Redesign eines Brokerage-Systems als Prototyp auf der Basis von Enterprise JavaBeans 2.0

DIPLOMARBEIT

(mit drei Monaten Bearbeitungsdauer)

zur Erlangung des Grades eines Diplom-Wirtschaftsinformatikers am Fachbereich Wirtschaft der Fachhochschule Nordostniedersachsen

eingereicht von:

Kai Donker

10. Fachsemester Wirtschaftsinformatik

Matrikelnummer: 131076

Luhdorfer Str. 123 a

21423 Winsen (Luhe)

Tel.: (04171) 780407

Prüfer

Prof. Dr. rer. nat. Jürgen Jacobs

Winsen, den _____

Vorwort

Die vorliegende Diplomarbeit entstand im Rahmen meines Praxissemesters bei der Netlife Internet Consulting und Software GmbH, Hamburg.

Die Diplomarbeit enthält somit den Praxisbericht.

Winsen, 25.04.2002

Inhaltsverzeichnis

Abkürzungsverzeichnis

BMP	*Bean Managed Persistence*
CMP	*Container Managed Persistence*
CMR	*Container Managed Relationships*
CORBA	*Common Object Request Broker Architecture*
DMZ	*demilitarisierte Zone*
EJB QL	*EJB Query Language*
EIS	*Enterprise Informationssystemen*
EJB	*Enterprise JavaBeans*
ERP	*Enterprise Resource Planning*
HTTPS	*Hypertext Transfer Protocol Secure*
IIOP	*Internet Inter-ORB Protocol*
ISIN	*International Securities Identification Number*
J2EE	*Java 2 Enterprise Edition*
JAAS	*Java Authentication and Authorization Service*
JAF	*Java Activation Framework*
Java IDL	*Java Interface Definition Language*
JAXP	*Java API for XML Processing*
JDBC	*Java Database Connectivity*
JNDI	*Java Naming and Directory Interface*
JSP	*JavaServer Pages*
JTA	*Java Transaction API*
JTS	*Java Transaction Service*
JMS	*Java Messaging Service*
MVC	*Model-View-Controller*
RMI	*Java Remote Method Invokation*
VM	*Virual Machine*
WKN	*Wertpapierkennnummer*
WTLS	*Wireless Transport Layer Security*
Xetra	*eXchange Electronic Trading*
XML	*eXtended Mark-up Language*

Darstellungsverzeichnis

1 Ziele der Diplomarbeit

1.1 Aufgabenstellung

Diese Arbeit ist im Rahmen der Konsolidierung verschiedener Systeme für den Wertpapierhandel (Brokerage-Systeme) entstanden. Bei der Entwicklung der bestehenden Brokerage-Systeme wurden Kompromisse im Design eingegangen. Es werden große Teile der Geschäftslogik durch Stored Procedures innerhalb der Datenbank ausgeführt. Dadurch ist es möglich, kurze Antwortzeiten mit wenig Datenaustausch zwischen Datenbank und Anwendungsserver zu erreichen. Mit diesem Ansatz sind jedoch Nachteile verbunden, so dass ein Redesign angestrebt wird. Zum einen sollte die Geschäftslogik von der Datenhaltung getrennt sein, und zum anderen ist auf diese Weise eine sehr hohe Abhängigkeit vom Datenbankhersteller entstanden.

In dieser Arbeit wird ein Redesign für einen zentralen Geschäftsvorfall eines Brokerage-Systems vorgenommen. Dabei wird angestrebt, möglichst große Teile der Geschäftslogik aus der Datenbank zu entfernen. Es wird ein Prototyp implementiert, um einen Performancevergleich mit dem bestehenden System durchzuführen. Aufgrund der Ergebnisse erfolgt eine Bewertung des redesignten Systems.

1.2 Abgrenzung

Bei der Entwicklung des Systems wird der aktuelle Stand der Technik berücksichtigt. Die Ergebnisse erlauben es, die Möglichkeiten im untersuchten Kontext zu beurteilen. Es erfolgt jedoch keine allgemeingültige Bewertung der zum Einsatz kommenden Technologie Enterprise JavaBeans 2.0 sowie des verwendeten Application Servers BEA WebLogic 6.1.

1.3 Aufbau

Der erste Teil bietet Einführungen in die technischen und fachlichen Aspekte der Arbeit. Zunächst werden die wesentlichen Merkmale der Technologie Enterprise JavaBeans dargestellt. Die Unterschiede zwischen den verschiedenen Versionen werden aufgezeigt. Im Anschluss daran wird kurz der Application Server BEA WebLogic 6.1 vorgestellt. Hier wird auf besondere Fähigkeiten

eingegangen, die nicht Bestandteil der Enterprise JavaBeans Spezifikation sind. Der Leser wird in die Lage versetzt, die fachlichen Anforderungen an das System nachvollziehen zu können, indem Grundlagen des Wertpapierhandels vermittelt werden.

Das Brokerage-System auf dem die Arbeit aufsetzt, wird im zweiten Teil vorgestellt. Das Gesamtsystem wird in einem Überblick veranschaulicht, der Geschäftsvorfall „Order aufgeben“ detailliert dargestellt.

Im dritten Teil werden Entwurf und Implementierung des Prototypen dargestellt und es wird ein Performancevergleich durchgeführt. Die Arbeit endet mit einer abschließenden Beurteilung.

2 Einführung in Enterprise JavaBeans

Enterprise JavaBeans (EJB) ist Bestandteil der Java 2 Plattform Enterprise Edition (J2EE). Bei J2EE handelt es sich um eine Technologie, die einen komponentenorientierten Ansatz zur Entwicklung von serverseitigen Unternehmensanwendungen bietet. Sie ist von Sun Microsystems entwickelt worden. Derzeit ist die Version 1.3 gültig. EJB's sind serverseitige Komponenten, welche die Form von modularen, strukturierten und lose gekoppelten Softwarebausteinen besitzen. Sie verfügen über eine bestimmte Funktionalität oder bieten einen Dienst an. EJB's machen einen wesentlichen Teil von J2EE aus. Weitere Technologien von J2EE sind:

- Java Naming and Directory Interface (JNDI), ein Namens- und Verzeichnisdienst, der den Zugriff auf registrierte Komponenten ermöglicht
- Java Remote Method Invokation (RMI) ermöglicht zusammen mit Java IDL[1] (Java Interface Definition Language) entfernte Methodenaufrufe
- Java Database Connectivity (JDBC) ist eine Schnittstelle für den Datenbankzugriff
- Java Transaction API (JTA) und Java Transaction Service (JTS) bieten Transaktionskontrolle
- Java Servlet und JavaServer Pages (JSP) sind Netzwerkkomponenten, die anfrage- / antwortorientiert über HTTP mit Clients kommunizieren
- Java Messaging Service (JMS) ermöglicht asynchrone Kommunikation in verteilten Systemen
- JavaMail und Java Activation Framework (JAF) stellen Email-Funktionen bereit
- Java API for XML Processing (JAXP) dient zur Verarbeitung von XML-Dokumenten
- J2EE Connector API dient zur Kommunikation mit Enterprise Informationssystemen (EIS) wie Mainframe- oder Enterprise Resource Planning-Systemen (ERP)

[1] Java IDL ist eine Implementierung der CORBA- Spezifikation (Common Object Request Broker Architecture) in Java. CORBA erlaubt Objekten einer heterogenen Umgebung miteinander zu kommunizieren.

- Java Authentication and Authorization Service (JAAS) bietet Sicherheit auf Benutzerebene

Die J2EE Plattform bietet die Möglichkeit, auf diesen Technologien basierend mehrschichtige verteilte Anwendungen zu entwickeln. Bei diesem Prozess können wiederverwendbare Softwarekomponenten entstehen.

2.1 Architektur von J2EE

Auf Softwareebene lässt sich eine Anwendung in die Präsentations-, die Anwendungs- und die Datenschicht unterteilen. Mit dieser Aufteilung wird das Ziel verfolgt, die Schichten voneinander zu isolieren und deren Implementierungen unabhängig voneinander ersetzen zu können. Physikalisch werden diese Schichten in eine zwei- oder mehrschichtige Architektur aufgeteilt. Eine Schicht kann beispielsweise durch einen Prozess oder eine Maschine begrenzt sein.

Zweischichtige Architekturen erfordern, dass sich in einer physischen Schicht zwei Softwareschichten befinden. Dabei kann entweder ein Fat-Client oder ein Thin-Client entstehen. Ein Fat-Client führt Geschäftslogik aus, so dass der Server ausschließlich der zentralen Datenhaltung dient. Werden große Teile der Geschäftslogik auf die Serverseite verlagert, so dass u. U. nur die Präsentationsschicht auf dem Client verbleibt, spricht man von einem Thin-Client.

Beide Ansätze sind mit Nachteilen behaftet. So kommt es auf Softwareebene zu einer Vermischung der Schichten, beispielsweise indem Stored Procedures Geschäftslogik innerhalb der Datenbank ausführen. Auf der physischen Ebene entsteht eine hohe Abhängigkeit von der Datenbank, da die Clients direkt mit ihr kommunizieren. Diese Abhängigkeiten führen bei Änderungen häufig zu hohen Kosten. Da jeder Client mit der Datenbank kommuniziert, entsteht ein hoher Ressourcenverbrauch.

Eine mehrschichtige Architektur besteht mindestens aus drei Schichten. Es gibt zu jeder Softwareschicht mindestens eine physische. In dieser Architektur befinden sich nur Thin-Clients. Sie kommunizieren mit einem speziellen Server auf dem die Geschäftslogik ausgeführt wird. Nur dieser ist mit der Datenbank verbunden. Gegenüber der zweischichtigen Architektur ergeben sich Vorteile. So lassen sich z.B. Ressourcen effizienter nutzen und Performanceprobleme

sind leicht lokalisierbar. Aber auch die mehrschichtige Architektur hat Nachteile. Die Aufteilung auf drei oder mehr Ebenen hat ein hohes Kommunikationsaufkommen zur Folge. Der Wartungsaufwand für diese Ebenen ist deutlich größer als der einer zweischichtigen Architektur.

Mit der J2EE Plattform ist es möglich, Anwendungen für mehrschichtige Architekturen zu entwickeln. J2EE definiert vier Anwendungsschichten und verschiedene Komponenten. Eine Komponente ist als unabhängig funktionale Softwareeinheit zu verstehen.

Die Komponenten der Clientschicht können webbasiert oder nicht webbasiert sein. Webbasierte sind Applets und Browser auf Clientrechnern, die innerhalb der virtuellen Maschine (VM, engl. virtual maschine) von Browsern ablaufen. Nicht webbasierte sind Anwendungen, die auf Clientrechnern ausgeführt werden.

Auf einem J2EE-Server befinden sich Web- und Geschäfts-Komponenten. Sie werden einer Web- und einer Geschäfts-Schicht zugeordnet. Web-Komponenten sind Servlets und JSP's, deren Aufgabe es ist, dynamisch Anfragen zu bearbeiten und Antworten zu generieren. Die Geschäfts-Komponenten zum Ausführen der Geschäftslogik sind EJB's.

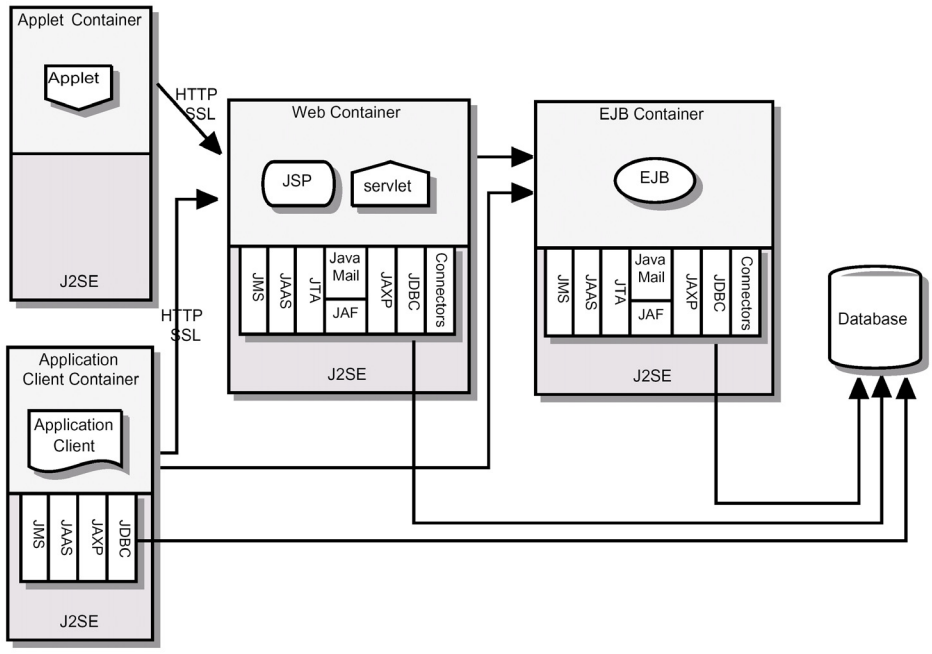

Abbildung 1: J2EE-Beispielarchitektur
(Quelle: Java 2 Platform Enterprise Edition Specification, 2001, S. 4)

Die EIS-Schicht integriert Mainframe- und ERP-Systeme sowie Datenbanken und Legacy-Systeme.

J2EE-basierte Systeme erfordern sogenannte Application Server. Ein Application Server setzt sich aus einem Container und einem Server zusammen. Der Container verwaltet die Komponenten über Schnittstellen. Der Server stellt die Laufzeitumgebung für den Container bereit und verwaltet Ressourcen. Der Application Server bietet verschiedene konfigurierbare und nicht konfigurierbare Dienste. Zu den konfigurierbaren zählen Transaktionsmanagement, Sicherheit, Namen- und Verzeichnisdienste sowie die entfernte Verfügbarkeit. Die nichtkonfigurierbaren Dienste sind Ressourcenmanagement, Verwaltung der Komponentenlebenszyklen und Persistenzmechanismen. Komponenten sind im wesentlichen vom Application Serverhersteller unabhängig. Es gibt jedoch auch herstellerspezifische konfigurierbare Dienste.

Der Entwicklungsprozess J2EE-basierter Systeme ist auf mehrere Rollen verteilt. Die EJB-Spezifikation definiert Rollen für das Entwickeln von Komponenten und Anwendungen, das Bereitstellen von Laufzeitumgebung und Diensten sowie das Installieren von Komponenten und das Überwachen von Systemen. Mit dieser Aufteilung wird das Ziel verfolgt, dass durch Spezialisierung auf einzelne Bereiche eine möglichst hohe Qualität der Gesamtsysteme entsteht. Der Komponentenentwickler kann sich auf die Implementierung der Geschäftslogik konzentrieren. Er greift dabei auf Dienste des Application Servers zurück.

2.2 Enterprise JavaBeans-Arten

Eine EJB besteht aus vier Teilen: dem Remote Interface, dem Home Interface, der Bean-Implementierungsklasse und dem Deployment Descriptor.

Das Remote Interface definiert Geschäftsmethoden, die Clients auf einer Bean aufrufen können. Im Home Interface sind eine oder mehrere create-Methoden definiert, um Zugriff auf eine EJB zu erlangen. In der Beanklasse sind die Geschäftsmethoden implementiert. Der Deployment Descriptor ist eine XML-Datei, in der die Umgebungsinformation und das Verhalten der Bean festgelegt werden.

Der Container verwaltet die Bean-Instanzen. Ein direkter Zugriff durch Clients ist nicht möglich. Remote und Home Interface müssen von bestimmten Schnittstellen, EJBObject und EJBHome, erben. Von zentraler Bedeutung sind konkrete Objekte dieser Schnittstellen, die Bestandteil des Containers sind. Die HomeObject-Instanz ist ein Factory-Objekt und erzeugt EJBObject-Instanzen. Diese kapseln den Zugriff auf Bean-Instanzen.

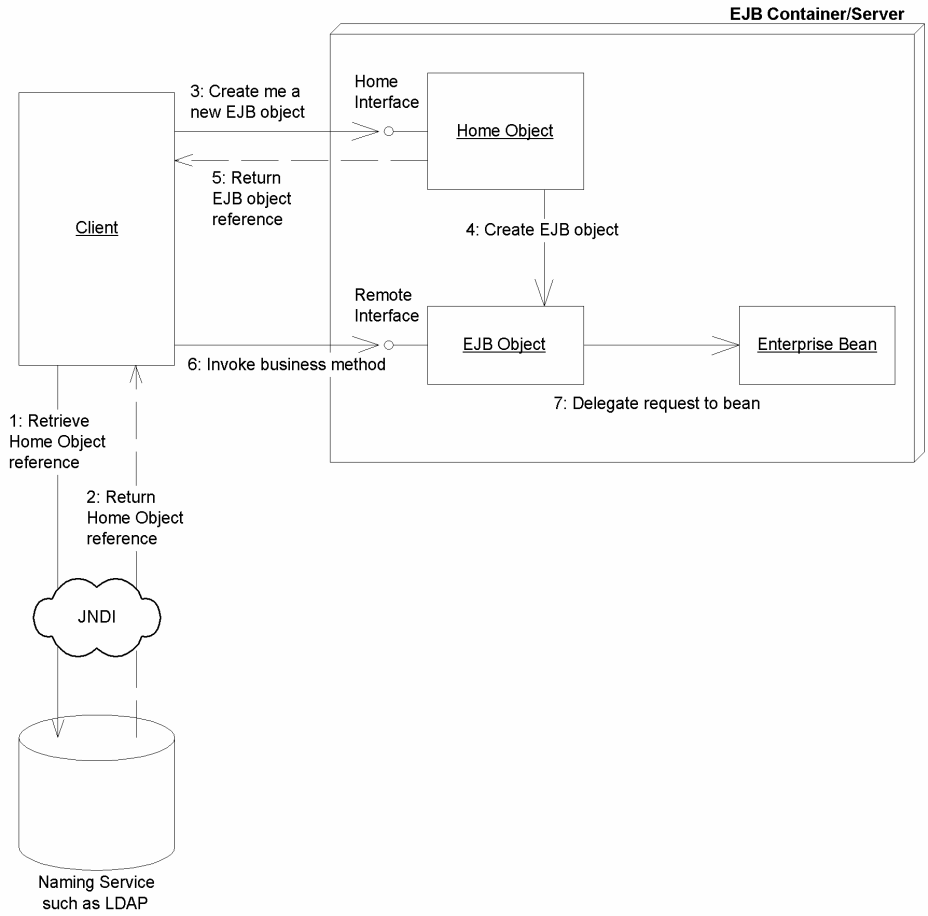

Abbildung 2: Referenzieren eines Home Objektes und Aufrufen einer Geschäftsmethode (Quelle: Roman, E., Mastering Enterprise JavaBeans, 1999, S. 93)

Ein Client hat die Möglichkeit sich durch JNDI Zugang zu der HomeObject-Instanz einer Beanklasse zu verschaffen. Der Aufruf einer create-Methode bewirkt, dass eine EJBObject-Instanz erzeugt wird. Methodenaufrufe auf dem Remote Interface werden über diese EJBObject-Instanz an eine Bean-Instanz weitergeleitet.

