Universität Kaiserslautern
AG Datenbanken und Informationssysteme

Seminar im Sommersemester 2001

Datenbankaspekte des E-Commerce, Schwerpunkt: Techniken

http://wwwdbis.informatik.uni-kl.de/courses/seminar/SS2001/

Java 2, Enterprise Edition
Einführung und Überblick

Martin Husemann
m_husema@informatik.uni-kl.de

Inhalt

1. Rahmenbedingungen in der heutigen Marktwirtschaft

1.1 Die Situation für Unternehmen

Durch den wachsenden Einfluß des Internet im Allgemeinen und des E-Commerce im Speziellen steigt die Bedeutung von Information und Wissen für Unternehmen aller Art. Insbesondere für Firmen der „New Economy“ stellt ihr Wissensstamm einen Aktivposten nicht zu unterschätzenden Wertes dar. Die Verlagerung wirtschaftlicher Schwerpunkte hin zu einer „Informationswirtschaft“ zwingt die Unternehmen zum Überdenken selbst grundlegender betrieblicher Verfahrensweisen. Der Einsatz neuer Technologien und vor allem auch die erfolgreiche Einbindung bisheriger Systeme sind dabei entscheidende Aspekte.
Bereits seit einiger Zeit wird die EDV zur Unterstützung geschäftlicher Tätigkeiten eingesetzt. In jüngerer Vergangenheit gingen die Bestrebungen vermehrt in Richtung einer stärkeren Integration der verschiedenen Systeme. Weiterhin gilt es zunehmend, Daten aus heterogenen, verteilten Quellen zu integrieren und einheitlich weiterzuverarbeiten. Beispielhaft sei dafür die Fusion zweier Firmen genannt.

1.2 Herausforderungen an Applikationsentwickler

Der harte Konkurrenzdruck und der sich schnell entwickelnde Markt zwingen die Unternehmen dazu, ihre betriebliche Software in kurzen Zyklen zu entwickeln und einzusetzen. Gleichzeitig müssen gewisse Qualitätsanforderungen an die Software eingehalten werden, um Nutzung und Weiterentwicklung nicht unnötig zu erschweren; die Komplexität der Problemstellungen und damit auch der Produkte zu ihrer Lösung ist nicht zu unterschätzen. Schließlich muß sich die neue Software auch möglichst nahtlos in bestehende Strukturen einpassen und einfach zu skalieren sein. Insgesamt ergeben sich folgende Herausforderungen [2]:

- Kurze Reaktionszeiten
  Der bekannte Aspekt der termingerechten Entwicklung verschärft sich in der modernen Wirtschaft dahingehend, daß auf neue Trends und Technologien schnell und angemessen reagiert werden muß, um wirtschaftlich nicht ins Abseits zu geraten.

- Produktivität der Programmierung
  Gedankenloses Befolgen neuer Trends alleine bringt keine Vorteile. Es gilt, neue Technologien sinnvoll mit bestehenden Systemen zu vereinbaren, um Effektivitätszuwachs zu erzielen. In einer Zeit rasch wechselnder Standards und rapide veraltenden Wissens kann es für Entwickler aber sehr schwierig sein, den nötigen Überblick zu behalten. Divergierende Technologien und Programmiermodelle kommen erschwerend hinzu.

- Verfügbarkeit und Zuverlässigkeit
  Mit der steigenden Bedeutung der EDV für die Existenz der Unternehmen wächst auch die Anforderung an die permanente Verfügbarkeit der Systeme; Ausfallzeiten bedeuten ernstzunehmenden wirtschaftlichen Schaden. Auch im regulären Betrieb hat die Software ihre Funktion zuverlässig zu erfüllen, zum Beispiel bei der Sicherung von Transaktionen.

- Sicherheit
  Die zunehmende intra- und interbetriebliche Vernetzung erfordert umfassende Sicherheitsmodelle. Unternehmenskritische Informationen müssen in allen Nutzungsszenarien ausreichend geschützt sein, ohne den Aufwand für die Nutzer unangemessen zu erhöhen.

- Skalierbarkeit
  Die verwendeten Systeme müssen hardware- und softwareseitig in der Lage sein, wachsenden Anforderungen und höherer Last durch möglichst einfachen Ausbau entgegen zu kommen. Entscheidend sind dabei effektive Ressourcenverwaltung, Plattformunabhängigkeit und die Möglichkeit zur Weiterverwendung einmal entworfener Systemstrukturen.

- Integration
  Bestehende Datenbanken und Informationssysteme beinhalten bereits einen Großteil der unternehmenskritischen Informationen. Um dieses Potential zu nutzen, muß neu entwickelte Software mit den vorhandenen Systemen integriert werden können. Gerade dieser Aspekt entscheidet über den effektiven Nutzen neuer Software für die Unternehmen.

2. Mehrschichtige Systemarchitekturen

2.1 Prinzipielle Aufteilung

Die genannten Herausforderungen lassen sich nur über eine Partitionierung der betrieblichen Software erfolgversprechend angehen. Nur durch eine sinnvolle Aufteilung nach funktionalen Gesichtspunkten können die Auswirkungen bei Neuentwicklungen und Änderungen auf Teile des Gesamtsystems begrenzt werden. Grundsätzlich zerfällt ein komplexes System dabei in folgende Teile [2]:

- Die Benutzerschnittstelle
  Sie wickelt die Interaktion des Benutzers mit dem System ab und legt dabei fest, wie Ausgaben dargestellt werden und auf welche Weise Eingaben getätigt werden können.

- Die Präsentationslogik
  Als logischer Unterbau der Benutzerschnittstelle steuert die Präsentationslogik sowohl den Inhalt der Ausgaben als auch die Reaktion auf Benutzereingaben.

- Die Geschäftslogik
  In der Geschäftslogik sind die Regeln für die Steuerung des gesamten Systems implementiert. Hier sitzt die „Intelligenz“, das Wissen über den Ablauf betrieblicher Prozesse.

- Die Datenbasis
  Sie umfaßt den eigentlichen Datenbestand des Unternehmens, auf dem das System aufsetzt, und die mit seiner Verwaltung betraute Steuerungssoftware.

- Die Infrastruktur
  Zur Infrastruktur zählen die verschiedenen Wege des Daten- und Informationsaustauschs zwischen den Komponenten des Systems, zum Beispiel Message Passing oder Unterstützung des Transaktionsparadigmas.

Die genaue Umsetzung dieser fünf elementaren Bestandteile in logische Schichten (*Layers*) und physikalische Ebenen (*Tiers*) kann auf verschiedene Weisen stattfinden, von denen jede ihre spezifischen Vor- und Nachteile besitzt. In der theoretischen Betrachtung vernachlässigt man meist die Infrastruktur und faßt Benutzerschnittstelle und Präsentationslogik zusammen, woraus die Begriffe der Präsentationsschicht, der Geschäftslogikschicht und der Datenschicht folgen. Idealerweise sollten die Schichten, womöglich auch Teile der Schichten, dem Gedanken der funktionalen Partitionierung folgend unabhängig voneinander austauschbar sein.
In der Praxis werden die drei Schichten meist auf zwei oder drei Ebenen aufgeteilt.

