Vor allem der Boom von Internet und E-Commerce hat die Anzahl der verwendeten Datenbank-Systeme in die Höhe getrieben. Nicht zu vergessen sind auch die klassischen Anwendungsfelder, wie das Bankwesen oder betriebliche Informations-systeme.
Durch die Weiterentwicklung der klassischen, relationalen Datenbanken zu größeren, verteilten Systemen haben sich viele neue Einsatzgebiete erschlossen, wie z.B. spezialisierte Informations- oder Ingenieurssysteme. [01]

1.2 Das Client-Server-Modell

Um mehr als einem Benutzer die Möglichkeit zu geben, auf bestimmte Daten zugreifen zu können, muss es eine zentrale Stelle geben, welche die Daten verwaltet. Diese Aufgabe übernimmt der Datenbank-Server. Die Benutzer verwenden einen Client um Zugriff zu bekommen. Dieser kann ein Programm oder ein komplettes System sein. So ist es möglich, dass viele Clients mit zentral verwalteten Daten arbeiten können.

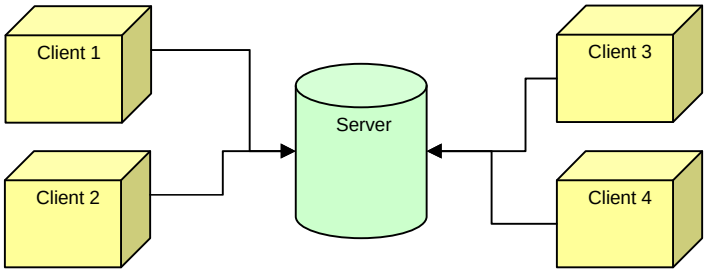

Abb. 1-1 Ein Client-Server-Modell [01]

Da die Clients meist auf anderen Rechnern arbeiten als der Datenbank-Server, wird dieser entlastet. Die zentrale Verwaltung der Daten erhöht die Effizienz, Redundanzen werden vermieden und die Integrität der Daten ist sichergestellt. [01]

1.3 Funktionale Verteilung

Betrachtet man die Aufgaben, die ein Client-Server-System zu bewältigen hat, lassen sich diese in verschiedene Funktionsgruppen einteilen:

A) *Die Präsentation der Daten, Benutzerinteraktion*
   Was der Anwender oder der Administrator zu sehen bekommt. Hier kann man Daten einsehen oder manipulieren.
B) *Die Anwendungslogik*
   Der eigentliche Kern der Anwendung. Hier wird die Anwendung ausgeführt.
C) *Der Datenzugriff*
   Die Schnittstelle der Anwendungslogik zu den eigentlichen Daten. Regelt z.B. Transaktionen.
D) *Die Datenerhaltung*
   Die Datenbank und die Methodik zur Benutzung.

Wenn man diese Unterteilung nicht nur theoretisch anwendet, sondern aus diesen Teilen Einzelkomponenten bildet, die dann untereinander

kommunizieren, entstehen mehrschichtige Systeme, die im Folgenden erklärt werden. [01]

1.4 Das Zwei-Schichten-Modell (2-tier)

Beim Zwei-Schichten-Modell wird die Funktionalität meist vollständig im Client implementiert. Den Datenzugriff auf den Datenbank-Server regelt das sog. Datenbank-Management-System, oder kurz DBMS.

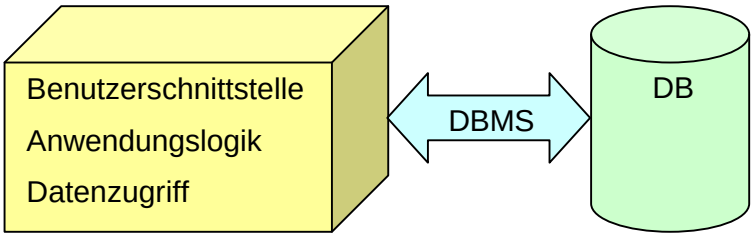

Abb. 1-2 Zwei-Schichten-Modell [01]

Um mit dem DBMS zu kommunizieren, existieren in vielen Programmiersprachen Standard-Schnittstellen wie z.B. in JAVA die Java-Database-Connectivity (JDBC).
Anwendungen auf diese Art und Weise zu erstellen ist vergleichsweise einfach, bringt jedoch einige Nachteile mit sich. Da der Client den größten Teil der Arbeit erledigt, muß er genügend Leistung haben. Wird die Software verändert, muß sie auf jedem Client neu aufgespielt werden (automatisch oder manuell). Jede Datenmanipulation setzt den Transfer der Daten zum Client voraus. Datenintensive Operationen haben so auch einen hohen Datentransfer zur Folge, der leicht zur Überlastung des verwendeten Netzwerkes oder des Datenbank-Servers führt. [01]

1.5 Das Drei-Schichten-Modell (3-tier)

Bei dem Drei-Schichten-Modell wird die eigentliche Anwendungslogik in eine zusätzliche Schicht ausgegliedert. Dadurch entsteht ein Applikationsserver,

der z.B. höherwertige Funktionen in Form von Objekten und deren Methoden anbietet. Diesen Teil nennt man auch Geschäftslogik oder „Business Logic“. Die Software auf dem Applikationsserver nennt man „Middleware“. Da der Client keinen direkten Zugriff auf den Datenbestand hat, wird sichergestellt, dass ein Client nur die für ihn freigegebenen Daten sehen und bearbeiten kann. Für die Kommunikation zwischen Client und Applikationsserver werden spezielle Middleware-Lösungen wie z.B. CORBA verwendet. Aber auch herkömmliche Web-Mechanismen wie HTTP lassen sich für den Client verwenden, wenn auch nur mit der Hilfe von Servlets oder serverseitigen Scriptsprachen. Der Applikationsserver und der Datenbank-Server kommunizieren hier wieder über das entsprechende DBMS-Protokoll.

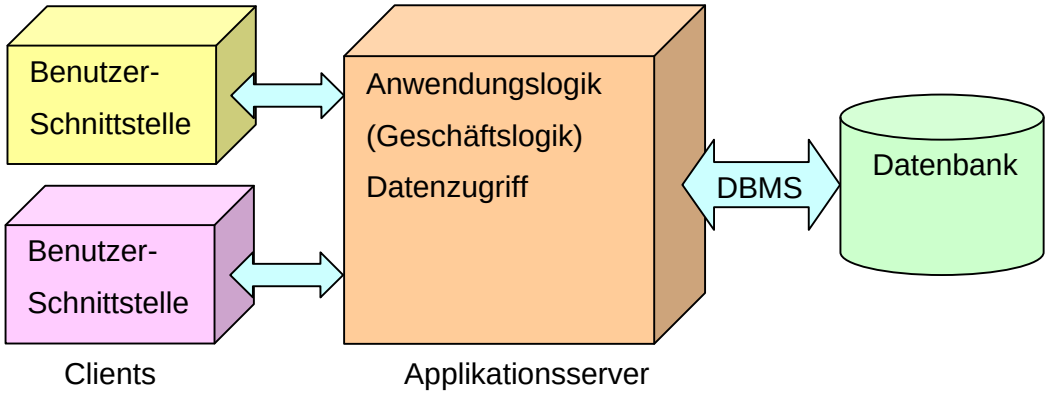

Abb. 1-3 Drei-Schichten-Modell [01]

Da die eigentliche Anwendung auf dem Applikationsserver liegt, müssen Änderungen auch nur dort vorgenommen werden. Man kann verschiedene Arten von Clients verwenden, ohne die Anwendungslogik verändern zu müssen. Bei datenintensiven Operationen bringt ein Applikationsserver Vorteile in der Geschwindigkeit, da die Verbindung zum Datenbank-Server meist größer dimensioniert ist als die Verbindung zu den Clients. Es genügt eine langsame Verbindung zum Client, da die eigentlichen Daten nur über die DBMS-Schnittstelle ausgetauscht werden. Der Entwicklungsaufwand ist zwar höher, das System ist aber besser zu warten und zu erweitern. [01]

Des weiteren ist es möglich, die Anwendungslogik in noch mehr funktionale Teile zu unterteilen oder den Datenzugriff von der Anwendungslogik zu

trennen. Bei jeder weiteren Teilung steigt natürlich der Kommunikationsaufwand. Deshalb lohnt sich dieser Aufwand nur, wenn das Projekt so komplex ist, dass die Kommunikation der Programmteile bei Betrachtung der Performance in den Hintergrund tritt.

Kapitel 2 Implementierungstechniken

Durch die Größe und Komplexität heutiger Datenbankanwendungen kommen eine Vielzahl verschiedener Techniken für die Umsetzung und Kommunikation zum Einsatz, von denen die wichtigsten in diesem Kapitel erklärt werden. Um den Zusammenhang und die Übersicht zu verbessern, werden die Techniken in 3 Kategorien eingeteilt:

- Der Datenbank-Server
- Der Applikationsserver
- Der Client

2.1 Der Datenbank-Server

Der Datenbank-Server hält die eigentlichen Daten. Er muß Techniken zur Verfügung stellen, um den Zugang zu den Daten zu gewährleisten.

2.1.1 Das Datenbank-Management-System (DBMS)

Das Datenbank-Management-System (DBMS) beschreibt alle Komponenten für den Datenzugriff und die Datenverarbeitung einer Datenbank. Der Aufbau eines DBMS wurde von der ANSI-SPARC[1] Gruppe bereits in den Siebziger Jahren definiert und gilt heute als allgemein anerkannt. Das DBMS läßt sich in verschiedene Komponenten mit speziellen Funktionen einteilen.

[1] ANSI: American National Standards Institute, SPARC: Standards Planning and Requirements Committee

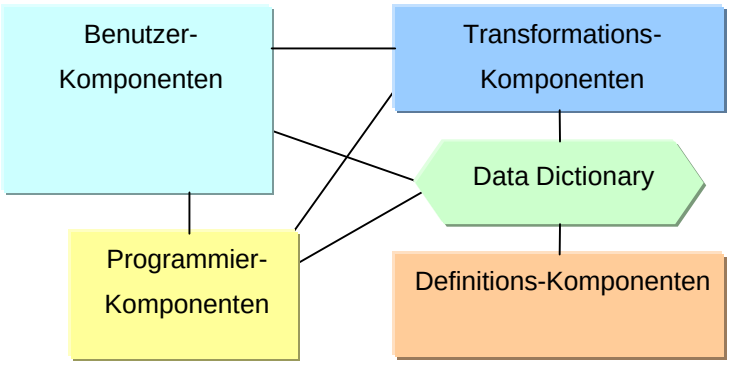

Abb. 2-1 Struktur eines DBMS [02]

Die *Definitionskomponenten* dienen zur Definition und Verwaltung der Daten durch einen Administrator.
Die *Programmierkomponenten* stellen eine Umgebung zur Verfügung, mit der man in einer höheren Programmiersprache eigene Zugriffsmöglichkeiten auf die Datenbank entwickeln kann, wie z.B. eine grafische Benutzeroberfläche.
Die *Benutzerkomponenten* beinhalten die Werkzeuge für die Manipulation des Datenbestandes durch den Benutzer.
Die *Transaktionskomponenten* wandeln die Datenbank-Befehle in reale Lese- und Schreiboperationen auf den eigentlichen Datenträger um.
Das *Data Dictionary* (Datenwörterbuch) enthält die in der Definitionskomponente festgelegten Datenstrukturen und gibt sie an die anderen Komponenten weiter. [02]

2.1.2 Open-/Java- Database Connectivity (ODBC, JDBC)

ODBC bzw. JDBC sind die Abkürzungen für *Open-/Java- Database Connectivity* und bezeichnen Treiber, um in Java oder einer anderen Programmiersprache relationale Datenbanksysteme anzusprechen. Durch diese Programmierschnittstelle können die Funktionen unterschiedlicher Datenbanken einheitlich genutzt werden, da das Datenbankprotokoll schon im Treiber implementiert ist. Der xDBC Treiber ist eine der wichtigsten Schnittstellen eines DBMS zu anderen Programmiersprachen oder

Systemen. Die Sprache SQL, auf die im nächsten Abschnitt eingegangen wird, steuert die Kommunikation mit der Datenbank. [04]

2.1.3 Standard Query Language (SQL)

Mit der „Standard Query Language" (SQL) kann man Anfragen (engl. „Queries") an eine relationale Datenbank senden, um Daten zu erstellen, auszulesen oder zu manipulieren. Entstanden ist diese Sprache aus SEQUEL („Structured English Query Language"). SEQUEL war in den frühen siebziger Jahren eine Anfragesprache für eine der ersten relationalen Datenbanken, dem „System R" von IBM. Im Jahre 1986 wurde die erste SQL-Norm vom ANSI Konsortium verabschiedet, 1992 entstand die zweite Version von SQL (SQL 2, bzw. SQL-95 genannt). SQL2 wird von allen wichtigen Datenbanken unterstützt und erlaubt somit die Steuerung verschiedenster Systeme mit einem Standard. [04]

2.1.4 Stored-Procedures / Transact-SQL

Die sog. „Stored-Procedures" sind Funktionen, die ein Datenbank-System intern ausführen kann. Sie ermöglichen es, datenintensive Operationen schnell und direkt auf dem Datenbank-Server auszuführen, ohne die Daten über eine langsame Schnittstelle zu einer verarbeitenden Applikation zu schicken. Dazu stehen zusätzlich zu den normalen SQL-Befehlen eine Verwaltung für Variablen und einige Kontrollstrukturen zur Verfügung, wie z.B. Schleifen und Fallunterscheidungen.
Im praktischen Teil der Diplomarbeit werden die Stored-Procedures in Transact-SQL programmiert. Diese Sprache ist der Standard bei Sybase-Datenbanken.

2.1.5 Embedded SQL in Java (SQLJ)

Um in Java auf eine relationale Datenbank zuzugreifen, existierte bisher nur die JDBC Schnittstelle. Um die Möglichkeit zu haben, SQL Befehle direkt in den Quelltext einzubetten (engl. „embedding“), wurde von einigen namhaften Softwareherstellern SQLJ als höhere Schnittstelle festgelegt. Wie schon lange in anderen Programmiersprachen wie C, Cobol und Fortran möglich, kann dies nun auch in Java die Entwicklung von Datenbankanwendungen vereinfachen.
Der ANSI Standard für „embedded SQL“, der für die anderen Programmier-sprachen existiert, wurde hier leicht abgewandelt, um die Objektorientierung von Java besser zu nutzen. Der SQLJ-Quellcode wird bei der Übersetzung durch einen Präprozessor in Java-Quellcode umgewandelt. Das hat den Vorteil, dass zur Übersetzungszeit schon eine Syntax- bzw. Typenprüfung stattfindet. Da die Operationen zu diesem Zeitpunkt schon festgelegt sein müssen, ist die Verwendung von SQLJ weniger flexibel als die Verwendung von JDBC. [19]

2.2 Der Applikationsserver

Der Begriff Applikationsserver als Beschreibung der mittleren Schicht eines Dreischichtsystems umschreibt alle möglichen, ausführenden Programme und verschiedenste Techniken.
Die wichtigsten werden in diesem Kapitel beschrieben.

2.2.1 Scriptsprachen auf dem Server (Server sided Script)

Es wurden im Laufe der Entwicklung des Internets immer mehr Script-sprachen entwickelt, die speziell auf die Ausführung von webbasierten Applikationen ausgerichtet sind. Die meisten dieser Scriptsprachen haben den Nachteil, dass sie für die Kommunikation mit dem Client auf HTML angewiesen sind. Da jede HTML-Seite für sich ausgeführt wird, muß die

Übergabe von Variablen innerhalb einer Web-Applikation über komplizierte Umwege geschehen, meist über Sessions oder Cookies.
Eine Gruppe dieser Scriptsprachen läßt sich direkt in den HTML-Code einbetten. Diese werden dann von einem Interpreter auf dem Server verarbeitet und die Ausgabe im HTML-Format weitergeleitet.
Zu dieser Gruppe gehören z.B. die Active Server Pages (**ASP**) von Microsoft und **PHP**, ein Open-Source Projekt der PHP Group[1].
Auch Sybase bietet mit **Dynamo** eine Scriptsprache, welche HTML mit der Funktionsvielfalt ihres Applikationsserver-Paketes erweitert, von Datenbank-Schnittstellen und SQLJ bis hin zur Nutzung des Komponentenservers.

2.2.2 Java

Java ist eine objektorientierte Programmiersprache, die im Serverbereich weit verbreitet ist. Ein Java-Programm wird immer erst zur Laufzeit kompiliert, das macht Java plattformunabhängig, aber auch recht langsam. Dieses Kapitel gibt eine Übersicht über die wichtigsten Merkmale dieser Sprache.

2.2.2.1 Die „Java Virtual Maschine“ (VM)

In den meisten Programmiersprachen wird ein Programm vor dem Gebrauch kompiliert, das heißt, es wird aus dem Quellcode ein Maschinencode erzeugt, der nur auf dem entsprechenden System lauffähig ist.
Ein Java-Compiler erzeugt den Maschinencode für eine „virtuelle Maschine“, dieser wird Bytecode genannt. Um ein Java-Programm ausführen zu können, muß ein Interpreter während der Laufzeit den Bytecode in den entsprechenden Maschinencode umwandeln und ausführen. Deshalb nennt man diesen Interpreter auch die „virtuelle Maschine“. Anstatt viele Versionen für die unterschiedlichsten Systeme zu kompilieren, hat man immer dieselbe

[1] HTTP://www.php.org

„virtuelle Maschine“. Das macht Java zu einer kompilierten und gleichzeitig interpretierten Sprache die plattformunabhängig ist. Durch die Interpretation vom Bytecode während der Ausführung leidet natürlich die Geschwindigkeit des Programms deutlich. Um diesem Problem entgegenzuwirken existieren die sog. „Just-in-Time Compiler“ (JIT). Hierbei wird der Bytecode *vor* der Ausführung komplett in den Maschinencode des Systems übersetzt und ohne Interpretation ausgeführt. [04]

2.2.2.2 Java Servlets / Java Server Pages (JSP)

Ein Servlet ist ein Java-Programm und funktioniert ähnlich wie die im Kapitel 2.2.1 beschriebenen Scriptsprachen auf dem Server. Der Vorteil von Servlets ist, dass sie die Anfragen des Clients speichern und Datenbankanbindungen für die Dauer der gesamten Sitzung offen halten können. Servlets können auch untereinander die Anfrage-Daten weitergeben (engl. „Servlet chaining“). Die Abhängigkeit von Sessions oder Cookies entfällt hier. [06]
Die Java Server Pages (**JSP**) sind eine Weiterentwicklung der Servlets, sie werden wie eine serverseitige Scriptsprache programmiert und vom Web-Server als Servlet ausgeführt. Der Fokus liegt hier in der einfachen Integration von HTML bzw. XML.