Die Beanklasse muss zwar alle Methoden des Remote Interface enthalten, darf es aber nicht implementieren. Da das Remote Interface EJBObject erweitert, müssen auch dessen Methoden in der Bean implementiert werden. Diese Me-

thoden sind speziell für den Clientzugriff und gehören nicht in die Bean. Wird das Interface dennoch implementiert, so ist die Bean automatisch vom Typ EJBObject. Dann kann es passieren, dass versehentlich anstelle der EJBObject-Instanz eine Referenz auf die Bean übergeben wird. Dieses Verhalten verbietet die Spezifikation.

Um den verschiedenen Anforderungen, Implementierung von Geschäftslogik und Abbildung persistenter Daten, gerecht werden zu können, definiert die Spezifikation zwei Arten von EJB's – Session Beans und Entity Beans.

2.2.1 Session Beans

Session Beans implementieren Geschäfts- und Workflowlogik. Sie repräsentieren Geschäftsprozesse und Vorgänge, die für Clients durchgeführt werden. Für jeden Client wird eine eigene Session Bean zur Verfügung gestellt.

Es gibt Stateful und Stateless Session Beans. Stateful Session Beans zeichnen sich dadurch aus, dass sie Geschäftsprozesse abbilden, die sich über mehrere Methodenaufrufe erstrecken. Diese Komponenten ermöglichen eine dialogorientierte Kommunikation, da sie ihren Zustand für ihren Client festhalten. Stateless Session Beans hingegen repräsentieren Geschäftsprozesse, die sich über einen Methodenaufruf erstrecken. Sie bieten einen unabhängigen Dienst, der sich für jeden Aufruf von Clients immer gleich darstellt. Zustandsänderungen gehen nach der Abarbeitung der Methode verloren. Dieses Verhalten spiegelt sich in den create-Methoden und Lebenszyklen wider. Da sich Stateless Session Beans nicht voneinander unterscheiden, hat die create-Methode keine Parameter. Im Gegensatz dazu können Stateful Session Beans über verschiedene create-Methoden verfügen. Der Lebenszyklus von Stateless Session Beans ist unabhängig vom Client. Sie werden aus einem Pool von Instanzen bedient. Der Container verwaltet den Pool und steuert die Größe. Bei Stateful Session Beans hingegen wird der Lebenszyklus durch die Aktionen des Clients beeinflusst. Der Container behält aber weiterhin die Kontrolle über Zustandsänderungen. Ein Poolen ist nicht möglich, da die Beans eine clientspezifische Identität besitzen.

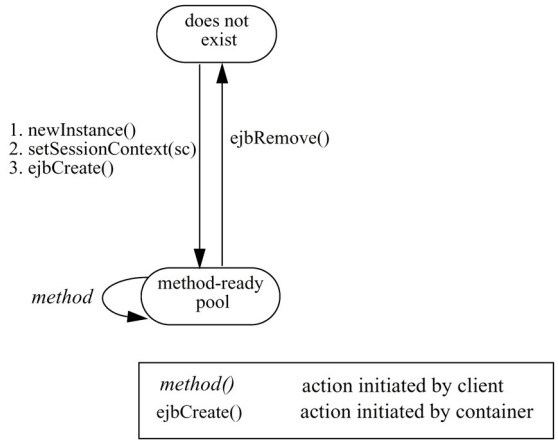

Abbildung 3: Lebenszyklus von Stateless Session Beans
(Quelle: Enterprise Java Beans Specification, 2001, S. 89)

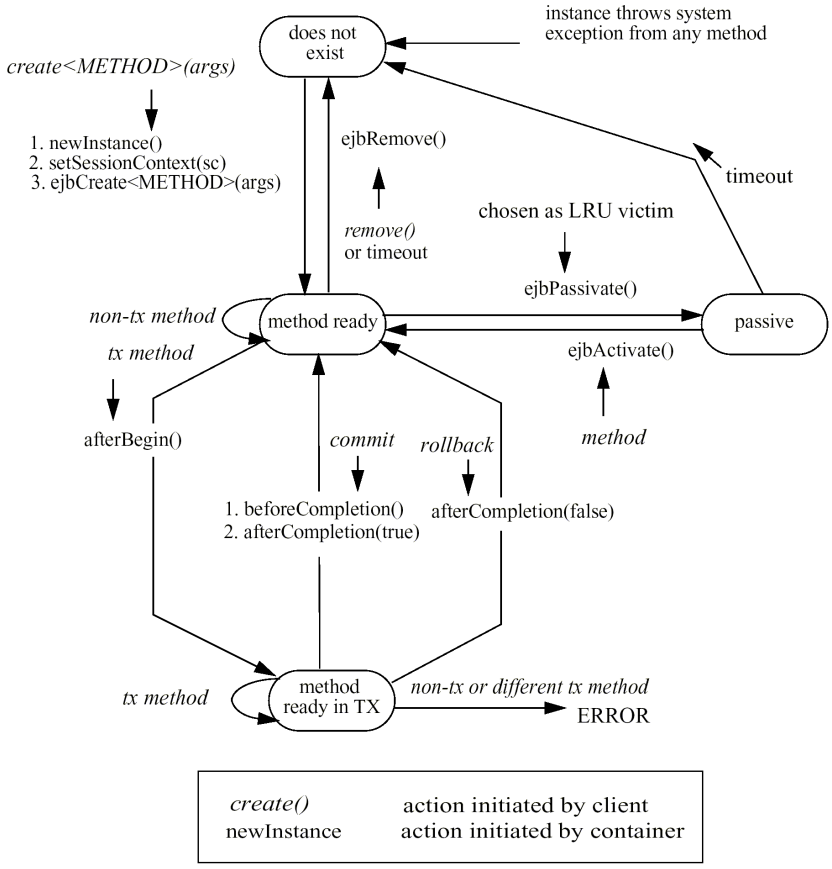

Abbildung 4: Lebenszyklus von Stateful Session Beans
(Quelle: Enterprise Java Beans Specification, 2001, S. 77)

Es ergeben sich also unterschiedliche Einsatzbereiche für Stateful und Stateless Session Beans. In einer E-Commerce-Anwendung kann beispielsweise eine Stateful Session Bean eingesetzt werden, um einen Warenkorb zu realisieren,

eine Stateless Session Bean, um Berechnungen für den Inhalt des Warenkorbs vorzunehmen.

2.2.2 Entity Beans

Entity Beans enthalten keine Geschäftslogik sondern modellieren Daten. Sie bieten eine objektorientierte Ansicht auf persistente Daten.

Befinden sich Daten in einer relationalen Datenbank, spricht man von einer objekt-relationalen Abbildung. Im Deployment Descriptor wird eine Verbindung zwischen den Attributen einer Datenbanktabelle und den Feldern einer Entity Bean hergestellt. Eine Entity Bean existiert solange wie der entsprechende Eintrag im Persistenzspeicher. Ihre Lebensdauer unterscheidet sich dadurch wesentlich von der von Session Beans, da sie auch kritische Situationen wie Serverabstürze übersteht. Im Gegensatz zu Session Beans steht nicht jedem Client eine eigene Entity Bean zur Verfügung, sonder eine Entity Bean bedient alle Clients. Die Clientzugriffe werden durch Transaktionen voneinander abgegrenzt. Werden dabei Werte geändert, bewirkt dies einen Zugriff auf den darunter liegenden Datenspeicher.

Entity Beans werden eindeutig durch Primary Key-Objekte identifiziert. Sie entsprechen dem Primärschlüssel einer zugrunde liegenden Tabelle. Mit der create-Methode des Home Interface werden neue Entity Beans erzeugt, mit der remove-Methode bestehende gelöscht. Um Zugriff auf bestehende Entity Beans zu erhalten, werden im Home Interface finder-Methoden zur Verfügung gestellt. Es muss eine geben, die zu einem Primary Key-Objekt die entsprechende Entity Bean zurückliefert. Weitere finder-Methoden können es ermöglichen, nach bestimmten Entity Beans zu suchen. Sie liefern jedoch nur eine Sammlung von Primary Key-Objekten. Das hat zur Folge, dass ein Zugriff auf den Persistenzspeicher notwendig ist, um eine Sammlung von Primary Key-Objekten zu erhalten und je ein weiterer um eine Entity Bean zu erzeugen. Dies wird auch das „n+1 Zugriffsproblem“ genannt.

Die EJB-Spezifikation definiert zwei Mechanismen, um die Persistenz von Entity Beans sicherzustellen. Eine Entity Bean wird im Deployment Descriptor als Bean- oder Container-managed gekennzeichnet.

Bei Bean Managed Persistence (BMP) muss der Entwickler die für die Persistenz notwendigen Vorgänge unter zu Hilfenahme von API's wie z.B. JDBC selbst implementieren. Der Entwickler hat in diesem Fall sehr viel Code schreiben, um Datensätze anzulegen, zu ändern, zu löschen oder zu suchen.

Wird eine Entity Bean mit Container Managed Persistence (CMP) realisiert, so ist es Sache des Containers, für die notwendigen Zugriffe auf den Datenspeicher zu sorgen. Im Deployment Descriptor wird spezifiziert wie die finder-Methoden zu implementieren sind. Es entsteht ein vergleichsweise geringer Entwicklungsaufwand.

Entity Beans werden eingesetzt, um persistente Daten zu modellieren. Die beispielhaft angeführte E-Commerce-Anwendung könnte sich folgendermaßen verhalten: Hat ein Kunde Waren gekauft, wird der Inhalt des Warenkorbs durch eine Entity Bean persistent gemacht. Später wird die Entity Bean von einem anderen Teil des Systems benutzt, um die Rechnung zu erstellen.

2.3 Enterprise JavaBeans 2.0

Im September 2001 ist die EJB-Spezifikation 2.0 veröffentlicht worden. In ihr gibt es gravierende Änderungen und Neuerungen. Lokale Clients können jetzt von einem leichgewichtigeren Zugriff profitieren. Ein überarbeitetes Konzept der Entity Beans ermöglicht Persistenzmanagern effizienter zu arbeiten, und es lassen sich persistente Beziehungen abbilden. Mit den Message-driven Beans ist JMS in EJB integriert worden. Vor einer detaillierten Betrachtung soll verdeutlicht werden, wo sich Schwachstellen in der EJB-Technologie befinden.

2.3.1 Schwachstellen in EJB 1.1

Die Idee, javabasierte Komponentenentwicklung mit der EJB-Technologie zu betreiben, ist im Vergleich zu anderen Ansätzen, z.B. das Microsoftprodukt COM, vergleichsweise jung. Mit der neuen Spezifikation sind einige Schwächen beseitigt worden, andere bestehen weiterhin.

Damit der Container in die Lage versetzt werden kann Beans zu verwalten, ist es nötig, dass er Informationen über die Zustände der einzelnen Beans hat und ggf. bestimmte Operationen auf diesen ausführen kann. Um dies zu erreichen, kapselt das Laufzeitsystem die Bean-Instanzen. Zugriffe auf die Beans erfolgen

immer durch diese Schicht, nie durch den Client direkt. Das verschlingt viel Zeit und Ressourcen und führt dazu, dass der Einsatz feingranularer Geschäftsobjekte verhindert wird.

In der objektorientierten Programmierung werden Callbacks eingesetzt. Ein Objekt A ruft eine Methode von Objekt B auf. Um diese Methode abarbeiten zu können, muss Objekt B wiederum Methoden von Objekt A aufrufen. Dieses Vorgehen ist bei EJB's jedoch sehr kritisch. Die Spezifikation verbietet es sogar, Callbacks auf Session Beans auszuführen, da eine Session Bean immer nur einen Client hat. Bei wiedereintrittsfähigen Entity Beans sind Callbacks zwar möglich, sollten aber vermieden werden, da der Container nicht unbedingt zwischen Callback und Simultanaufruf eines anderen Clients unterscheiden kann. Ein gleichzeitiger Zugriff ist verboten und kann zu unvorhersagbaren Ergebnissen führen. Auch in der neuen Spezifikation gelten diese Einschränkungen.

Zwischen EJB und dem Standard Java-Modell kommt es zu Abweichungen. Um z.B. die Objektidentität von EJBObject-Instanzen zu überprüfen, muss die Methode *isIdentical()* verwendet werden. In Standard Java hingegen werden „==“ sowie die Methoden *equals()* und *hashcode()* verwendet. Diese werden häufig in Frameworks eingesetzt. Das macht es schwierig, Frameworks in Bezug auf EJB anzuwenden. In diesem Punkt hat die Spezifikation ebenfalls keine Änderungen erfahren.

Die Entwicklung von Komponenten unterscheidet sich von der von Standard Java-Objekten. Dem wird nicht hinreichend Rechnung getragen. Dinge wie Sicherheit, Transaktionen und Persistenz bleiben bei der Vererbung auf Klassenebene unberücksichtigt. Es besteht Bedarf, die Vererbung auch auf Komponentenebene anwenden zu können. Derzeit ist dies nur möglich, indem jede einzelne Klasse bzw. Schnittstelle vererbt und der Deployment Descriptor kopiert wird. Die Komponenten-Vererbung wird voraussichtlich Bestandteil zukünftiger Versionen der Spezifikation sein.

Es gibt „System“-Objekte, die Dienste des Containers wie Sicherheit, Transaktionen und Verteilung ermöglichen. Sie kapseln und mischen sich mit den Geschäftsobjekten. Die Klassen dieser Dienst-Objekte sind statisch, d.h. klassenspezifisch, generiert. Nach jeder Änderung müssen sie neu erzeugt und verteilt werden. Dadurch wird die Akzeptanz von EJB gehemmt; die zusätzlichen

Klassen sind eine Last für das System und machen es für Application Server-Hersteller und Entwickler komplizierter. Würden dynamische, also unspezifische, Dienstobjekte verwendet, gäbe es pro Dienst nur eine zusätzliche Klasse, die für alle Beans eingesetzt werden kann. Die Komponentenentwicklung ließe sich dadurch vereinfachen. Auch in der Spezifikation 2.0 ist keine Änderung erfolgt.

Die EJB Spezifikation 1.1 legt sich nicht auf ein Kommunikationsprotokoll fest. Es kann sowohl RMI als auch RMI/IIOP[2] eingesetzt werden. Somit ist auch das Verhalten für entfernte Aufrufe nicht eindeutig. RMI bietet verteilte Garbage Collection und die Parameter werden als Referenzen übergeben. RMI/IIOP hingegen bietet keine verteilte Garbage Collection an und Parameter werden als Werte übergeben. Diese Unterschiede müssen berücksichtigt werden, wenn verschiedenartige Clients Beans referenzieren. In der neuen Spezifikation wird dieses Problem beseitigt. Als einheitliches Protokoll wird RMI/IIOP festgelegt. Es kommt zu einer Annäherung zwischen EJB und CORBA, da die Dienste der EJB 2.0 kompatiblen Server auf CORBA basieren. Der Grund dafür ist, dass ein wachsender Bedarf für serverseitige Komponenten entsteht, die in einer heterogenen Umgebung miteinander kommunizieren.

Die Spezifikation der Entity Beans ist in einigen Punkten unzureichend. Es ist nicht klar, wie Objekte innerhalb von Entity Beans persistent zu machen sind. Kritische Aspekte, wie das Abbilden und Verwalten der referenziellen Integrität sowie das Teilladen von Objektgraphen in den Speicher, werden nicht berücksichtigt. Beziehungen zwischen Beans lassen sich demzufolge nur mit BMP umsetzen, indem sie fest einprogrammiert werden. Der Einsatz von Entity Beans bringt bei großen Datenmengen Performanceprobleme mit sich. Deren Einsatz ist dann u. U. gar nicht möglich. Ein weiterer Entity Bean-Typ für rein lesenden Zugriff könnte dieses Problem lindern. Es ist geplant, so einen Typ in Zukunft in die Spezifikation zu integrieren. In diesem Bereich hat es einige Änderungen gegeben. Diese werde im Abschnitt „Neuer Kontrakt für Entity Beans“ ausführlich betrachtet.

[2] Steht für RMI over IIOP. Entfernte Methodenaufrufe werden über ein spezielles Protokoll ausgeführt, dass Bestandteil der CORBA-Spezifikation ist. IIOP steht für Internet Inter-ORB Protocol.

Ein Client hat nur die Möglichkeit, über das Remote Interface auf Beans zuzugreifen. Auch lokale Clients, sie befinden sich in derselben VM, können nicht von ihrer Nähe zur Bean profitieren. Es entsteht immer ein Performance-Overhead bei Remote-Aufrufen. Zusätzlich entsteht Overhead für die Übergabe von Werten anstelle von Referenzen. Aus diesen Gründen sollte es vermieden werden, große Objekte zu übergeben. In EJB 2.0 werden lokale Schnittstelen definiert, die lokalen Clients einen leichtgewichtigeren Zugriff auf Beans gestatten. Eine nähere Betrachtung erfolgt im Abschnitt „Das Local Interface“.

Die Portabilität von Beans leidet darunter, dass verschiedene Aspekte in der Spezifikation nicht berücksichtigt werden. Propietäre Lösungen der Application Server Hersteller führen zu Inkompatibilitäten. Es entstehen z.B. bei objekt-relationaler Abbildung und Komponentensuche datenspeicherabhängige Implementierungen mit geringer Portabilität. Die objekt-relationale Abbildung ist in EJB 2.0 weiterhin herstellerspezifisch, aber die Suche nach Komponenten wurde standardisiert. Dies wird ebenfalls detailliert im Abschnitt „Neuer Kontrakt für Entity Beans“ besprochen.

Der Einsatz von JMS innerhalb von Beans ist zwar möglich, wird aber nicht besonders unterstützt. Eine Integration in die EJB-Konzepte ist wünschenswert, damit asynchrone Kommunikation auch als Konsument angemessen durchgeführt werden kann. Die Integration wird mit EJB 2.0 vollzogen und im Abschnitt „Message-driven Beans“ dargestellt.

2.3.2 Das Local Interface

EJB 2.0 führt neben dem Remote Interface das Local Interface ein. Es kann von Clients genutzt werden, die sich innerhalb derselben VM und derselben Anwendung befinden. Local Interfaces haben den Vorteil des leichtgewichtigeren Zugriffs, mit dem eine gesteigerte Performance erzielt wird. Für sie gilt Standard Javaaufrufsemantik und es entsteht kein RMI-Overhead. Der Code wird semantisch besser lesbar. In der Regel werden Beans nur ein Interface anbieten. Dadurch kann man erkennen, ob Beans für lokale oder entfernte Clients zugreifbar sind. Der Einsatz von Local Interfaces führt zu einer lokalen Abhängigkeit, die Standorttransparenz geht verloren.