2.2 Zwei-Ebenen-Architekturen

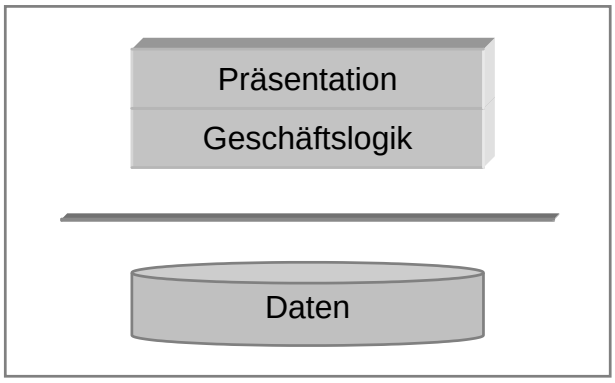

Bild 1: Zwei-Ebenen-Architekturen

Der traditionelle Ansatz bei der Arbeitsteilung zwischen zwei Ebenen besteht darin, Präsentations- und Geschäftslogik auf der Ebene der Clients zusammenzufassen, während der Server lediglich die Daten verwaltet [2]. Diese auf den ersten Blick einleuchtende Variante weist eine Reihe schwerwiegender Nachteile auf.
Zum einen ist der Wartungsaufwand sehr hoch, da in den meisten Fällen die Software auf den Clients betroffen ist. Selbst bei kleinen Änderungen muß so jeder Client einzeln aktualisiert werden. In der Praxis ist es nahezu unmöglich, derart verteilte Installationen unternehmensweit konsistent zu halten.
Zum anderen ist eine solche Systemarchitektur nur schwerlich skalierbar. Ein Großteil der Arbeitslast liegt bei den Clients, außerdem ist die Belastung der Netzwerke aufgrund der vielen nötigen DB-Anfragen vergleichsweise hoch. Ein Austausch des Servers bringt so nur wenig Leistungszuwachs, weiterer Fortschritt kann nur durch Ausbau der Netzwerkkapazitäten und der Client-Leistung erzielt werden, was einen ungleich höheren Aufwand und größere Kosten bedeutet.
Seit einiger Zeit existieren auch Architekturen, bei denen Teile der Geschäftslogik auf die Ebene der Datenhaltung, also des Servers, verschoben werden [3]. Hierbei handelt es sich typischerweise um die Teile der Geschäftslogik, die mit der Verwaltung der persistenten Daten befaßt sind.
Auf diese Weise verschiebt sich auch ein relevanter Teil der Arbeitslast auf die Datenebene, was gegenüber dem traditionellen Ansatz Vorteile bei der Skalierbarkeit und der Belastung des Netzwerkes zur Folge hat. Auch die Gesamtperformanz des Systems steigt, da komplexe Datenbank-Operationen jetzt serverintern abgearbeitet werden können, so daß

nur die eigentliche Anfrage und ihr Ergebnis über das Netzwerk versandt werden müssen, nicht aber Daten zu jedem Zwischenschritt.

Jedoch wird durch die Verteilung der Geschäftslogik auf zwei Ebenen das Konzept der Modularität konterkariert: Da die Implementierung von Geschäftsregeln auf der Datenebene in der Regel nur unter Zuhilfenahme proprietärer *Stored-Procedure*-Sprachen des jeweiligen DBS möglich ist, wird auf diese Weise die Geschäftslogikschicht fest an eine bestimmte Realisierung der Datenschicht gebunden.

2.3 Drei-Ebenen-Architekturen

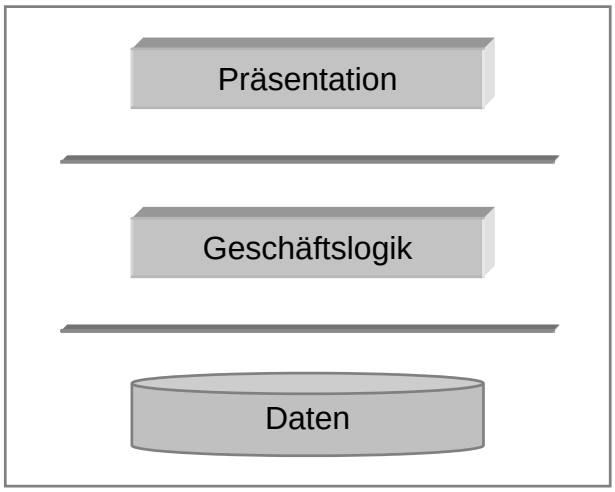

Bild 2: Drei-Ebenen-Architekturen

Bei Drei-Ebenen-Architekturen werden die drei logischen Schichten eines Systems in drei getrennten Ebenen realisiert. Die Trennung zwischen den Ebenen muß nicht unbedingt physikalischer Natur sein, es kann sich auch lediglich um Prozeßgrenzen auf einem Rechner handeln [3]. Der Definition folgend kann aber eine Schicht nicht auf mehrere Ebenen verteilt sein; somit sind Abhängigkeiten wie die in 2.2 geschilderten ausgeschlossen. Wenn nicht durch ungeschickte Modellierung der Schnittstellen auf anderem Wege Abhängigkeiten geschaffen werden, ergibt sich so eine sehr flexible und leistungsfähige Systemarchitektur.

So besteht zum Beispiel die Möglichkeit, verschiedene Client-Typen zu konstruieren, die alle auf derselben, einheitlichen Geschäftslogik aufsetzen. Da der Großteil des Systems zentral auf verschiedene Server plaziert wird, sind Aufwand und Kosten bei den meisten Änderungen überschaubar. Weiterhin ergibt sich durch die Zentralisierung der Geschäftslogik eine erhöhte Sicherheit, da schutzbedürftige Unternehmensdaten sowohl physikalisch als auch logisch geschützt im Rechenzentrum vorgehalten werden. Auch die Leistungsfähigkeit des Systems einschließlich der Möglichkeiten zur Skalierung wächst, da die relevanten Komponenten nicht mehr über das Netzwerk verteilt sind. Engpässe oder Ausfälle einer Schicht legen nicht mehr das gesamte System lahm. Zwar ist normales Arbeiten auch bei Drei-Ebenen-Architekturen nicht mehr möglich, wenn etwa die Schicht der Geschäftslogik ausfällt. Eine unabhängige Präsentationsschicht kann aber wenigstens in geordneter Weise auf den Fehler hinweisen oder sogar alternative Verbindungen zu den tieferen Schichten suchen, anstelle kommentarlos die Funktion einzustellen.

Als Nachteil der Drei-Ebenen-Architekturen erweist sich zunächst die höhere Komplexität des Gesamtsystems [3]. Um die potentiellen Vorteile auszuschöpfen, müssen die einzelnen Komponenten und ihre Schnittstellen sorgfältig modelliert und implementiert werden. Durch die Trennung der einzelnen Schichten ergibt sich auch ein höherer Kommunikationsaufwand, als dies bei stärkerer Integration der Fall wäre. Trotz dieser Aspekte zeigt sich jedoch bei konsequenter Nutzung der gebotenen Möglichkeiten die Überlegenheit der Drei-Ebenen-Architekturen.

3. Serverseitige Komponenten-Architekturen

Das Konzept der Mehrebenen-Systemarchitekturen existiert bereits seit einigen Jahren. Zentrales Element dieser Architekturen ist die Ebene der Geschäftslogik. Geschäftslogik implementierende Komponenten werden auf sogenannten *Application-Servern* eingesetzt, die den Komponenten eine Laufzeitumgebung mit verschiedenen Verbindungen zur darunterliegenden Plattform zur Verfügung stellen.
Ursprünglich existierte jedoch kein einheitlicher Standard für derartige Komponenten, so daß das Zusammenspiel mit dem Application-Server von Produkt zu Produkt auf andere Weise ablief. Diese proprietären Lösungen verringerten die Portabilität und erschwerten die effektive Nutzung des neuen Konzepts. Es fehlte eine klare Definition der Schnittstellen zwischen Komponenten und Application-Server. Idealerweise sollte ein solcher Standard einen Markt sowohl für Komponenten als auch für Application-Server eröffnen, so daß ein Unternehmen ein speziell an die eigenen Bedürfnisse angepaßtes System aus gekauften und selbst erstellten Modulen zusammenbauen können würde.
Mittlerweile sind verschiedene Standards zur Lösung der Problematik vorgestellt worden. Die bekanntesten unter ihnen stammen von Microsoft, der Object Management Group und Sun Microsystems.

3.1 Microsoft Distributed interNet Applications Architecture (DNA)

Microsoft stellte vor einiger Zeit *Windows DNA* vor. Zugrundeliegende Plattform für alle Elemente war ursprünglich *Windows NT*; mittlerweile ist der Nachfolger *Windows 2000* etabliert. Hauptbestandteile von *Windows DNA* sind der *SQL Server* und der *Internet Information Server*. Für den Bereich der Middleware kommen das *Distributed Component Object Model (DCOM)* als Komponententechnologie und der *Transaction Server* als Laufzeitumgebung zum Einsatz. Die Kommunikation zwischen den Komponenten läuft über *Message Queues*.
Entscheidender Aspekt an *Windows DNA* ist die Tatsache, daß das gesamte System von einem einzelnen Hersteller stammt. Daraus ergibt sich auf den ersten Blick der Vorteil der reibungslosen Zusammenarbeit der einzelnen Teile. Doch wird der Gedanke herstellerübergreifender Modularität und Kompatibilität insofern zunichte gemacht, als daß zumindest für die zugrundeliegende Plattform ausschließlich Microsoft-Produkte eingesetzt werden können. Gerade im Server-Bereich sind jedoch UNIX-Systeme hinsichtlich der Skalierbarkeit von Rechenleistung und sonstigen Ressourcen weit überlegen.
Diesertags arbeitet Microsoft an einem neuen Konzept, der sogenannten *.NET*-Strategie, die auf offenen Standards wie HTTP und XML basiert. Sie umfaßt neben einem Framework für die Applikationsentwicklung und einer Familie von Server-Produkten auch Unterstützung für mobile Geräte wie Mobiltelefone und PDAs sowie für die Einbindung externer Dienstleistungen über das Internet. Ob *.NET* jedoch derart breit und universell nutzbar ist, wie augenblicklich versprochen wird, bleibt abzuwarten.