2.2.2.3 Enterprise JavaBeans (EJB)

JavaBeans ist der Name für ein Softwarekomponentenmodell in Java. Ein Bean ist folgendermaßen spezifiziert: “Ein Java-Bean ist eine wieder-verwendbare Softwarekomponente, die mit Hilfe eines visuellen Generierungs-Tools manipuliert werden kann”.
Um mit JavaBeans Softwarekomponenten generieren und benutzen zu können, gibt es verschiedene Klassen und Interfaces:

- zur Erstellung der Beans
- zur Definition und zum Auslesen der Methoden und Events

- zum Verwenden der Beans in Applikationen

Enterprise JavaBeans ist die Beschreibung für die serverseitige Version der JavaBeans. Mit Hilfe von Enterprise JavaBeans kann man Komponenten generieren, die auf dem Server ausgeführt werden. Die Besonderheit von EJB ist, dass diese meist kein User-Interface besitzen, da sie nur zur Datenverarbeitung auf dem Server benutzt werden. Enterprise JavaBeans sind kompatibel zum CORBA Standard und verwenden hierzu die „Java Transaction Services“ (JTS), eine Java-Version der „CORBA Objekt Transaction Services“ (OTS). Zur Kommunikation wird die CORBA IIOP verwendet. Mehr zur CORBA Technik erfahren sie in den nächsten Abschnitten. [06]

2.2.3 Die „Common Object Request Broker Architecture” (CORBA)

CORBA steht für „Common Object Request Broker Architecture” und wurde von der Object Management Group (OMG) entwickelt. Es ist ein System, welches einzelne Objekte über ein Netzwerk zur Verfügung stellt. Das Besondere daran ist, dass es die Kommunikation von Applikationen ermöglichen soll, egal wo sie im Netz liegen oder womit sie entwickelt worden sind. Um dies zu erreichen, muß CORBA eine vollständige Verteilungstransparenz bieten, die sich folgendermaßen erklären läßt:

- Der Client muß nicht wissen, welche Hard- und Software die Gegenstelle benutzt.
- Ein Client benötigt keine Informationen über den physikalischen Ort oder die interne Repräsentation des benutzten Server-Objektes.
- Die Art der Kommunikation und die verwendeten Netzwerkprotokolle sind für den Client transparent.
- Clients sind vom Betriebssystem, dem Maschinentyp und der Implementierungssprache der Objekte unabhängig.

Um die Verteilungstransparenz zu erreichen, werden standardisierte Techniken und Protokolle verwendet, die im Folgenden erklärt werden. [04]

2.2.3.1 Der „Object Request Broker" (ORB)

Der „Object Request Broker" ist die zentrale Komponente bei CORBA. Der ORB sucht das vom Client aufgerufene Objekt, leitet den Aufruf und die übergebenen Werte weiter und gibt das Ergebnis an den Client zurück. Dazu benötigt der Client nur noch die Objektspezifikation, um die Ein- und Ausgabe korrekt auszuführen. Zusätzlich verwaltet der ORB ein Verzeichnis aller Server-Objekte, die verwendet werden können (das sog. „Repository").
Der Client und der Server sind nicht direkt mit dem ORB verbunden, sie werden jeweils auf beiden Seiten durch eine Proxy-Schicht getrennt, welche die Ein- und Ausgabeschnittstellen implementiert.
Der Clientproxy wandelt den Aufruf eines Objektes vor der Weiterleitung in ein plattformunabhängiges Format um und wartet auf die Rückgabewerte, die dann wieder in das systemspezifische Format umgewandelt werden. Diese Clientproxies werden auch als **Stubs** bezeichnet.
Auf der Seite des Server-Objekts übernimmt ein Serverproxy die Umwandlung der Ein- und Ausgabewerte in das serverspezifische Format. Die Serverproxies werden **Skeletons** genannt.

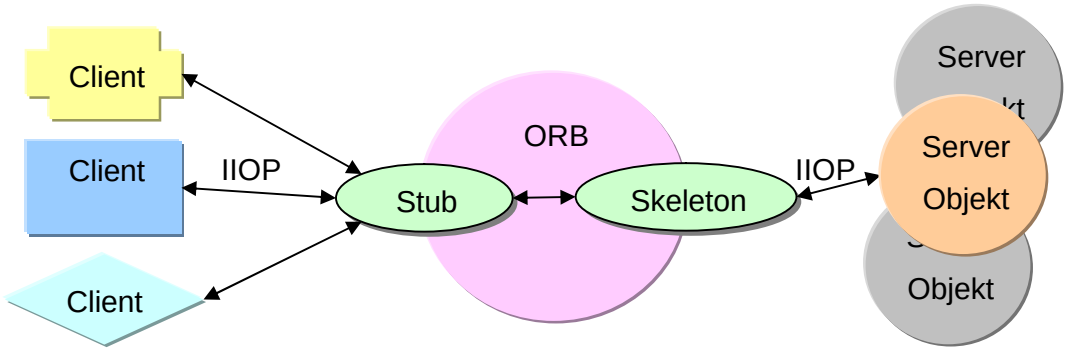

Abb. 2-2 Der Object Request Broker (ORB)

Kommuniziert wird über das standardisierte Inter-ORB Protokoll (**IIOP**), um die Skalierbarkeit und Interoperabilität mit anderen CORBA-Implementierungen zu gewährleisten. [06]

2.2.4 „Java 2 Platform, Enterprise Edition Version 1.3“ (J2EE)

Die Enterprise Edition von Java 2 oder kurz J2EE wurde von Sun entwickelt und steht für eine auf JAVA 2 basierende Standard-Architektur zur Definition und Unterstützung von mehrschichtigen Programmmodellen. Die Stärken von J2EE liegen in dem Zusammenspiel von kleinen Client-Applikationen (sog. „Thin-Client applications”) mit der Applikationslogik auf dem Server (z.B. über CORBA). Durch die Verlagerung der Applikationslogik benötigen die „Thin-Client applications” kein leistungsfähiges System bzw. Netz. [07]

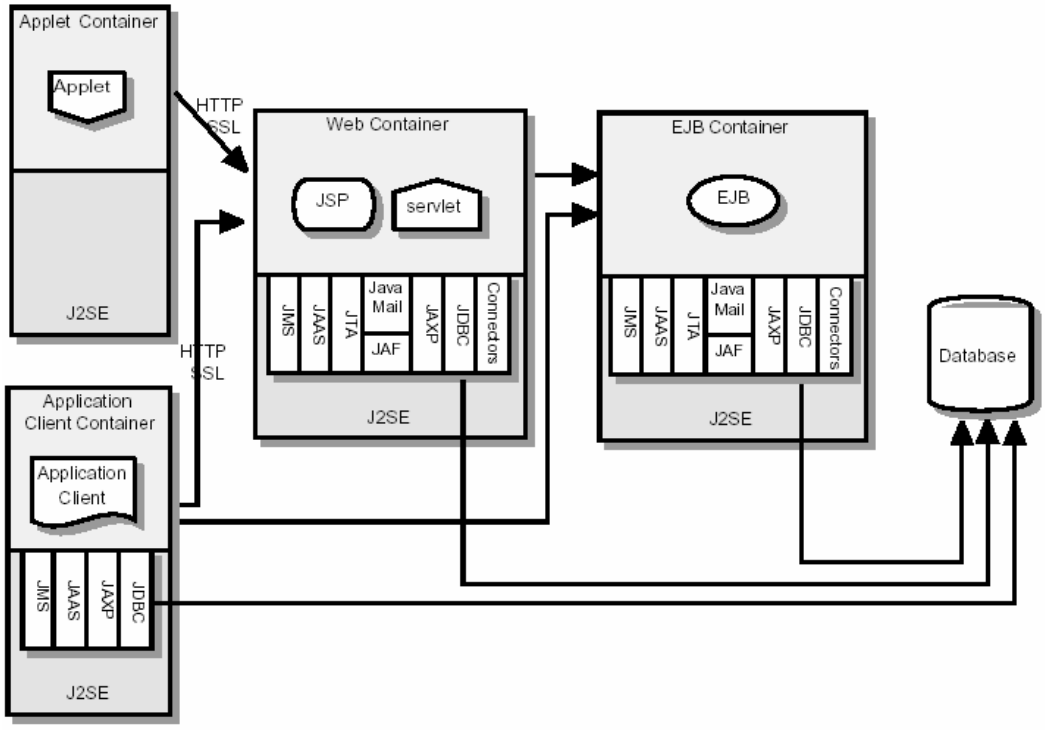

Abb. 2-3 J2EE Architektur [13]

Die „Enterprise Edition“ faßt unter anderem die besprochene Java-Technik in einem Standard zusammen und ist die Grundlage der meisten Applikationsserver. Die wichtigsten Module lassen sich wie folgt beschreiben:

- **JMS** – „Java Message Service“
  Regelt den Nachrichtentransport (z.B. für „Message-driven Beans“).
- **JAAS** – „Java Authentication and Authorization Service“
  Der Authentifizierungs- und Sicherheitsservice von Java.
- **JDBC** – „Java Database Connectivity”
  Für Datenbankverbindungen (siehe Kapitel 2.1.2).
- **JCA** – „Java Connector Architecture”
  Zur Verbindung mit Back-End Systemen, wie z.B. SAP und Peoplesoft.
- **JTA** – „Java Transaction API“
  Regelt Transaktionen, z.B. zwischen Applets und zugehörigen Containern.
- **Java Mail** – „Java Mail Service“
  Regelt den Mailverkehr.
- **JAF** – „Java Activation Framework“
  Wird für den Java Mail Service benötigt.
- **JAXP** – „Java API for XML Parsing”
  Unterstützt das Parsen von XML Dateien nach den Standards SAX, DOM und XSLT.
- **Connectors** – „J2EE Connector Architecture”
  Für die Verbindung zu anderen J2EE kompatiblen Servern, mit Server-Pooling, Transaction Management und Security.

[13]

2.2.5 “Simple Open Access Protocol” (SOAP)

SOAP ist ein Standard für Kommunikation und Datenaustausch. Er ermöglicht den transparenten Datenaustausch von Applikationen über XML für die Datenformatierung und HTTP als Übertragungsprotokoll.

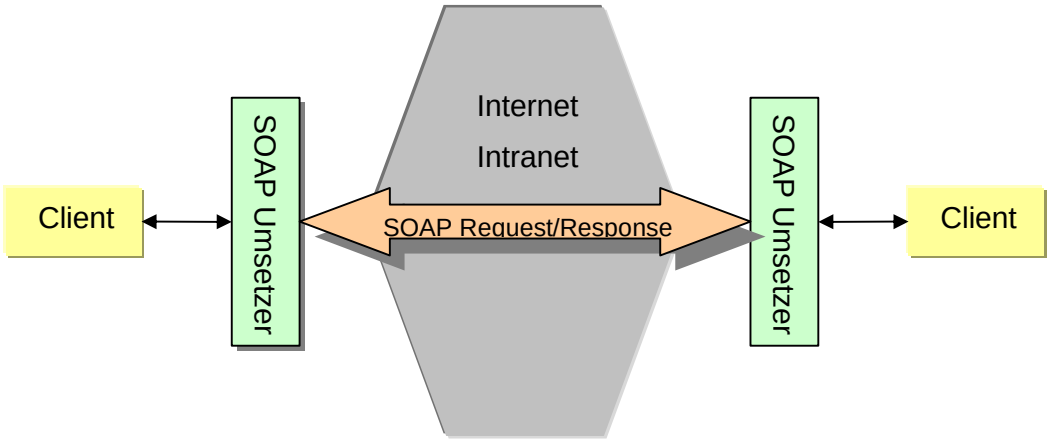

Abb. 2-4 Simple Open Access Protokoll [16]

Damit können Applikationen, unabhängig von der verwendeten Hard- und Software, über schon vorhandene HTTP-Netze miteinander kommunizieren. [16]

2.2.6 „Web Services Description Language“ (WSDL)

WDSL ist ein XML-Format zur Beschreibung von Netzwerkdiensten für den Austausch von Nachrichten, die dokumentenorientierte bzw. prozess-orientierte Daten übertragen. Dazu werden die Operationen und Nachrichten auf Sender- und Empfängerseite abstrakt definiert und an ein existierendes Protokoll gebunden (z.B. SOAP, HTTP GET/POST). So ist die Daten-übertragung frei definierbar. [21]

2.3 Der Client

Der Endnutzer verwendet einen Client, um eine Applikation über ein Netzwerk zu nutzen. Bei heutigen Inter- bzw. Intranet-Anwendungen ist dies meist ein WWW-Browser der die HTML-Seiten anzeigt, die der Applikations-server generiert hat. Es können aber auch Applikationen in anderen Sprachen genutzt werden, wie z.B. Java oder C++.

2.3.1 „Extensible Markup Language“ (XML)

XML ist eine Methode, um strukturierte Daten in einer Textdatei zu speichern. Die Regeln für die Semantik von XML basieren auf dem schon 1986 standardisierten SGML.
XML verwendet Tags zur Definition von Daten, wie auch in HTML üblich. Nur können diese im Gegensatz zu HTML frei definiert werden. XML Dateien lassen sind leicht generieren bzw. verarbeiten und sind plattformunabhängig. Durch zusätzliche Stylesheets (CSS) und die „Extensible Style Language“ (XSL) sind XML-Dateien auf einem (XML-fähigen) Browser formatiert darstellbar. [03]

2.3.2 Hyper Text Markup Language (HTML)

HTML ist die universelle Sprache zur Darstellung von Text und Grafik im Internet. Der HTML-Standard wird vom W3-Consortium[1] gepflegt und ständig erweitert.
Über diverse Plug-Ins ist es möglich, Videos, Musik oder auch Java-Applikationen zu nutzen. So genannte Formulare ermöglichen die Verarbeitung von Benutzereingaben. Die Interaktion mit dem Benutzer ist jedoch sehr beschränkt, da der Datenfluß der Nutzdaten nur in eine Richtung geht. Zur Datenverarbeitung braucht man zusätzlich eine Scriptsprache.
HTML-Dokumente werden auf dem sog. Browser betrachtet, die Standard-anwendung zum Surfen im Internet. Die beiden erfolgreichsten Anbieter von Browsern sind Microsoft mit dem Internet Explorer und AOL mit dem Netscape Communicator. Beide Hersteller haben ihre eigene Interpretation von HTML implementiert. Es ist schwierig, komplizierte HTML-Seiten kompatibel für alle Browser zu machen. [03]

[1] http://www.w3.org

2.3.3 Javascript / Visual Basic Script

Javascript bzw. Visual Basic Script (kurz „VBScript“) sind einfache Scriptsprachen zur Verarbeitung von Daten in HTML-Dokumenten. Sie werden auf dem Client (engl. „Client-Sided“) ausgeführt, wo sie direkte Einflußnahme auf die Darstellung und Datenverarbeitung haben (z.B. für Schalter, Popups etc.). Neben Javascript gibt es noch eine Variante von Microsoft, die auf Visual Basic basiert. Der Funktionsumfang beider Sprachen ist annähernd gleich.

2.3.4 Client Applikationen in Java

Es gibt zwei Möglichkeiten, einen in Java geschriebenen Client zu nutzen.
Zum einen als Java-Applet, welches im Browser ausgeführt wird, zum anderen als Java-Applikation. Man braucht aber auf jeden Fall eine Java Virtual Maschine (siehe Kapitel 2.2.2.1). Die meisten Browser sind javafähig, sie haben bereits die Java-VM integriert. Ein Java-Applet läuft, im Gegensatz zur Applikation, im sog. „Sandbox Modus“. Dies wiederum bedeutet zum Beispiel, dass es keine unsicheren bzw. systemnahen Aktionen ausführen darf. Diese Aktionen werden, sofern überhaupt möglich, durch die Virtuelle Maschine verhindert. So kann man z.B. nicht ohne weiteres eine JDBC-Verbindung (siehe Kapitel 2.1.2) zu anderen Rechnern aufnehmen. Um mit einem Java-Client eine verteilte Applikation zu benutzen, bietet sich z.B. die Verwendung von CORBA (siehe Kapitel 2.2.3) an, da die Kommunikation über die CORBA Schnittstelle erlaubt ist.

2.3.5 Client Applikationen in anderen Sprachen

Wenn man Applikationen in anderen Programmiersprachen entwickeln möchte, muß man darauf achten, dass die entsprechenden Standards (z.B. CORBA) zur Kommunikation mit dem Applikationsserver unterstützt werden.

Kapitel 3 Applikationsserver – Klassifikation

Die Anwendungsfelder für Applikationsserver wachsen ständig, und so gibt es immer mehr Anbieter von Applikationsserver-Lösungen. Die meisten Systeme basieren auf der J2EE (siehe Kapitel 2.2.4). Es gibt aber auch Systeme, die nicht auf Java basieren und keine bzw. nur in einigen Schnittstellen eine Java-Unterstützung bieten. In diesem Kapitel werden einige der großen Hersteller und ihre Produkte vorgestellt, um einige technische und konzeptionelle Unterschiede aufzuzeigen.

3.1 Applikationsserver ohne Java-Kern

Der größte Anbieter von Applikationsserver-Software die nicht auf Java basiert, ist **Microsoft**. Es gibt aber auch serverseitige Scriptsprachen wie **PHP,** oder **CGI** kompatible Sprachen wie **Perl**, die ähnlich einem Applikationsserver arbeiten. Viele Firmen haben neue Applikationsserver-Lösungen entwickelt, die verschiedene Techniken kombinieren und dadurch ein breites Anwendungsspektrum bieten. Als Beispiel ist hier der **Cold Fusion** Server von Marcomedia beschrieben, der zusätzlich zur Verwendung multimedialer Inhalte mit Flash und Shockwave noch viele Server-funktionalitäten mitbringt.

3.1.1 Microsoft IIS – ASP – COM – DCOM

Mit dem „Internet Information Server“ (**IIS**) bietet Microsoft einen Web-Server mit der integrierten, serverseitigen Scriptsprache **ASP** (siehe Kapitel 2.2.1).
Mit dem „Component Object Model“ (**COM**) bzw. der Erweiterung „Distributed Component Object Model“ (**DCOM)** hat Microsoft eine CORBA ähnliche Architektur entwickelt (siehe Kapitel 2.2.3).