2.3.3 Neuer Kontrakt für Entity Beans

CMP 2.0 hat weitreichende Änderungen erfahren. Durch das neue Local Interface ist es für den Container möglich flexiblere, inkrementelle Feldzugriffe durchzuführen und mit Beziehungen zwischen Entity Beans umzugehen. Mit dem entsprechenden Wissen ist der Container in der Lage, abhängige Objekte selbständig zu laden und zu speichern. Die neuen Konzepte der Container Managed Relationships (CMR) und der abhängigen Objekte ermöglichen einen feinkörnigen Zugriff auf abhängige Objekte durch träges und inkrementelles Laden.

Der Container ist in der Lage Zustand und Persistenz der Beziehung, die auch verschachtelt sein können, zu behandeln und referentielle Integrität sicherzustellen. Es ist möglich 1 zu 1-, 1 zu n- und n zu m-Beziehungen abzubilden. Der Zugriff auf andere Beans erfolgt über die Methoden des Local Interface. Beziehungen dürfen nicht über das Remote Interface veröffentlicht werden.

Mit der EJB Query Language (EJB QL) ist eine standardisierte Abfragesprache auf Komponentenebene eingeführt worden. EJB QL basiert semantisch auf SQL 92. Mit ihr werden finder-Methoden und die neuen select-Methoden formuliert. Dadurch wird die Portabilität von Entity Beans erhöht. Select-Methoden dienen dazu, datenspezifische Geschäftslogik zu realisieren. Sie sind nicht für Clients zugänglich. Der Entwickler kann intern Gebrauch von ihnen machen. EJB QL bietet in dieser Version leider nur eingeschränkte Möglichkeiten. So werden z. Z. noch keine Aggregatoperationen unterstützt. Es ist aber geplant, dies in Zukunft zu tun.

Besondere Bedeutung kommt dem Persistenzmanager zu. Er ist vom Container entkoppelt und kann unabhängig von Drittanbietern bereitgestellt werden. Er hat die Aufgabe, auf Basis eines abstrakten Persistenzschemas für die konkrete Implementierung der Bean-Klasse zu sorgen. Der Entwickler definiert Felder, Beziehungen und Abfragen im Deployment Descriptor sowie eine abstrakte Klasse mit abstrakten getter-, setter- und finder-Methoden. Der Persistenzmanager sorgt für einen optimierten Datenzugriff und ermöglicht so träges und inkrementelles Laden sowie ein höherwertiges Caching. Bei diesem Vorgang wird EJB QL in eine native Abfragesprache überführt.

Im Home Interface von Entity Beans können so genannte home-Methoden deklariert werden. Sie sind nicht instanzenspezifisch und dürfen nicht auf den Zustand einzelner Beans zugreifen. Ihre Aufgabe besteht darin, globale Operationen auszuführen.

Der neue Kontrakt bewirkt, dass CMP 1.1 und 2.0 nicht kompatibel zueinander sind. Die Containerhersteller müssen jedoch beide Formen unterstützen. Auf diesem Weg wird die Abwärtskompatibilität sichergestellt.

2.3.4 Message-driven Beans

Mit Message-driven Beans wird eine neue Bean-Art eingeführt. Sie sind transaktionsbewusste Komponenten, die asynchrone Kommunikation auf der Basis von JMS ermöglichen. Andere Messaging-Typen sind bisher leider nicht integriert. Es wird jedoch angestrebt, dies in zukünftigen Versionen zu ändern. Es entsteht eine lose Kopplung zwischen dem Produzenten, dem Client, und dem Konsumenten, der Message-driven Bean. Dadurch werden Abhängigkeiten der Komponenten untereinander reduziert. Der Lebenszyklus von Message-driven Beans ähnelt dem von Stateless Session Beans. Message-driven Beans besitzen jedoch weder Remote noch Home Interface. Aufrufe erfolgen durch den Container, wenn eine Nachricht zu verarbeiten ist. Der Container stellt auch Transaktions-, Sicherheits- und Nebenläufigkeitsdienste bereit.

3 WebLogic Server

BEA Systems ist Marktführer im Bereich der Application Server. Das Produkt WebLogic Server in der Version 6.1 mit Servicepack 1 ist eine Implementierung der J2EE 1.3 Technologien.

Der WebLogic Server besitzt zusätzliche, nicht spezifizierte Eigenschaften in Bezug auf EJB. Zu diesen gehören insbesondere die Möglichkeiten mehrere Application Server in einem Cluster zu betreiben und Entity Beans auf lesenden Zugriff zu beschränken. Das Verhalten von EJB's in einem Cluster ist in der Spezifikation nicht berücksichtigt. Der Einsatz von Clustern ist aber nötig um große, skalierbare Systeme realisieren zu können. Entity Beans auf lesenden Zugriff zu beschränken ermöglicht dem Application Server, Daten zwischenzuspeichern. Die Performance kann gesteigert werden, da Datenbankzugriffe reduziert werden können.

3.1 Cluster-Fähigkeiten

Ein Cluster ist eine Gruppe von lose gekoppelten Servern. Sie arbeiten zusammen und stellen nach außen eine Einheit dar. Cluster unterscheiden sich von Einzelserver-Lösungen besonders durch Skalierbarkeit, Verfügbarkeit und Wartbarkeit. Die Anzahl der Server eines Clusters kann an veränderte Lastbedingungen angepasst werden. Dienste eines Clusters stehen mit hoher Wahrscheinlichkeit immer zur Verfügung, da Ausfälle einzelner Server kompensiert werden können. Für Cluster entsteht jedoch auch ein höherer Wartungsaufwand, da mehrere Server zu pflegen sind.

Application Server bieten einen JNDI-kompatiblen Namensdienst an. Dort werden Dienste öffentlich gemacht, indem dort HomeObject-Stubs registriert und in einer Baumstruktur organisiert werden. Clients können Zugriff auf die dort gebundenen Stubs erhalten. Innerhalb eines Clusters wird ein einziger logischer JNDI-Baum erzeugt, der alle Dienste des Cluster enthält. Darin sind sowohl diejenigen enthalten, welche nur auf einzelnen Server verfügbar sind als auch diejenigen, welche auf mehreren Servern des Clusters verfügbar sind. Jeder Server des Clusters besitzt eine lokale Kopie vom JNDI-Baum. Bei dessen Aufbau werden die Dienste zunächst lokal gebunden. Die speziellen Stubs

für Dienste, die auf mehreren Servern verfügbar sind, werden an alle Server verteilt.

Die speziellen Stubs werden als „replica-aware“ (von Kopien wissend) bezeichnet. Sie repräsentieren nicht ein einzelnes Objekt, sondern eine Sammlung von Kopien. Sowohl HomeObject-Stubs als auch EJBObject-Stubs sind replica-aware. Sie wissen, auf welchen Servern ein Dienst angeboten wird und ermöglichen so Lastverteilung und Ausfallsicherung.

Die Lastverteilung hat das Ziel, alle Server gleichermaßen auszulasten. Der WebLogic Server benutzt standardmäßig einen Round-Robin-Algorithmus, bei dem die Server der Reihe nach ausgewählt werden. Dieser Algorithmus kann auch gewichtet werden, wenn die Server nicht die gleich Leistungsfähigkeit haben. Anstelle des Round-Robin-basierten Verfahrens können die Server auch zufällig ausgewählt werden.

Tritt bei einer Methodenausführung ein Systemfehler auf, wird versucht, die Methode erneut auf einem anderen Server auszuführen. Die Methode muss dafür als idempotent gekennzeichnet werden. Das bedeutet, dass sie mehrfach aufrufbar ist, ohne sich dabei unterschiedlich zu verhalten. Das trifft auf jeden Fall dann zu, wenn sie keine permanenten Seiteneffekte hat.

Replica-aware-Stubs verhalten sich bei der Ausfallsicherung folgendermaßen:

Bei Methodenaufrufen von HomeObject-Stubs oder EJBObject-Stubs von Stateless Session Beans wird eine beliebige Kopie ausgewählt. Create- und finder-Methoden können auf einem beliebigen Server aufgerufen werden. Die dabei erzeugten EJBObject-Instanzen hängen nicht von einem bestimmten Server ab. Die Geschäftsmethoden der EJBObject-Stubs von Stateless Session Beans können einen beliebigen Server auswählen, da alle Bean-Instanzen gleich sind. Bei Stateful Session Beans und Entity Beans sind EJBObject-Stubs auf eine Kopie festgelegt, da die zugehörige Bean-Instanz an einen Server gebunden ist.

Stateful Session Beans bieten eine besondere Ausfallsicherung. Der Zustand der Bean-Instanz wird auf einen Backupserver kopiert. Erst wenn die ursprüngliche Bean-Instanz nicht mehr benutzbar ist, wird auf dem Backupserver eine neue erzeugt und mit dieser weitergearbeitet. Dann wird auch ein neuer Back-

upserver festgelegt. Es werden nur Zustandsänderungen auf den Backupserver kopiert. Das geschieht entweder, wenn die Transaktion beendet wird, an der die Bean beteiligt ist oder nach der Methodenausführung. Der Einsatz dieses Verfahrens ist jedoch mit einem geringen Risiko behaftet. In bestimmten Situationen kann es bei einem Serverabsturz passieren, dass der Backupserver nicht über den aktuellen Zustand oder über gar keinen Zustand verfügt. Stürzt der Backupserver ebenfalls ab, ist der Zustand der Session in jedem Fall verloren.

3.2 Read-Mostly-Muster und Entity Bean-Erweiterungen

Der WebLogic Server bietet die Möglichkeit Entity Beans in einem herstellerspezifischen Deployment Descriptor als nur lesefähig zu deklarieren. Das ermöglicht dem Container, die Beans zwischenzuspeichern. In einem Cluster werden sie auf alle Server kopiert. Es erfolgen nur Datenbankzugriffe, um die Bean zu laden. Wenn sie aus dem Speicher entfernt wird, wird ihr Zustand nicht gespeichert. Von dem Cache kann in dieser Version leider nur profitiert werden, wenn auf Beans über das Primary Key-Objekt zugegriffen wird. Erfolgt ein Zugriff über eine spezielle finder-Methode, so ist ein Datenbankzugriff nötig. Für die read-only Beans muss ein Timeout angegeben werden, nach dessen Ablauf sie ungültig werden. Wenn dann ein Zugriff auf die Bean erfolgt, muss sie neu geladen werden. Auf diese Weise werden regelmäßige Datenänderungen, die von externen Systemen durchgeführt werden, trotzdem berücksichtigt. Die Bean kann auch durch einen Methodenaufruf einer bestimmten Schnittstelle ungültig gemacht werden. Die read-only Beans unterliegen noch weiteren Einschränkungen. Sie dürfen nicht in Transaktionen einbezogen werden und ihre Methoden müssen idempotent sein.

Im Read-Mostly-Muster werden read-only und read-write Beans miteinander kombiniert, indem sie die gleichen Daten modellieren. Zum einen wird so eine gute Performance erreicht, da read-only Beans zwischengespeichert werden. Zum anderen ist Sicherheit gewährleistet, da Zugriffe auf read-write Beans innerhalb von Transaktionen stattfinden.

Für Entity Beans können im herstellerspezifischen Deployment Descriptor Gruppen für das Laden von CMP- und CMR-Feldern definiert werden. Die Daten der Gruppen werden als Einheit betrachtet. Sie können mit Beziehungen und Abfragen assoziiert werden. Die Daten werden geladen, wenn die Abfrage

ausgeführt, oder auf die Relation zugegriffen wird. Standardmäßig werden die Daten der default-Gruppe geladen, in der alle Felder enthalten sind.

Der WebLogic Server ist bei CMP in der Lage, eine Sammlung von Entity Beans mit einem Datenbankzugriff zu erzeugen. Das Problem der n+1 Datenbankzugriffe besteht bei CMP also nicht.

4 Einführung in den Wertpapierhandel

4.1 Grundlagen und Begriffe

Grill und Perczynski definieren den Begriff Wertpapier wie folgt:

„Ein Wertpapier ist eine Urkunde, in der ein privates Vermögensrecht so verbrieft ist, dass zur Ausübung des Rechts der Besitz an der Urkunde erforderlich ist.“[3]

Eine Urkunde ist eine rechtlich oder wirtschaftlich wichtige Erklärung in schriftlicher Form.

Ein Wertpapier verkörpert ein Vermögensrecht. Das kann ein Forderungsrecht (Geldforderung) oder Mitgliedschaftsrecht (Teilhaberrecht an einem Unternehmen) sein.

Um das Recht auszuüben, ist der Besitz des Wertpapiers erforderlich. Recht und Urkunde sind folglich so eng miteinander verbunden, dass sie eine Einheit bilden.

Börsenfähige Wertpapiere werden als Effekten bezeichnet und sind vertretbare Kapitalwertpapiere. Vertretbar bedeutet, dass Papiere desselben Ausstellers mit dem gleichen Wert auch die gleichen Rechte verkörpern und beliebig austauschbar sind. Kapitalwertpapiere sind langfristige Forderungen oder Teilhaberrechte. Schuldverschreibungen, Aktien und Investmentzertifikate erfüllen diese Merkmale. Rechte aus Schuldverschreibungen und Aktien stehen entweder beliebigen Inhabern (Inhaberpapiere) oder dem namentlichen genannten (Namenspapiere) zu.

Schuldverschreibungen stellen Forderungsrechte dar. Der Aussteller (Schuldner) beschafft sich durch einen Kredit Fremdkapital[4] vom Kapitalmarkt[5]. Der Gesamtbetrag des Kredits wird als Anleihe, Schuldverschreibung oder Obligation bezeichnet. Der Anleger (Gläubiger) hat Anspruch auf Rückzahlung des Darlehens und auf Zinsen. Erfolgen regelmäßige Zinszahlungen, so spricht man auch von Renten.

---

[3] Vgl. Grill, W./Perczynski, H., Wirtschaftslehre, 1998, S. 203
[4] der Teil des Kapitals, der von außerhalb des Unternehmens kommt
[5] Finanzmarkt, an dem langfristige Kapitalanlagen gehandelt werden

Aktien sind Teilhaberrechte an einer Aktiengesellschaft. Durch den Erwerb von Aktien ist man am Grundkapital einer Aktiengesellschaft beteiligt. Es wird zwischen Nennbetrags- und Stückaktien unterschieden. Nennbetragsaktien lauten auf einen glatten Eurobetrag. Ihr Anteil am Grundkapital entspricht dem Verhältnis des Nennbetrages zum Grundkapital. Stückaktien lauten auf einen Anteil am Grundkapital.

Neben Inhaber- und Namensaktien gibt es noch die Sonderform der vinkulierten Namensaktie. Aktien dieses Typs dürfen nur mit Zustimmung der Gesellschaft verkauft werden.

Investmentzertifikate sind Anteile an einem Wertpapierfond. Wertpapierfonds werden von Kapitalanlagegesellschaften (Investmentgesellschaft) verwaltet. Mit dem Vermögen werden Wertpapiere gekauft. Sie können nach unterschiedlichen Gesichtspunkten zusammengesetzt werden. Dabei wird das Ziel verfolgt mögliche Risiken, die beim Kauf einzelner Wertpapiere entstehen, durch eine breite Streuung auszuschließen.

4.2 Handel an Wertpapierbörsen

Wertpapierbörsen sind Marktplätze, die schnelle Geschäftsabschlüsse und Preisfindung sowie Markttransparenz ermöglichen. Börsenplätze können Präsenzbörsen (Parkettbörsen) sein, an denen Geschäfte über Makler abgeschlossen werden, oder Computerbörsen, an denen dies automatisch durch Computersysteme geschieht.

In Deutschland gibt es acht Präsenzbörsen in Berlin, Bremen, Düsseldorf, Frankfurt am Main, Hamburg, Hannover, München und Stuttgart sowie die Computerbörse Xetra (Xetra steht für Exchange Electronic Trading). An der Frankfurter Börse werden die meisten Geschäfte getätigt. Die Börsenteilnehmer sind Kreditinstitute, die sowohl mit eigenen Wertpapieren als auch mit denen ihrer Kunden handeln, und Makler, die Geschäfte vermitteln und Kurse feststellen.

Wertpapiere werden an verschiedenen Märkten gehandelt. Für die dort notierten Unternehmen gelten verschieden strenge Vorschriften.

Die Kurse von Aktien werden als Stückkurse, die von Schuldverschreibungen als Prozentkurse ermittelt. Die kleinste handelbare Einheit ist auf 0,01 Euro

festgelegt. Für alle Wertpapiere wird einmal am Börsentag der Einheitskurs, auch Kassakurs genannt, festgestellt. Hierbei werden alle zu diesem Zeitpunkt vorliegenden Aufträge berücksichtigt. Bei diesem Kurs muss der größtmögliche Umsatz zustande kommen, und es müssen alle Geschäfte mit bestimmten Kriterien ausgeführt werden können. Der Kurs gilt für alle Geschäfte. Bei häufig gehandelten Wertpapieren werden zusätzlich Eröffnungs- und Schlusskurse sowie der fortlaufende Preis ermittelt. Dies wird als variable Notierung bezeichnet. Eröffnungs- und Schlusskurs werden wie der Einheitskurs bestimmt, der fortlaufend festgestellt Kurs bezieht sich nur auf einzelne Geschäfte.

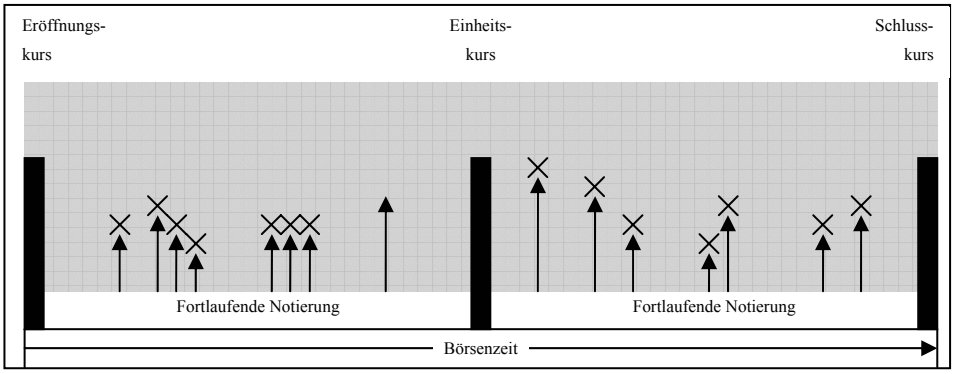

Abbildung 5: Handel bei variabler Notierung
(Quelle: Grill, W./Perczynski, H., Wirtschaftslehre, 1998, S. 265)

4.3 Verhältnis zwischen Kreditinstituten und Kunden

Kreditinstitute sind durch das Wertpapierhandelsgesetz dazu verpflichtet, Dienstleistungen immer im Interesse der Kunden zu erbringen und dabei die ordnungsgemäße Durchführung sicherzustellen. Kunden müssen angemessen beraten und über Konsequenzen informiert werden.