3.2 OMG Common Object Request Broker Architecture (CORBA)

Die *Object Management Group (OMG)* hat es sich zur Aufgabe gemacht, Standards für den Umgang mit Objekten festzulegen und so ein allgemeines Framework für die Entwicklung von Applikationen zur Verfügung zu stellen, um letztendlich einen Markt für Software-Komponenten zu etablieren. Die bisher wichtigsten Beiträge zur Erreichung dieses Ziels sind die *Common Object Request Broker Architecture (CORBA)* und das *Internet Inter-ORB Protocol (IIOP)*, die portable verteilte Objekte und ihre Interaktion beschreiben. Komponenten, die CORBA/IIOP verwenden, können

mit Komponenten oder Application-Servern anderer Hersteller zusammenarbeiten. Hier liegt der entscheidende Unterschied zu Microsofts *Windows DNA*: CORBA ist lediglich eine Spezifikation, kein konkretes Produkt. Spezifikationsgemäß erstellte Komponenten verschiedener Hersteller können zu einem System zusammengebaut werden; den Unternehmen bietet sich also eine größere Auswahl bei der Abstimmung ihrer betrieblichen Software.
Allerdings legt CORBA ursprünglich keine Standards für komplette Software-Komponenten und ihre Laufzeitumgebungen fest, sondern beschränkt sich auf Objekte und ihre Interaktion. In letzter Zeit hat die OMG daher die Erweiterung *CORBA Components* begonnen, die den Begriff der Komponente für CORBA definiert und auch Aussagen über Application-Server macht. Die Zukunft von *CORBA Components* ist jedoch unklar, da die Notwendigkeit eines solchen Standards angesichts der mittlerweile vorgestellten *Java 2 Enterprise Edition* von Sun Microsystems in Frage gestellt wird.

3.3 Sun Microsystems Java 2 Platform, Enterprise Edition (J2EE)

Nach der Einführung der neuen Programmierplattform Java zeigten sich bald Schwächen beim Einsatz auf Client-Ebene. Abgesehen von der geringen Verarbeitungsgeschwindigkeit ließ die Euphorie über die Möglichkeiten von Applets schnell nach, nachdem Versionsinkompatibilitäten zwischen den verschiedenen *Virtual Machines* der Browser die praktische Nutzung erschwerten. Dafür erkannte Sun das Potential von Java auf Server-Ebene. Im Gegensatz zu den diversifizierten Client-Installationen läßt sich in diesen abgegrenzten Umgebungen sicherstellen, daß die korrekte Laufzeitumgebung zum Einsatz kommt, und die Verarbeitungsgeschwindigkeit ist sekundär, da typischerweise 80% der Prozessorzeit auf Datenbank- oder Netzwerkebene verbraucht werden. Die Plattformunabhängigkeit von Java-Applikationen kommt als Vorteil gegenüber anderen Implementierungssprachen hinzu [3].
Sun begann also mit der Entwicklung verschiedener Einzeltechnologien für die Nutzung in Unternehmen, darunter Verzeichnisdienste, Transaktionsunterstützung und eine erste Version der *Enterprise JavaBeans*. Die einzelnen Produkte waren jedoch nur schlecht aufeinander abgestimmt, ihre genaue Verwendung teilweise unklar und durch Widersprüchlichkeiten der Implementierungen erschwert. Nach entsprechenden Klagen der ersten Nutzer reorganisierte Sun die Strukturen und veröffentlichte mit dem Schritt auf *Java 2* drei verschiedene Plattformen: *Java 2 Micro*, *Standard* und *Enterprise Edition*. Die *Micro Edition* ist dabei im Sprachumfang deutlich eingeschränkt und als Entwicklungsplattform für Java-fähige Geräte wie PDAs oder Pager gedacht. Die *Standard Edition* enthält den normalen Sprachumfang für die Entwicklung von Einzelplatzapplikationen mit grundlegender Netzwerkunterstützung.
Mit *Java 2, Enterprise Edition* liegt eine integrierte Plattform für die Entwicklung portabler Software-Komponenten auf Server-Ebene vor. Zentrales Element sind die *Enterprise JavaBeans*, die die eigentlichen Komponenten für den Einsatz auf Application-Servern darstellen. Dennoch handelt es sich bei J2EE wie bei *CORBA* um eine Spezifikation, nicht um ein Produkt. Obwohl J2EE auch eine Referenzimplementierung des Konzeptes umfaßt, ist Java als Implementierungssprache nicht zwingend vorgeschrieben, ebenfalls möglich sind zum Beispiel *C(++)* oder *Visual Basic*. Jedem Hersteller steht es frei, die Spezifikationen auf seine Weise umzusetzen. Damit kommt J2EE dem Bestreben der OMG nach, einen Markt für Komponenten zu etablieren, geht aber gegenüber CORBA bei der Festlegung von Standards den entscheidenden Schritt weiter, tatsächlich Aussagen über die Schnittstellen und das Verhalten auf Komponentenebene zu treffen, während CORBA lediglich die Kommunikation auf Objektebene definiert.

4. Aufbau der J2EE-Plattform

J2EE stellt im Grunde eine verteilte Application-Server-Umgebung dar, liefert also vor allem Mittel für die Realisierung der Ebene der Geschäftslogik. Hauptbestandteile sind eine Laufzeit-Infrastruktur für den Einsatz und eine Anzahl von APIs für die Entwicklung von Applikationen. Dabei wird nicht vorgeschrieben, wie eine Laufzeitumgebung für Geschäftslogik implementierende Komponenten im einzelnen auszusehen hat, sondern es werden Rollen und Schnittstellen der verschiedenen beteiligten Parteien definiert. Auf diese Weise läßt sich eine Trennung von Applikationen und Laufzeitumgebung erreichen, die es der Laufzeitumgebung ermöglicht, den Applikationen Dienste des zugrundeliegenden Rechnersystems einheitlich und abstrahiert von dessen konkreten Eigenarten zur Verfügung zu stellen. Komponenten können so auf verschiedenen Betriebsplattformen mit gleichen Umgebungsbedingungen rechnen [2]. Außerdem kann die Entwicklung von Applikationen und Laufzeitumgebung entkoppelt werden, so daß auf Applikationsebene in rascher Folge Anpassungen vorgenommen werden können, ohne daß bewährte Basisdienste dadurch beeinflußt werden.

4.1 Ebenen

Unabhängig von der Konzentration auf die mittlere Ebene umfaßt J2EE eine komplette Drei-Ebenen-Architektur. So werden in der Spezifikation *Client*-, *Middle*- und *Enterprise Information System (EIS) Tier* definiert.
Auf Client-Ebene sind primär Web-Browser und Java-Applikationen vorgesehen, deren Aufgabe sich normalerweise auf die Präsentation von Ausgaben und die Entgegennahme von Benutzereingaben beschränkt. Gegebenenfalls kann aber mehr Funktionalität in diese Ebene verlagert werden; auch können Stand-Alone-Programme in einer anderen Sprache als Java implementiert werden.
Die EIS-Ebene bindet DBVS verschiedener Hersteller über ein einheitliches API, *Java Database Connectivity (JDBC)*, an. Dazu müssen die DBVS-Hersteller einen entsprechenden Treiber zur Verfügung stellen. In Zukunft soll dieser Bereich mit dem Konzept des *Connector* erheblich erweitert werden. Ziel ist dabei die Möglichkeit zur Benutzung auch anderer EIS als relationaler Datenbanken über ein einheitliches API.
In der mittleren Ebene wird die Geschäftslogik mit Hilfe von *Enterprise JavaBeans (EJB)* implementiert, die als Komponenten in der Laufzeitumgebung eines EJB-Servers eingesetzt werden. Damit bilden die EJB das Rückgrat des gesamten J2EE-Konzepts.