Das „Component Object Model“ wurde in der gesamten Familie der Microsoft Betriebssysteme unter anderem als Modell für die komponentenorientierte Programmierung entwickelt.

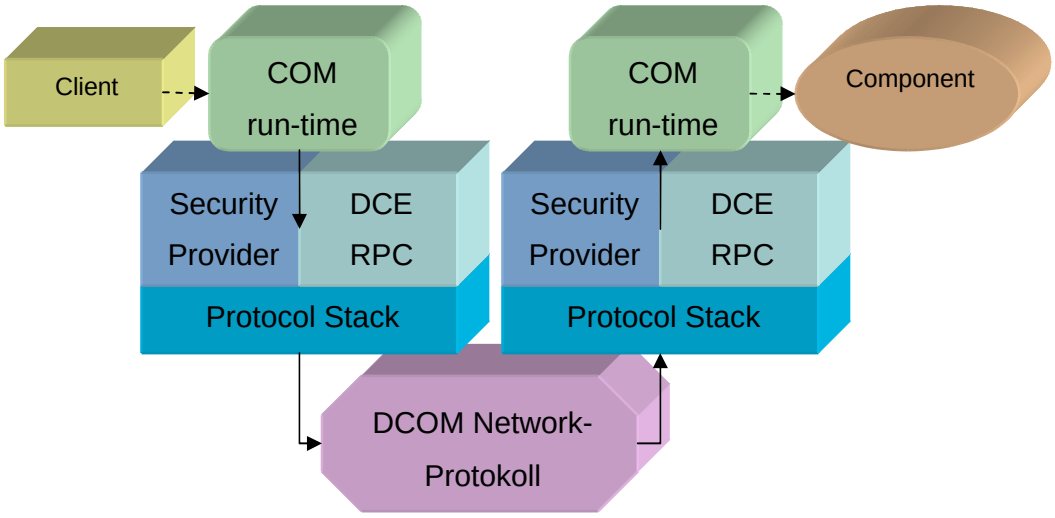

Abb. 3-1 Microsoft COM / DCOM [11]

Das DCOM ist eine Erweiterung des COM und beinhaltet die Mechanismen für einen entfernten Prozeduraufruf (engl. „Remote Procedure Call“). Im Prinzip funktioniert es ähnlich wie CORBA, deshalb hier nur eine kurze Übersicht.
Der Client ruft über die COM-Engine eine Komponente auf einem entfernten Server auf. Die Kommunikation erfolgt über das "Remote Procedure Call" Protokoll (RPC), welches an das "Distributed Computing Environment“ (DCE) angelehnt ist, ein Standard der "Open Software Foundation" (OSF). DCOM nutzt die vorhandenen Netzwerkprotokolle wie z.B. TCP/IP oder HTTP für die Kommunikation mit dem entfernten Server. Ein Client kann zum Beispiel per ActiveX-Schnittstelle auf das DCOM zugreifen. [11]

3.1.2 Microsoft .NET

Die .NET Strategie von Microsoft beschreibt nicht nur eine Programmierschnittstelle für die Entwicklung verteilter Systeme, sondern

eine neue Betriebssystem-, Unternehmens- und Marketingstrategie. Microsoft stellt mit .NET eine hardware- und betriebssystemunabhängige Lösung für die Entwicklung verteilter Systeme vor. Die .NET Komponenten nutzen offene Standards wie XML und SOAP zur Datenhaltung und Kommunikation. Für die Entwicklung der Komponenten kommen alle Sprachen zum Einsatz, die sich an das eigens dafür aufgestellte Regelwerk „Common Language Specification“ (CLS) halten. Kompatibel sind unter anderen die neuen Versionen von Visual Basic und C++, das „Visual Basic.NET“ und C# (sprich engl. „C-Sharp“).

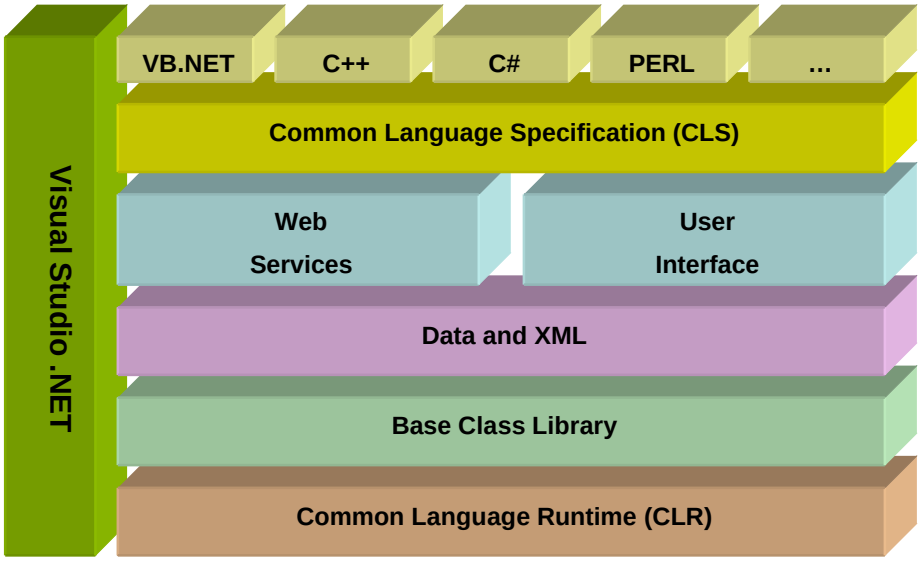

Abb. 3-2 Microsoft .NET [05]

Alle Programme werden zuerst in die Metasprache „Microsoft Intermediate Language“ (MSIL) übersetzt, um betriebssystemunabhängig zu bleiben. Genau wie bei der Java Virtual Maschine entsteht ein Bytecode, der bei Microsoft „Portable Executable“ (PE) genannt wird und von der „Common Language Runtime“ (CLR) ausgeführt wird (vergleiche Kapitel 2.2.2.1). Da die CLS eine Verwendung von Sun-Bibliotheken verbietet, ist die Benutzung von JavaBeans und ähnlichen Konzepten nicht ohne weiteres möglich (siehe Kapitel 2.2.2.3). [11]

3.1.3 Serverseitige Scripte als Applikationsserver-Ersatz

Ein Applikationsserver beschreibt im Allgemeinen ein verteiltes System mit mindestens einer ausführenden Einheit. Durch serverseitige Scriptsprachen ergibt sich die Möglichkeit, auf Datenbanken oder das Dateisystem zuzugreifen und dadurch Funktionalität zur Verfügung zu stellen. Der Applikationsserver ist hier der Interpreter der Sprache.
Durch die Verwendung von mehr als einem Web-Server für ein Projekt ist eine begrenzte Skalierung möglich.
Die meistgenutzte Scriptsprache im Internet ist **PHP**. Die Scriptsprache PHP ist ein Open-Source Projekt der PHP Group[1]. Sie hat einen mächtigen Funktionsumfang und ist für viele Serversysteme erhältlich. Neben der Integration von ODBC bietet es unter anderem die native Unterstützung der größten Datenbank-Systeme, persistente Datenbankverbindungen (wie Java-Servlets) und einen XML-Parser.

3.1.4 Das „Common Gateway Interface“ (CGI)

Es gibt noch eine Gruppe von Sprachen, die für die Internet-Programmierung verwendet werden können, obwohl sie nicht direkt in HTML eingebettet werden. Den Datentransfer mit dem Web-Client regelt hier das „Common Gateway Interface“ (**CGI**). Dazu leitet das CGI z.B. Formulardaten in die Standardeingabe, bzw. die Standardausgabe eines Programms zum Web-Client weiter.

[1] HTTP://www.php.org

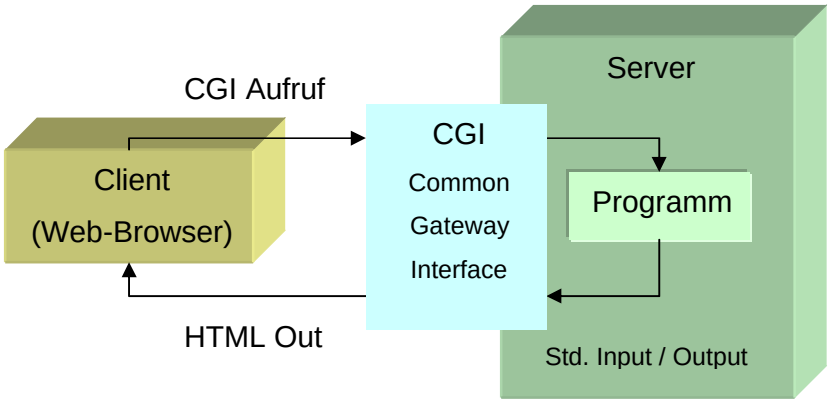

Abb. 3-3 CGI Aufruf [20]

Alle zum CGI kompatiblen Programmiersprachen können verwendet werden, um Web-Seiten zu programmieren. Zu der gebräuchlichsten Kombination gehören hier **Perl** und **CGI**.
Damit kann man den vollen Funktionsumfang einer Sprache nutzen, auch wenn sie nicht direkt für die Verwendung über den Web-Browser konzipiert ist. [20]

3.1.5 Macromedia Cold Fusion 5

Die Applikationsserver-Lösung von Macromedia basiert auf der „Cold Fusion Markup Language“ (CFML) und läßt sich mit vielen Plug-Ins erweitern. Die CFML läßt sich wie eine serverseitige Scriptsprache in HTML oder ähnliche Dokumente integrieren. Die Stärken von Cold Fusion liegen in der Chart- und Grafikverarbeitung und in der Integration von Shockwave und Flash. Der Server beherrscht unter anderem das Clustering, Security-Management und Datenbankverbindungen. Durch die effiziente Verwendung von Caches und JIT-Compilern soll der Server recht schnell sein.

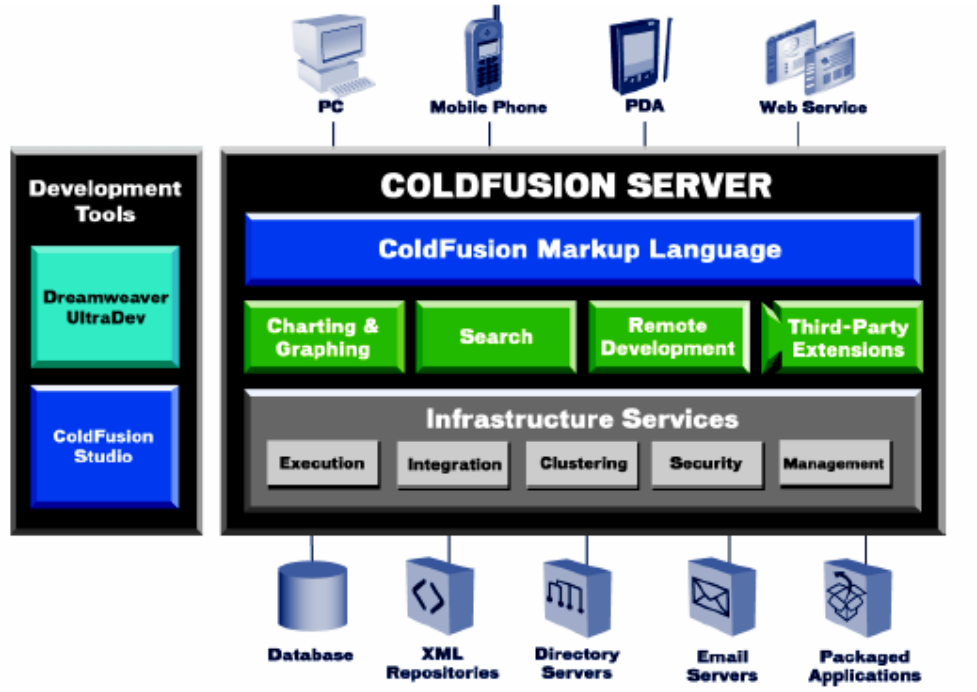

Abb. 3-4 Cold Fusion Server [12]

Es ist auch möglich COM, CORBA, EJB und andere Komponentendienste zu verwenden. Durch JRun, ein anderes Produkt von Macromedia, wird der Cold Fusion Server um die J2EE-Funktionalität erweitert (siehe Kapitel 2.2.4). [12]

Es handelt sich bei Cold Fusion also eher um ein Mischprodukt, welches auch Java-Funktionalität bietet.

3.2 Applikationsserver basierend auf J2EE

Viele große Applikationsserver-Lösungen basieren auf dem J2EE-Standard von Sun (siehe Kapitel 2.2.4). Um diesen Standard werden eine Vielzahl von einzelnen Programmen oder Servern entwickelt, je nachdem wo die Schwerpunkte der einzelnen Hersteller liegen.
Die großen Datenbankanbieter **Oracle**, **IBM** und **Sybase** haben ihre Applikationsserver mit ihrer Datenbank-Technik kombiniert. IBM bietet sogar

komplette Hardware-Lösungen an. **BEA** bietet mit „WebLogic Enterprise“ eine auf E-Business spezialisierte Applikationsserver-Lösung an und ist damit ein erfolgreicher Anbieter. Auch die Open-Source Gemeinde entwickelt Applikationsserver, wie z.B. **JBoss**. Der Applikationsserver von Apple nennt sich **WebObjects** und ist auch für andere Plattformen erhältlich.

Dies ist nur eine kleine Auswahl von Anbietern. Für welchen man sich letztendlich entscheidet, hängt von der spezifischen Ausgangssituation und der gewünschten Infrastruktur ab.
In den folgenden Abschnitten werden einige der Applikationsserver vorgestellt, um eventuelle Unterschiede der Architekturen kennenzulernen.

3.2.1 Oracle 9iAS

Oracle beschreibt seinen Applikationsserver als das umfangreichste Software-Paket am Markt. Es arbeitet vollständig mit standardisierten Methoden und Protokollen. Nach Meinung des Herstellers handelt es sich um die schnellste und speicherschonenste J2EE-Umgebung. Der 9iAS unterstützt die integrierte Verarbeitung von Web-Seiten, Web-Services und J2EE 1.3 Services.
Die J2EE-Umgebung unterstützt die Verwendung von Servlets 2.3, JSP 1.2 und EJB 2.0 (siehe Kapitel 2.2.4). Es gibt außerdem einige Zusatzmodule, wie z.B. die „Business Components for Java“ (BC4J). Als Kommunikationsschnittstellen stehen unter anderem SOAP, XML, RMI, RMI-IIOP (z.B. CORBA) und Wireless-Protokolle zur Verfügung. Das zugehörige Entwicklungswerkzeug nennt sich JDeveloper. Zur visuellen Modellierung von Objekten dient die Sprache UML (Unified Modeling Language), unterstützt durch das „Oracle MVC Framework“ (MVC).

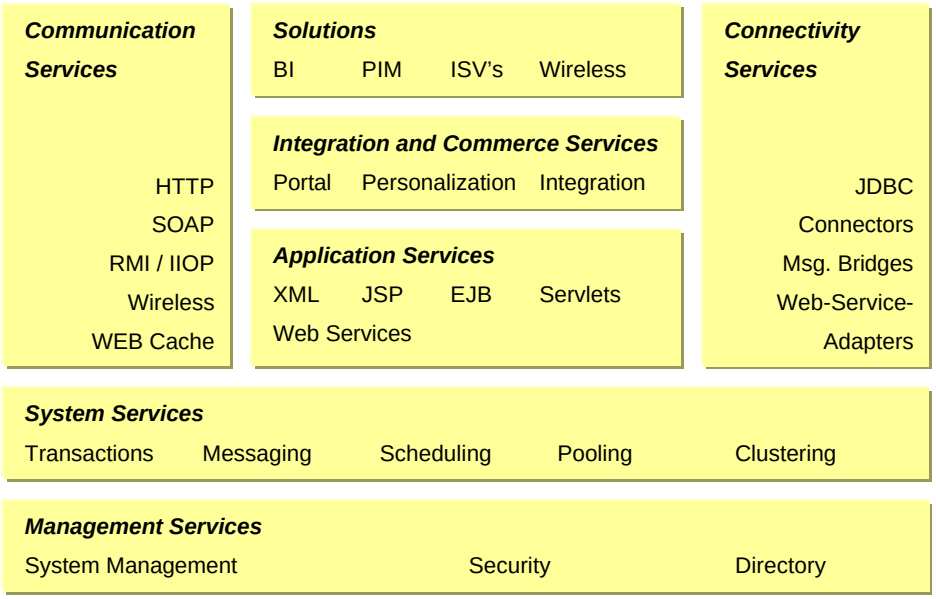

Abb. 3-5 Oracle 9iAS konzeptionelle Architektur [15]

Der 9iAS hat einen eigenen Web-Server integriert, der auf dem Apache-Server basiert. Mit zusätzlichen Modulen wird dieser Web-Server in der Funktionalität erweitert. Es existieren Module für die Kommunikation zu J2EE-Komponenten (hier Container), für eine einheitliche Authentifizierung des Benutzers über alle 9iAS Komponenten sowie Module für die Kommunikation mit Datenbanken, Stored-Procedures und Perl-Programmen. Eine eigene CGI und FastCGI Schnittstelle für C, C++ und Java (siehe Kapitel 3.1.4) wird auch mitgeliefert. Der 9iAS hat zusätzlich einen Perl-Interpreter integriert. Diese Funktionalitäten stehen durch verschiedene Web-Listeners auch für andere Web-Server zur Verfügung. Oracle unterstützt das „Web-based Distributed Authoring and Versioning“ (WebDAV), eine Erweiterung des HTTP 1.1 Protokolls für verteiltes Arbeiten.

Durch die „9iAS Forms Services“ lassen sich Java-Datenbank-Formulare erstellen und nutzen.