Bei Aufträgen müssen die börsenmäßige Bezeichnung des Wertpapiers und die zu handelnde Menge angegeben werden. Die Bezeichnung erfolgt durch nationale Wertpapierkennnummern (WKN). Im März 2003 werden die nationalen Bezeichnungen durch die weltweit eindeutige International Securities Identification Number (ISIN) ersetzt.[6]

Kunden haben die Möglichkeit, Aufträge mit einem Kurslimit zu versehen. Wird kein Limit angegeben, spricht man von einem unlimitierten Auftrag. Kreditinstitute sollten diese Aufträge auf jeden Fall ausführen. Kaufaufträge

[6] Vgl. Wertpapier-Mitteilungen Datenservice: ISIN-Einführung, 2001, S. 3

werden möglichst billig („billigst“), Verkaufaufträge möglichst teuer („bestens“) ausgeführt. Wird ein Limit angegeben, so handelt es sich um einen limitierten Auftrag. Dieser Kurs darf beim Kauf nicht über-, beim Verkauf nicht unterschritten werden. Wird ein circa Auftrag erteilt, so darf vom Limitwert um eine bestimmte Spanne abgewichen werden. Stop-Aufträge werden erst beim Kurslimit aktiv. So werden Stop-loss-Aufträge „bestens“ ausgeführt, sobald das Limit unterschritten wird. Damit können bereits erzielte Gewinne gesichert und mögliche Verluste beschränkt werden. Stop-buy-Aufträge werden „billigst“ ausgeführt, sobald das Limit überschritten wird. In beiden Fällen werden die Aufträge beim nächsten ermittelten Börsenkurs ausgeführt. Aufträge können auch als interessewahrend gekennzeichnet werden. Dies kommt nur bei größeren unlimitierten Verkaufsaufträgen vor. Das Kreditinstitut muss den Auftrag so ausführen, dass der Kurs möglichst wenig beeinflusst wird. Die Ausführung kann sich über mehrere Börsentage erstrecken.

Vor der Weiterleitung von Aufträgen muss eine Deckungsprüfung durchgeführt werden. Bei Verkäufen muss der Kunde über eine ausreichende Anzahl an Wertpapieren besitzen. Bei Käufen muss er über das voraussichtlich erforderliche Guthaben oder über eine ausreichende Kreditlinie verfügen.

Kreditinstitute ordnen Wertpapiere und Kunden häufig in Risikoklassen ein. Erteilen Kunden Aufträge, muss diese Einstufung berücksichtigt werden. Ist die Risikoklasse nicht ausreichend, muss der Kunde explizit darüber aufgeklärt werden, damit der Auftrag trotzdem angenommen werden darf.

Finanzderivate[7] dürfen nur gehandelt werden, wenn der Kunde börsentermingeschäftsfähig ist. Kaufleute besitzen Börsentermingeschäftsfähigkeit. Nichtkaufleute können sie für einen Zeitraum von maximal drei Jahre erwerben.

Kunden können für ihre Aufträge eine befristete oder unbefristete Gültigkeitsdauer festlegen. Wird keine Angabe gemacht, sind unlimitierte Orders nur am selben Börsentag gültig (Tagesorder). Ist die Börse bereits geschlossen, sind Orders am nächsten Börsentag gültig. Limitierte Aufträge ohne Gültigkeitsdauer sind nur bis zum letzten Börsentag des laufenden Monats gültig (Ultimo-

[7] Derivate sind Termingeschäfte. Sie verkörpern Rechte, die von der Entwicklung anderer Werte abhängen.

Order). Wird eine Order am letzen Börsentag des Monats aufgegeben, ist sie für den kommenden Monat gültig.

Aufträge werden vom Kreditinstitut an die Börse weitergeleitet, wobei der Kunde den Börsenplatz bestimmt. Wird keine Börse vorgegeben, muss das Kreditinstitut einen im Interesse des Kunden auswählen. Kreditinstitute sind dazu verpflichtet, Kundenaufträge als Kommissionäre auszuführen. Laut §383 HGB führt ein Kommissionär den Wertpapierhandel für andere durch. Bei der einfachen Wertpapierkommission werden Geschäfte mit anderen Marktteilnehmern abgeschlossen. Hat ein Kreditinstitut keinen Zugang zur Börse, wird ein weiteres Kreditinstitut mit der Ausführung beauftragt. Dann spricht man von der Zwischenkommission.

Wertpapiere werden von speziellen Kreditinstituten, den Wertpapiersammelbanken, verwahrt und verwaltet. In der Regel werden Wertpapiere in Sammelverwahrung genommen. Dabei verliert der Kunde sein Sondereigentum, erwirbt aber Miteigentum am Gesamtbestand. Er hat das Recht auf Herausgabe von Wertpapieren in Höhe des hinterlegten Nennbetrages bzw. der hinterlegten Stückzahl. Vinkulierte Namensaktien sind nicht für die Sammelverwahrung geeignet, da der Eigentümer namentlich auf der Urkunde eingetragen ist. Solche Bestände werden in Sonderverwahrung genommen. Der Kunde behält sein Eigentum am hinterlegten Wertpapier.

5 Das bestehende Brokerage-System

Der Betreiber bietet seinen Kunden die Möglichkeit, über dieses System Wertpapiere zu handeln. Das System bedient die Vertriebswege Internet, mobile Endgeräte (Handy und PDA), Call Center und Filiale. Das System besteht im wesentlichen aus Technologien der J2EE-Plattform. Zusätzlich werden Stored Procedures eingesetzt. Die für den Betrieb erforderlichen Daten werden fast vollständig in einer relationalen Datenbank vorgehalten. Zeitkritische Daten werden durch Fremdsysteme verfügbar gemacht.

5.1 Funktionen

Kunden können das System sowohl über das Internet als auch über mobile Endgeräte nutzen. Es ermöglicht den Wertpapierhandel und bietet zusätzlich Marktinformationen und weitere Dienstleistungen. In den Vertriebswegen Call Center und Filiale bedienen Mitarbeiter des Betreibers für Kunden das System. Ihnen stehen zusätzlich Verwaltungs- und Kontrollfunktionen zur Verfügung.

Kundenfunktionen

Um das System benutzen zu können, muss der Kunde über ein Verrechnungskonto bei einem bestimmten Kreditinstitut verfügen. Mit der Kontonummer und einer PIN zum Autorisieren erfolgt die Anmeldung. Ein Kunde kann mehrere Depots besitzen, jedoch nur eins zur Zeit verwalten.

An allen deutschen Börsenplätzen können Aktien, Renten und Optionsscheine gekauft und verkauft werden. Der Wertpapierkauf schließt das Zeichnen von Neuemissionen mit ein. Kunden können außerbörslich mit Fondsanteilen handeln. Beim Fondssparplan werden regelmäßig Anteile eines Fonds für einen festen Betrag gekauft werden. Der Betreiber entscheidet, welche Wertpapiere von den Nutzern des Systems gehandelt werden dürfen.

Durch das Orderbuch erhält der Kunde Informationen über die Wertpapierbestände in seinem Depot. Noch nicht ausgeführte Aufträge können geändert oder gestrichen werden. Der Kunde kann sich detailliert anzeigen lassen, wie sich sein Wertpapierbestand im Laufe der Zeit verändert hat.

Der Kunde hat direkten Zugriff auf sein Verrechungskonto und kann Kontostand und Umsätze abfragen sowie Überweisungen auf ein festgelegtes Referenzkonto tätigen.

In das System sind die Webseiten eines Kursinformationsproviders integriert. Kunden können sich einen Marktüberblick verschaffen, Kurse abfragen und ein Musterdepot führen. Sie können Chartanalyse betreiben, sich über Neuemissionen, Neuigkeiten und Fachbegriffe des Wertpapierhandels informieren.

Kunden können verschiedene weitere Dienste in Anspruch nehmen. Dazu gehören ein umfangreiches Informationsangebot sowie die Verwaltung von PIN und TAN-Liste.

Mitarbeiterfunktionen

Die Mitarbeiter des Betreibers verfügen über umfangreiche Verwaltungsfunktionen, dazu gehören uneingeschränkte Zugriffe auf Depots, Depotbestände und Aufträge. Durch die Mitarbeiter wird außerdem festgelegt, welche Wertpapiere über dieses System gehandelt werden können.

5.2 Architektur und Design

5.2.1 Architektur

Das Brokerage-System verfügt über eine mehrschichtige Hardware-Architektur. Sie besteht aus einem Webserver-Cluster, einem Application Servercluster und einem Datenbankserver-Cluster. Durch eine Firewall vor dem Webserver-Cluster und eine zwischen Web- und Application Servercluster wird eine demilitarisierte Zone[8] (DMZ) geschaffen. Dadurch entsteht ein hohes Maß an Sicherheit.

Die Systemarchitektur entspricht auf Softwareebene einer J2EE-Architektur. Das System wird sowohl von webbasierten als auch von nicht webbasierten Clients benutzt. Auf Basis der zu Grunde liegenden Hardware-Architektur werden Web- und Geschäftsschicht physisch voneinander getrennt. In beiden Schichten befinden sich Komponenten der J2EE-Technologien. In der EIS-

[8] Eine demilitarisierte Zone ist ein halböffentlicher Teil eines Netzwerkes. Er ist nach außen durch eine Firewall gesichert, die aber bestimmte Verbindungen zulässt. Durch eine zweite Firewall entsteht ein privater Teil. Diese Firewall ist restriktiver als die erste, so dass der private Teil abgeschottet wird.

Schicht werden Daten in einer relationalen Datenbank vorgehalten und verschiedene Fremdsysteme angebunden.

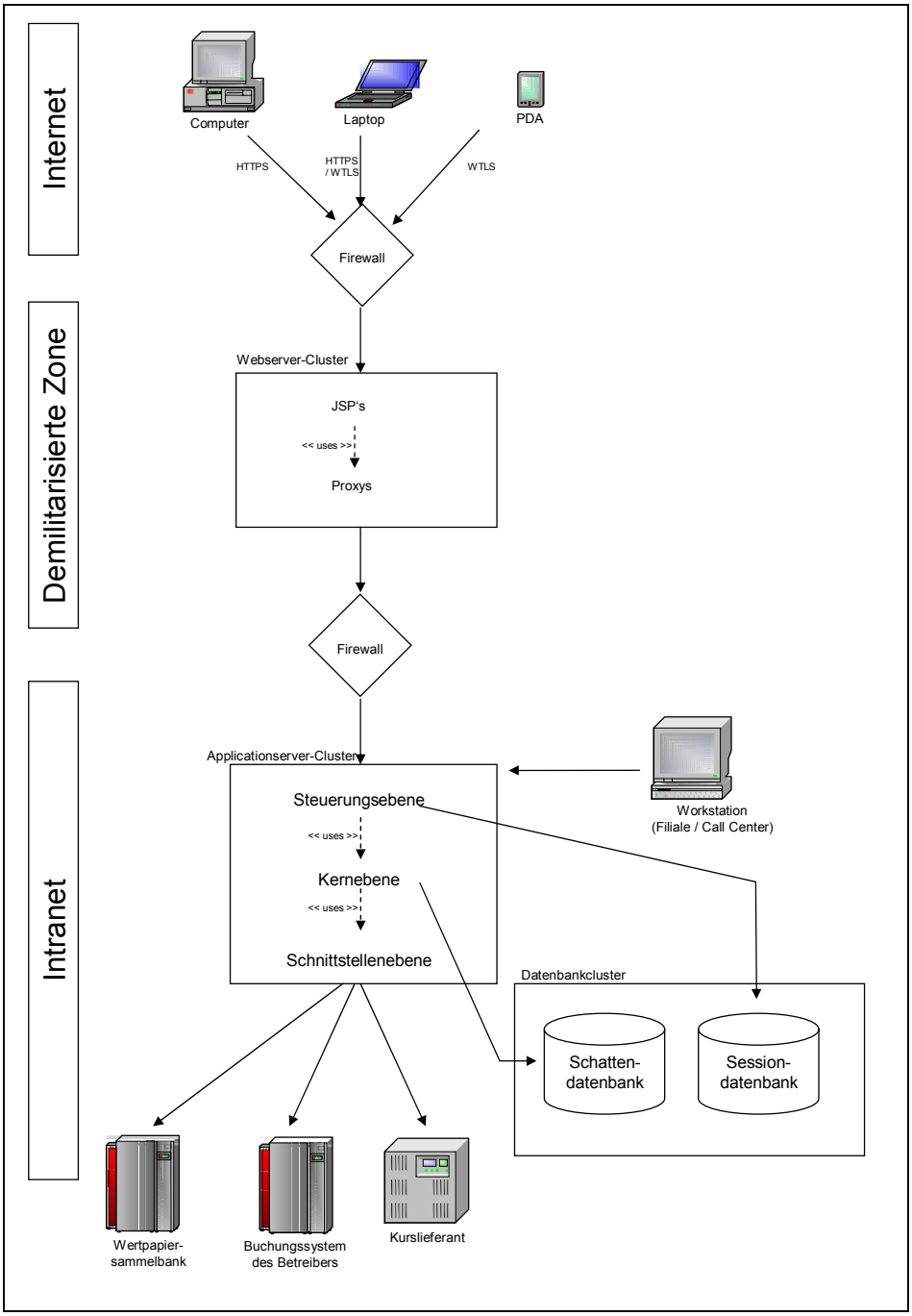

Abbildung 6: Architektur des Brokerage-Systems

Im Webserver-Cluster werden Apache 1.3.14, ein reiner Webserver, und Tomcat 3.2.3, ein Container für Web-Komponenten, miteinander kombiniert.

Für den Application Servercluster wird WebLogic 5.1 verwendet. Aufgrund der Trennung von Web- und Geschäftsschicht wird vom Application Server ausschließlich der Container für Geschäftskomponenten verwendet. Der

WebLogic Server 5.1 ist konform zur J2EE-Spezifikation 1.2 und damit auch zur EJB-Spezifikation 1.1. Es verfügt nicht über die in Kapitel drei aufgeführten Fähigkeiten des WebLogic Servers 6.1.

Als Datenbank wird Oracle in der Version 8.1.7 eingesetzt.

5.2.2 Design

Das Design orientiert sich am Model-View-Controller-Entwurfsmuster (MVC). Dieses Muster teilt die Anwendung in drei verschiedene Funktionsbereiche. Der Bereich „Model“ bezieht sich auf die Daten der Anwendung sowie die anzuwendenden Geschäftsregeln. Hier werden Zustand und Funktionalität der Anwendung abstrahiert. Mit „View“ ist die Darstellung der Daten in einem bestimmten Kontext gemeint. Der „Controller“ setzt das, was in „View“ passiert, in Methodenaufrufe des „Model“ um und stellt dem Benutzer die entsprechenden „Views“ zur Verfügung.

Die Präsentation der Daten erfolgt bei nicht-webbasierten Clients durch eine grafische Oberfläche. In diese ist die Funktion des „Controllers“ integriert. Bei den webbasierten Clients geschieht die Darstellung durch HTML bzw. WML. Browser bzw. mobile Endgeräte schicken ihre Anfragen an die JSP’s der Webschicht – die „Controller“. Die verschiedenen „Controller“ übersetzten die Aktionen in entsprechende Methodenaufrufe der Geschäftsschicht, dem „Model“, um.

Webschicht

Die JSP’s haben die Aufgabe, dynamisch für den Inhalt der Präsentationsschicht zu sorgen. Sie sind clientspezifisch und produzieren entweder HTML für Browser oder WML für mobile Endgeräte. An der Beantwortung einer Anfrage eines Browsers sind mehrere JSP’s beteiligt. Die aufgerufene JSP steuert die Beantwortung und schließt jeweils weitere JSP’s mit ein. Diese dienen sowohl dazu, die Beantwortung zu steuern, als auch dazu, HTML zu erzeugen. Anfragen von Clients über WAP werden immer von einer JSP bearbeitet. Um Geschäftsvorfälle abzuarbeiten, werden Methoden der Geschäftsschicht aufgerufen. In einigen Fällen wird auch von den JSP’s Geschäftslogik ausgeführt. Der Zugriff auf die Geschäftsschicht wird durch Proxy-Objekte gekapselt. Der Einsatz der Proxy-Objekte entspricht dem Business Delegate Entwurfsmuster.

Im wesentlichen werden die Ziele verfolgt, die Komplexität der EJB-API zu verbergen und eine Entkopplung von der EJB-Schicht herbeizuführen.[9]

Geschäftsschicht

Die Geschäftsschicht besteht aus Stateless Session Beans und einfachen Java-Objekten. EJB's zeichnen sich dadurch aus, dass sie während der Laufzeit ersetzt werden können, wohingegen Java-Objekte den Vorteil haben, dass sie schneller und einfacher entwickelt werden können und eine bessere Performance aufweisen. In Abhängigkeit davon, ob Teile des Systems leichter änderbar oder performanter sein müssen, sind sie entweder als Java-Objekt oder als EJB implementiert. Der wesentliche Vorteil von Stateless Session Beans gegenüber Stateful Session Beans ist, dass sie vom Container gepoolt werden können. Auf diese Weise kann mit den vorhandenen Ressourcen effizient umgegangen werden.

Es hat sich in der Praxis gezeigt, dass der Einsatz von Entity Beans problematisch ist. Sie besitzen eine schlechte Performance, da viele Datenbankzugriffe entstehen. Es muss immer erst der Primärschlüssel gesucht werden, mit dessen Hilfe dann die Entity Bean erzeugt wird. Aus diesem Grund werden keine Entity Beans eingesetzt.

Stateful Session Beans in einem Cluster zu verwenden, bringt verschiedene Nachteile mit sich. Für die Dauer des Vorgangs kommuniziert der Client mit einem bestimmten Server. Zum einen sind in dieser Zeit Ressourcen des Servers gebunden und zum anderen wird die Möglichkeit, eine Lastverteilung durchzuführen, eingeschränkt. Ein Serverabsturz hat außerdem zur Folge, dass der Konversationszustand unwiderruflich verloren geht. Wegen der genannten Nachteile wird gänzlich auf Stateful Session Beans verzichtet.

Da es Geschäftsvorfälle gibt, die sich über mehrere Methodenaufrufe erstrecken und da häufig Kundeninformationen benötigt werden, müssen dennoch Sessions verwaltet werden. Das ließe sich mit den HTTP-Session-Objekten des Webservers realisieren. Der Webserver befindet sich jedoch in der DMZ und ist dadurch der Gefahr eines Angriffs ausgesetzt. Um ein hohes Maß an Sicherheit zu gewährleisten, dürfen keine Kundendaten zwischen den Aufrufen

[9] Vgl. Marinescu, F., EJB Design Patterns, 2002, S. 98 ff.

im Arbeitsspeicher verbleiben, sie werden deshalb in der Datenbank verwaltet. Eine Session wird beim Login-Vorgang angelegt. Die benötigten Informationen werden in einem Hashtable zusammengestellt und in der Sessiondatenbank gespeichert. Zu jeder Session wird ein 128 Bit langer Schlüssel generiert. Dieser wird nicht auf dem Webserver gespeichert, sondern bei jeder Anfrage und jeder Antwort zwischen Clientrechner und Webserver hin- und hergeschickt. Da die Kommunikation über das Internet durch eine abhörsichere Verbindung erfolgt, können Angreifer den Schlüssel nur in Erfahrung bringen, wenn sie Zugang zum Endgerät des Kunden haben. Alle Methoden, die vom Webserver aus aufgerufen werden, erfordern, dass ein Schlüssel mitgeschickt wird. Über den Schlüssel wird auf die Sessiondaten zugegriffen. Auf diese Weise ist es Angreifern nicht möglich Methodenaufrufe durchzuführen. Lediglich ein Login wäre möglich, wofür jedoch Kontonummer und PIN nötig sind.