4.2 Container

Als Laufzeitumgebung kommen auf der mittleren Ebene sogenannte *Container* zum Einsatz. Sie bieten bestimmte Dienste an, die Komponenten auf jeder J2EE-konformen Plattform als vorhanden voraussetzen können. Im Falle von *EJB-Containern* sind dies zum Beispiel Transaktionsunterstützung und Verzeichnisdienste, bei *Web-Containern* Schnittstellen für die Bearbeitung von Client-Anfragen. Alle *Container* stellen darüber hinaus Zugang zu den verschiedenen Kommunikationstechnologien von J2EE bereit. Die Architektur eines *Containers* läßt sich in vier Bereiche einteilen [2].

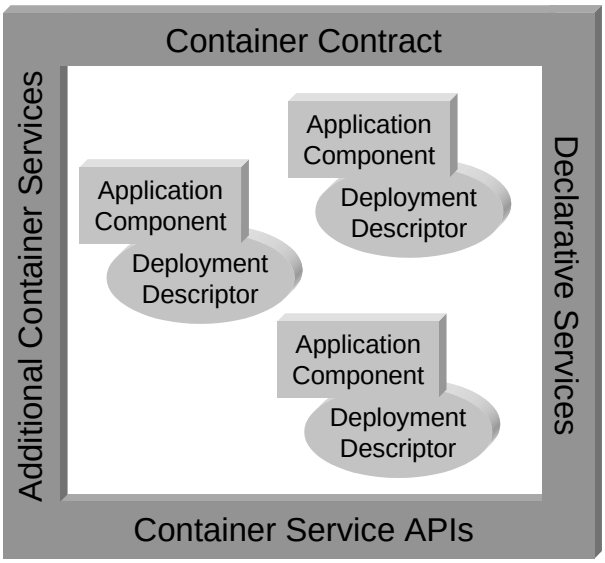

Bild 3: Container-Architektur

Da es als Laufzeitumgebung für Komponenten die Aufgabe des *Containers* ist, in ihm eingesetzte Komponenten zu verwalten, etwa hinsichtlich ihres Lebenszyklus oder ihrer Ressourcenbeanspruchung, müssen diese sich an bestimmte Vorgaben halten, um die Verwaltung durch den *Container* zu ermöglichen. Diese Vorgaben werden im *Component Contract* des *Containers* spezifiziert. In J2EE besteht der *Component Contract* aus *Interfaces* und Klassen, die von einzusetzenden Komponenten implementiert bzw. erweitert werden müssen. Hinzu kommen einige weitere Regeln zur Gestaltung der Komponenten.

Umgekehrt bietet der *Container* den eingesetzten Komponenten über *Container Service APIs* Zugang zu den verschiedenen Technologien und APIs von J2EE. Die Schnittstelle zwischen Komponenten und *Container* ist dabei so gestaltet, daß die Komponenten die externen APIs nutzen können, als ob der *Container* selbst sie implementieren würde. Tatsächlich vermittelt der *Container* die Aufrufe jedoch nur an die zuständigen Dienste. Die Komponenten greifen auf die API-Funktionen über Methoden entsprechender Objekte zu, die der *Container* bereitstellt. Damit können an sich stark heterogene APIs einheitlich angesprochen werden.

Ein ähnliches Konzept verfolgen die *Declarative Services*. Sie ermöglichen es, Komponenten generisch zu implementieren und erst bei der Einsetzung in eine bestimmte Laufzeitumgebung über die Realisierung von Funktionalitäten mit Einbeziehung externer Dienste zu entscheiden. Die Konfiguration geschieht dabei über *Deployment Descriptors*, XML-Dateien mit Angaben über Verhalten und Schnittstellen der einzelnen Komponenten, die zusammen in einem *Container* eingesetzt werden. Die Komponenten übertragen so die konkrete Umsetzung der Funktionen, die mit der Interaktion mit Gegenstellen außerhalb des *Containers* befaßt sind, auf den *Container*, was *als Declarative Invocation* bezeichnet wird. Da jegliche Kommunikation über *Container*-Grenzen hinweg zwingend vom *Container* abgewickelt wird, kann dieser bei der Durchleitung der Nachrichten Zwischenbearbeitungsschritte einfügen und auf diese Weise Dienste für die Komponenten übernehmen, ohne daß diese dafür angepaßt werden müssen. Damit wird es möglich, ein und dieselbe Komponente für verschiedene Umfelder zu konfigurieren und zu nutzen.

Weitere *Container Services* umfassen zum Beispiel die Verwaltung der Lebenszyklen der einzelnen Komponenten, *Resource Pooling* und die Aktualisierung von Verzeichnisdiensten.

4.3 Vorteile

Aus der Betrachtung der Konzepte von J2EE ergeben sich folgende Vorteile mit Blick auf die eingangs formulierten Herausforderungen [1]:

- Vereinfachte Entwicklung

  Durch die Plattformunabhängigkeit von Java kann einmal entwickelte Software auf verschiedenen Rechnertypen und Betriebssystemen eingesetzt werden, wenn eine entsprechende Laufzeitumgebung existiert. Da keine Portierung mehr nötig ist, müssen die Entwickler auch nicht mehr eventuell unterschiedliche Programmiermodelle berücksichtigen. Die Möglichkeiten zur Wiederverwendung fertig implementierten Kodes werden auf diese Weise

ebenfalls erweitert, da sowohl auf Objekt- als auch auf Komponentenebene über alle genutzten Plattformen hinweg einheitliche Bedingungen vorliegen.

Der komponentenbasierte Ansatz von J2EE ermöglicht außerdem eine leichte Modellierung des Systems nach der gewünschten Funktionalität. Da das System nicht monolithisch aufgebaut ist, können einzelne Komponenten leicht und unabhängig voneinander ausgetauscht oder geändert werden. Ein großer Teil der Konfigurationsentscheidungen kann auf den Zeitpunkt des Zusammensetzens der Komponenten zu Modulen oder ihre Einsetzung in Laufzeitumgebungen verschoben werden, da zur Konfiguration nicht in den Kode eingegriffen werden muß.

- Leichte Skalierung

  Die Drei-Ebenen-Architektur von J2EE erlaubt die gezielte Skalierung des Systems an Stellen mit Leistungsengpässen. Durch die Trennung von Applikation und Betriebsplattform mit Hilfe der *Container* können Änderungen und Erweiterungen auf die betroffenen Systemteile begrenzt werden. Weiterhin können *Container* durch den Server-Hersteller derart implementiert werden, daß sie selbständig skalieren, so daß die in ihnen eingesetzten Komponenten sich um potentielle Engpässe nicht zu kümmern brauchen.

- Integration in bestehende *Enterprise Information Systems*

  J2EE bietet bereits jetzt eine Anzahl APIs, die einen einheitlichen Zugriff auf EIS verschiedener Hersteller ermöglichen. Durch die kurz vor der Einführung stehende *Connector*-Architektur wird die Menge der nutzbaren EIS nochmals erweitert.

- Wahl der Produkte

  Die Unternehmen legen sich nicht auf einen Hersteller fest. Serverplattformen und Komponenten können von verschiedenen Herstellern angeboten werden, so daß sich ein Markt für diese Produkte bilden kann. Die Einhaltung des J2EE-Standards garantiert dabei die Kompatibilität und Interoperabilität der verschiedenen Produkte.

  Die Spezifikationen der Komponenten sind außerdem so angelegt, daß einfache Entwicklung und Manipulation durch spezielle Werkzeuge ermöglicht wird. Zusätzlich lassen sich viele Entwicklungsschritte automatisieren. So kann sich auch ein Markt für Entwicklungs- und Administrationswerkzeuge bilden. Unternehmen können auf diese Weise die für ihre Bedürfnisse am besten geeigneten Produkte wählen.

- Einheitliches Sicherheitsmodell

  J2EE erlaubt sogenanntes *Single Signon*, das heißt lediglich einmaliges Anmelden am System selbst bei Zugriffen über die Grenzen verschiedener Systemteile hinweg, um dem Benutzer maximalen Komfort zu bieten. Sicherheitsrichtlinien können von Entwicklern direkt im Kode implementiert werden, um den Zugriff auf kritische Operationen effektiv zu beschränken. Die Rollen- und Gruppenzuweisung von Benutzern erfolgt jedoch zur Einsetzungszeit, also der Konfiguration einer Applikation bei der Zusammenstellung von Komponenten in einem *Container*, was die Flexibilität und Transparenz der Richtlinienumsetzung erhöht.