Dies ist eine kurze Zusammenfassung der Funktionen dieser Software. Die kompletten „Technical Whitepapers“ vom 9iAS umfassen etwa 70 Seiten, gefüllt mit einer Vielzahl von unterstützen Standards und Möglichkeiten zu deren Nutzung. [15]

Zusammenfassend

- Volle J2EE V1.3 Unterstützung
- Unterstützt viele Sprachen (z.B. Java, Perl, C, Cobol und PL/SQL)
- Unterstützt eine Vielzahl von Standards (J2EE1.3, Web Services, RosettaNet 1.1/2.0, ebXML, WebDAV, LDAP v3, SSL v3 und einige XML-Standards)
- Eher für große Unternehmen ausgelegt

3.2.2 BEA – WebLogic

BEA bewirbt WebLogic als flexible, komponentenbasierte E-Business Lösung. Zusätzlich haben sie verschiedene Komponenten und Erweiterungen ihres WebLogic-Servers im Angebot. Hier eine kleine Auswahl der wichtigsten Komponenten:

- BEA WebLogic-Server
- BEA WebLogic-Enterprise
- BEA Tuxedo
- BEA eLink

Der **WebLogic-Server** stellt die Basis des Systems dar. Er beinhaltet die Java Applikationslogik (engl. „Business Logic“) mit EJB Unterstützung und die Präsentationslogik mit JSP, Servlets und HTML/XML.

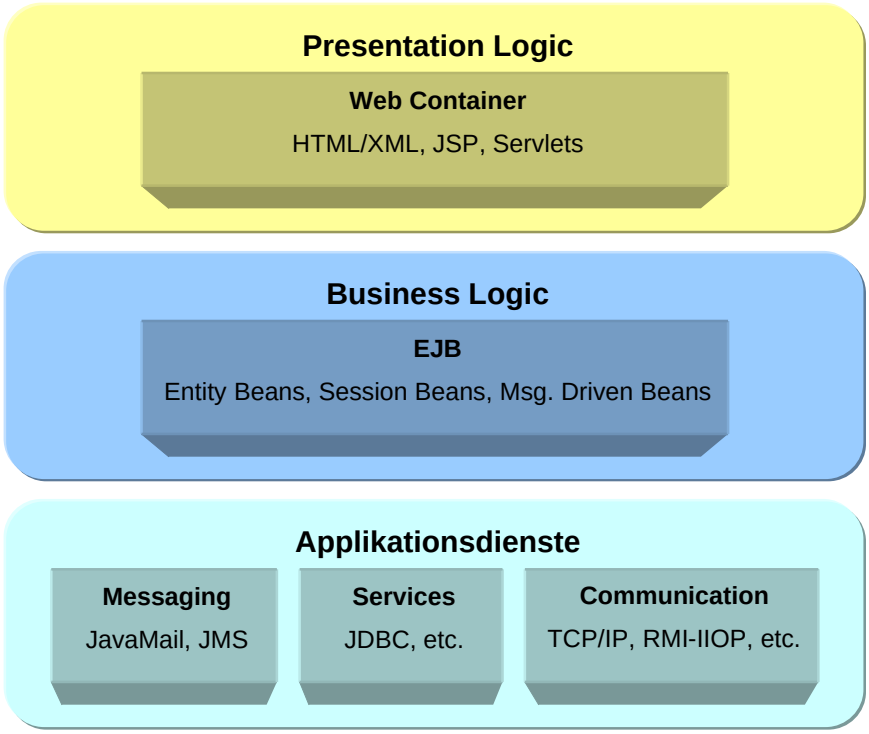

Abb. 3-6 BEA WebLogic Server-Architektur [14]

BEA Tuxedo ist ein Server, der im wesentlichen für schnelle Transaktionen, Routing, Load-Balancing und Security in einer verteilten Datenbank-Anwendung auf der Basis von CORBA oder anderen Kommunikations-standards zuständig ist.

Der **BEA WebLogic Enterprise-Server** beschreibt ein Komplettpaket aus der Kombination der hier beschriebenen Produkte und anderen Zusatz-produkten.

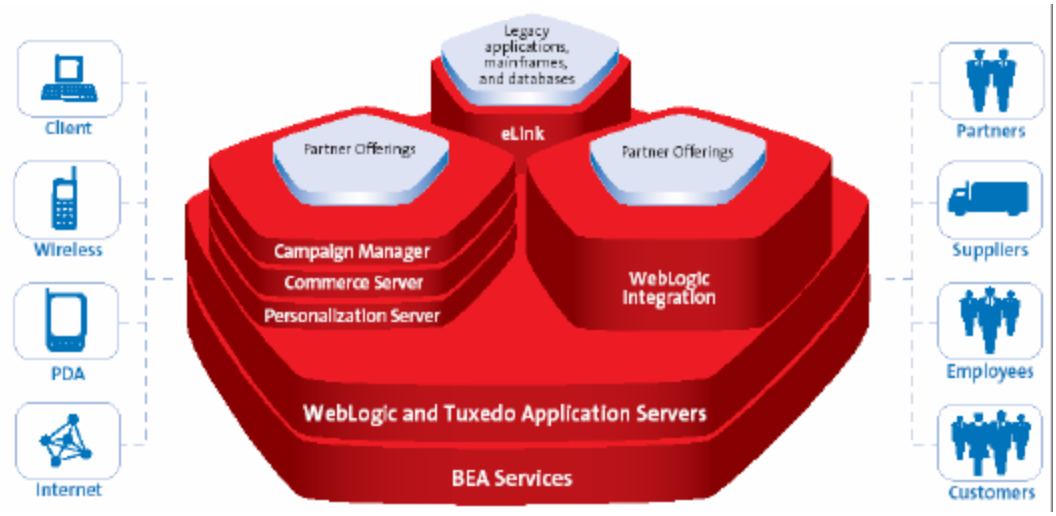

Abb. 3-7 BEA WebLogic Enterprise [14]

BEA eLink ist für die Kommunikation zwischen den einzelnen Servern und zu externen Mainframe-Anwendungen bzw. Datenbanksystemen zuständig. [14]

Zusammenfassend

- Sehr gute Java Unterstützung (J2EE 1.3)
- Mit Tuxedo sehr gutes Transaktionsmanagement
- Unterstützt CORBA, COM und Web-Services
  (COM nur über Java-Wrapper)
- Unterstützt „Enterprise Resource Planning“ (ERP)
- Schnelle EJB-Unterstützung (mit Load Balancing und Failover)
- Keine eigenen Entwicklungstools
  (Orientiert sich an Symantecs „Visual Cafe Enterprise“)
- Sehr teuer

3.2.3 IBM WebSphere Version 4.0

Der Applikationsserver von IBM basiert auf einer Servlet-Engine, ist voll kompatibel zur Version 1.2.1 des J2EE Standards und hat den Support für einige Features der Version 1.3 implementiert (siehe Kapitel 2.2.4). So

unterstützt er in der Advanced-Version alle J2EE-Dienste wie Servlets, EJB, JSP und die Standard-Schnittstellen wie z.B. JMS und CORBA (siehe Kapitel 2.2.4).

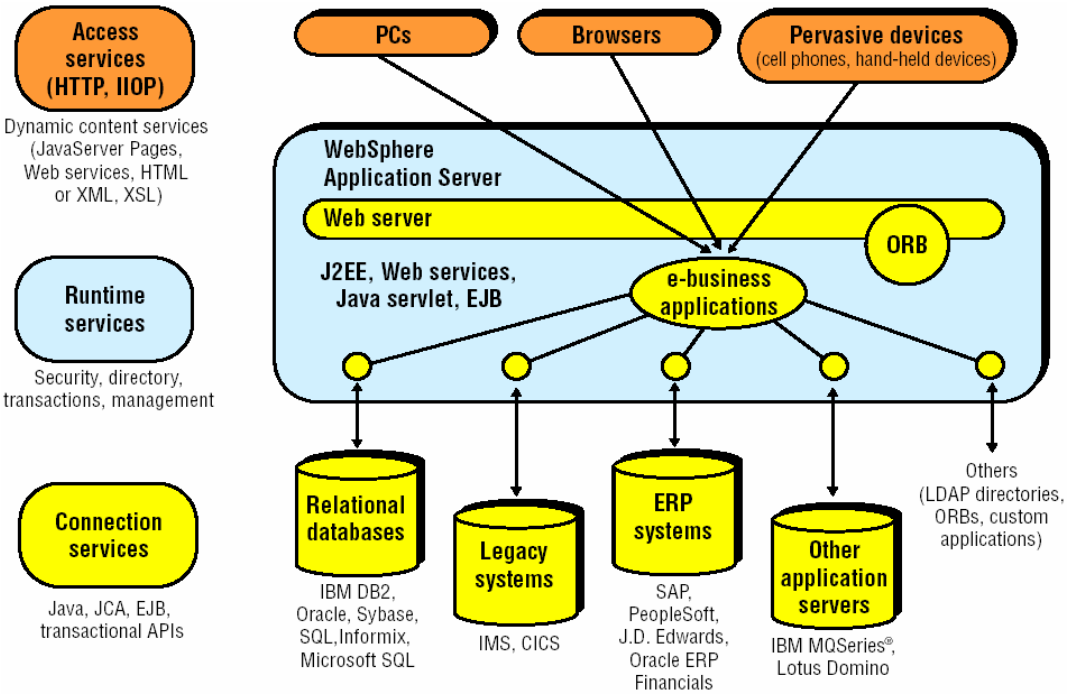

Abb. 3-8 IBM WebSphere Architektur [17]

WebSphere hat in der Advanced-Version einen Apache-Webserver integriert und unterstützt auch alle Web-Services wie z.B. das „Simple Object Access Protocol“ (SOAP), „Universal Description, Discovery and Integration“ (UDDI) und die „Web Services Description Language“ (WSDL). Weitere unterstützte Standards kann man aus der Abbildung entnehmen. WebSphere läuft neben den NT- und Unix-Versionen auch auf einigen Mainframes.[17]

Zusammenfassend

- J2EE V1.2.1 mit einigen Features der Version 1.3 (z.B. JCA, JMS)
- Basiert auf der Servlet-Engine
- Volle Unterstützung von Web-Services
- Native Unterstützung vieler Datenbanken

3.2.4 Sybase Enterprise Application Server 4.1 (EAServer)

Um einen fairen Produktvergleich zu ermöglichen, wird die neueste Version des EAServers herangezogen, obwohl die praktische Aufgabe dieser Diplomarbeit mit der Version 3.5 realisiert wurde.
Die neuste Version des EAServer ist voll kompatibel zum J2EE 1.3 Standard. Sie unterstützt EJB, JSP und Servlets mit dem „Java Messaging Service“ (JMS), der „Java Connector Architecture“ (JCA) und den für die Authentifizierung notwendigen „Java Authentication and Authorization Service“ (JAAS) (siehe Kapitel 2.2.4). Mit dem Web-Service-Toolkit besitzt der EAServer die Unterstützung der SOAP- (siehe Kapitel 2.2.5), WDSL- und UDDI- Standards.

Für die Entwicklung der Komponenten und Anwendungen gibt es verschiedene Entwicklungsumgebungen:

- **PowerJ -** eine Java-Entwicklungsumgebung
- **PowerBuilder** – Ein „Rapid Application Development“ (RAD) Tool für die visuelle Erstellung von Applikationen
- **PowerSite** – Ein HTML-Editor, ähnlich wie Microsofts „Frontpage“
- **PowerDesigner** – Ein Programm zur visuellen Erstellung von relationalen Datenbankmodellen

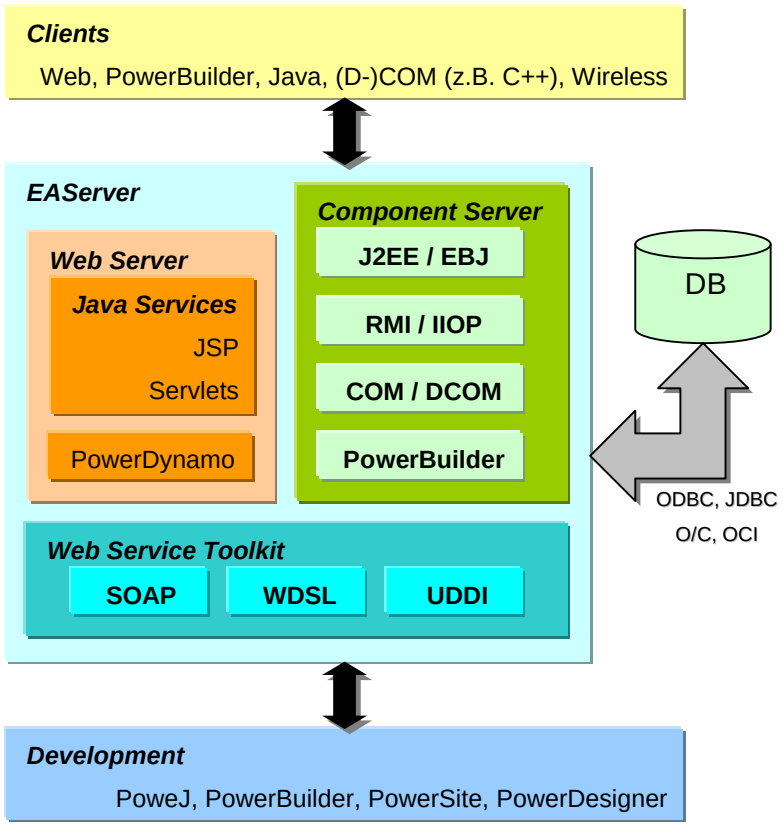

Abb. 3-9 EAServer 4.1 Architektur

Die meisten Teile des EAServers sind Einzelprodukte, die schon länger existieren und für die Sybase-Datenbanksysteme entwickelt wurden. Der Komponentenserver (Jaguar) ist der wichtigste Bestandteil des EAServers. Zusammen mit dem PowerBuilder und PowerJ ist das Sybase-Paket eine gute Applikationsserver-Lösung, insbesondere wenn schon Produkte von Sybase eingesetzt werden. [22]

Zusammenfassend

- Kompatibel zum J2EE V1.3 Standard
- Unterstützt Komponenten in vielen Sprachen (Java und DCOM)
- Eigene, serverseitige Scriptsprache PowerDynamo
  (Nur in Zusammenhang mit Sybase-Datenbank einsetzbar)
- RAD Integration (PowerBuilder)

- Keine Komplettlösung, sondern eine Sammlung von einzelnen, teilweise schon länger existierenden Sybase-Tools
  (Besonders geeignet, wenn schon Sybase eingesetzt wird)

3.3 Fazit: Wer braucht welchen Server?

Welche Applikationsserver-Lösung kann man empfehlen?
Kleine, abgeschlossene Anwendungen lassen sich mit einer serverseitigen Scriptsprache bzw. mit dem CGI einfach umsetzen. Mit der Möglichkeit auf Datenbanken zuzugreifen, die Transaktionen und Stored-Procedures unterstützen, ist man für die meisten Fälle schon gut genug ausgerüstet. Das Szenario 1 beschreibt die Einsatzgebiete von einfachen serverseitigen Scriptsprachen und CGI.

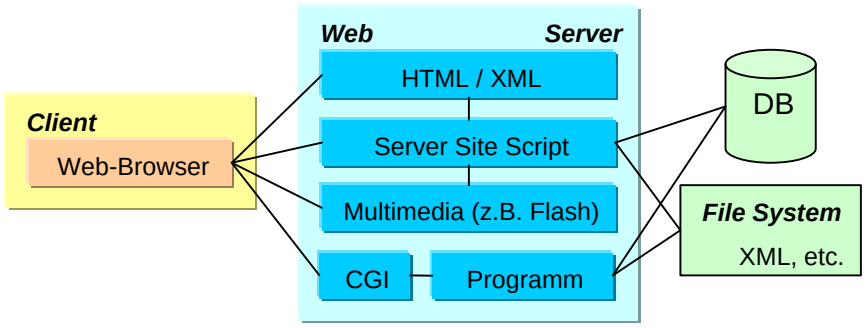

Abb. 3-10 Szenario 1, kleine Web-Anwendungen

Der Cold Fusion Server von Macromedia bietet ohne den JRun-Server eine leistungsfähige Scriptsprache, einige Multimediaerweiterungen und erweiterte Performance- bzw. Sicherheitstechnik. Zusammen mit dem JRun-Server gehört das Cold Fusion Paket zu den J2EE-Servern.
Es gibt bei den verschiedenen Java-Architekturen feine Unterschiede in der Unterstützung der Standards für die Kommunikation und Entwicklung der Komponenten. Da die meisten auf den J2EE-Standard aufbauen, bringen sie die selben Grundfunktionalitäten mit. Alle hier vorgestellten J2EE-Server können mit Produkten aus eigenem Hause oder Fremdprodukten erweitert werden, bzw. lassen sich in verschiedenen Ausbaustufen erwerben. Der

Einsatz solcher Systeme lohnt sich nur, wenn man komponentenorientiert programmiert und diese Komponenten in vielen, verschiedenen Clients nutzen will. Diese Systeme sind auch mit erweiterten Performance- und Sicherheitstechniken ausgerüstet, die bei serverseitigen Scriptsprachen nicht vorhanden sind. Das Szenario 2 zeigt ein Beispiel einer verteilten, komponentenbasierten Anwendung.

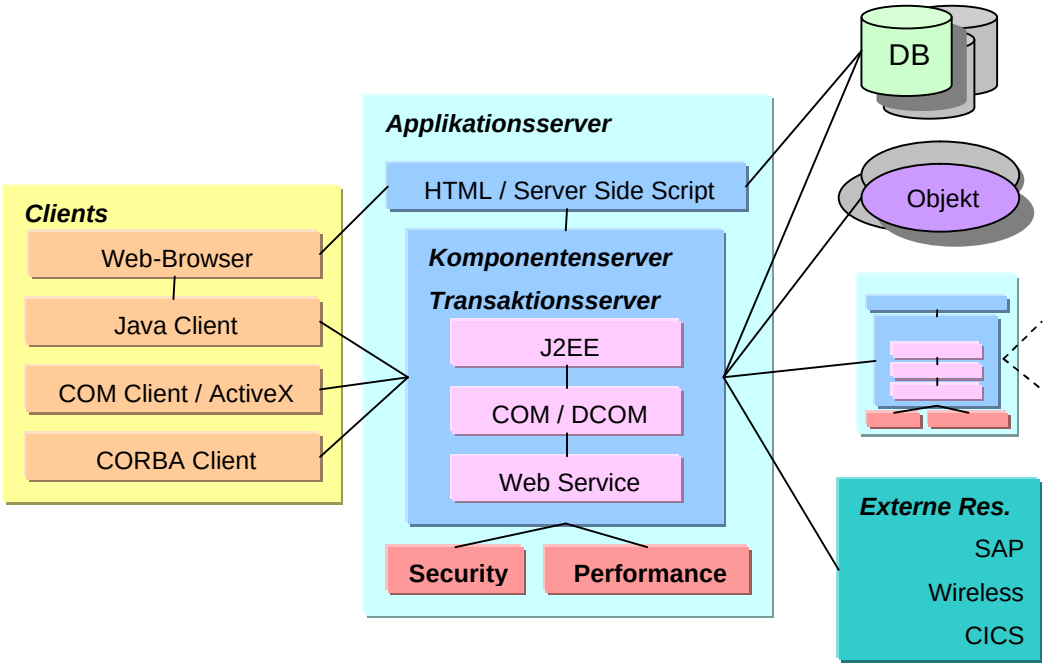

Abb. 3-11 Szenario 2, große komponentenbasierte Anwendung

Vergleicht man die J2EE-Server mit dem Microsoft .NET, so erkennt man das gleiche Prinzip, basierend auf anderen Schnittstellen (siehe Kapitel 3.1.1). Die meisten aktuellen J2EE-Server können mit COM/DCOM-Objekten umgehen. Die Unterstützung von EJB oder CORBA unter Microsoft .NET ist nicht ohne weiteres möglich. Immerhin besteht die Möglichkeit, über vorhandene Web-Services (wie SOAP) zu kommunizieren. So eignet sich .NET besonders für Nutzer, die schon Microsoft-Server einsetzen und auf die Kombination aus ActiveX und DCOM aufbauen. Wenn Java benutzt wird, sollte man einen J2EE-Server einsetzen.