Durch diese Sessionverwaltung ergibt sich der Vorteil, dass sie nicht an einen einzelnen Server des Clusters gebunden ist. Bei einem Serverabsturz können nur die momentan durchgeführten Transaktionen verloren gehen. Der Datenbankserver ist durch ein Notfallsystem gesichert, so dass ebenfalls nur die momentan durchgeführten Transaktionen verloren gehen können.

Die Geschäftsschicht ist in drei logische Ebenen unterteilt. Netlife benutzt die Bezeichnungen Steuerungsschicht, Geschäftsschicht und Schnittstellenschicht. Um Verwechslungen mit den Namen der J2EE-Archtektur zu vermeiden, werden im folgenden die Bezeichnungen Steuerungsebene, Kernebene und Schnittstellenebene verwendet.

Die Bean-Klassen der Komponenten der Steuerungs- und der Kernebene erben von einer gemeinsamen Oberklasse für jede Ebene. Die Oberklassen implementieren jeweils die von der EJB-Spezifikation geforderten Methoden, um eine Steuerung durch den Container zu ermöglichen.

Die Steuerungsebene stellt eine Fassade für den Clientzugriff dar. Zwischen Web- und Geschäftsschicht sowie zwischen den Komponenten der Geschäftsschicht erfolgt die Kommunikation über Datentransferobjekte. Daten werden nicht modelliert.

Es gibt jeweils ein Paar von Session Beans in der Steuerungs- und der Kernebene, um Geschäftsprozesse in bestimmten Bereichen abzubilden. Diese Be-

reiche sind Kunde, Konto, Depot, Order, Wertpapiere, Sicherheit und Systeminformationen. Ein Hostsystem des Betreibers übernimmt verschiedene Aufgaben. Es ist durch eine Bean der Schnittstellenschicht angebunden. Die Funktionen des Hostsystems werden in verschiedene Zuständigkeitsbereiche integriert. Das Hostsystem stellt beispielsweise die Kundenautorisierung sicher. Diese Funktion ist Bestandteil des Bereichs Sicherheit.

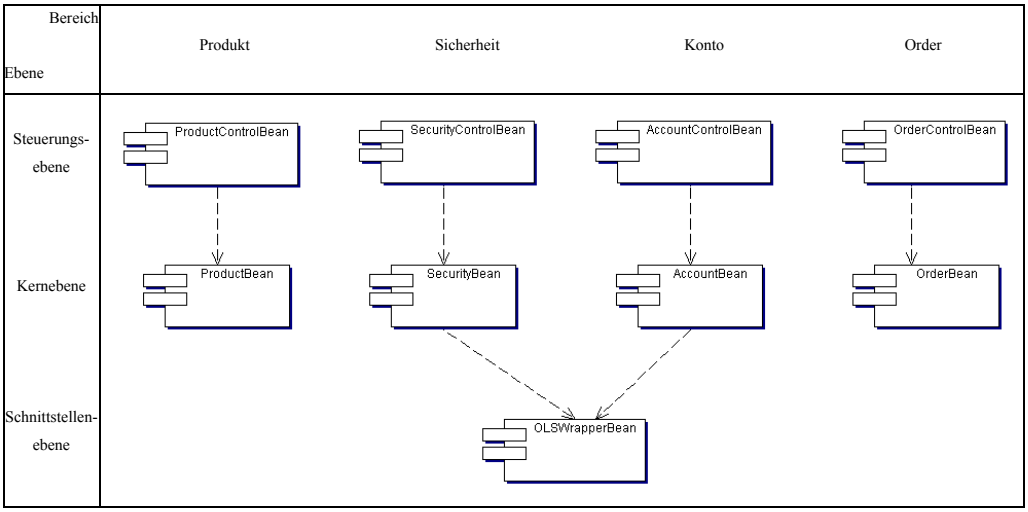

Abbildung 7: einige Komponenten der Geschäftsschicht

Den Beans der Steuerungsebene wird neben einem Datentransferobjekt der Schlüssel für die Sessiondaten übergeben. Diese Beans haben die Aufgabe die Sessiondaten zu laden und zu speichern. Wenn erforderlich benutzen sie Komponenten und Objekte der Kernebene, um dem Datentransferobjekt weitere Informationen hinzuzufügen. Die Steuerungsebene und die Kernebene sind eng miteinander verbunden. Zu jeder Bean der Steuerungsebene gibt es eine zugehörige in der Kernebene. Innerhalb eines Methodenaufrufs einer Bean der Steuerungsebene wird immer eine Methode mit dem gleichen Namen der zugehörigen Bean der Kernebene aufgerufen. Neben dem Datentransferobjekt werden dann die geladenen Sessiondaten übergeben.

In der Kernebene wird die Geschäftslogik ausgeführt. Dazu werden die Dienste der Schnittstellenebene benutzt und Stored Procedures in der Datenbank angestoßen. Die Daten des Systems werden nicht in der Geschäftsschicht abgebildet. Der Grund dafür ist, dass viele Daten für die Geschäftslogik benötigt werden. Um diese in der Kernebene verfügbar zu machen, wären viele Datenbankzugriffe notwendig. Daraus würden langsame Antwortzeiten und eine hohe Netzwerklast entstehen. Darum wird die Geschäftslogik, die auf den vorgehal-

tenen Daten basiert, in Stored Procedures abgebildet. Um die Datenbankzugriffe durchführen zu können, wird ein Verbindungs-Pool verwaltet.

In der Schnittstellenebene befinden sich Beans und Objekte, durch die eine Anbindung an Fremdsysteme erfolgt.

EIS-Schicht

Die notwendigen Daten für den Betrieb werden in der EIS-Schicht gehalten. Sie befinden sich zum Großteil in verschiedenen Fremdsystemen. Um unabhängig von der Verfügbarkeit dieser Systeme zu sein, werden alle Daten mit geringer Änderungshäufigkeit wie Produkt- und Kundendaten in einem eigenen Datenbanksystem vorgehalten. Bei Bedarf werden Anfragen an die Fremdsysteme gestellt, um zeitkritische Informationen wie Kurse oder Kontostände zu erhalten. Zusätzlich sind Daten im System vorhanden, die sich nur sehr selten ändern und die von Netlife gepflegt werden. Dazu gehören z.B. Informationen über Börsenplätze.

Zwischen der Schattendatenbank[10] und einem Hostsystem findet ein regelmäßiger Datenaustausch statt. Das Hostsystem liefert täglich aktualisierte Stammdatendaten sowie mehrmals täglich Informationen über ausgeführte Aufträge.
Neue Aufträge werden zunächst in der Schattendatenbank gespeichert und dann an das Hostsystem weitergeleitet. Die Daten werden durch C-Programme und Stored Procedures in Batch-Läufen in die Schattendatenbank importiert.

5.3 Datenmodell

Das Datenmodell des Brokerage-Systems ist sehr umfangreich. Es soll hier nur ein Überblick gegeben werden.

Die Tabellen lassen sich in die Kategorien Import, Produkt, Order, Depot, Kunden sowie Administration gliedern.

In mehreren Batchläufen werden die Import-Tabellen mit Daten aus einem Hostsystem gefüllt. Es gibt verschiedene Tabellen für Wertpapiere, Depots, Aufträge und Kundendaten.

[10] Kopie einer Datenbank, auf der parallel oder zeitversetzt alle Änderungen ebenfalls nachvollzogen werden.

Informationen über Wertpapiere befinden sich in den Produkt-Tabellen. Dies sind zum einen Informationen über die Wertpapiere (*Product*) selbst und zum anderen börsenplatzabhängige Daten (*ProductExchange*). Dazu gehört beispielsweise, an welcher Börse ein Wertpapier gehandelt werden kann. Zur Tabelle Product ist anzumerken, dass die Datensätze zwar eindeutig durch die WKN identifiziert werden, aber trotzdem ein zusätzlicher künstlicher Primärschlüssel eingeführt wurde. Der Grund dafür ist die anstehende Einführung der ISIN. Das Datenmodell ist unabhängig von der fachlichen Identifizierung der Wertpapiere. Bei der Umstellung des fachlichen Schlüssels muss nur die Tabelle Product angepasst werden. Fremdschlüssel anderer Tabellen referenzieren immer den künstlichen Schlüssel und sind so von fachlichen Änderungen unabhängig. Die Tabelle *ProductName* dient zusätzlich zum Abbilden der WKN's auf künstliche Schlüssel.

Die Fonds der verschiedenen Kapitalanlagegesellschaften, sowie die Fondsparpläne befinden sich in weiteren Tabellen (*KAG*, *FundProduct*, *FSP*).

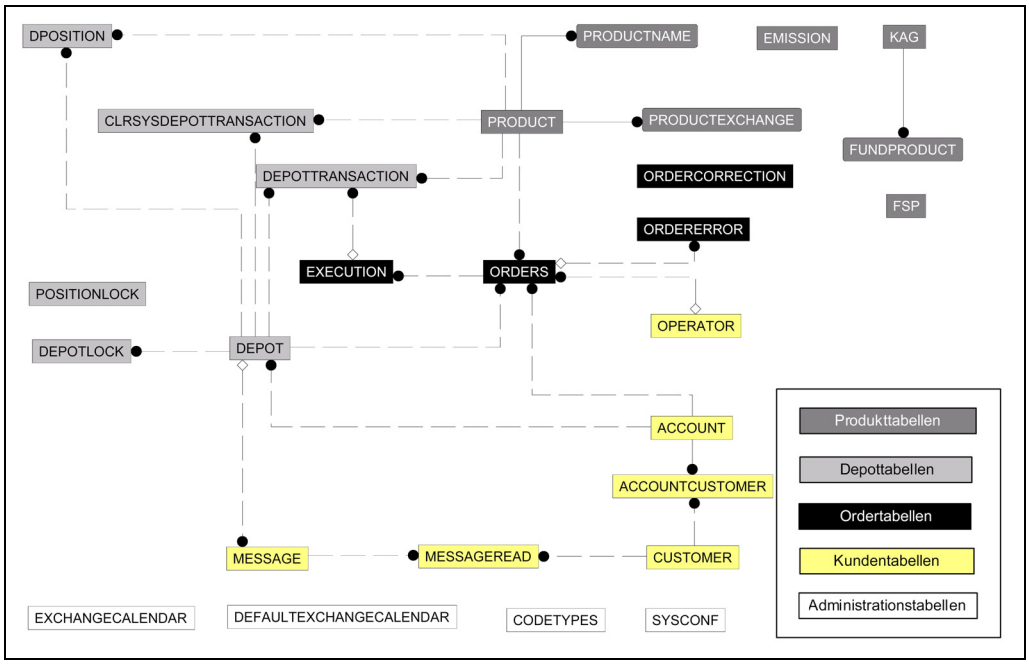

Abbildung 8: Teildarstellung des Datenmodells

In den Order-Tabellen sind im wesentlichen die Daten der aufgegebenen Aufträge (*Orders*) sowie deren Ausführungen (*Executions*) enthalten.

Die Depot-Tabellen enthalten Daten der Wertpapierdepots (*Depot*). Außerdem sind Bestände (*DPosition*), Umsätze (*DepotTransaction*, *ClearingSystemDepotTransaction*) und Sperren (*DepotLock*, *PositionLock*) hinterlegt.

Informationen über Kunden und deren Verrechnungskonten (*Customer*, *AccountCustomer*, *Account*) sowie über die Mitarbeiter des Betreibers (*Operator*) befinden sich in den Kundentabellen.

Die Administrations-Tabellen nehmen Daten auf, die nicht von Hostsystemen bereitgestellt werden. Hier erfolgt die Aufschlüsselung von Werten, die in den anderen Tabellen verwendeten werden (*CodeTypes*), und die Konfiguration des Systems (*SysConf*). Ebenso sind hier die Öffnungszeiten (*DefaultExchangeCalendar*) und Börsenfeiertage (*ExchangeCalendar*) der Börsenplätze abgelegt.

5.4 Schwachstellen

Technisch gesehen sind JSP's und Servlets gleichwertig. JSP's sind jedoch auf die Ausgabe einer Markup Language fokussiert, wohingegen Servlets eher eine Java-zentrische Sicht darstellen. Es wäre besser, neben JSP's auch Servlets einzusetzen. Damit ließe sich eine klare Trennung zwischen der „Controller-Funktion" und der Aufbereitung des Inhalts herbeiführen.

Es ist nicht nötig, clientspezifische JSP's zu haben. Wenn der Inhalt in XML dargestellt wird, ließe sich die benötigte clientspezifische Rückgabe durch XSLT erzeugen. Dadurch könnte erreichen werden, dass alle webbasierten Clients denselben „Controller" benutzen. Dieser Ansatz geht aber zulasten der Performance, da XSLT-Processing sehr zeitaufwendig ist.

Das Design der Geschäftsschicht sollte in jedem Fall um eine Datenzugriffsschicht erweitert werden. Die Beans der Kernebene führen sowohl Logik als auch Datenbankzugriffe durch. Grundsätzlich sollten Beans unabhängig vom Datenspeicher sein. Es sollte eine klare Trennung zwischen Geschäftslogik und Datenzugriff angestrebt werden. Um den Datenzugriff zu realisieren, bieten sich Data Access Objects an. Hierbei handelt es sich um eine Anwendung des Abstract Factory-Patterns[11]. Eine Fabrik erzeugt datenspeicherspezifische Zugriffsobjekte, welche von Beans über eine Schnittstelle angesprochen werden. Auf diese Weise wird die Implementierung der Beans unabhängig vom Datenspeicher.

[11] Vgl. Gamma, E./Helm, R./Johnson, R./Vlissides, J., Entwurfsmuster, 1996, S. 65

Das Brokerage-System verwaltet einen eigenen Verbindungspool. Die Verwaltung von Ressourcen ist jedoch Aufgabe des Application Servers. Es gibt keinen Grund, an dieser Stelle anders vorzugehen, als dies die Spezifikation vorsieht.

Ein weiterer ganz wesentlicher Nachteil der Lösung, Stored Procedures zu verwenden und keine Daten in der Geschäftsschicht abzubilden, ist, dass eine große Abhängigkeit vom Datenbankhersteller entstanden ist. Zum einen bieten nicht alle Datenbanken die Möglichkeit Stored Procedures auszuführen, zum anderen erfolgt deren Implementierung in herstellerabhängiger Sprache. Der Einsatz einer anderen Datenbank ist also nur bedingt möglich. Der Umstieg von PL/SQL auf eine andere Sprache kann mit großem Aufwand verbunden sein.

Nach der J2EE-Architektur soll die Geschäftslogik in die dafür vorgesehene Geschäftsschicht implementiert werden. Im Brokerage-System ist aber in allen Schichten Geschäftslogik enthalten. Dadurch gehen wesentliche Vorteile der Schichtenarchitektur verloren. Die Schichten sind nicht mehr unabhängig voneinander und Änderungen der Geschäftslogik wirken sich u. U. in mehreren Schichten aus. Eine weitere Folge ist, dass bei Tests alle Schichten mit einbezogen werden müssen.

In einer mehrschichtigen Architektur befinden sich normalerweise verschiedene Komponenten-Arten. Dazu gehören Steuerungskomponenten und fachliche Komponenten. Dass wesentliche Teile der Geschäftslogik nicht in der Geschäftsschicht ausgeführt werden, hat dazu beigetragen, dass diese Komponenten-Arten nicht im Brokerage-System identifiziert werden können.

In einigen Fällen müssen vom Client innerhalb eines Anwendungsfalls mehrere Komponenten aufgerufen werden. Der Client muss folglich die Geschäftslogik implementieren, um zu entscheiden, welche Komponenten aufzurufen sind. Genau das ist aber Aufgabe einer Anwendungsfallsteuerung-Komponente, mit der Clients der Geschäftsschicht ausschließlich kommunizieren sollten.

Die Steuerungsebene hat wesentliche Bedeutung für die Sicherheit des Systems. In einigen Fällen werden zwar noch Informationen von anderen Komponenten eingeholt, im Grunde werden Aufrufe aber nur an die Kernebene weitergeleitet, so dass man nicht von einer fachlichen Steuerung sprechen kann.

Die Kernebene kann weder als Steuerungskomponente noch als fachliche Komponente bezeichnet werden. Im Regelfall wird eine Stored Procedure aufgerufen, die den größten Teil der Geschäftslogik ausführt.

6 Betrachtung des Geschäftsvorfalls „Order aufgeben“

Die Kernfunktionalität des Systems besteht darin, dass Kunden die Möglichkeit erhalten, über das Internet Wertpapiere zu handeln. Insbesondere diese Funktionen müssen für Kunden in annehmbarer Zeit durchgeführt werden können. Kann das redesignte System diese Anforderung nicht erfüllen, ist es nicht sinnvoll, den gewählten Ansatz weiter zu verfolgen. Aus diesem Grund wird für die Realisierung des Prototypen exemplarisch der Geschäftsvorfall „Order aufgeben“ betrachtet.

Ein Geschäftsvorfall endet mit einem fachlich sichtbaren Ergebnis. Beim Aufgeben einer Order ist das die Weitergabe an die Börse. Da der Betreiber selbst keinen Zugang zur Börse hat, tritt die Wertpapiersammelbank als Zwischenkommissionär ein. Das Annehmen und Weiterleiten von Aufträgen wird unabhängig voneinander durchgeführt. Zeitkritisch ist nur der Vorgang, in dem ein Auftrag angenommen wird. Aus diesem Grund wird für die Realisierung des Prototypen nur das Annehmen von Aufträgen berücksichtigt. Die Kommunikation mit dem Hostsystem der Wertpapiersammelbank wird nicht mit einbezogen.

Für den Geschäftsvorfall werden detailliert die Anforderungen beschrieben, um deren Komplexität deutlich zu machen. Das Design des bestehenden Brokerage-Systems wird für diesen konkreten Fall noch mal betrachtet und bewertet, um die allgemein beschriebenen Schwächen zu verdeutlichen.

6.1 Anforderungen

Zunächst werden vom Kunden alle für den Auftrag erforderlichen Informationen angegeben. Die Angabe eines Börsenplatzes ist erst später erforderlich. Das System muss sicherstellen, dass das Papier für den Kunden in der angegebenen Menge am angegebenen Termin gehandelt werden kann. Wenn der Handel mit der Mengenangabe nicht möglich ist, muss sie vom Kunden auf einen möglichen Wert angepasst werden. Das Ausführungsdatum ist nötigenfalls vom System anzupassen. Beim Kauf kann es erforderlich sein, dass der

Kunde den Handel zusätzlich bekräftigen muss. Um die Order aufzugeben, muss der Kunde den Handel rechtskräftig machen.