5. Typische Interaktionsstrukturen

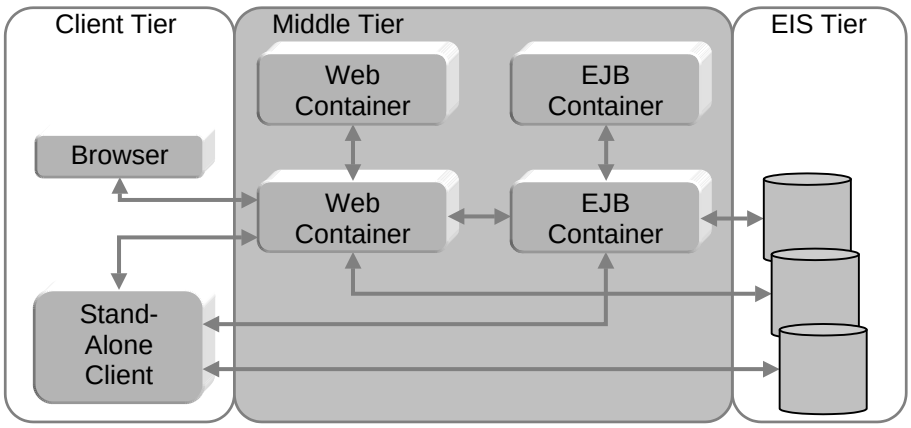

Bild 4: Allgemeiner Aufbau eines J2EE-Systems

Ein J2EE-System ist im allgemeinen wie in Bild 4 ersichtlich aufgebaut: Auf Client-Ebene werden bevorzugt Web-Browser, wo nötig auch *Stand-Alone*-Clients eingesetzt. Auf der mittleren Ebene wird die Geschäftslogik durch Komponenten in *Web-* und *EJB-Containern* realisiert. Auf EIS-Ebene schließlich kommen beispielsweise relationale Datenbanksysteme zum Einsatz. Es zeigt sich, daß die Halbwertszeit der Software mit steigender Entfernung von der EIS-Ebene rasch abnimmt. Aus diesem Grund empfiehlt sich auf Client-Ebene die Verwendung von Web-Browsern anstelle eigenständiger Applikationen, soweit dies möglich ist. Während eine Aktualisierung der *Rendering Engine* der Browser nur selten notwendig sein dürfte, liegt im Falle von *Stand-Alone*-Clients naturgemäß mehr Funktionalität in der Client-Ebene, so daß diese von Änderungen am Gesamtsystem häufiger betroffen sein wird.

In der Praxis kann die Systemarchitektur je nach Anforderung verschiedene Formen annehmen, von denen einige charakteristische im Folgenden erläutert werden [1].

5.1 Mehr-Ebenen-Applikation

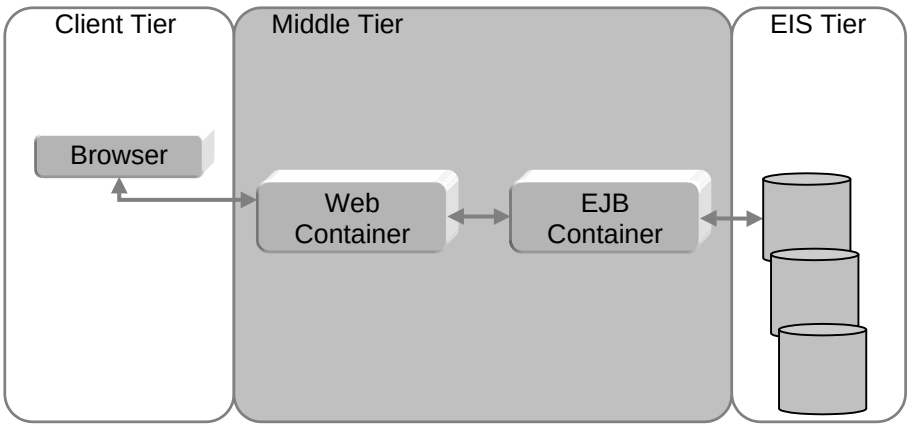

Bild 5: Mehr-Ebenen-Applikation

Diese Systemstruktur stellt sozusagen den Maximalausbau eines J2EE-Systems dar. Der Browser als Client hat lediglich die Aufgabe, die direkte Benutzerinteraktion abzuwickeln. Dafür kommuniziert er, bevorzugt per normalem HTTP, mit dem *Web-Container* in der mittleren Ebene. Die Web-Komponenten im *Web-Container* realisieren ausschließlich die internen Teile der Präsentationslogik. Sie nehmen *HTTP-Requests* des Browsers entgegen und beantworten diese mit dynamisch erstellten HTML-Seiten. Die eigentliche Geschäftslogik steckt in *Enterprise JavaBeans*, die *im EJB-Container* ablaufen. Sie steuern das gesamte System und interagieren dabei mit dem *Web-Container* einerseits und der Datenbank andererseits. So werden Datenzugriff und Interaktion mit dem Benutzer weitestgehend entkoppelt. Mehr-Ebenen-Applikationen weisen maximale Mächtigkeit und Skalierbarkeit auf, sind jedoch entsprechend aufwendig umzusetzen.

5.2 Stand-Alone-Client

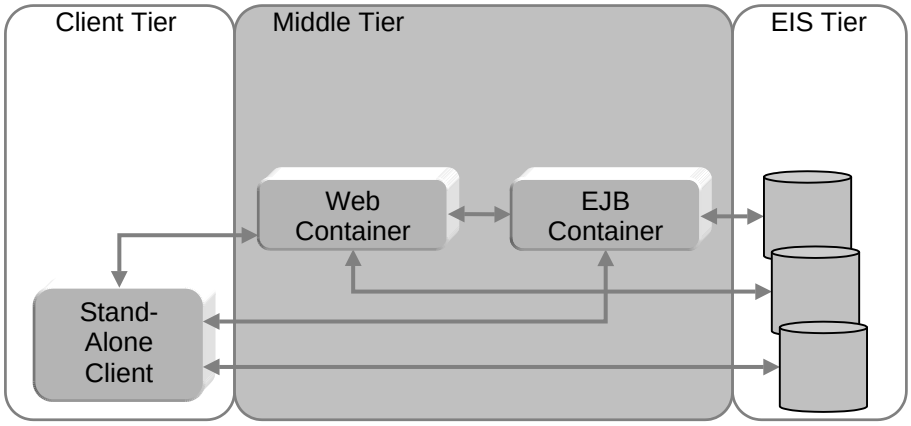

Bild 6: Stand-Alone-Client

Wird auf Client-Ebene eine größere Funktionalität benötigt, kommen meist *Stand-Alone*-Applikationen zum Einsatz. Hieraus ergeben sich mehrere mögliche Systemstrukturen.

Der Client kann unter Umgehung von Web-Komponenten direkt mit dem *EJB-Container* kommunizieren, normalerweise mit Hilfe von *Remote Method Invocation (RMI)*. In diesem Fall regelt der Client die Präsentationslogik selbständig, die Geschäftslogik verbleibt jedoch auf der mittleren Ebene.

Analog kann auch der *EJB-Container* umgangen werden. Dies ist der Fall, wenn der Client in der Lage ist, dynamische Web-Inhalte, meist in Form von XML, zu verarbeiten. Der *Web-Container* bietet dem Client dann Zugang zu diesen Inhalten, die auf der EIS-Ebene gelagert werden. Auch bei dieser Variante trägt die Client-Ebene die Präsentationslogik. Die Geschäftslogik kann je nach Komplexität im *Web-Container* oder auf der EIS-Ebene umgesetzt werden.

Schließlich kann der Client auch direkt mit den Datenbanken der EIS-Ebene kommunizieren. Hierbei werden Präsentations- und Geschäftslogik, also die gesamte Funktionalität der mittleren Ebene, auf Client-Ebene untergebracht. Damit entspricht diese Architektur dem klassischen 2-Ebenen-Client-Server-Modell mit allen daraus resultierenden Nachteilen, die in Abschnitt 2.2 erläutert wurden.