Kapitel 4 Die Praxis

In diesem Kapitel geht es um die Lösung eines exemplarischen Problemfalles. An diesem praktischen Beispiel wird die Erstellung einer Client-Server-Anwendung und einer Java-Komponente veranschaulicht. Durch die Umsetzung in zwei verschiedenen Versionen werden die Unterschiede der klassischen Client-Server Programmierung zur komponentenorientierten Programmierung aufgezeigt.

4.1 Die Aufgabe: Teilnahme an einem Kurs

Für einen Anbieter von Aufbau- und Lernkursen existiert eine Datenbank zur Speicherung der Kurse, Lehrer, Teilnehmer und Material.
Die Teilnahme an einem Kurs soll über ein Java-Applet möglich sein, welches über das Internet aufgerufen wird.

4.1.1 Die Datenbank

Die Datenbank läuft auf einem Datenbank-Server vom Typ „Adaptive Server Enterprise“ (ASE) von Sybase. Die Darstellung im „Entity-Relationship-Model“ (ER-Modell) hilft, den Ablauf bei der Aufnahme einer neuen Teilnahme zu verstehen.
Das ER-Modell beschreibt die Struktur der Datenbank.

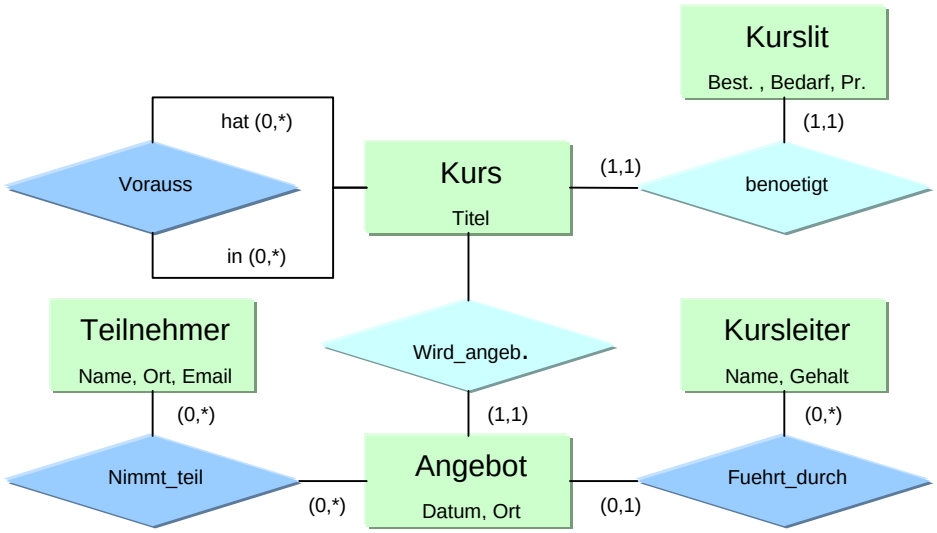

Abb. 4-1 ER-Modell der KursDB [09]

Mit Hilfe des ER-Modells wird eine Datenbank erstellt, die aus acht Tabellen besteht und wie folgt aufgebaut ist:
(PK = Primärschlüssel, FK = Fremdschlüssel)

Kurs			
KursNr (PK)	Titel		
Teilnehmer			
TnNr (PK)	Name	Ort	Email
Kursleiter			
PersNr (PK)	Name	Gehalt	
Kurslit			
KursNr (PK) (FK Kurs)	Bestand	Bedarf	Preis
Angebot			
AngNr (PK)	*KursNr (FK Kurs)*	Datum	Ort
Vorauss			
VorNr (PK) (FK Kurs[KursNr])	*KursNr (PK) (FK Kurs)*		
Nimmt_teil			
AngNr (PK) (FK Angebot)	*KursNr (PK) (FK Angebot)*	*TnNr (PK) (FK Teilnehmer)*	
Fuehrt_durch			
AngNr (PK) (FK Angebot)	*KursNr (PK) (FK Angebot)*	*PersNr (PK) (FK Kursleiter)*	

Abb. 4-2 Datenbank KursDB

Die Teilnahme an einem Kurs führt zu Lese- bzw. Schreiboperationen in insgesamt fünf der acht Tabellen. [09]

4.1.2 Datenbank-Anfragen bei einer Teilnahme

Eine neue Teilnahme setzt viele Anfragen und Änderungen in den verschiedenen Tabellen der KursDB voraus. Worauf bei Anmeldung einer neuen Teilnahme zu achten ist, zeigt diese Liste:

1. Ist der Teilnehmer neu oder schon vorhanden?
2. Ist das Angebot vorhanden?
3. Setzt das Kursangebot andere Kurse voraus?
4. Ist das Angebot an einem noch gültigen Datum?
5. Ist genug Literatur vorhanden?

Das sind die Abfragen, die an die Datenbank gestellt werden, entweder einzeln oder in Kombination. Zusätzlich gibt es folgende auszuführende Aktionen:

6. Teilnehmer eintragen
7. Teilnahme eintragen
8. Kursliteraturbedarf erhöhen

4.2 Die Server- und Entwicklungsumgebung

Die Serverumgebung besteht aus der Sybase „Adaptive Server“ Datenbank in der Version 12.5 und dem Sybase EAServer in der Version 3.5. Im Gegensatz zur Version 4.1, die im letzten Kapitel beschrieben wurde, ist diese Version noch nicht voll kompatibel zum neuen J2EE 1.3 Standard und bietet keine Web-Services. Da man für die Umsetzung des Beispiels nur einen CORBA kompatiblen Komponentenserver und eine Datenbank-anbindung benötigt, reicht die ältere Version des EAServers aus. Sie beinhaltet den „Jaguar Server“ als Komponentenserver.
Als Entwicklungswerkzeug wird PowerJ benutzt, das eine gute Integration des Jaguar Servers in die Java-Umgebung bietet. Für die Erstellung der Beispiel-Komponente kommt der „Application Integrator for stored

Procedures“ zum Einsatz. Er ermöglicht das einfache Erstellen einer Komponente für den Zugriff auf eine Stored-Procedure.

4.3 Die klassische Client-Server Lösung

Die Teilnahme an einem Kurs läßt sich als Client-Server Anwendung umsetzten. Ein Java-Client nutzt eine Verbindung zum Datenbankserver über die JDBC-Schnittstelle, um Daten zu lesen und zu ändern.

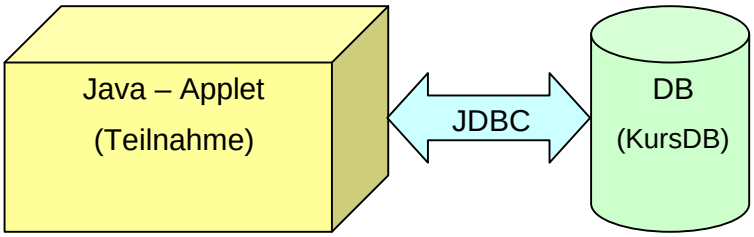

Abb. 4-3 Java JDBC Applet

So ließe sich diese Anwendung auch mit einer einfachen, serverseitigen Scriptsprache lösen. Um jedoch den Unterschied zur komponentenbasierten Lösung zu veranschaulichen, wird bei allen Lösungen ein Java-Applet entstehen. Die Umsetzung als Java Applet mit JDBC-Zugriff bringt einen Nachteil mit sich:
Ein Applet arbeitet auf dem Browser im sog. „Sandbox Modus“. Das bedeutet unter anderem, dass alle Funktionen verboten sind, die ein Sicherheitsrisiko darstellen. Eine JDBC-Verbindung zu einem fremden Server stellt ein Sicherheitsrisiko dar. In der Entwicklungsumgebung ist dieser Mechanismus nicht aktiv, da das Applet als Applikation gestartet wird.

4.3.1 Verwendete Java-Objekte (Client-Server)

Neben den Standard Java-AWT-Objekten werden einige spezielle Objekte von PowerJ verwendet, um auf die Sybase Datenbank zuzugreifen und diese Daten zu verarbeiten:

- **powersoft.powerj.db.java_sql.Transaction**
  Dieses Objekt stellt das Transaktionsmanagement für Datenbanken zur Verfügung und wird für das Query-Objekt benötigt
- **powersoft.powerj.db.java_sql.Query**
  Dieses Objekt verarbeitet Anfragen an die Datenbank und nutzt das Transaction-Objekt
- **powersoft.powerj.ui.grid.DBGrid**
  Dieses Objekt stellt die Ergebnisse einer Datenbankanfrage in einer Tabelle dar und ermöglicht die Auswahl bzw. Änderung einzelner Daten.

4.3.2 Der Client-Server Quellcode

Es werden nur die wichtigsten Befehlszeilen des Quellcodes beschrieben. Die meisten Befehle sind durch das Anwenden der Drag-and-Drop Funktion bzw. das Ändern der Objekteigenschaften in PowerJ entstanden. Der komplette Quellcode befindet sich auf der beiliegenden CD-ROM.

Als erstes wird mit dem Transaction-Objekt eine Verbindung zur Datenbank aufgebaut. Der Benutzername und das Passwort sind bei diesem Beispiel fest im Programm verankert. Die wichtigsten Zeilen im Quellcode für den Verbindungsaufbau sind:

```
transaction_1.setDataSource("jdbc:sybase:Tds:192.168.39.38:12000/jag
demo" );
transaction_1.setUserID( "jagdemo" );
transaction_1.setPassword( "jagdemo" );
```

Ein Query-Objekt kann nun über das Transaction-Objekt auf die Datenbank zugreifen. Das erste Query-Objekt ist für die Ausführung der Anfragen im Hauptteil zuständig. Das zweite Query-Objekt ist für die Darstellung der Angebote zuständig. Beide Objekte werden an das Transaction-Objekt gebunden:

```
query_1.setTransactionObject( transaction_1 );
query_2.setTransactionObject( transaction_1 );
```

Damit das DBGrid die Angebote darstellen kann, wird die entsprechende Query an das DBGrid-Objekt gebunden und eingegeben:

```
grid_1.setDataSource( query_2 );
query_2.setSQL( "select * from Angebot_voll where Datum>getdate()
order by Datum" );
```

Im Hauptteil, welcher beim Drücken der Schaltfläche „Teilnehmen“ ausgeführt wird, werden die einzelnen SQL-Anfragen abgearbeitet. Sie werden immer im gleichen Stil aufgerufen:

```
sql="begin transaction";
query_1.setSQL(sql);
query_1.execute();
```

Da die Eingabe- und Kontrollstrukturen nicht von Bedeutung sind, werden hier nur die SQL-Befehle beschrieben, die zur Kommunikation mit der Datenbank verwendet werden. Um die Variablen leicht erkennbar zu machen, werden sie hier mit einem @ gekennzeichnet, was dem Transact-SQL-Standard entspricht, der bei Sybase zum Einsatz kommt.

A) `SELECT * FROM Angebot_voll where Datum>getdate() order by Datum`
   Holt alle Angebote mit den Kursbeschreibungen aus der View „Angebot_voll“, für die Darstellung im DBGrid-Objekt.

B) `BEGIN TRANSACTION`
   Mit diesem Befehl wird eine Transaktion gestartet. Alle SQL-Befehle innerhalb einer Transaktion können rückgängig gemacht werden, falls beispielsweise ein Fehler auftritt.

C) `SELECT TnNr from Teilnehmer where Name=@name and Ort=@ort`
   Diese Funktion prüft, ob der angegebene Teilnehmer vorhanden ist und holt ggf. die Teilnehmer-Nummer.

D) `SELECT max(TnNr)+1 FROM Teilnehmer`

Ist der Teilnehmer nicht vorhanden, gibt diese Funktion die nächste, freie Teilnehmer-Nummer zurück.

E) `INSERT INTO Teilnehmer (TnNr,Name,Ort,Email) VALUES (@tnnr,@name,@ort,@email)`

Hier wird ein neuer Teilnehmer eingetragen, wenn er nicht vorhanden ist.

F) `SELECT count(VorNr) FROM Vorauss WHERE KursNr=@kursnr AND VorNr NOT IN (SELECT KursNr FROM Nimmt_teil WHERE TnNr=@tnnr)`

Mit dieser verschachtelten Anfrage wird geprüft, ob die Voraussetzungen für den gewünschten Kurs erfüllt sind.

G) `INSERT INTO Nimmt_teil (AngNr,KursNr,TnNr) VALUES (@angnr,@kursnr,@tnnr)`

Hier wird die Teilnahme gespeichert.

H) `UPDATE Kurslit SET Bedarf=Bedarf+1 WHERE KursNr=@kursnr`

Durch dieses Update wird der Literaturbedarf des betroffenen Kurses um eins erhöht.

I) `COMMIT TRANSACTION`

Mit diesem Befehlt wird die Transaktion abgeschlossen und als gültig erklärt.

J) `ROLLBACK TRANSACTION`

Kommt es zu einem Fehler, so kann dieser Befehl alle SQL-Befehle rückgängig machen, die dem `BEGIN TRANSACTION` folgten.

Zur Messung der von den Funktionen in Anspruch genommenen Zeit, dient ein Timer, der die Zeit vor und nach Abarbeitung der Funktionen nimmt:

```
java.lang.Long starttime = new
java.lang.Long(java.lang.System.currentTimeMillis( ));
…
java.lang.Long endtime = new
java.lang.Long(java.lang.System.currentTimeMillis( ));
java.lang.Long diff = new java.lang.Long(endtime.longValue()-
starttime.longValue());
```

4.3.3 Das fertige Client-Server Applet

Das fertige Applet funktioniert nur in der Entwicklungsumgebung, da im Browser der „Sandbox Modus“ die Datenbankverbindung verhindert. Das User Interface des ersten Beispiels ist in der Abbildung 4-4 dargestellt.

Das Einlesen der Kursdaten löst durch seine Datenmenge den größten Netzverkehr aus. Der Engpass kann hier in der Übertragung der Daten liegen.

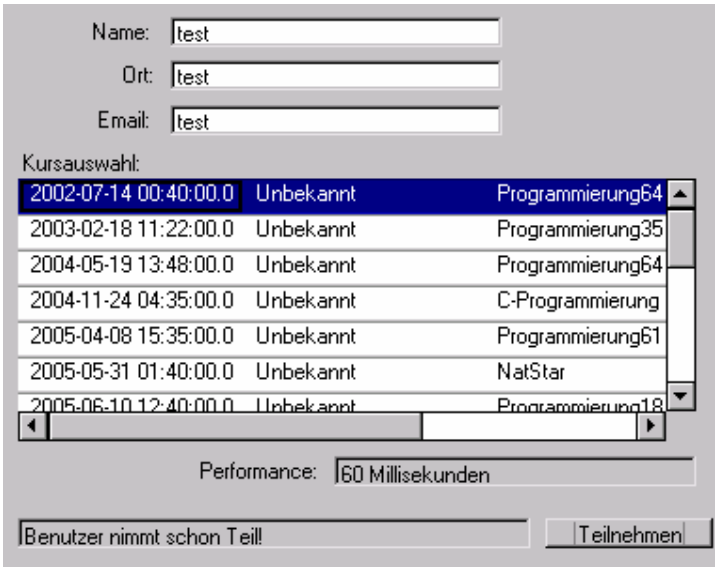

Abb. 4-4 User Interface vom Client-Server Applet

Alle Daten, ob zur Bearbeitung oder nur zur Auswertung im Applet müssen über die Netzwerkschnittstelle übertragen werden. Bei vielen Clients kann so die Netzwerkanbindung am Server an seine Grenzen geraten.

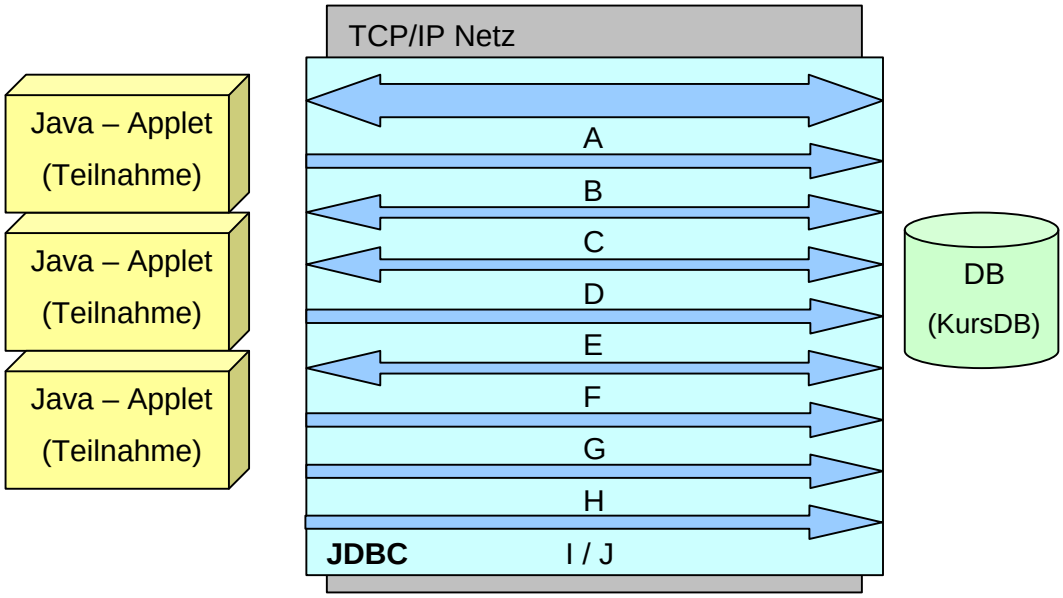

Abb. 4-5 Beispiel 1 - Netzverkehr

Die Verarbeitung der kompletten Daten passiert beim Client, der zusammen mit der Geschwindigkeit des JDBC und des Netzes die Performance bestimmt. Die Abbildung 4-5 zeigt die schematische Darstellung des nötigen Datenbanktransfers. Die Buchstaben kennzeichnen die Datenbankaufrufe aus Kapitel 4.3.2.