Orders können auch durch Mitarbeiter des Betreibers für Kunden angelegt werden. Mitarbeiter handeln im Auftrag des Kunden und müssen Vorgänge gegenüber dem System nicht besonders bestätigen.

Aktienkauf

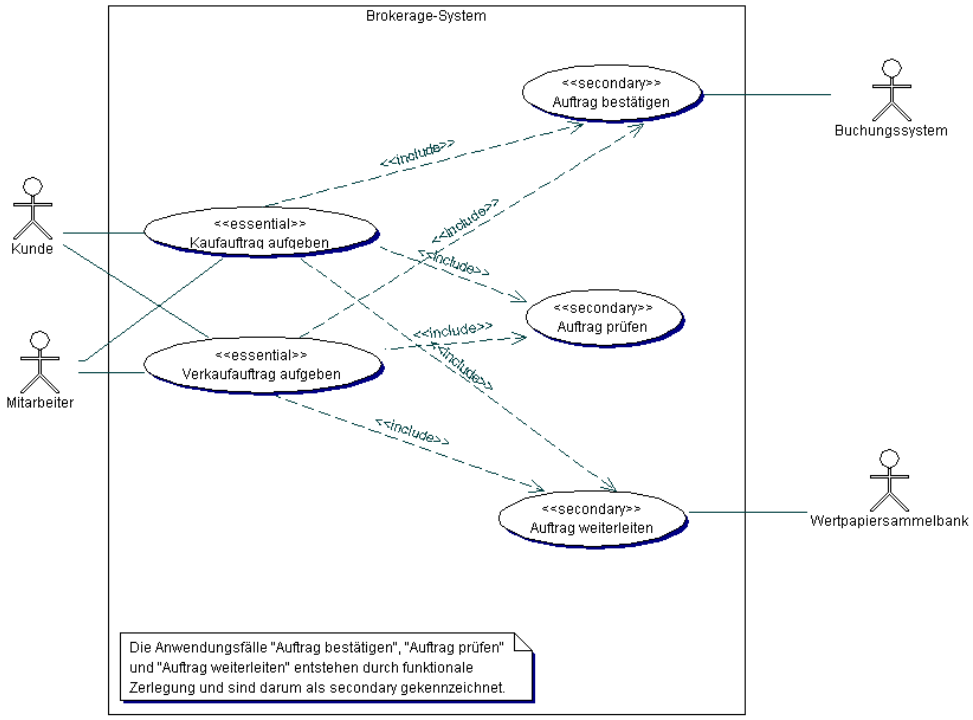

Abbildung 9: Geschäftsanwendungsfalldiagramm

Um einen Aktienkauf durchzuführen, sind immer WKN, Stückzahl bzw. Nominale und Orderzusatz anzugeben. Ein Auftrag muss mit einem Gültigkeitsdatum befristet werden. Standardmäßig erfolgt der Handel mit fortlaufender Notierung (variable). Der Kunde muss explizit angeben, dass er den Handel zum Kassakurs (Auktion) wünscht. Orders sind bestens auszuführen, wenn kein Limit (Limitpreis oder Stop Buy-Preis) festgelegt wird.

Vom System sind nur syntaktisch korrekte Eingaben zu akzeptieren. Andernfalls ist ein Hinweis auszugeben und die Verarbeitung abzubrechen. Die WKN ist eine Zahl mit genau sechs Stellen. Stückzahlen sind als positive ganze Zahlen, prozentuale Anteile als eine Zahl mit einer Vor- und bis zu vier Nachkommastellen anzugeben. Für Stücke und Nominale ist der Wert null nicht

zulässig. Das Datum ist im Format „TT.MM.JJJJ“ anzugeben und darf nicht in der Vergangenheit liegen. Das Limit ist als eine Zahl mit beliebig vielen Vor- und bis zu zwei Nachkommastellen einzugeben. Liegt der aktuelle Kurs unterhalb von zehn Eurocent, darf die Limitangabe bis zu drei Nachkommastellen haben.

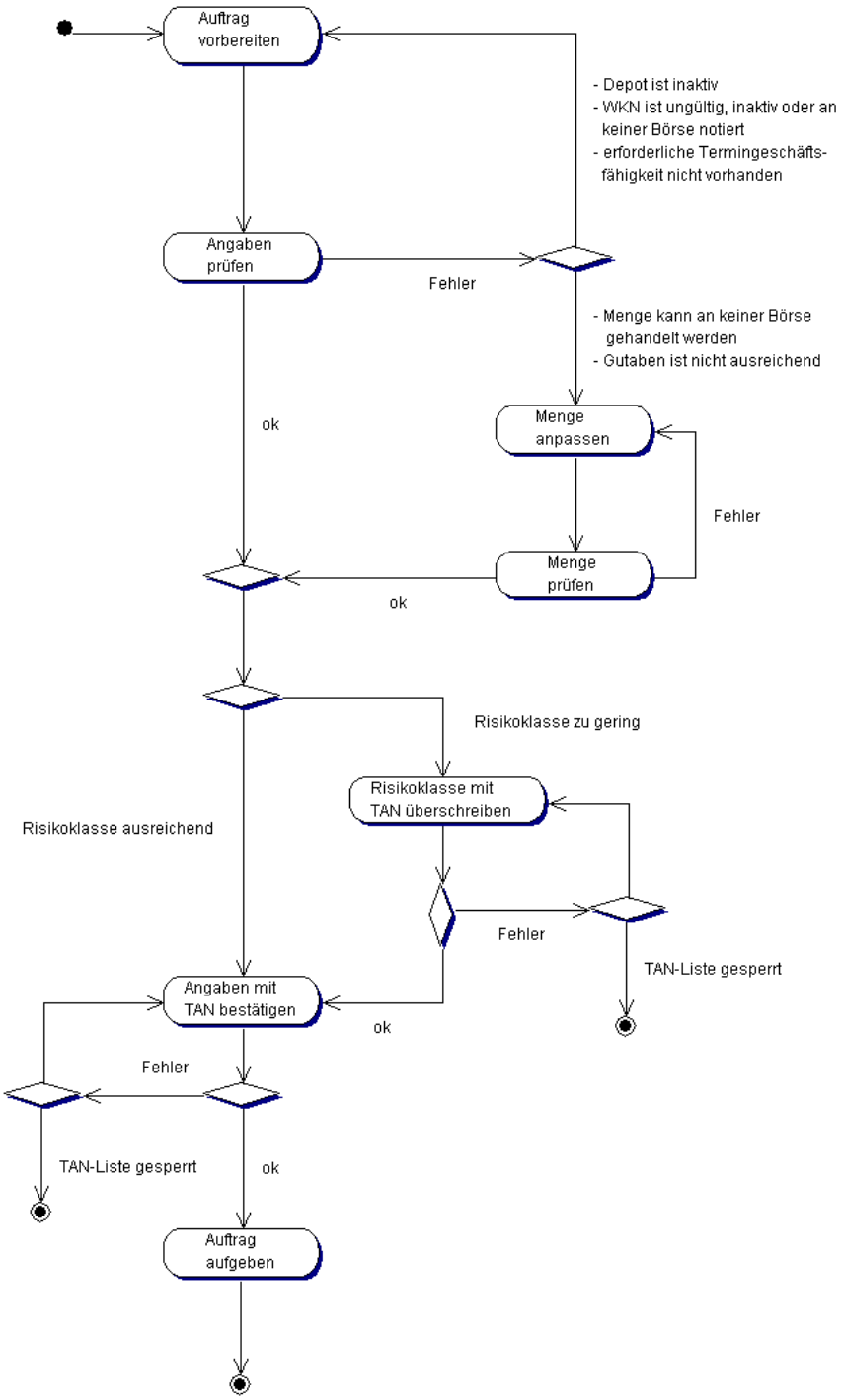

Abbildung 10: Aktivitätsdiagramm für den Geschäftsfall „Kaufauftrag aufgeben“

Für eine Kauforder sind verschiedene Dinge zu prüfen. Das Depot des Kunden darf nicht inaktiv sein. Die WKN muss dem System bekannt sein und der Betreiber des Systems muss den Handel mit dem Papier gestatten. Das Produkt darf nicht inaktiv sein und muss an mindestens einer Börse gelistet sein. Bei einem Verstoß sind die Prüfungen zu beenden, und dem Kunden ein Hinweis anzuzeigen.

Ist das Produkt nicht in der gewünschten Stückelzahl handelbar, ist diese im Rahmen der Dispositionsprüfung anzupassen. Der Kunde ist auf die Anpassung hinzuweisen.

Bei der Dispositionsprüfung ist der voraussichtliche Kaufpreis mit dem Kontostand des Verrechnungskontos zu vergleichen. Der Kaufpreis errechnet sich bei Stückkursen aus dem Ausführungskurs multipliziert mit der Anzahl der Stücke (Ausführungskurs × Anzahl der Stücke). Bei Prozentkursen berechnet er sich aus dem Ausführungskurs multipliziert mit dem Prozentwert dividiert durch 100 (Ausführungskurs × Prozentwert / 100). Den aktuellen Kontostand liefert das Buchungssystem des Betreibers. Es ist nur das Guthaben zu berücksichtigen, nicht der Verfügungsrahmen der Kunden. Vom Guthaben sind die Kosten für offene und die Kosten für ausgeführte, dem Verrechnungskonto noch nicht gut geschriebene Kauforders abzuziehen sowie die Erträge für ausgeführte, dem Verrechnungskonto noch nicht gut geschriebene Verkauforders hinzuzufügen.

Der wahrscheinliche Ausführungskurs ist auf Basis des aktuellen Kurses und der Limitangabe zu ermitteln. Ist kein Limit angegeben, ist der aktuelle Kurs zu nehmen, da die Ausführung nicht eingeschränkt ist. Liegt das Limit unterhalb des aktuellen Kurses, so ist das Limit zu nehmen, da dies der höchste Kurs ist, bei dem die Order ausgeführt wird. Im umgekehrten Fall ist der aktuelle Kurs zu nehmen. Wird ein Limit für Stop Buy angegeben, das unter dem aktuellen Kurs liegt, so ist der aktuelle Kurs heranzuziehen, da das Börsensystem welches die Order ausführt, nicht berücksichtigt, ob der Kurs fällt oder steigt und die Order sofort ausführt. Ist das Stop Buy-Limit größer als der aktuelle Kurs, so ist das Limit für die Berechnung zu benutzen, da erst ab diesem Kurs die Order tatsächlich ausgeführt wird.

Um die Berechnung mit einem möglichst stabilen Kurs durchzuführen, ist die Kursanfrage an einer Börse durchzuführen, an der möglichst viel gehandelt wird. Dabei ist folgende Rangfolge einzuhalten: Frankfurt, Düsseldorf, Berlin, München, Stuttgart, Hamburg, Bremen, Hannover, XETRA. Der Kurs ist u. U. in Euro umzurechnen, da die Währung des Verrechnungskontos in Euro lautet.

Verfügt der Kunde nicht über genügend Geld, so sind Stücke bzw. Nominale auf den größtmöglichen Wert anzupassen. Gleiches gilt, wenn die gewünschte Stückelung nicht handelbar ist. Der Kunde bekommt dann die Möglichkeit die Order zu ändern.

Der Betreiber ist in der Lage, die Dispositionsprüfung für das gesamte System abzuschalten. Es ist dann nicht zu prüfen, ob genügend Geld zur Verfügung steht. Kunden dürfen dann keine Orders aufgeben. Sie erhalten einen entsprechenden Hinweis. Orders können dann nur noch von Mitarbeitern des Betreibers aufgeben werden.

Kann die Dispositionsprüfung nicht durchgeführt werden, weil der aktuelle Kontostand oder der aktuelle Kurs nicht ermittelt werden konnten, so ist der Vorgang abzubrechen und dies durch einen entsprechenden Hinweis dem Kunden anzuzeigen. Wenn der aktuelle Kurs nicht ermittelt werden kann, aber ein Limitbetrag angegeben ist, so ist die Dispositionsprüfung mit diesem Wert durchzuführen.

Handelt es sich um eine vinkulierte Namensaktie, so ist der Kunde darauf hinzuweisen.

Das Datum ist auf Gültigkeit zu prüfen und ggf. auf ein korrektes Datum anzupassen, so dass die Order ausgeführt werden kann. Dies ist dem Kunden anzuzeigen. Eine Order, die in einem Halbjahr aufgegeben wird, darf nicht im nächsten gehandelt werden. Eine Order darf nicht später als am letzten Tag des Folgemonats ausgeführt werden. Werden diese Grenzen überschritten, ist der Ausführungstermin auf einen Börsentag soweit vorzuziehen wie nötig. Ist die Ausführung einer Order nicht möglich, weil die Börse am Handelstag nicht öffnet oder bereits geschlossen hat, so ist der Termin auf den nächsten Börsentag zu verlegen.

Name:	Auftrag aufgeben
Kurzbeschreibung:	Ein Kunde möchte einen Auftrag aufgeben.
Akteure:	Kunde, Mitarbeiter, Wertpapiersammelbank
Auslöser:	Kunde möchte einen Auftrag aufgeben
Vorbedingungen:	Der Kunde hat sich erfolgreich gegenüber dem System autorisiert.
eingehende Informationen:	WKN bzw. Depotposition, Menge, Limit, Orderzusatz, Ausführungsdatum
Ergebnisse:	Auftragsannahme wird bestätigt
Nachbedingungen:	Der Auftrag ist zur Ausführung an die Wertpapiersammelbank weitergeleitet worden.
Invarianten:	Der Kunde verfügt über ein Verrechnungskonto bei einem bestimmten Kreditinstitut. Der Kunde hat sich erfolgreich gegenüber dem System autorisiert.
essentieller Ablauf:	1. Auftrag vorbereiten 2. Angaben prüfen 3. Angaben bestätigen 4. Auftrag aufgeben
Ausnahmen:	zu 2. Die Prüfungen schlagen fehl, weil - das Depot inaktiv ist, - die WKN ungültig ist, - das Wertpapier nicht gehandelt werden darf, - das Wertpapier an keine Börse notiert ist, - die gewünschte Stückzahl nicht handelbar ist, - die Dispositionsprüfung nicht durchgeführt werden konnte, - sich nicht genügend Geld auf dem Verrechnungskonto befindet, - das Ausführungsdatum nicht angepasst werden kann, - die Risikoklasse zu gering ist. zu 3. Die angegebene TAN ist ungültig.

Abbildung 11: Geschäftsfallbeschreibung "Kaufauftrag aufgeben“

Der Betreiber teilt Kunden und Wertpapiere in Risikoklassen ein. Papiere werden für jeden Börsenplatz, an dem sie notiert sind, klassifiziert. Diese differenzierte Einteilung kann durch eine generelle Einstufung durch den Betreiber überschrieben werden. Dies dient dazu, ein Produkt zeitweilig in eine andere Risikoklasse einzuordnen. Die Risikoklasse des Kunden muss ausreichend sein, um das Papier handeln zu können. Schlägt die Prüfung fehl, so ist dem Kunden anzuzeigen, dass seine Risikoklasse für diesen Handel nicht ausreicht. Er erhält die Möglichkeit, ihn trotzdem auf eigenen Wunsch durchzuführen. Das ist möglich, indem eine TAN angegeben wird. Besitzt der Kunde keine

Termingeschäftsfähigkeit, obwohl das Produkt dies erfordert, so ist dies durch einen entsprechenden Hinweis anzuzeigen. Ein Handel dieses Papiers ist dann nicht erlaubt.

Sind die Prüfungen positiv verlaufen, wird dem Kunden ein voraussichtlicher Kaufpreis und eine Liste mit Börsenplätzen angezeigt, an denen das Papier gehandelt werden kann. Der Kunde gibt einen Börsenplatz an und muss das Geschäft durch Eingabe einer TAN rechtkräftig machen. Ist während der Prüfungen das Datum geändert worden, so erhält der Kunde einen Hinweis, aber keine Änderungsmöglichkeit. Aufträge dürfen nur angenommen werden, wenn sie auch ausgeführt werden können.

Aufträge sind in der Datenbank des Brorekage-Systems zu speichern. In regelmäßigen kurzen Abständen sind die Daten neuer Aufträge an die Wertpapiersammelbank zu schicken. Die Daten an die Wertpapiersammelbank müssen auch beinhalten, ob die Risikoklasse überschrieben wurde und welcher Geldbetrag dem Kunden für weitere Käufe nicht zur Verfügung steht, so lange der Auftrag nicht ausgeführt ist.

Verkauforder

Wird eine Verkauforder aufgegeben, so ist anstelle der WKN eine Depotposition zu spezifizieren. Aus dieser ergibt sich eine WKN sowie die Anzahl maximal handelbarer Stücke bzw. Nominale. Ansonsten gelten die gleichen Anforderungen wie beim Kauf. Der Kunde kann einen Limit- oder einen Stop Loss-Preis angeben.

Die Eingaben sind in gleicher Weise syntaktisch zu prüfen, wie beim Kauf.

Eine Verkaufsorder darf nicht ausgeführt werden, wenn das Depot oder die Depotposition gesperrt sind oder das Depot inaktiv ist. Eine Sperrung kann sowohl durch den Betreiber als auch durch die Wertpapiersammelbank erfolgen. Es ist eine Fehlermeldung anzuzeigen, wenn die Depotposition nicht im System enthalten ist.

Ein Verkauf kann ebenfalls nicht erfolgen, wenn das zugehörige Wertpapier nicht im System enthalten, inaktiv, gesperrt, oder nicht handelbar ist.

Es ist zu prüfen, ob der Kunde über genügend Stücke bzw. Nominale für den Verkauf verfügt. Hierfür ist von der bestätigten, verfügbaren Menge auszuge-

hen. Von dieser Menge ist die Summe der Stücke bzw. Nominale aller Orders zu dieser Position abzuziehen, die sich aus nicht ausgeführten Verkäufen und nicht ausgeführten Stornierungen von Käufen sowie Abbuchungen ergibt. Die verfügbare Menge ist um die Summe der Stücke bzw. Nominale der Teilausführungen dieser Orders zu erhöhen, die zu diesem Zeitpunkt noch nicht in der von der Wertpapiersammelbank bestätigten Menge enthalten sind. Ebenfalls hinzuzuaddieren ist die Summe der Stücke bzw. Nominale aller noch nicht bestätigten Ausführungen, aller Kauf- und Stop Loss-Orders sowie der Zubuchungen zu dieser Position. Dadurch wird Intradayfähigkeit erreicht, d.h. Aktien können am selben Tag erst ge- und dann verkauft werden. Die angegebene Stückzahl darf die handelbare Menge nicht überschreiten. Die Dispositionsprüfung muss immer erfolgen.

Die Prüfung des Ausführungsdatums erfolgt wie beim Kauf.