5.3 Web-basierte Applikation

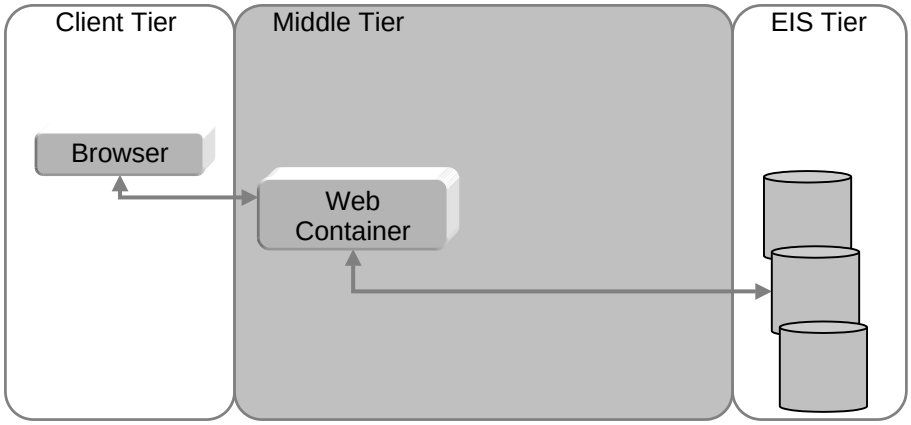

Bild 7: Web-basierte Applikation

In vielen Fällen mag der Aufwand, der durch den Einsatz eines EJB-Servers entsteht, im Verhältnis zur gegebenen Problemstellung übertrieben sein. Die J2EE-Spezifikation schreibt nicht vor, daß alle potentiell gegebenen Möglichkeiten auch ausgeschöpft werden müssen. Für überschaubare Aufgabenbereiche sind Drei-Ebenen-Web-Applikationen weit verbreitet. Der *Web-Container* übernimmt dabei neben der Präsentations- auch die Geschäftslogik. Auf der Seite des Clients bleibt es bei den aus dem Maximalausbau bekannten Verfahren, der *Web-Container* kommuniziert aber jetzt direkt mit der EIS-Ebene. Sollte die Arbeitslast im Laufe der Zeit anwachsen, läßt sich eine solche Architektur leicht um einen EJB-Server ergänzen. Die mit dem Datenzugriff betrauten Komponenten

werden dann in den *EJB-Container* migriert, was bei sinnvoller Modellierung mit nur minimalem Aufwand verbunden sein sollte.

5.4 Interaktion zwischen Containern

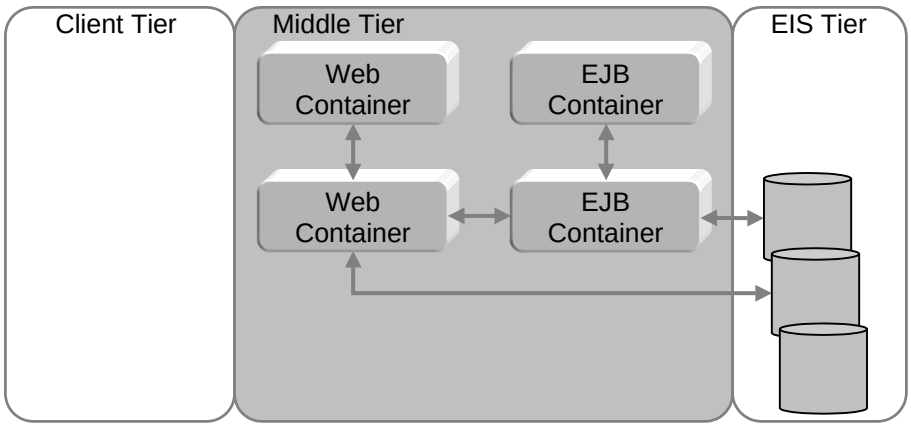

Bild 8: Interaktion zwischen Containern

Für die Kommunikation zwischen *Web-Containern* sieht J2EE primär den Austausch von XML-Nachrichten über HTTP vor. Dieses Verfahren entspricht der in diesen Bereichen typischen losen Kopplung. Die Bindungen zwischen *EJB-Containern* hingegen sind enger; hier kommt zur Zeit vor allem RMI zum Einsatz. In Zukunft wird es jedoch mit zunehmender Verbreitung des *Java Message Service (JMS)* möglich werden, auch *EJB-Container* stärker voneinander zu entkoppeln.

6. Technologien und APIs

6.1 Komponenten-Technologien

Komponenten sind die kleinsten Software-Einheiten in der J2EE-Spezifikation. Eine Komponente ist nicht allein lauffähig, sondern nur für Teilaufgaben zuständig. Mehrere Komponenten werden zu einem Modul zusammengefaßt und in einen *Container* eingesetzt. Die Zusammenarbeit mehrerer Module schließlich ergibt eine vollständige Applikation [2].
In den Komponenten wird somit die eigentliche Funktionalität eines Systems realisiert. Im Gegensatz zu den unterstützenden Technologien, darunter vor allem die von Application-Servern bereitgestellten Laufzeitumgebungen, die in der Regel als fertiges Produkt eingekauft werden, werden Komponenten oft im Unternehmen selbst oder gegebenenfalls von Dienstleistern nach Auftrag entwickelt. J2EE spezifiziert drei Haupttypen von Komponenten: *Java Server Pages (JSPs), Servlets und Enterprise JavaBeans*.
Die Aufgabe von JSPs und *Servlets* ist die Verarbeitung von *HTTP-Requests*. Diese werden in der Regel vom Benutzer ausgelöst, so daß den Komponenten auch die Präsentationslogik zufällt. Der Unterschied zwischen den beiden Komponententypen liegt darin, daß *Servlets* als aktive Komponenten agieren, während JSPs vorgefertigte Webseiten sind, die um dynamischen Inhalt ergänzt werden. Dementsprechend müssen *Servlet-Container* neben der üblichen Umgebung vor allem Netzwerkdienste zur Verfügung stellen; *JSP-Container* müssen zusätzlich noch einen Interpreter für die Skript-Anweisungen in den Webseiten besitzen [2].
Enterprise JavaBeans sind eine Technologie für die Entwicklung und den Einsatz von Geschäftslogikkomponenten auf Server-Ebene. Sogenannte *Session Beans* existieren normalerweise nur für die Dauer einer Client-Server-Verbindung und führen verschiedene Dienste für den Client aus. *Entity Beans* hingegen repräsentieren Daten der EIS-Ebene und

existieren typischerweise so lange wie die ihnen zugeordneten Daten selbst. *EJB-Container* bieten neben den Standarddiensten vor allem Unterstützung für Transaktionslogik und Persistenzverwaltung [2].

6.2 Service-Technologien

Die Service-Technologien von J2EE ermöglichen den Komponenten den Zugriff auf verschiedene externe Dienste. Dabei vereinheitlichen sie gegenüber der Applikation die heterogenen Schnittstellen zugrundeliegender Dienste des Rechnersystems.

Java DataBase Connectivity (JDBC) bindet relationale Datenbanken an. Dies umfaßt den Verbindungsauf- und abbau, Authentifizierung, Transaktionsverwaltung, Vorabüberprüfung von SQL-Statements, Ausführung von *Stored Procedures* und Auswertung von Anfragen. Dabei baut die JDBC-Schnittstelle auf einen vom DBVS-Hersteller zu liefernden Adaptertreiber für die jeweils eingesetzte Datenbank auf [2].

Mit der kurz vor der Veröffentlichung stehenden *Connector*-Architektur soll zusätzlich die Anbindung anderer *Enterprise Information Systems* möglich werden, darunter zum Beispiel *Enterprise Resource Planning (ERP)*- oder *Mainframe*-Systeme. Auch hier kommt ein Adaptertreiber des jeweiligen Herstellers zum Einsatz, um eine einheitliche Schnittstelle zu ermöglichen [1].

Das *Java Transaction API (JTA)* stellt Applikationen eine implementierungsunabhängige Unterstützung für Transaktionen zur Verfügung. Dazu werden standardisierte Java-*Interfaces* zwischen einem Transaktions-Manager und den an den Transaktionen teilnehmenden Parteien spezifiziert. Mit dem *Java Transaction Service (JTS)* existiert ein JTA-konformer Transaktions-Manager, der die OMG-Spezifikation bezüglich Objekt-Transaktionsdiensten umsetzt und von *Enterprise JavaBeans* für die Transaktionsverwaltung verwendet wird. Ein solcher Transaktions-Manager unterstützt die Isolation und Synchronisation nebenläufiger Transaktionen und verwaltet darüber hinaus die benötigten Ressourcen. In J2EE werden Transaktionen üblicherweise von *Containern* kontrolliert, so daß bei der Entwicklung von Komponenten in dieser Hinsicht kein großer Aufwand betrieben werden muß [2].

Über das *Java Naming and Directory Interface (JNDI)* können Namen- und Verzeichnisdienste genutzt werden. Es stellt dazu Methoden für typische Verzeichnisoperationen wie die Suche nach Objekten oder anderen Ressourcen und die Assoziation von Attributen zur Verfügung. *Enterprise JavaBeans* verwenden JNDI für komponentenübergreifende Zugriffe. Ähnlich wie JTA ist JNDI implementierungsunabhängig, so daß diverse populäre Verzeichnisdienste wie LDAP, NDS, DNS und NIS angesprochen werden können. Dies erleichtert die Integration in bestehende Systeme [3].