4.3.4 Verwendung von Stored-Procedures

Erweitert man das vorherige Beispiel um eine Stored-Procedure (siehe Kapitel 2.1.4), so fallen die meisten SQL-Anfragen im Java-Applet weg. Diese Prozedur wird auf dem Datenbank-Server gespeichert und ausgeführt. Dadurch wird ein Großteil der Funktionalität auf den Datenbank-Server übertragen (siehe Abb. 4-6). Auch der Transfer über die JDBC-Schnittstelle wird optimiert, da keine Kontrollwerte mehr an das Java-Applet übertragen werden müssen. Die meisten SQL-Anfragen aus dem vorherigen Beispiel können in einer Stored-Procedure zusammengefasst werden. Es bleiben nur noch folgende SQL-Anfragen übrig:

A) `SELECT * FROM Angebot_voll where Datum>getdate() order by Datum`
Holt alle Angebote mit den Kursbeschreibungen aus der View „Angebot_voll“, diese werden in einer Liste zum Auswählen angeboten.

B) `Teilnahme @name, @ort, @email, @angnr, @kursnr`
Ruft die Stored-Procedure Teilnahme auf.

Diese Vorgehensweise spart einiges an Netzbandbreite. Die komplette Datenkontrolle und Manipulation übernimmt der Datenbank-Server. Bei sehr vielen, gleichzeitigen Aufrufen der Prozedur besteht aber die Gefahr, dass der Server diese nicht mehr schnell genug verarbeiten kann. Die Auswahlliste der Angebote muß komplett an den Client übertragen werden, da sie dort angezeigt wird.

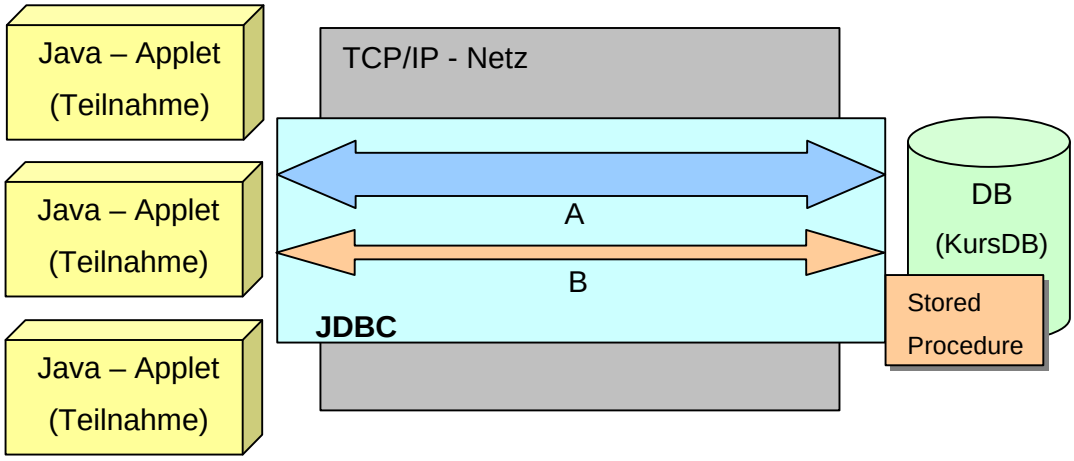

Abb. 4-6 Beispiel 1 mit Stored-Procedure - Netzverkehr

Der komplette Quellcode der Stored-Procedure befindet sich auf der beiliegenden CD-ROM.

4.4 Die verteilte, komponentenbasierte Lösung

Bei der zweiten Möglichkeit der Umsetzung wird keine JDBC-Schnittstelle benötigt. Es werden über den Komponentenserver „Jaguar“ zwei Methoden aufgerufen:

- Die Ausgabe der Angebote (Angebot_holen)
- Die Teilnahme (Teilnahme)

4.4.1 Die benötigte Infrastruktur

Die Einarbeitung in die Infrastruktur dieses verteilten Systems nimmt etwas Zeit in Anspruch. Eine Vielzahl von Servern, Protokollen und Sicherheitseinstellungen müssen korrekt miteinander harmonieren, damit ein solches System stabil funktioniert. Hier ist man auf gut durchdachte Administrationswerkzeuge angewiesen.

Für die Umsetzung der Teilnahmefunktion benötigt man den Komponentenserver „Jaguar“, einen Datenbank-Server, PowerJ als Entwicklungsumgebung und den „Application Integrator for stored Procedures“, der die Erstellung der Komponenten vereinfacht. Bevor es zur eigentlichen Implementierung bzw. zur Entwicklung des Clients kommt, muß etwas Vorarbeit geleistet werden:

1. Erstellen eines Paketes mit dem Namen „Schwarzb“ im Jaguar.
2. Erstellen einer Komponente mit dem Namen „jagdemo“ im Jaguar.
3. Erstellen der Methoden in der Komponente (Implementierung).

4. Einrichten eines Benutzers und einer Sicherheitsrolle im Jaguar.
5. Verknüpfung der Sicherheitsrolle mit den Komponenten.

6. Einrichten eines Zugriffsprofils für den Jaguar-Server in PowerJ.
7. Einrichten der eigenen Komponenten in PowerJ
   (für Drag-and-Drop Programmierung).
8. Entwicklung des Clients.

Nach den Vorbereitungen gestaltet sich die eigentliche Entwicklung der Anwendung recht einfach. Per Drag-and-Drop lassen sich Verbindungen zum

Jaguar-Server aufbauen und die Komponenten einfach einbinden. In den folgenden Abschnitten wird beschrieben, wie und warum diese Vorarbeiten geleistet werden müssen.

4.4.2 Manuelle Erstellung von Komponenten und Methoden

Um die Arbeitsweise des „Application Integrators“ im nächsten Abschnitt zu verstehen, wird die manuelle Bereitstellung einer Komponente auf dem Jaguar-Server beschrieben. Die folgenden Einstellungen werden im Jaguar-Administrationstool vorgenommen:

1. *Ein neues Paket erstellen*
   Die Komponenten werden auf dem Jaguar-Server in sog. Paketen installiert, um sie thematisch und funktional zu trennen und so die Übersicht zu verbessern.
2. *Eine neue Komponente definieren*
   Um eine neue Komponente in einem Paket zu erstellen, braucht man eine Beschreibung, den Komponententyp und einen Java-Klassennamen.
3. *Eine Methode und dessen Interface erstellen*
   In der Komponente können nun Methoden definiert werden. Neben dem Namen der Methode muß man noch die Ein- und Ausgabevariablen (das „Interface“) definieren.
4. *Generierung der Stubs/Skeletons für Java*
   Der Jaguar-Server kann nun für die ausgewählte Komponente die Stubs und Skeletons generieren (siehe Kapitel 2.2.3.1). Dazu wählt man den CORBA- und Java-Standard aus und gibt das Basisverzeichnis der Java-Klassen an.

Für die folgenden Aktionen benötigt man einen Editor und einen Java Compiler.

5. *Kompilieren des Stub-Codes*
   Mit dem Compiler wird nun der Stub-Code kompiliert, der im Verzeichnis („[*JAGUAR ROOT*]\html\classes\[*KOMPONENTENNAME*]“) generiert wurde.
6. *Implementieren der Funktion*
   Nun muß die gewünschte Funktion implementiert werden. Dazu bearbeitet man die Datei mit der Endung „NEW“ im Verzeichnis („[*JAGUAR ROOT*]\html\classes\[*KLASSENBAUM DER METHODE*]“), und speichert sie ohne die Endung „NEW“, wodurch eine Java-Quelldatei entsteht.
7. *Kompilieren der Implementierung und des Skeleton-Codes*
   Zum Abschluss muß diese Datei und der Skeleton-Code kompiliert werden, die sich beide im selben Verzeichnis befinden.

Nun steht die Komponente zur Benutzung auf dem Jaguar-Server bereit.

4.4.3 Benutzung des „Application Integrator for stored Procedures“

Im letzten Abschnitt wurde beschrieben, wie eine Komponente erstellt wird. Für die Beispielaufgabe benötigt man jedoch keine eigene Implementierung, da die Datenbankfunktionalität schon in den Stored-Procedures „**Teilnahme**“ (siehe Kapitel 4.3.4) und „**Angebot_holen**“ gekapselt ist.
Die Prozedur „Angebot_holen“ beinhaltet eine einfache Select-Anweisung zur Ausgabe der gültigen Angebote.
Der „Application Integrator“ (kurz: AI) kann eine Stored-Procedure auf eine Methode einer Jaguar-Komponente abbilden. Dazu verfügt der AI über verschiedene, fertige Komponenten für den Zugriff auf Datenbanken und die Transformation von Daten.
Um Zugriff auf die gewünschte Datenbank zu bekommen, richtet man im AI eine neue Verbindung (engl. „Connection“) ein. Dazu werden der Typ der Datenbankverbindung, die Adresse des Datenbank-Servers und die Login-Daten benötigt.

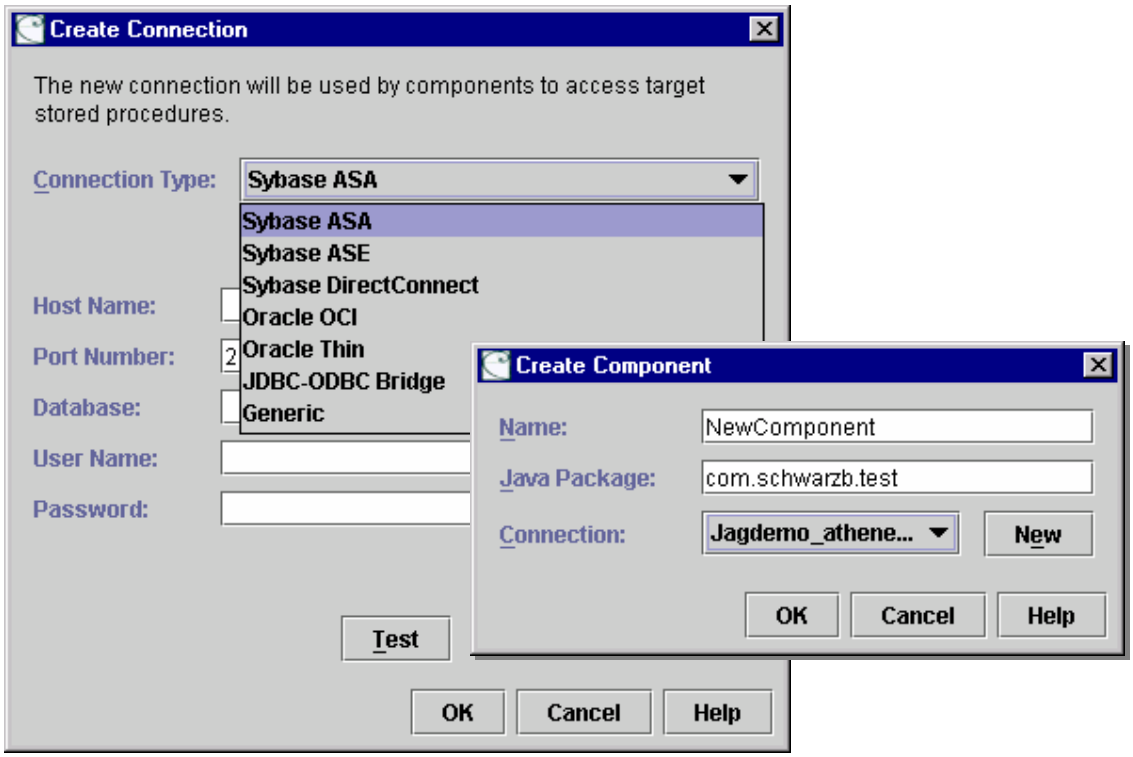

Abb. 4-7 AI for Stored-Procedures – Create Connection & Component [18]

Mit dieser Verbindung kann nun eine neue Komponente generiert werden. Dazu werden der Name der Komponente und der zu generierende Klassenpfad angegeben.

Als nächstes müssen Methoden für die Komponente definiert werden, welche die entsprechenden Stored-Procedures aufrufen. Dazu werden der Datenbank-Name (hier „Catalog“), der Besitzer (hier „Schema“) und der Name der Stored-Procedure benötigt.

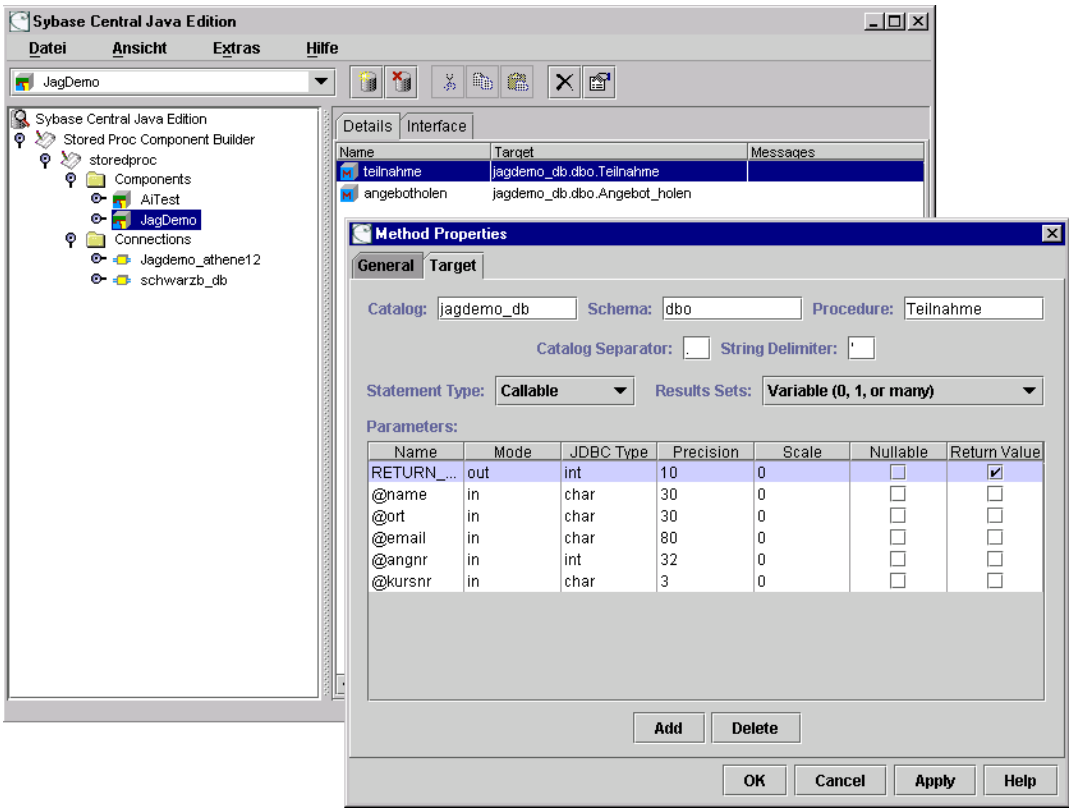

Abb. 4-8 AI for Stored-Procedures – Method Properties [18]

Es ist wichtig zu wissen, welche Variablen in welcher Reihenfolge erwartet und zurückgegeben werden. Diese Variablen werden dann automatisch oder manuell auf die entsprechenden Java-Typen abgebildet. Die Methode stellt sie später als Ein- und Ausgabewerte zur Verfügung. Es können nur Prozeduren benutzt werden, die über die verwendete Verbindung aufrufbar sind.

Auf diese Art und Weise werden die Prozeduren „Angebot_holen“ und „Teilnahme“ als Methoden der Komponente „Jagdemo“ erstellt.

Nachdem die Methoden definiert sind, kann man die Komponente in ein Paket auf dem Jaguar-Server installieren (hier „deploy“).