Dem Kunden ist der voraussichtliche Verkaufpreis und eine Liste mit möglichen Börsenplätzen anzuzeigen. Der Kunde muss wieder einen Börsenplatz auswählen und das Geschäft durch Eingabe einer TAN rechtskräftig machen. Analog zum Kauf ist ein vom System angepasstes Ausführungsdatum nicht änderbar. Vom System dürfen nur ausführbare Aufträge akzeptiert werden.

Auch Verkaufaufträge sind in gleicher Weise wie Kaufaufträge im eigenen System zu speichern und in regelmäßigen Abständen an die Wertpapiersammelbank zu schicken.

6.2 Umsetzung im Brokerage-System

Der Geschäftsvorfall „Order aufgeben“ wird in drei Prozesse unterteilt. Vorbereiten einer Kauforder, Vorbereiten einer Verkauforder und Ausführen einer Order. Die Geschäftsschicht stellt in mehreren Komponenten der Steuerungsebene verschiedene Dienste für Clients bereit. Um die Prozesse durchzuführen übernehmen Clients Aufgaben einer Anwendungsfallsteuerung und implementieren Geschäftslogik. Sie kommunizieren mit der Geschäftsschicht über Proxy-Klassen. Aus den bereits in Kapitel fünf angeführten Gründen handelt es sich bei allen Komponenten um Stateless Session Beans.

Ein Client prüft die Eingaben semantisch auf Korrektheit. Um sicherzustellen, dass der angegebene Limitwert den Geschäftsregeln entspricht, wird über die

Komponente ProductControlBean der aktuelle Kurs des zu handelnden Wertpapiers in Erfahrung gebracht. Diese Komponente bietet Dienste in Bezug auf Wertpapiere. ProductControlBean leitet Kursanfragen über die zugehörige Komponente in der Kernebene ProductBean an die Klasse QuoteFeeder weiter. Diese bindet das System an zwei Kurslieferanten an, um aktuelle Devisen- und Wertpapierkurse verfügbar zu machen. Die Klasse QuoteFeeder benutzt intern einige weitere Klassen, um auch bei Ausfall eines der Lieferanten die Kursversorgung sicherzustellen.

Die Eingabe für das Gültigkeitsdatum wird an die Proxy-Klasse SystemInfoProxy übergeben. Sie stellt sicher, dass die Eingabe ein in der Zukunft liegendes Datum ist.

Die Komponente SecurityControlBean bietet Dienste im Bereich Sicherheit an. Über diesen Bereich führen Clients TAN-Prüfungen durch. Wenn der Kunde eine TAN angibt, um ein Geschäft rechtskräftig zu machen oder um sich über seine zu niedrige Risikoklassifizierung hinwegzusetzen, muss dieser Dienst in Anspruch genommen werden. SecurityControlBean stellt sicher, dass der Client über eine gültige Session verfügt und leitet den Aufruf an die zugehörige Komponente SecurityBean weiter. Die eigentliche Überprüfung findet in einem Hostsystem des Betreibers statt. Dieses ist durch die Komponente OLSWrapperBean an das Brokrerage-System angebunden. SecurityBean leitet den Aufruf dorthin weiter.

Im Kern der Prozesse werden vom Client die Dienste *checkBuyOrder()*, *checkSellOrder()* und *execOrder()* der Komponente OrderControlBean genutzt. Die Methoden *checkBuyOrder()* und *checkSellOrder()* stellen sicher, dass der Handel grundsätzlich möglich ist. Sie bekommen in einem Datentransferobjekt die Angaben des Auftrags übergeben. Darin sind WKN, Menge, Limit, Orderzusatz und Ausführungsdatum enthalten. Beim Kauf wird zusätzlich angegeben, ob der Kunde seine Risikoklasse überschrieben hat; beim Verkauf wird die Depotposition, von der verkauft werden soll, spezifiziert. Alle vorgeschriebenen Prüfungen werden durchgeführt und das Ergebnis in einem Datentransferobjekt zurückgegeben. Darin sind Börsenplätze, an denen der Handel möglich ist, die Beschreibung des Wertpapiers, eventuell das angepasste Ausführungsdatum, der voraussichtliche Wert der Order, ggf. wie viele Stücke bzw.

Nominale maximal gehandelt werden können und Hinweise sowie der Status enthalten. Die Hinweise sind eine Liste von Codes, die über das Ergebnis der Prüfung informieren. Darüber wird beispielsweise angegeben, warum das Ausführungsdatum angepasst wurde. Der Status zeigt in codierter Form an, ob ein Systemfehler aufgetreten ist. Die Methode *execOrder()* wird aufgerufen um eine bereits geprüfte Order aufzugeben. Ihr werden die gleichen Angaben wie beim Vorbereiten übergeben. Es wird nochmals sichergestellt, dass der Auftrag ausgeführt werden kann.

Bei allen Methoden wird geprüft, ob der Client über eine gültige Session verfügt. Kaufaufträge erfordern, dass der Kontostand vorliegt. Diesen bereitzustellen, ist Aufgabe des Bereichs Konto. Der Kontostand wird durch die Komponente AccountBean der Geschäftsschicht für OrderControlBean verfügbar gemacht. AccountBean leitet den Aufruf an die Komponente OLSWrapperBean in der Schnittstellenebene weiter, da der Kontostand beim Buchungssystem angefragt werden muss. OrderControlBean fügt den Kontostand in das Datentransferobjekt ein. Die Anfrage wird zusammen mit den Sessiondaten an die zugehörige Komponente OrderBean in der Kernebene weitergeleitet. Dort wird nochmals der aktuelle Kurs über die Klasse QuoteFeeder angefragt, um den Ausführungskurs bestimmen zu können. Beim Vorbereiten eines Auftrags werden der Inhalt des Datentransferobjektes sowie der Ausführungskurs an Stored Procedures übergeben. Innerhalb dieser erfolgen alle Prüfungen, für die persistente Daten benötigt werden. Beim Ausführen einer Order wird die Klasse OrderRouter aufgerufen. Sie hat die Aufgabe, sämtliche Aufträge in der Datenbank persistent zu machen. Sie ruft Stored Procedures auf, um neue Aufträge anzulegen und bestehende zu ändern oder löschen.

Ist eine Order erfolgreich im Sinne der Geschäftsregeln geprüft worden, wird dies in der Session hinterlegt. Nur wenn das der Fall ist, kann die Methode *execOrder()* ausgeführt werden. Ist ein Auftrag ausgeführt worden, wird dies ebenfalls in den Sessiondaten gekennzeichnet. Auf diese Weise wird erreicht, dass das Vorbereiten und Annehmen einer Order keine unabhängigen Dienste sind, sondern immer einen zusammenhängenden Vorgang bilden. Es kann jedoch immer nur genau ein Auftrag zur Zeit aufgegeben werden.

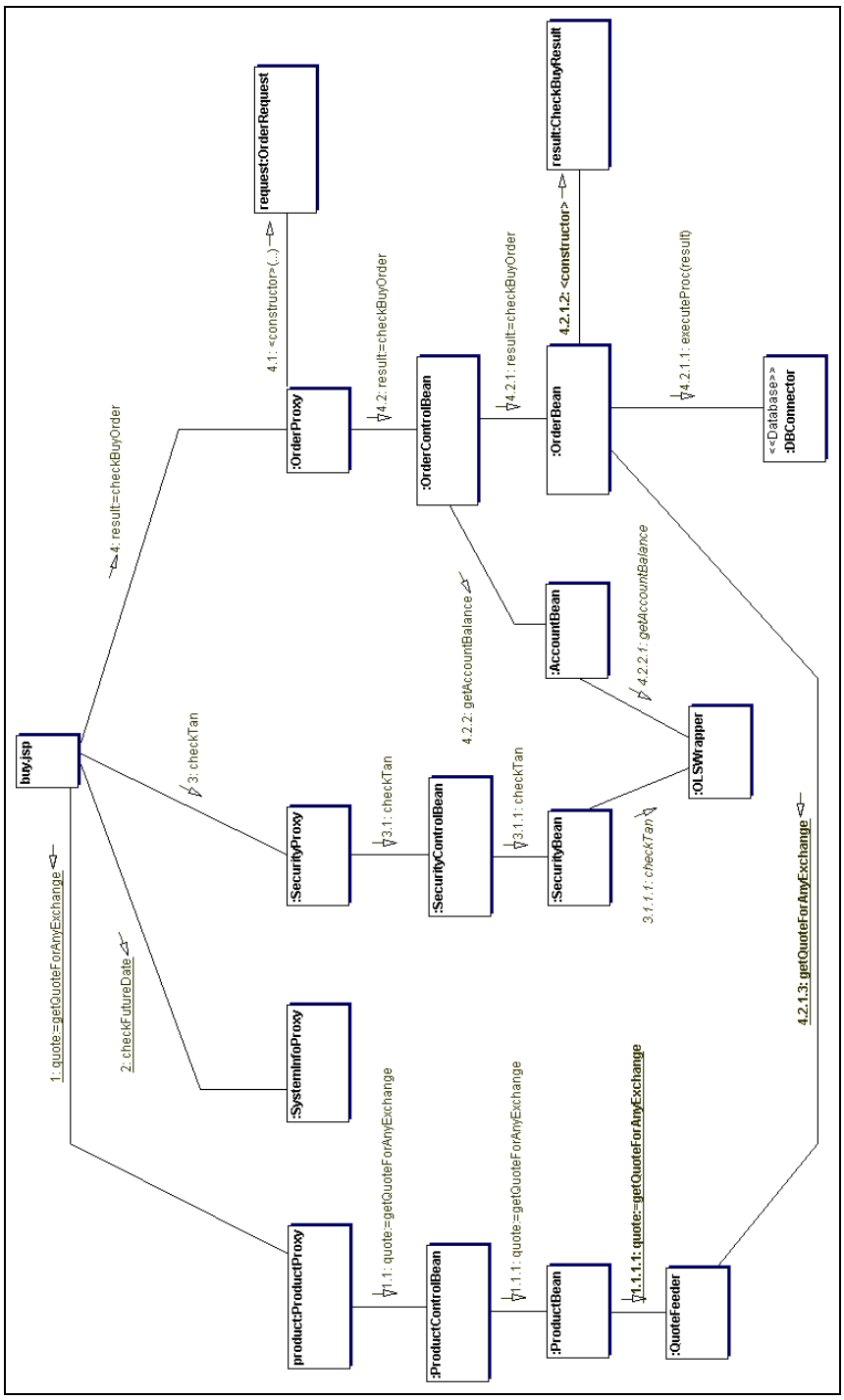

Abbildung 12: Kommunikation beim Vorbereiten einer Kauforder über das Internet

6.3 Bewertung

Anhand der Umsetzung des Geschäftsvorfalls lassen sich konkret die bereits im vorherigen Kapitel angesprochen Kritikpunkte aufzeigen.

Der Client der Geschäftsschicht ist in die Bearbeitung eines Anwendungsfalls mit einbezogen. Er muss mit drei verschiedenen Komponenten kommunizieren und übernimmt Aufgaben einer Anwendungsfallsteuerung. In Web- und

Clientschicht sind also Geschäftslogik enthalten. Wie bereits angeführt, soll sich die Geschäftslogik in dieser Architektur ausschließlich in der Geschäftsschicht befinden.

Die Komponenten der Geschäftsschicht sind weder Steuerungskomponenten noch fachliche Komponenten. Sie führen wenig Geschäftslogik aus und leiten Aufrufe häufig nur an eine andere Komponente weiter.

Die Datenbankzugriffe finden direkt innerhalb der Beans statt. Dadurch entsteht eine starke Kopplung zwischen den Beans und der Datenbank.

In der detaillierten Betrachtung ist eine weitere Schwäche des Systems offensichtlich geworden. Clients der Geschäftsschicht rufen nur Stateless Session Beans auf. Es ist demnach zu erwarten, dass alle Methoden unabhängige Dienste bereitstellen. Tatsächlich wird aber in den Sessiondaten der Zustand des Auftrags abgelegt. Das ist ein Widerspruch in sich. Wenn ein Kunde mehrere Aufträge parallel vorbereitet, kann nur der zuletzt vorbereitete auch tatsächlich ausgeführt werden. Zwischen Client und Geschäftsschicht findet also eine zustandsbasierte Kommunikation statt, die ihren Zustand aber in einer bestimmten Situation verliert.

7 Prototyp

7.1 Anforderungen

Für das System wird vorgegeben, dass es über mehrschichtige Hard- und Softwarearchitektur verfügen soll. Die Softwarearchitektur soll auf der J2EE-Plattform basieren. Dies wird durch eine entsprechende Serverarchitektur unterstützt. Das System muss in einem Servercluster lauffähig sein, damit es skalierbar ist. Die zentrale Anforderung besteht darin, die hohe Abhängigkeit von der Datenbank zu reduzieren. Dies gilt es, durch eine konsequente Anwendung des in der J2EE-Architektur definierten Schichtenmodels zu erreichen. Darüber hinaus soll so Unabhängigkeit und Austauschbarkeit der einzelnen Schichten erzielt werden. Die benötigte Umgebung liefert der Application Server BEA WebLogic 6.1. Alle durch J2EE und Application Server gebotenen Möglichkeiten können genutzt werden. Für den Prototyp steht die Geschäftslogik des Geschäftsvorfalls „Order aufgeben“ im Vordergrund. Andere Bereiche wie Sessionverwaltung und Anbindung an Hostsysteme und Kommunikation mit diesen sind zu vernachlässigen. Grafische Oberfläche und Webschicht sind nicht Bestandteil des Prototypen. Änderungen am Datenmodell sind aus betriebtechnischen Gründen nur für Daten möglich, die von Netlife selbst bereitgestellt und gepflegt werden. Die Ergebnisse des Prototypen dienen als Entscheidungsgrundlage für die Weiterentwicklung der konsolidierten Brokerage-Systeme. Somit sind in den Prototypen möglichst viele Neuerungen in Enterprise JavaBeans 2.0 einzubringen.

7.2 Entwurf

Um den Entwurf durchführen zu können, muss die Umgebung des Systems berücksichtigt werden. Ganz wesentliche Bedeutung hat die Anwendungsarchitektur. Aus ihr ergeben sich Möglichkeiten und Einschränkungen für die Problemlösung.[12] In einem Prototypen sollte ermittelt worden sein, dass die Anwendungsarchitektur für die Lösung geeignet und angemessen ist. In diesem Fall besteht die Aufgabe darin, einen Prototypen zu entwickeln, an den konkrete Anforderungen gestellt werden. Im Entwurf wird für eine mehrschichtige

[12] Vgl. Oestereich, B., Objektorientierte Softwareentwicklung, 2001, S. 145

Komponentenarchitektur die Geschäftsschicht modelliert. Dabei wird berücksichtigt, dass die Umsetzung mit EJB 2.0 erfolgt und dass der Prototyp in einem WebLogic 6.1 Servercluster eingesetzt wird.

7.2.1 Geschäftslogik

Der Geschäftsvorfall „Order aufgeben“ wird mit mehreren Komponenten umgesetzt. Für die beiden Anwendungsfälle „Kaufauftrag aufgeben“ und „Verkaufauftrag aufgeben“ gibt es die Steuerungskomponente OrderFacadeBean. Als Fassade[13] verbirgt sie ein Subsystem und ist für entfernte Clients über ein Remote-Interface zugreifbar. Innerhalb des Subsystems befinden sich verschiedene fachliche Komponenten, welche die Geschäftslogik enthalten. Diese benutzen weitere fachliche Komponenten, die Daten modellieren.

Der Ablauf der Anwendungsfälle ist fast gleich. Zunächst wird der Auftrag vorbereitet, u. U. muss die Mengeangabe geändert werden und dann wird die Order aufgegeben. Für das Aufgeben muss zwischen Kunde und Mitarbeiter unterschieden werden, da Mitarbeiter Vorgänge gegenüber dem System nicht rechtskräftig machen müssen. Beim Kauf kann es zusätzlich erforderlich sein, dass sich der Kunde über eine zu geringe Risikoklassifizierung hinwegsetzt. Es liegt nahe, beide Anwendungsfälle mit Stateful Session Beans zu modellieren. Jeder Schritt könnte mit einer Methode dargestellt werden. Stateful Session Beans besitzen gegenüber Stateless Session Beans entscheidende Nachteile, so dass davon abgesehen wird, sie einzusetzen. Ausschlaggebend ist, dass Stateless Session Beans gepoolt werden können, wohingegen Stateful Session Beans bis zum Ende eines Anwendungsfalls ständig Ressourcen beanspruchen. Ein weiterer gewichtiger Grund ist, dass bei einem Serverabsturz der Zustand verloren geht. Durch die replica-aware Stubs des WebLogic Servers wird dieses Risiko zwar stark reduziert, es bleibt aber bestehen. Es wäre fatal, wenn ein Kunde einen Auftrag ein zweites Mal aufgeben würde, weil es beim ersten Mal in einem ungünstigen Moment zu einem Serverabsturz gekommen ist.

Die Nachteile von Stateful Session Beans zu vermeiden, hat Auswirkungen auf das Design. Alle benötigen clientspezifischen Daten müssen immer an Stateless Session Beans übergeben werden, da sie unabhängige Dienste bereitstel-

[13] Vgl. Gamma, E./Helm, R./Johnson, R./Vlissides, J., Entwurfsmuster, 1996, S. 212

len. Die Ressourcen des Servers zu schonen, führt zu einer höheren Netzwerkbelastung. Es ist problematisch, beim Kaufauftrag die Risikoklasse des Kunden zu überschreiben. Nach den Geschäftsregeln darf dies kein unabhängiger Dienst sein, denn ein Kunde darf seine Risikoklasse immer nur für ein bestimmtes Geschäft überschreiben. Also ist es nötig, diesen Schritt in einen anderen zu integrieren. Hierfür kommt nur der letzte Schritt, das Aufgeben einer Order, in Frage, da in einer Kauforder die Information enthalten sein muss, ob der Kunde sich über eine zu geringe Risikoklasse hinweggesetzt hat.

Die Lösung mit unabhängigen Diensten macht es erforderlich, bei jedem Schritt die Prüfungen durchzuführen, da sichergestellt werden muss, dass nur Aufträge aufgegeben werden, die auch ausgeführt werden können. Das ist aber kein Nachteil, denn das wäre auch bei einem Einsatz von Stateful Session Beans notwendig, da sich zwischen den Schritten die den Prüfungen zugrunde liegenden Daten geändert haben können.

Client und Anwendungsfallsteuerung kommunizieren normalerweise über Datentransferobjekte miteinander[14]. Es wird jedoch davon abgesehen, die Parameterübergabe vom Client an die Steuerungskomponente mit Datentransferobjekten durchzuführen. Für jede Methode wird ein Datentransferobjekt benötigt, um Daten an die Anwendungsfallsteuerungskomponente zu übergeben und evtl. ein weiteres, um Daten an den Client zurückzugeben. Die Anzahl der Datentransferobjekte übersteigt schnell die Anzahl der Komponenten um ein Vielfaches und kann immens hoch sein. Das Design der Anwendung wird dadurch insgesamt unübersichtlich und die Wartbarkeit sinkt. Um dieser Entwicklung entgegenzuwirken werden Datentransferobjekte nur eingesetzt, um Daten an den Client zurückzugeben.