6.3 Kommunikations-Technologien

Über die Kommunikations-Technologien kommunizieren Komponenten- und Service-Technologien miteinander. Neben der Verwendung von TCP/IP und HTTP für die Kommunikation mit Web-Browsern kommen hauptsächlich drei Technologien zum Einsatz.

Remote Method Invocation (RMI) erlaubt es, Methoden entfernter Objekte so aufzurufen, als ob sie lokal verfügbar wären. Dazu werden lokal *Interfaces* definiert, die die entfernten Objekte beschreiben. J2EE bedient sich dabei des *Internet Inter-ORB Protocol*, so daß auch Objekte angesprochen werden können, die nicht in Java implementiert sind, da die in Java spezifizierten *Remote Interfaces* gegebenenfalls in die allgemeine *Interface Definition Language* konvertiert und anschließend die zugehörigen Objekte in einer beliebigen CORBA-konformen Sprache implementiert werden können [2].

Der *Java Message Service (JMS)* bietet ein API für die Nutzung sogenannter *Message-Oriented Middleware*. Auf diese Weise können Komponenten Informationen in Form von wohldefinierten Nachrichten austauschen. Der Nachrichtenaustausch geschieht zuverlässig und asynchron, was gegenüber synchroner Kommunikation bessere Skalierbarkeit bedeutet. Da Komponenten nicht durch die Nutzung des Kommunikationsdienstes blockiert werden, sondern die Abarbeitung ihrer Aufgaben zunächst fortsetzen können, fallen Verzögerungen auf den Nachrichtenwegen nicht so stark ins Gewicht. JMS unterstützt sowohl *Point-to-Point* als auch *Publish-Subscribe Messaging*. Beim *Point-to-Point Messaging* senden Komponenten ihre Nachrichten direkt an die Eingangswarteschlange des Empfängers. Beim *Publish-Subscribe Messaging* existieren zentrale Verteilerknoten, die jeweils für Nachrichten eines Typs zuständig sind. Komponenten registrieren sich bei Verteilern, deren Nachrichten für sie von Belang sind, und senden eigene Nachrichten an den jeweils zuständigen Verteiler. So sind Sender und Empfänger nicht in direktem Kontakt. Die Verteilerknoten verwalten die Nachrichtenweiterleitung und können gegebenenfalls auch Nachrichten puffern, falls registrierte Empfänger gerade inaktiv sind [3].
JavaMail schließlich bietet über verschiedene Klassen und Interfaces Zugang zu den wichtigen Protokollen für den Versand und Empfang von Emails. Damit wird wie bei JMS ein asynchroner Nachrichtenaustausch ermöglicht. *JavaMail* ist jedoch primär für die Verarbeitung von Benutzernachrichten bestimmt. Im Vergleich zu JMS ist es langsamer und weniger zuverlässig, so daß die Verwendung für systeminterne Kommunikation sich nicht empfiehlt [3].

7. Client-Typen

Der Eindruck, den die Benutzer vom gesamtem verteilten System gewinnen, hängt entscheidend von der Qualität der Client-Ebene ab. Die J2EE-Plattform unterstützt viele Client-Typen, vom simplen Browser bis zur komplexen *Stand-Alone*-Applikation mit direktem Datenbank-Zugriff. Die Wahl des jeweils angemessenen Clients ist essentiell für die Akzeptanz durch die Benutzer und damit den erfolgreichen Einsatz des Systems. Es gilt, den richtigen Trade-Off bei der Verteilung der Zuständigkeiten zwischen Client und Server zu finden. Die vom Benutzer wahrgenommene Qualität steigt mit zunehmender Funktionalität auf Client-Ebene. Wartung und Verwaltung fallen hingegen mit größerem Anteil der Funktionalität auf Server-Ebene leichter [1].

7.1 Aspekte für die Wahl der Clients

Der wichtigste Gesichtspunkt bei der Wahl des Client-Typs ist die Betriebsumgebung des Clients. Soll der Zugriff auf das System über das Internet erfolgen, kann nur von minimaler Qualität des Netzwerks und Leistungsfähigkeit der Laufzeitplattform ausgegangen werden. Hinzu kommt eine hohe Diversität der verwendeten Zugänge und Benutzerarbeitsplätze. So wird man sich hier auf den kleinsten gemeinsamen Nenner beschränken müssen. Niedrige Bandbreite beschränkt die Interaktion des Clients mit dem Server und die Interaktivität der Bedienung. Daher müssen gegebenenfalls Teile der Präsentationslogik auf die Client-Ebene verschoben werden; zusammen mit einer ausgeklügelten Pufferung der Netzwerkkommunikation läßt sich so noch am ehesten komfortables Arbeiten ermöglichen [1].
Im Intranet eines Unternehmens herrschen ganz andere Bedingungen. Die Netzwerkbandbreite ist höher, die Latenzzeiten sind geringer. Weiterhin können gewisse Standards hinsichtlich der Benutzerausrüstung durchgesetzt werden. Es behindern normalerweise keine *Firewalls* die Kommunikation zwischen Client und Server, und vertrauliche Daten

fließen nicht über öffentliche Leitungen. In einer solchen Umgebung sind damit wesentlich weniger Einschränkungen bei der Wahl des richtigen Clients zu berücksichtigen [1].
Der nächste zentrale Punkt ist die Verteilung der Client-Software. Wie oft und auf welche Weise müssen Änderungen durchgeführt und neue Versionen auf die Arbeitsplätze verteilt werden? Hier eröffnet sich ein Gegensatz zu der Idee, über das Internet angebundene Clients mit mehr Eigenfunktionalität auszustatten und damit größer und komplexer werden zu lassen, da für die dann entsprechend häufiger nötigen Aktualisierungen höchstwahrscheinlich auch kein besserer Kanal bereitsteht, als dies für die alltägliche Arbeit der Fall ist. Vom Standpunkt der Verwaltung drängen sich statt dessen wieder möglichst simple Clients auf, idealerweise Web-Browser [1].
Schließlich muß die eventuelle Heterogenität der Arbeitsplätze beachtet werden. Müssen verschiedene Plattformen mit dem Client beschickt werden, bietet sich eine plattformunabhängige Implementierung an. In erster Linie sind dann browser- oder javabasierte Clients zu wählen. In diesem Zusammenhang stellt sich auch die Frage, wie hoch die Leistungsfähigkeit der Zielplattform ist. So kann man leistungsfähigen Arbeitsplatzrechnern zumindest Teilaufgaben wie aufwendige Grafikberechnungen übertragen, ohne direkt die ganze Mehrebenen-Architektur des Gesamtsystems zu unterwandern [1].

7.2 Web-Clients

Web-Clients laufen normalerweise in einem Browser. Die Benutzerschnittstelle wird serverseitig generiert und durch die *Rendering Engine* des Browsers dargestellt. Das zur Übertragung verwandte HTTP ist weit verbreitet, robust und einfach und kann die meisten *Firewalls* passieren. Als nachteilig erweist sich jedoch das Fehlen von Protokollzuständen und Transaktionsunterstützung. Zustände müssen umständlich über *Cookies* simuliert werden. Weiterhin ist HTML als Format für übertragene Inhalte zwar ebenfalls allgemein akzeptiert, in seiner Darstellungsmächtigkeit jedoch begrenzt.
Ein einfacher Web-Browser kann damit nur bei verhältnismäßig überschaubaren Anforderungen als Benutzerschnittstelle genutzt werden. Dennoch ist in der Praxis dieses Modell das am weitesten verbreitete und bei Zugang zum System über das Internet meist Client der Wahl. Da aufgrund des simplen Clients alle relevanten Daten und Funktionalitäten auf der Server-Ebene angesiedelt sind, ist auch sogenannter *Roaming Access* möglich, das heißt, der Benutzer kann von beliebiger Stelle aus auf das System zugreifen und findet immer die gleichen Bedingungen vor [1].
Sind komplexere Inhalte darzustellen und mehr Funktionalität auf Client-Ebene nötig, können *Applets* genutzt werden. Sie laufen ebenfalls innerhalb von Web-Browsern und sind damit ähnlich portabel einzusetzen wie die Browser selbst, bieten aber komfortablere Benutzerschnittstellen und sind in der Lage, komplexere Arbeitsschritte selbst durchzuführen. In der Praxis treten allerdings Schwierigkeiten beim Einsatz auf. So müssen die Applets zu Beginn jeder Arbeitssitzung neu vom Server angefordert werden, und oftmals sind die Laufzeitumgebungen der Browser veraltet und inkompatibel. Diesen Problemen hat Sun Microsystems das *Java Plug-In* entgegengesetzt, das an die Stelle der browsereigenen Laufzeitumgebungen tritt. Damit werden Inkompatibilitäten verhindert, und neuere Versionen des *Java Plug-Ins* sind in der Lage, *Applets* auf dem Arbeitsplatzrechner zu speichern, um die Anzahl der Neuanforderungen zu reduzieren. Doch gehen mit Nutzung dieser Vorteile wieder Flexibilität und Portabilität verloren, da nun die Installation des *Plug-Ins* auf dem Client-Rechner vorausgesetzt wird.
Ähnlich verhält es sich mit anderen *Plug-Ins*, die die Funktionalität des Browsers steigern könnten. Sie sind in der Regel nicht plattformunabhängig und müssen einzeln installiert und verwaltet werden. Außerdem steigt mit der Anzahl der auf Client-Ebene involvierten Komponenten das Sicherheitsrisiko für den Betrieb des Gesamtsystems.