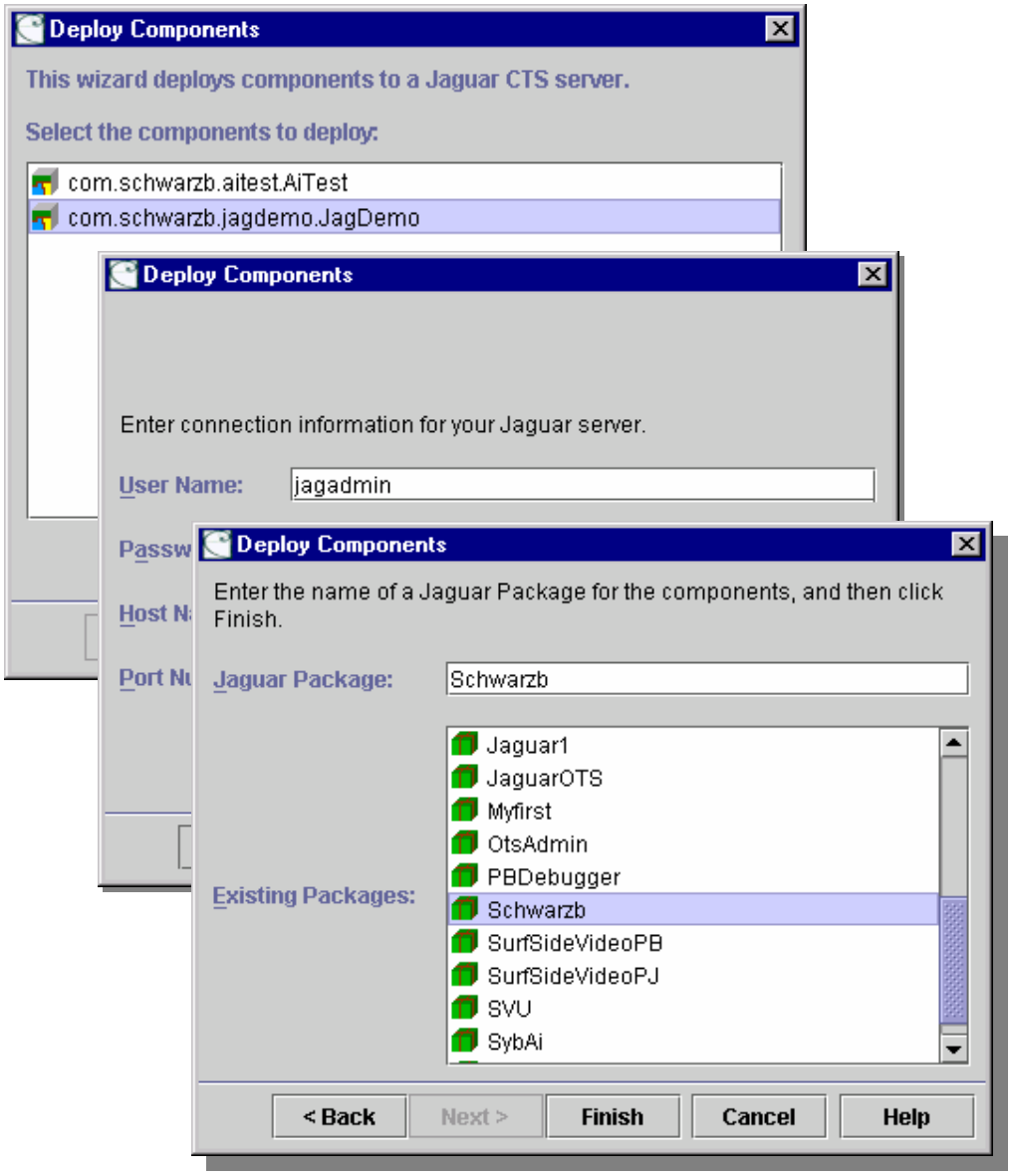

Abb. 4-9 AI for Stored-Procedures – Deploy Components Wizard [18]

Für die Installation der Komponente werden die Login-Daten des Jaguar-Administrators benötigt. Die Komponente wird im Paket „Schwarzb“ installiert.

4.4.4 Sicherheitsmechanismen des Jaguar Servers

Der Jaguar-Server unterstützt die Authentifizierung eines Benutzers und die Verschlüsselung der Kommunikation zwischen Client und Server. Es besteht die Möglichkeit, die Kommunikation per SSL (engl. „Secure Socket Layer“) mit verschiedenen Verschlüsselungsalgorithmen durchzuführen. Über die PKI (engl. „Public Key Infrastructure“) können private und öffentliche Schlüssel genutzt werden. Die Authentifizierung passiert entweder direkt auf dem Jaguar-Server, über die betriebssysteminterne Benutzeranmeldung oder per ID’s über Zertifizierungsstellen (intern oder extern).

In diesem Beispiel wird nur die unverschlüsselte, direkte Authentifizierung mittels Namen und Passwort verwendet. Der Authentifizierungsmechanismus des Jaguar-Servers basiert auf sog. Sicherheitsrollen. Eine Rolle (engl. „role“) ist eine Sammlung von Benutzern, Benutzergruppen und digitalen ID’s, denen der Zugriff erlaubt bzw. verboten werden kann. Diese Rollen können Jaguar-Paketen, einzelnen Komponenten oder deren Methoden zugeordnet werden.

Als erstes wird ein neuer Benutzer bzw. eine neue Identität (engl. „Identity“) auf dem Jaguar-Server eingerichtet, mit den Namen „jagdemo“ und dem Passwort „jagdemo“ ohne Verschlüsselung und Zertifizierung.

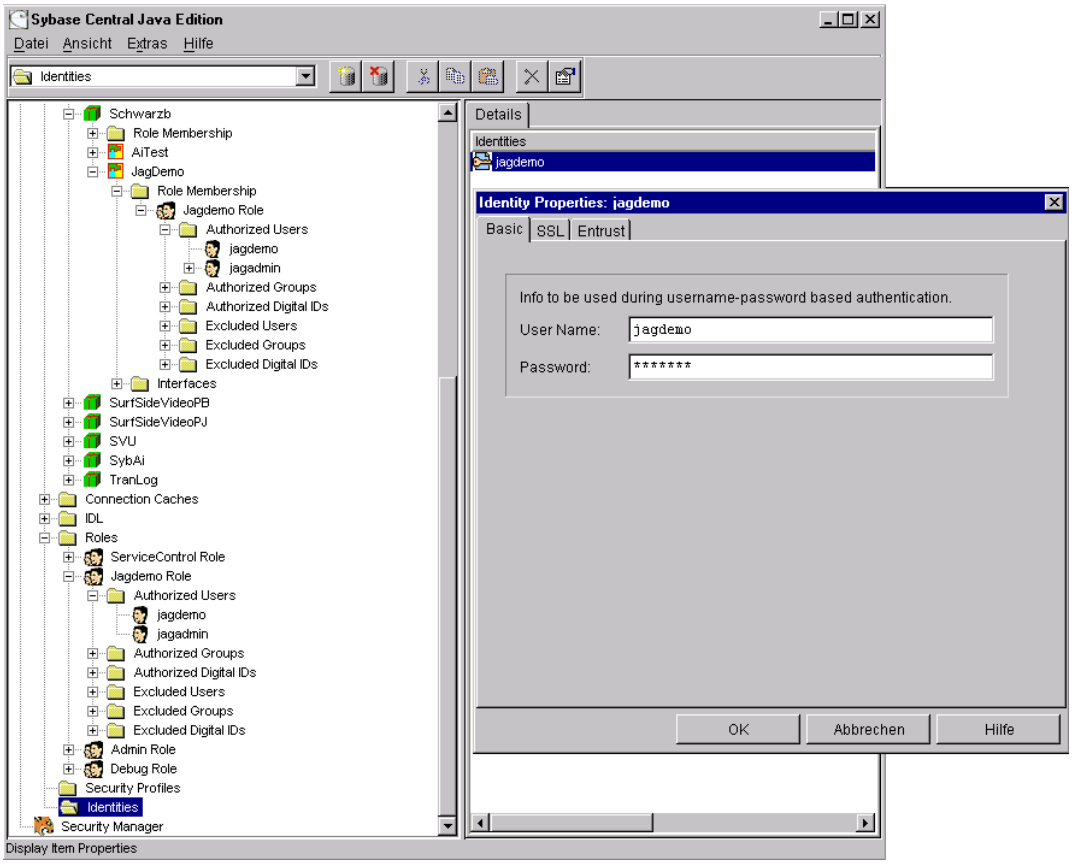

Abb. 4-10 Benutzer und Sicherheitsrollen auf dem Jaguar-Server [18]

Nun wird eine neue Rolle mit dem Namen „Jagdemo Role“ erstellt. Der Benutzer „jagdemo“ wird in die Gruppe der erlaubten Benutzer (engl. „authorized Users“) eingetragen. Anschließend wird diese Rolle mit der Komponente „Jagdemo“ verknüpft, die sich im Paket „Schwarzb“ befindet. Jetzt kann nur noch der Benutzer „jagdemo“ auf die Komponente zugreifen.

4.4.5 Manuelle Erstellung eines CORBA-Clients

Bevor der Client mit der Hilfe von PowerJ per Drag-and-Drop zusammengesteckt wird, sollte man zumindest wissen, wie eine Verbindung zum ORB und der Aufruf einer Methode von Hand zu erstellen ist. Es werden nur die wichtigsten Befehlszeilen des Quellcodes beschrieben. Diese sind

teilweise aus dem Kontext genommen, und einzeln nicht lauffähig. Diese Anleitung ist aus dem Tutorial-Client des Jaguar-Servers entstanden.

Als erstes wird die Hauptklasse von CORBA importiert:

```
import org.omg.CORBA.*;
```

Um eine Komponente zu nutzen, muss diese erst gefunden werden. Dazu wird ein Namensdienst (engl. „naming service“) benötigt. Um verständliche Fehlermeldungen auszugeben, wird ein Context-Objekt benötigt:

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
```

Als nächstes wird das Paket importiert, welches die Stub-Klasse der zu verwendenden Komponente enthält:

```
import Schwarzb.*;
```

Im Hauptprogramm wird der Name der Komponente definiert, und eine leere Instanz der Stub-Klasse gebildet:

```
String _compName = "Schwarzb/JagDemo";
Schwarzb.JagDemo _comp = null;
```

Nun kann der ORB initialisiert werden. Die Einstellung (engl. „properties“) der Verbindungsdaten müssen vorher festgelegt werden:

```
java.util.Properties props = new java.util.Properties();
ORB orb = ORB.init(this, props);
```

Über das ORB-Objekt wird der Namensdienst aufgerufen:

```
org.omg.CORBA.Object objRef = null;
objRef = orb.resolve_initial_references("NameService");
```

```
NamingContext nc = null;
nc = NamingContextHelper.narrow(objRef);
```

Um die Komponente benutzen zu können, wird die Instanz des zugehörigen Stub-Objektes gebildet, und damit eine Verbindung hergestellt:

```
NameComponent compNc[] = { new NameComponent(_compName, "") };
Factory compFactory = FactoryHelper.narrow ( nc.resolve(compNc) );

_comp = Schwarzb.JagDemoHelper.narrow( compFactory.create("jagdemo",
"jagdemo") );
```

Wie in Java üblich, werden die Ausnahmefehler über einen Try-Catch-Block behandelt, der in diesem Beispiel folgende Fehler verarbeitet:

`catch (NotFound nfe)`
Wenn die Komponente nicht gefunden wurde.

`catch (org.omg.CORBA.UserException ue)`
Wenn ein Fehler im Namensdienst aufgetreten ist.

`catch (org.omg.CORBA.SystemException se)`
Wenn ein CORBA-Systemfehler aufgetreten ist.

Sind keine Fehler aufgetreten, können die Methoden der Komponente auf folgende Art und Weise aufgerufen werden:

```
_comp.teilnahme(retval, Name, Ort, Email, AngNr, KursNr);
```

Es ist jedoch in dem meisten Fällen nicht nötig, den Quellcode für den Client von Hand einzugeben. Es stehen mit der Entwicklungsumgebung PowerJ leistungsfähige Werkzeuge zur Verfügung, um die Arbeit zu erleichtern.

4.4.6 Die Entwicklung des Clients mit PowerJ

Der Vorteil bei der Nutzung von PowerJ als Entwicklungsumgebung liegt in der guten Integration des Jaguar-Servers. Mit PowerJ lassen sich die meisten Objekte per Drag-and-Drop einfach in ein Programm integrieren. Die Eigenschaften eines Objektes (engl. „Properties“), können über die Benutzeroberfläche editiert werden, ohne in den Quellcode eingreifen zu müssen. Die Programmierung, die von Hand vorgenommen werden muß, beschränkt sich auf ein Minimum. Der Zugriff auf den Jaguar-Server und den darauf installierten Komponenten läßt sich dadurch vereinfachen.
Um die zuvor erstellte Komponente zu nutzen, muß diese jedoch erst in PowerJ integriert werden. Mit dem Verbindungsprofil des Jaguar-Administrators bekommt man Zugriff auf das Repository des Servers.

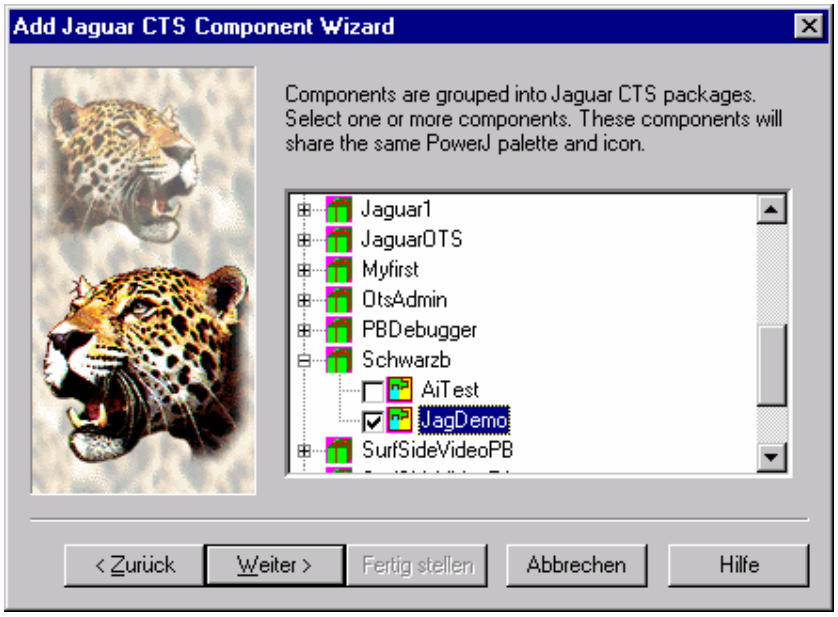

Abb. 4-11 PowerJ – Add Jaguar CTS Component Wizard [18]

Daraus werden die Komponenten gewählt, die PowerJ als Drag-and-Drop Objekt bereitstellen soll.

4.4.7 Verwendete Java-Objekte (CORBA-Client)

Im Gegensatz zum ersten Beispiel brauchen wir keine JDBC-Schnittstelle mehr. Neben den Standard Java-AWT-Objekten werden einige spezielle Objekte von PowerJ verwendet, um auf den Jaguar-Server zuzugreifen und die Komponenten aufzurufen:

- **powersoft.powerj.jaguar.InitialContext**
  Dieses Objekt stellt die Verbindung zum Jaguar-Server her.
- **org.omg.CORBA.Object**
  Dieses Objekt beinhaltet die CORBA-Funktionalität (siehe Kapitel 2.2.3).
- **com.schwarzb.jagdemo.JagDemo**
  Dies ist das Objekt, welches vom „Application Integrator“ erstellt wurde und die Methoden für den Datenbankzugriff enthält.

Zusätzlich werden noch einige Objekte aus dem Kapitel 4.3.1 verwendet, um die Angebote darzustellen.

4.4.8 Der CORBA-Client Quellcode

Es werden nur die wichtigsten Befehlszeilen des Quellcodes beschrieben. Diese sind teilweise aus dem Kontext genommen und einzeln nicht lauffähig. Die meisten Befehlszeilen sind durch das Anwenden der Drag-and-Drop Funktion bzw. das Ändern der Objekteigenschaften in PowerJ entstanden. Der komplette Quellcode befindet sich auf der beiliegenden CD-ROM.

Um die Verbindung zum Jaguar-Server aufzubauen, wird ein sog. InitialContext-Objekt verwendet. Als erstes müssen die Login-Daten für den Jaguar-Server gesetzt werden:

```
jctx_init.create( "main.jctx_init" );
jctx_init.setUser( "jagdemo" );
jctx_init.setPassword( "jagdemo" );
```

```
jctx_init.setURL( "iiop://192.168.39.14:9000" );
```

Nun wird eine Verbindung aufgebaut, um auf die Komponente „Jagdemo“ zugreifen zu können:

```
jctx_init.connect();  // ignore error
powersoft.powerj.jaguar.InitialContext jctxt_jagdemo_1 =
powersoft.powerj.jaguar.InitialContext.findByName( "main.jctx_init"
);
jctxt_jagdemo_1.connect();
```

PowerJ erstellt automatisch mehrere Möglichkeiten, die entfernte Komponente zu finden und bereitzustellen. Je nachdem welcher Namensdienst oder ob Verschlüsselung verwendet wird, entsteht das Objekt auf eine andere Weise:

```
org.omg.CORBA.Object cobj = (org.omg.CORBA.Object)
jctxt_jagdemo_1.lookup("Schwarzb.JagDemo");
jagdemo_1 = com.schwarzb.jagdemo.JagDemoHelper.narrow(cobj);
```

oder

```
jagdemo_1 = com.schwarzb.jagdemo.JagDemoHelper.narrow(
jctxt_jagdemo_1.getORB().string_to_object("Schwarzb/JagDemo") );
```

oder

```
fact_jagdemo_1 = SessionManager.FactoryHelper.narrow(
(org.omg.CORBA.Object) jctxt_jagdemo_1.lookup( "Schwarzb/JagDemo" )
);
jagdemo_1 = com.schwarzb.jagdemo.JagDemoHelper.narrow(
fact_jagdemo_1.create() );
```

Mit dem neu entstandenen Objekt „jagdemo_1“ kann man nun die Methoden aufrufen. Bei der Ausgabe der Angebote wird die Ergebnismenge über ein Query-Objekt zur Ausgabe gebracht:

```
TabularResults.ResultSet[] data = jagdemo_1.angebotholen();
```

```
query_1.setResultSetObject(data[0]);
grid_1.setDataSource(query_1);
grid_1.setVisible(true);
```

Nach der Kontrolle der Eingabewerte muß nur noch die Methode „Teilnahme“ aufgerufen werden. Der Rückgabewert nennt sich „retval“, wird hier aber nicht benutzt. Für die Fehlerausgabe wird eine Exception verwendet.

```
jagdemo_1.teilnahme(retval, Name, Ort, Email, AngNr, KursNr);
```

Zur Messung der von den Funktionen in Anspruch genommenen Zeit, dient ein Timer, der die Zeit vor und nach Abarbeitung der Funktionen nimmt.