Standalone-Web-Clients invertieren das Browser-Prinzip, indem sie Browser-Funktionalität in eigenständig lauffähige Programme auf Client-Ebene einbinden. Auch auf diese Weise lassen sich komplexere Benutzerschnittstellen realisieren und mehr Aufgaben auf Client-Ebene bearbeiten, doch müssen diese *Stand-Alone*-Clients aufwendig für jede Plattform entwickelt und anschließend auf jedem Arbeitsplatzrechner installiert werden. Damit sind sie für den Einsatz in stark verteilten Umgebungen eigentlich ungeeignet [1].
Jede Art von Web-Clients besitzt also ihre spezifischen Vor- und Nachteile. Die Entscheidung zugunsten einer Variante hängt stark von der jeweiligen Situation ab. Generell ist die Anbindung von Clients über das Internet immer mit Kompromissen verbunden, während Web-Clients im lokalen Netzwerk durchaus eine interessante Möglichkeit darstellen.

7.3 Enterprise JavaBeans-Clients

EJB-Clients sind Applikationen auf Client-Ebene, die direkt mit *Enterprise JavaBeans* auf der mittleren Ebene interagieren. Sie verwalten ihre Benutzerschnittstelle selbst und sind aus Sicht des Benutzers nativen Applikationen sehr ähnlich. Dennoch erfolgt die zentrale Steuerung durch die Geschäftslogik in der mittleren Ebene. Die Kommunikation zwischen Client und mittlerer Ebene erfolgt über RMI; dadurch stehen dem Client auch andere Dienste der mittleren Ebene, wie JNDI, JMS oder JDBC, zur Verfügung.
Da es sich bei EJB-Clients nicht um vollwertige native Applikationen des Client-Rechners handelt, benötigen sie einen *Container* als Laufzeitumgebung, der auch die Verbindungen zur mittleren Ebene verwaltet. Im Vergleich zu den bekannten J2EE-*Containern* zeigen sich *Container* für EJB-Clients allerdings deutlich leichtgewichtiger, da sie weniger Funktionalitäten bieten müssen [1].
Mit EJB-Clients können leistungsfähige und flexible Benutzerschnittstellen realisiert werden. Sie können mit komplexen Datenmodellen umgehen, die die Möglichkeiten von Web-Clients jeder Art übersteigen. Auch hinsichtlich Transaktionsunterstützung und der Fähigkeit, Teile der Arbeitslast zu tragen, sind sie den Web-Clients überlegen. Jedoch müssen EJB-Clients explizit auf den Arbeitsplatzrechnern installiert werden und sind allgemein komplexer im Aufbau. *Firewalls* stellen meist ein Problem dar, da selten die Möglichkeit besteht, RMI-Kommunikation passieren zu lassen. Aufgrund dieser Überlegungen eignen sich EJB-Clients vor allem für den Einsatz im lokalen Netzwerk. Durch entsprechende Unternehmensstandards läßt sich in diesem Umfeld der Aufwand für Verteilung und Installation auf erträglichem Niveau halten.

7.4 Enterprise Information System-Clients

EIS-Clients greifen unmittelbar auf die Datenbanken zu und verwalten sowohl ihre Präsentations- als auch die Geschäftslogik eigenständig. Diese Art von Client sollte mit Vorsicht eingesetzt werden. Im allgemeinen empfiehlt sich ein derart direkter Zugang zu den sensiblen Unternehmensdaten schon aus Sicherheitsgründen nicht für große Benutzerkreise. Außerdem sind Clients mit solch mächtiger Eigenfunktionalität normalerweise recht komplex und müssen komplett auf den Arbeitsplatzrechnern installiert werden, was letztendlich eine Rückkehr zur klassischen Zwei-Ebenen-Architektur mit allen daraus resultierenden Problemen bedeutet (siehe dazu Abschnitt 2.2). Die Verwendung von EIS-Clients sollte daher auf die Administration des Systems durch Fachleute beschränkt werden. Einsatzgebiete sind zum Beispiel automatisierte Wartung der Datenbanken oder gelegentliche manuelle Eingriffe in das System [1].

7.5 Auswirkungen auf die Server-Ebene

In der Regel wird man sich in großen Unternehmen nicht auf einen Client-Typ beschränken können, da die Umgebungsbedingungen zu heterogen sind. Die aus der Nutzung verschiedener Client-Typen entstehenden Konsequenzen für die Server-Ebene dürfen dann nicht übersehen werden. Software-Komponenten auf EIS- und Middleware-Ebene sollten von vornherein so gestaltet werden, daß die Anbindung unterschiedlicher Client-Typen mit minimalem Aufwand zu realisieren ist [1]. Eine saubere System-Modellierung minimiert außerdem die unnötige Duplizierung von Kode, da generisch realisierte Funktionalitäten von verschiedenen Client-Typen genutzt werden können. So ist zum Beispiel die Geschäftslogik normalerweise für alle Operationen bindend, und auch die Präsentationslogik dürfte bei geschickter Partitionierung zumindest teilweise wiederverwendbar sein. Eine gute Systemarchitektur umfaßt daher nicht nur die Aufteilung der Arbeitslast auf die drei Ebenen, sondern auch ebeneninterne Entwurfsentscheidungen.

8. Zusammenfassung

Bei Betrachtung der heutigen Marktwirtschaft und den daraus folgenden Anforderungen an betriebliche Software zeigt sich, daß kein Weg an mehrschichtigen Systemarchitekturen vorbeiführt. Von den verschiedenen Realisierungsvarianten mehrschichtiger Systeme erweist sich die Drei-Ebenen-Architektur als die leistungsfähigste.
Mit J2EE legt Sun Microsystems ein Framework für die Entwicklung serverseitiger Komponenten vor, das nicht an bestimmte Produkte gebunden ist. Im Gegensatz zu CORBA werden nicht nur auf Objekt-, sondern auch auf Komponentenebene Spezifikationen festgelegt.
J2EE ermöglicht eine Vielzahl von Systemstrukturen und erlaubt den Einsatz verschiedenster Client-Typen, wobei Drei-Ebenen-Architekturen im Vordergrund stehen. Über einen Anzahl von APIs und das *Container*-Konzept werden die Applikationen von den verwendeten Rechnersystemen entkoppelt, so daß sie plattformübergreifend einsetzbar sind.
Dennoch bietet auch ein augenscheinlich gelungenes Framework wie J2EE keine Garantie für perfekte verteilte Applikationen. Die nicht von der Hand zu weisende Komplexität mehrschichtiger Systeme muß von Applikationsentwicklern beherrscht werden, wenn die potentiellen Vorteile nicht in ungeschickter Modellierung untergehen sollen.
Weiterhin ist die Akzeptanz von J2EE im realen Einsatz noch begrenzt. Es bleibt abzuwarten, ob sich Suns Vorstoß auf breiter Front durchsetzen kann

Literaturverzeichnis

[1] N. Kassem: Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition (Blueprints), ISBN 0-20-1702770, http://java.sun.com/j2ee/blueprints/aboutthebook.html

[2] S. Allamaraju et. al.: Professional Java Server Programming J2EE Edition, Wrox Press Ltd., Birmingham, UK, 2000

[3] E. Roman: Mastering Enterprise Java Beans, and the Java 2 Platform, Enterprise Edition, Wiley Computer Publishing, New York, 1999