```
java.lang.Long starttime = new
java.lang.Long(java.lang.System.currentTimeMillis( ));
...
java.lang.Long endtime = new
java.lang.Long(java.lang.System.currentTimeMillis( ));
java.lang.Long diff = new java.lang.Long(endtime.longValue()-
starttime.longValue());
```

4.4.9 Das fertige CORBA-Client Applet

Dieses Applet funktioniert in allen Browsern, die eine Netzverbindung zum Jaguar-Server aufbauen können. Das User Interface ähnelt dem ersten Beispiel. Hinzugekommen ist eine Schaltfläche zum Holen der Angebote.

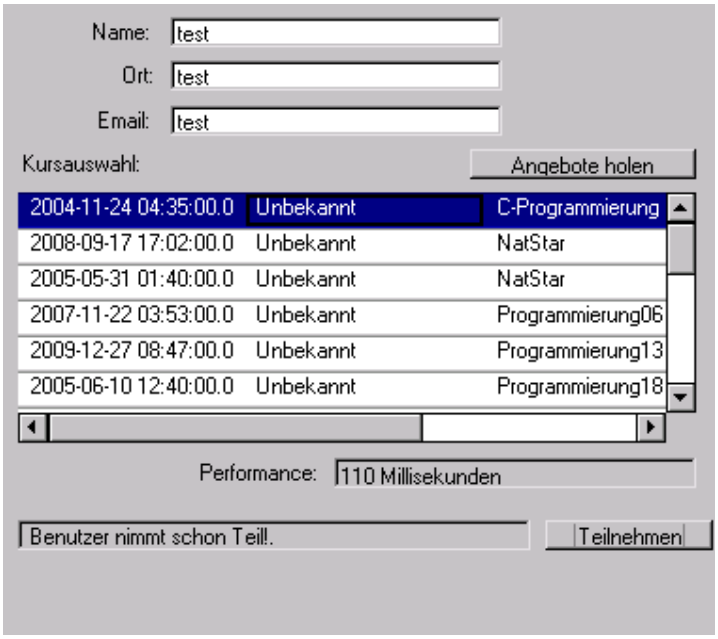

Abb. 4-12 User Interface vom CORBA-Client Applet

Die Datenlast ist wie beim Client-Server Beispiel verteilt, da die Angebote für die Darstellung auch komplett zum Client übertragen werden müssen. Die Teilnahme wird vollständig auf dem Datenbank-Server verarbeitet.

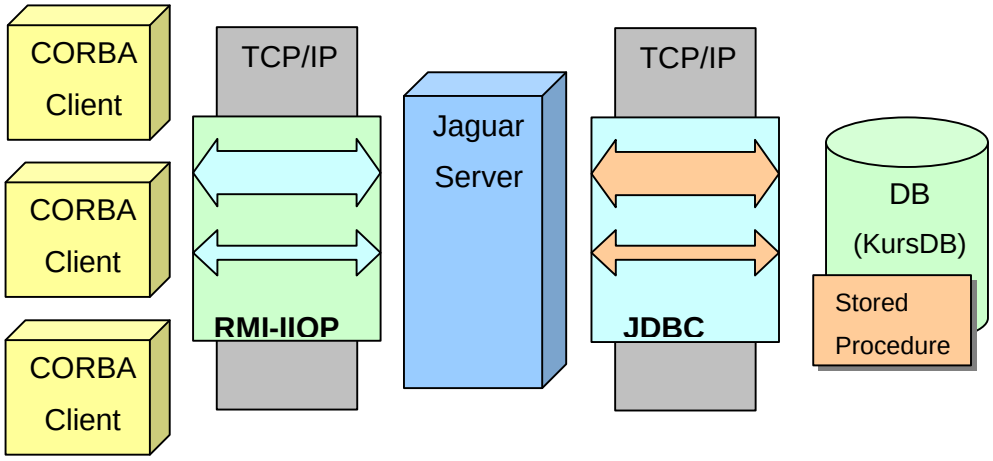

Abb. 4-13 CORBA Client – Netzverkehr

Der Kommunikationsaufwand steigt, da die Daten erst an den Jaguar-Server und dann, über ein anderes Interface, nochmals zum Client bzw. Server geschickt werden.

Läßt man die Bereitstellung der Infrastruktur außer Betracht, fällt die Implementierung der Funktionen leichter als beim Client-Server Beispiel.

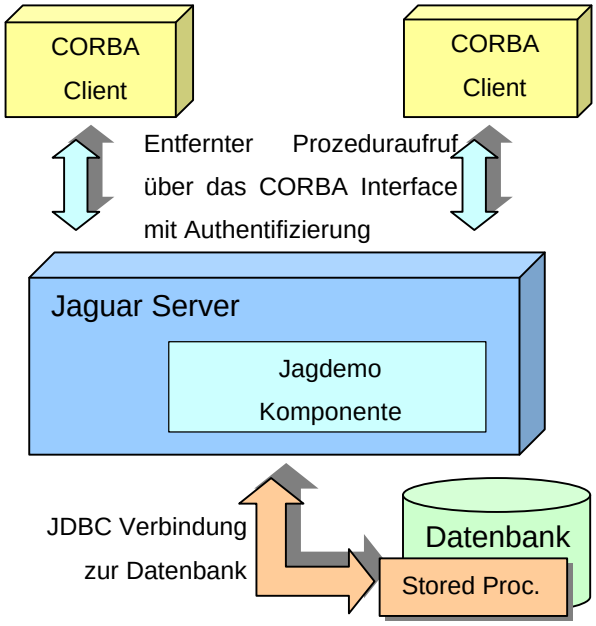

Abb. 4-14 Java-CORBA-Applet

Die größten Vorteile dieser Implementierung sind die Wiederverwendbarkeit der Komponenten in anderen Clients und die erweiterte Sicherheit. Durch die festgelegte Infrastruktur können viele Programmteile durch geeignete Entwicklungswerkzeuge automatisch generiert werden. Dadurch sinkt der Programmieraufwand, besonders bei komplexeren Projekten.

Kapitel 5 Resümee

Alle Ergebnisse und Erfahrungen des theoretischen und praktischen Teils dieser Diplomarbeit werden hier zusammengefasst betrachtet.

5.1 Praktische Erfahrung

Im praktischen Teil der Diplomarbeit werden zwei verschiedene Methoden gegenübergestellt:

- Die klassische Client-Server Programmierung
- Die komponentenbasierte Programmierung

Die Programmierung des Client-Server Beispiels gestaltet sich einfach, hier werden klassische Programmiertechniken eingesetzt. Dennoch ist der *selbst erstellte* Quell-Code umfangreicher als bei der Verwendung des Jaguar-Servers, da die Funktionalität im Client implementiert wird. Verwendet man Stored-Procedures, verschiebt man einen Grossteil der Verarbeitung auf den Datenbank-Server. Die Umsetzung in einem Java-Applet eignet sich nur zu Vergleichszwecken, eine serverseitige Scriptsprache ist die beste Lösung für eine Web-Applikation in diesem Umfang. Dies ist aber keine direkte Client-Server Lösung mehr, da hier der Web-Server die Verarbeitung übernimmt.

Für die Umsetzung der komponentenbasierten Lösung muß zuerst Vorarbeit geleistet werden. Man sollte als erstes die Infrastruktur verstehen und einrichten. Eine gut durchdachte Administrationsoberfläche ist hier entscheidend. Viele Funktionen lassen sich gut automatisieren. So ist die automatische Erstellung einer Komponente für den Zugriff auf eine Stored-Procedure ein einfacher Prozess, der sich jedoch schwer im Detail nachvollziehen läßt. Auch die Verwendung der Komponenten in der Entwicklungsumgebung ist dank Drag-and-Drop Funktion einfach zu handhaben. Dabei entsteht viel Quellcode automatisch, den man vielleicht

nicht versteht, bzw. verändern kann und darf. Solange keine Fehler auftreten, hilft der Komfort beim schnellen Arbeiten, andernfalls gestaltet sich die Fehlersuche als problematisch. Der selbst erstellte Quellcode beschränkt sich nur noch auf das Aufrufen der entfernten Methoden, der Quellcode für den Verbindungsaufbau und die Kommunikation wird automatisch erstellt. Im Idealfall kennt man sich gut mit der Materie aus und nutzt diese vielen Hilfen nur zur Arbeitsersparnis.

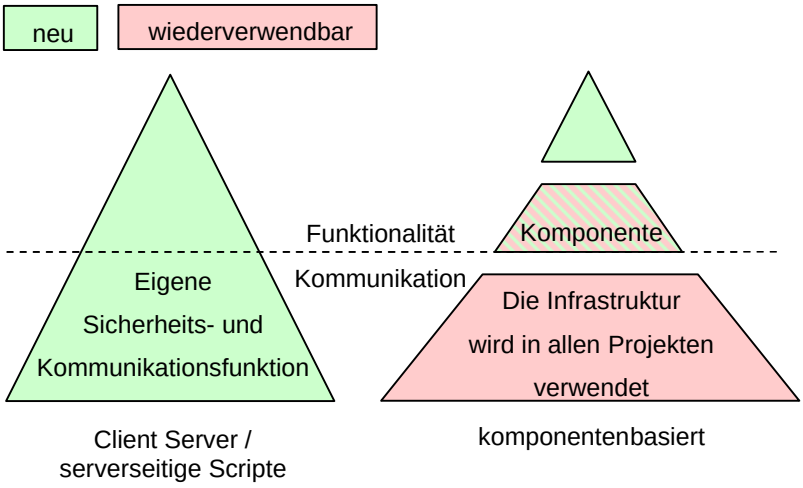

Abb. 5-1 Arbeitsaufwand

Ein so erstelltes Programm ist immer nur so gut, wie die Entwicklungsumgebung es zulässt. Die Fähigkeiten des einzelnen Programmierers sind nicht mehr so entscheidend für Qualität und Performance eines Programms.

5.2 Sicherheit

Die Sicherheit spielt eine wichtige Rolle bei der Wahl eines Applikationsservers.

Der Zugriff auf Server mit sicherheitsrelevanten Daten sollte nicht für jeden zugänglich sein. Bei direkten Client-Server Systemen läßt sich das nicht vermeiden. Die zusätzliche Schicht bei der Verwendung von Applikationsservern trennt im Idealfall die wichtigen Server von der Öffentlichkeit. So

erlauben viele Web-Applikationen nur den Zugriff auf den Web-Server, der Datenbank-Server ist nicht direkt erreichbar.
Verschiedene Authentifizierungsmechanismen sorgen für zusätzlichen Schutz vor unerlaubtem Zugriff.
Die Kommunikation kann durch verschiedene Verschlüsselungstechniken geschützt werden. So nutzen z.B. einige Homebanking-Systeme den verschlüsselten HTTPS-Standard. Es entstehen immer bessere Schutzmethoden wie z.B. die Biometrie für die Authentifizierung oder neue Verschlüsselungstechniken.

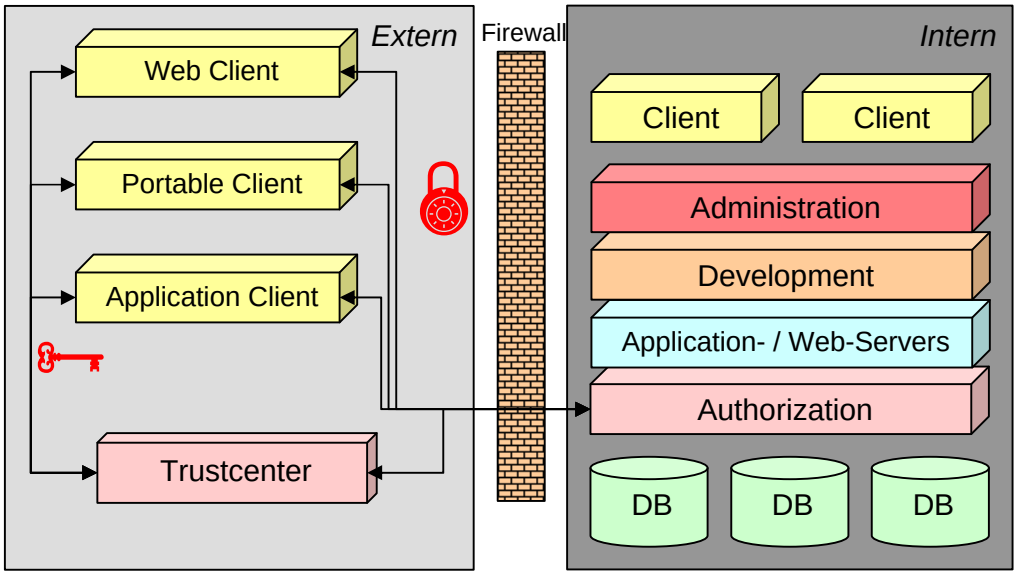

Abb. 5-2 Sicherheit bei Applikationsservern

Mit der Anzahl der unterschiedlichen Clients steigt auch die Zahl der zu verwendenden Sicherheitsmechanismen. Im Idealfall sind einheitliche Sicherheitsstandards vorhanden, um durchgehende Sicherheit zu erreichen.

5.3 Komplexität, Performance und Nutzen

In dieser Arbeit werden viele verschiedene Systeme beschrieben, um eine verteilte Anwendung zu erstellen. Es gibt die klassischen, direkten Client-

Server Applikationen, wie z.B. FTP-Clients, Peer-to-Peer Clients oder spezielle Clients im Businessbereich.

Die meisten Web-Applikationen werden heutzutage im Browser dargestellt und nutzen serverseitige Scriptsprachen oder Serversysteme, welche HTML-Output liefern. Diese Systeme stellen schon einfache Applikationsserver dar. Durch den geringen Kommunikationsaufwand können solche einfachen Systeme schneller sein als große, komplexe Applikationsserver. Die Interaktion über den Web-Browser ist jedoch stark eingeschränkt. In Zukunft werden diese Systeme mehr und mehr durch komplexere, komponenten-basierte Systeme ersetzt, was aber eine schnelle Verbindung zum Internet voraussetzt.

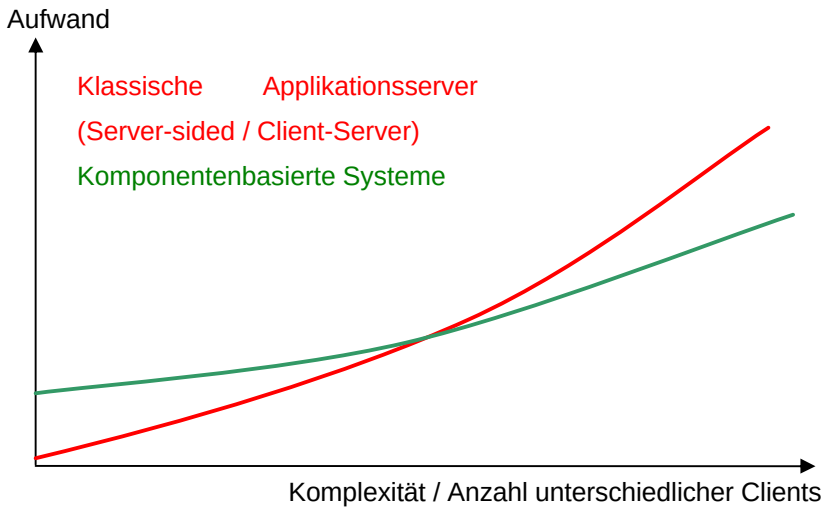

Abb. 5-3 Komplexität - Aufwand

Komponentenbasierte Systeme kommen momentan nur in größeren Firmen zum Einsatz. Sie besitzen einen großen Kern an Funktionalität, der auf verschiedene Arten verwendet werden kann, sei es als Web-Seite oder als interner Client. Damit diese komplexen Systeme einwandfrei funktionieren, muß eine gut durchdachte Infrastruktur vorhanden sein. Ab einem gewissen Grad an Komplexität ist ein System auf Komponentenbasis besser zu handhaben als andere Systeme (siehe Abb. 5-3). Ein großer Vorteil liegt in der Wiederverwendbarkeit und der Portabilität der Komponenten. Diese Systeme lassen sich sehr gut skalieren und bieten mit „Load Balancing”,

„Clustering” und „Fail-Save” Mechanismen sehr gute Performance und Sicherheit. Für ein Client-Server System oder eine Web-Applikation muß keine komplexe Infrastruktur aufgebaut werden, dafür sind die Ausbaufähigkeit, Sicherheit und Performance begrenzt.

Es gibt mit der Enterprise Edition von Java jetzt schon einen weit verbreiteten Standard für die komponentenbasierte Programmierung, der von den meisten Applikationsservern unterstützt wird. Microsoft wird mit dem .NET Projekt nachziehen und damit die Windows-Betriebssystemstruktur mit einem eigenen Konzept grundlegend umgestalten.

Kapitel 6 Anhang

6.1 Abbildungsverzeichnis

6.2 Literaturverzeichnis

[01] Buch: G. Saake, K.-U. Sattler - *Datenbanken und Java*, ISBN 3-932588-54-1

[02] Buch: A. Heuer, G. Saake - *Datenbanken: Konzepte und Sprachen*, ISBN 3-8266-0619-1

[03] Internet: *W3 Konsortium*, http://www.w3.com

[04] Buch: C. Ullenboom - *Java ist auch eine Insel*, ISBN 3898421740

[05] Internet: C. Lauer - *Introducing Microsoft .NET*, http://www.dotnet101.com/articles/art014_dotnet.asp

[06] Internet: Uni. Trier / Uni. Kaiserslautern (1998) - „*Middleware: Betriebssysteme der Zukunft*“ (PDF-Dokument)

[07] Internet: *TechMetrix Research*, http://www.techmetrix.com

[08] Dokument: Sybase - *Enterprise Application Server 3.5,* Online Dokumentation

[09] Dokument: Prof.Dr.habil.M.Günther - *Demo-Datenbank KursDB*, FH-Brandenburg

[10] Internet: Sybase - *„Neues im ASE“ (Dokumenten-ID: 34598-01-1250-01)*, http://www.sybase.com

[11] Dokument: A. Koch, M. Pietz und S. Seichter FH Bonn-Rhein-Sieg - *Verteilte und parallele Systeme: „Microsofts Welt“* (PDF-Dokument)

[12] Internet: Macromedia - *Macromedia Cold Fusion 5 Datasheet*, http://www.Macromedia.com

[13] Internet: Sun - *Java™ 2 Platform Enterprise Edition Specification, v1.3*, http://www.sun.com

[14] Internet: BEA - *WebLogic Enterprise Datasheet*, http://www.bea.com

[15] Internet: Oracle - *9iAS Datasheet und Whitepaper vom Januar 2002*, http://otn.oracle.com

[16] Internet: Microsoft – *SOAP,* http://msdn.microsoft.com/xml/general/soap_webserv.asp

[17] Internet: IBM - *WebSphere Application Server Version 4.0, Advanced Edition Whitepaper*, http://www.ibm.com

[18] Screenshots: Sybase - *EAS 3.5 Serverpaket*

[19] Internet: *SQLJ Online Resource*, http://www.sqlj.org

[20] Internet: *Perlunity Webseite – CGI Grundlagen*, http://www.perlunity.de

[21] Internet*: W3C – WSDL - Web Service Definition Language* http://www.w3.org/TR/wsdl

[22] Internet: *Sybase – EAServer 4.1 Technical Whitepaper* http://www.sybase.com

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt wurde.

Datum/Ort

Unterschrift