Die Steuerungskomponente OrderFacadeBean stellt sicher, dass der Client über eine gültige Session verfügt und benutzt die fachlichen Komponenten OrderBuilderBean, OrderCheckerBean und OrderRouterBean. Um einen bestimmten Schritt durchzuführen, sammelt OrderBuilderBean alle Daten die nötig sind in einem Order-Objekt. Für Kauf und Verkauf können jeweils verschiedene Order-Objekte erstellt werden, um einen Auftrag vorzubereiten, die Mengen-

[14] Vgl. Oestereich, B.: Objektorientierte Softwareentwicklung, 2001, S. 146 f.

angabe zu ändern und einen Auftrag aufzugeben. In Order-Objekten sind einfache Datentypen, Objekte und Stubs von Entity Beans enthalten.

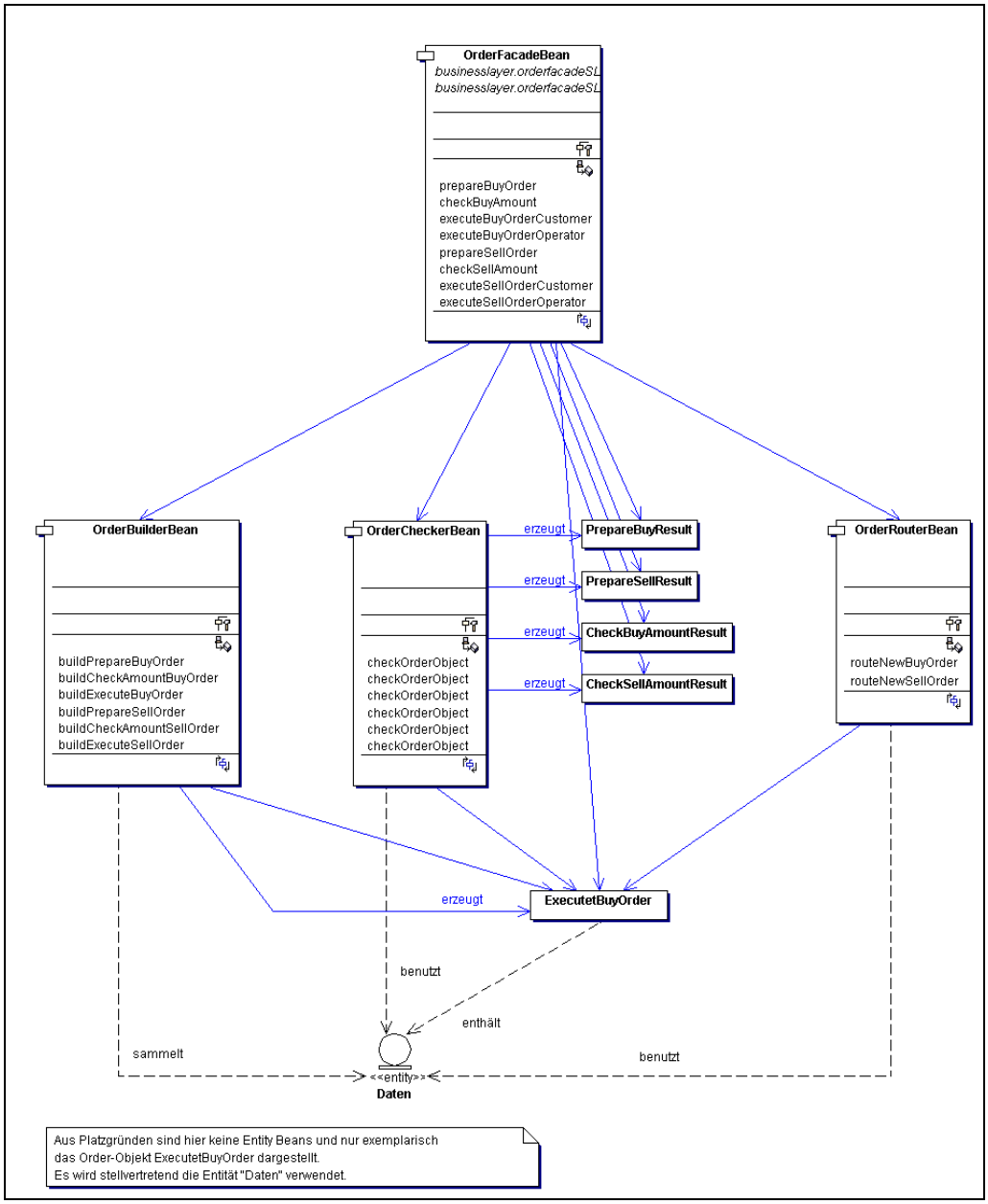

Abbildung 13: Vereinfachte Darstellung der Komponenten, die Geschäftslogik ausführen

Order-Objekte werden einer entsprechenden Methode von OrderCheckerBean übergeben, um Prüfungen durchzuführen. Alle für die Prüfungen benötigten Daten befinden sich in den Order-Objekten. Beim Vorbereiten einer Order und beim Ändern der Menge muss dem Client das Ergebnis der Prüfungen mitgeteilt werden. OrderCheckerBean erstellt für die verschiedenen Fälle Result-Objekte. Dabei handelt es sich um Datentransfer-Objekte, in denen nur einfa-

che Datentypen enthalten sind. Die Result-Objekte werden von OrderFacade-Bean an den Client zurückgegeben.

Soll ein Auftrag aufgegeben werden, so veranlasst OrderFacadeBean erforderliche TAN-Prüfungen. Geprüfte Order-Objekte werden an die Komponente OrderRouterBean übergeben. Die Komponente extrahiert die Daten aus dem Order-Objekt, um diese persistent zu machen und schließt so den Vorgang ab. Wenn sich die Geschäftsregeln ändern und neue Aufträge sofort an die Wertpapiersammelbank weitergeleitet werden sollen, kann dafür einfach eine weitere Bean-Klasse bereitgestellt werden. Über den Deployment Descriptor kann konfiguriert werden, welche Bean-Klasse zu verwenden ist. Die Implementierung anderer Komponenten kann dann unverändert bleiben. Es könnten sogar verschiedene Komponenten mit verschiedenen Bean-Klassen im Deployment Descriptor definiert werden, so dass sich die Anwendung in ihrem Verhalten an verschiedenen Situationen anpassen kann.

Die Komponenten OrderBuilderBean, OrderCheckerBean und OrderRouter-Bean bieten unabhängige Dienste an und können daher als Stateless Session Beans realisiert werden. Ihre Dienste werden nur von OrderFacadeBean in Anspruch genommen, so dass sie nur über ein Local-Interface verfügen. So kann der Overhead für Remote-Zugriffe eingespart werden. Entfernte Clients sollen und dürfen nicht mit den fachlichen Komponenten kommunizieren, denn entfernte Clients richten ihre Anfragen immer an eine Anwendungsfallsteuerung, in diesem Fall also an die Komponente OrderFacadeBean.

7.2.2 Abbildung der Daten

Für die Prüfungen werden Daten benötigt, die teils vom Kunden angegeben werden, teils von Hostsystemen verfügbar gemacht werden und teils in persistenter Form in einer relationalen Datenbank vorgehalten werden.

Der Kunde hat die Möglichkeit, seinen Auftrag mit einem Limit zu versehen. Die Angaben über Wert und Art (limitiert, Stopp loss usw.) werden in einem Limit-Objekt zusammengefasst. Ein Limit kann entscheiden, ob es in einem bestimmten Kontext, also für einen bestimmten Kurs, gültig ist.

Die Anbindung an Hostsysteme erfolgt ebenfalls durch Komponenten. Auf diese Weise kann flexibel auf geänderte Anforderungen reagiert werden. Wie

bei der Komponente OrderRouterBean können Anpassungen durch neue Bean-Klassen und Konfiguration im Deployment Descriptor erfolgen. Die anzubindenden Hostsysteme sind das Buchungssystem des Betreibers sowie zwei Kurslieferanten. Das Buchungssystem ist im bestehenden System durch die Komponente OLSWrapperBean angebunden. Diese Komponente wird auch im Prototypen verwendet. Da sie auf der EJB-Spezifikation 1.1 basiert, verfügt sie über ein Remote-Interface. Aus bereits aufgeführten Gründen wird dieses für den Prototyp durch ein Local-Interface ersetzt.

Im bestehenden System wird die Kursversorgung durch die Klasse QuoteFeeder sichergestellt. Sie kapselt den Mechanismus, um Anfragen an die verschiedenen Provider zu stellen. Dieses Verhalten kann mit Komponenten realisiert werden, indem jeder Kurslieferant durch eine Komponente angebunden wird und eine weitere Komponente die Logik implementiert, welche Komponente benutzt wird. Über den Deployment Descriptor könnte dann das Verhalten festgelegt werden. Eine andere Möglichkeit besteht darin, nur eine Komponente zu benutzen und die Anbindung der Kurslieferanten durch Anwendung des Abstact Factory-Pattern[15] zu realisieren. Da die Anbindung an Hostsysteme für den Prototypen keine Rolle spielt, wird der Einfachheit halber die Klasse QuoteFeeder als Bean-Klasse für die Komponente QuoteFeederBean verwendet. Sie wird als Stateless Session Bean realisiert und bietet ihre Dienste lokalen Clients an.

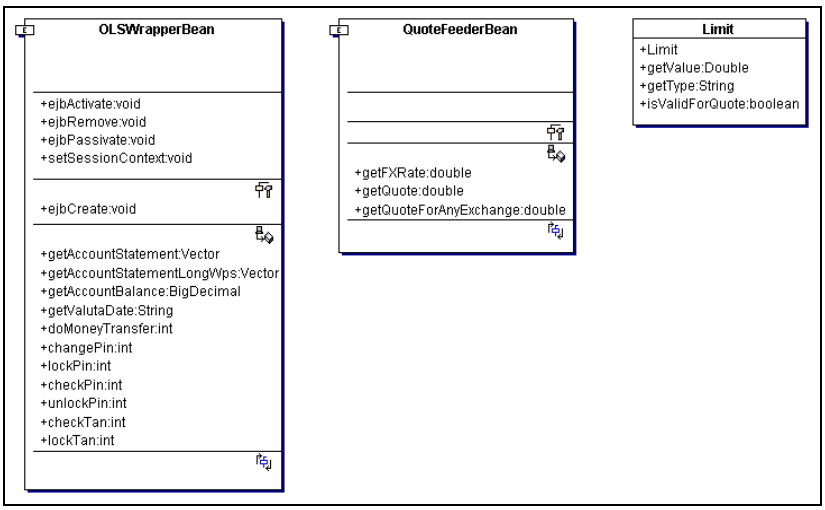

Abbildung 14: Vereinfachte Darstellung der Komponenten, die Hostsysteme anbinden, sowie der Klasse Limit

[15] Vgl. Gamma, E./Helm, R./Johnson, R./Vlissides, J., Entwurfsmuster, 1996, S. 65

Um persistente Daten zu modellieren sieht die EJB-Spezifikation Entity Beans vor. Der Prototyp muss auf Basis des bereits vorhandenen Datenmodells entworfen werden. In dem bestehenden Datenmodell spiegeln sich aber nur die Bedürfnisse der Daten wider. Für die Modellierung von Entity Beans muss neben den Daten auch deren Verhalten berücksichtigt werden.[16] Änderungen sind nur an wenigen Stellen möglich, so dass das Design der Entity Beans stark durch das Datenmodell vorgegeben und eingeschränkt wird.

Die Möglichkeiten der Modellierung von Entity Beans haben sich durch CMP 2.0 wesentlich verbessert. Bei Anwendungen, die auf der EJB-Spezifikation 1.x basieren, treten Performanceprobleme auf, da die Kommunikation mit Entity Beans über das Remote Interface erfolgt. Dadurch sind die Entwurfsmuster Value Object[17] und Aggregate Entity[18] entstanden. Bei Value Objects handelt es sich um einfache Java-Objekte, welche die Daten von Entity Beans aufnehmen. So kann die Anzahl der Zugriffe auf Entity Beans reduziert werden. Mit dem Aggregate Entity-Muster werden in einer Entity Bean abhängige persistente Objekte mit BMP abgebildet. Insbesondere durch das neue Locale Interface ist es jetzt möglich, fein-granulare Entity Beans zu modellieren und auf Value Objects zu verzichten.[19] Darum verfügen alle Entity Beans über das Locale Interface. Da der Container durch das Persistenzschema Datenbankzugriffe nun effizienter durchführen kann, und da für fein-granulare Entity Beans kein BMP erforderlich ist, wird für alle Entity Beans CMP verwendet. Dazu kommt, dass der WebLogic Server in der Lage ist, mit einem Datenbankzugriff eine Sammlung von Entity Beans zu erzeugen und nicht n+1 Zugriffe nötig sind, wie es bei BMP der Fall wäre.

---

[16] Roman, E./Ambler, S./Jewell, T., Mastering Enterprise JavaBeans Second Edition, S. 375
[17] Vgl. Sun Java Center J2EE™ Patterns – Value Object
[18] Vgl. Sun Java Center J2EE™ Patterns – Aggregate Entity
[19] Vgl. Marinescu, F., EJB Design Patterns, 2002, S 199 und S. 201

Folgender Teils des Datenmodells muss abgebildet werden:

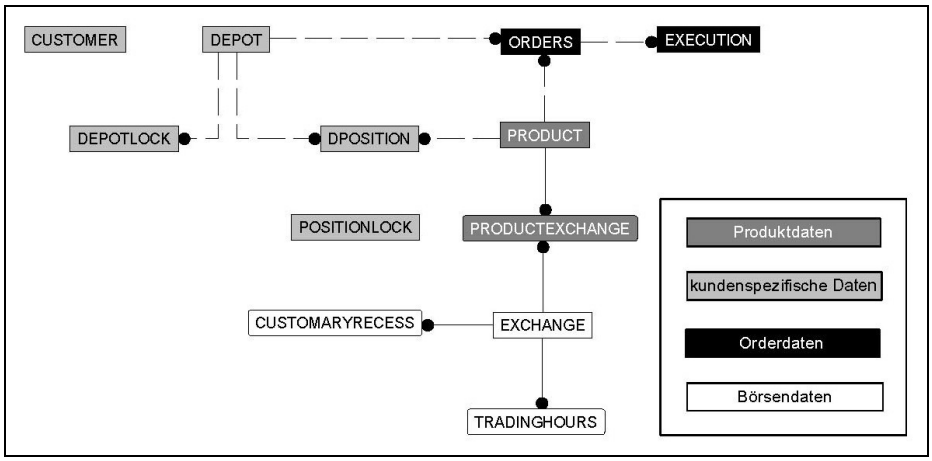

Abbildung 15: Abzubildender Teil des Datenmodells

Börsendaten

Börsendaten werden von Netlife bereitgestellt und gepflegt, so dass hier die Möglichkeit besteht, das Datenmodell den Bedürfnissen entsprechend anzupassen. Auf die Börsendaten wird nur lesend zugegriffen. Die Tabelle Exchange war nicht Bestandteil des bestehenden Systems. Sie wurde hinzugefügt, um die Börsenplätze sinnvoll als Entity Beans abbilden zu können. Im bestehenden System waren diese Daten in der Tabelle CodeTypes enthalten. Dort sind zum einen noch viele weitere Daten enthalten und zum anderen gibt es für jeden Kurslieferanten zwei Datensätze pro Börse. Die Tabellen TradingHours und CustomaryRecess sind aus dem bestehenden System übernommen und umbenannt worden. Ihre Bedeutung hat sich im Laufe der Zeit geändert, so dass die ursprünglichen Bezeichnungen DefaultExchangeCalendar und ExchangeCalendar nicht mehr aussagekräftig sind. Es wird davon abgesehen, die Tabelle CustomaryRecess als Entity Bean umzusetzen, da sie neben dem Fremdschlüssel nur ein Attribut für Börsenfeiertage enthält. Es gibt nur wenige Börsenfeiertage und durch die Geschäftslogik bedingt sind nur die Feiertage in einem ganz bestimmten Zeitraum relevant. Bei einer Lösung mit Entity Beans muss damit gerechnet werden, dass die finder-Methode in der überwiegenden Zahl der Fälle keine Daten liefern wird. Die relevanten Daten können jedoch einfach über SQL abgefragt und bedenkenlos zwischengespeichert werden, da sich die Börsenfeiertage nicht ändern. In der Implementierung muss dafür Sorge getragen werden, dass die relevanten Daten immer vorhanden sind. Fähigkeiten der En-

tity Beans werden hier nicht benötigt, so dass auf diese Weise das System entlastet wird. Zur Tabelle TradingHours hingegen wird eine entsprechende Entity Bean abgebildet. Die Tabellen TradingHours und CustomaryRecess referenzieren jeweils den Primärschlüssel von Exchange. Diese Beziehung wird auch auf Bean-Ebene modelliert, so dass die Komponente ExchangeBean die Börsenöffnungszeiten und die Börsenfeiertage verkapselt. ExchangeBean bietet Methoden an, mit denen festgestellt werden kann, ob eine Börse an einem bestimmten Tag geöffnet hat und ob an einer Börse eine bestimmte Wertpapierart zu einer bestimmten Tagenszeit gehandelt werden kann.

Produktdaten

Auf die Daten der Tabellen Product und ProductExchange wird ebenfalls nur lesend zugegriffen. Beide Tabellen werden als Entity Beans abgebildet, wobei ProductExchangeBean durch ProductBean gekapselt wird. Die Geschäftslogik erfordert, dass Clients Zugriff auf die Daten der Börsenplätze bekommen können. Die Beziehung der Tabellen ProductExchange und Exchange zueinander kann auch für die entsprechenden Komponenten modelliert werden. Die Komponente ProductBean ist in der Lage, mit ExchangeBean-Komponenten umzugehen, um so zu den zugehörigen ProductExchangeBean-Komponenten zu gelangen. Die Abhängigkeit der Komponenten ProductBean und ExchangeBean drückt sich im Datenmodell durch die Kreuztabelle ProductExchange aus.

Neben dem Zugriff auf die eigenen Daten bietet ProductBean Clients börsenplatzabhängige Operationen an. Clients können prüfen lassen, ob das Produkt überhaupt oder an einer bestimmten Börse gelistet ist. Sie können eine Sammlung mit Börsenplätzen erhalten, an der das Produkt gehandelt werden kann. Sie können bestimmen lassen, ob eine bestimmte Menge gehandelt werden kann, welches die kleinste handelbare Menge ist und über welche Risikoklasse ein Produkt verfügt. Damit ProductBean diese Dienste anbieten kann, verfügt ProductExchangeBean über eine finder-Methode, um die Börsenplätze ausfindig zu machen, an denen der Handel eines bestimmten Wertpapiers möglich ist.

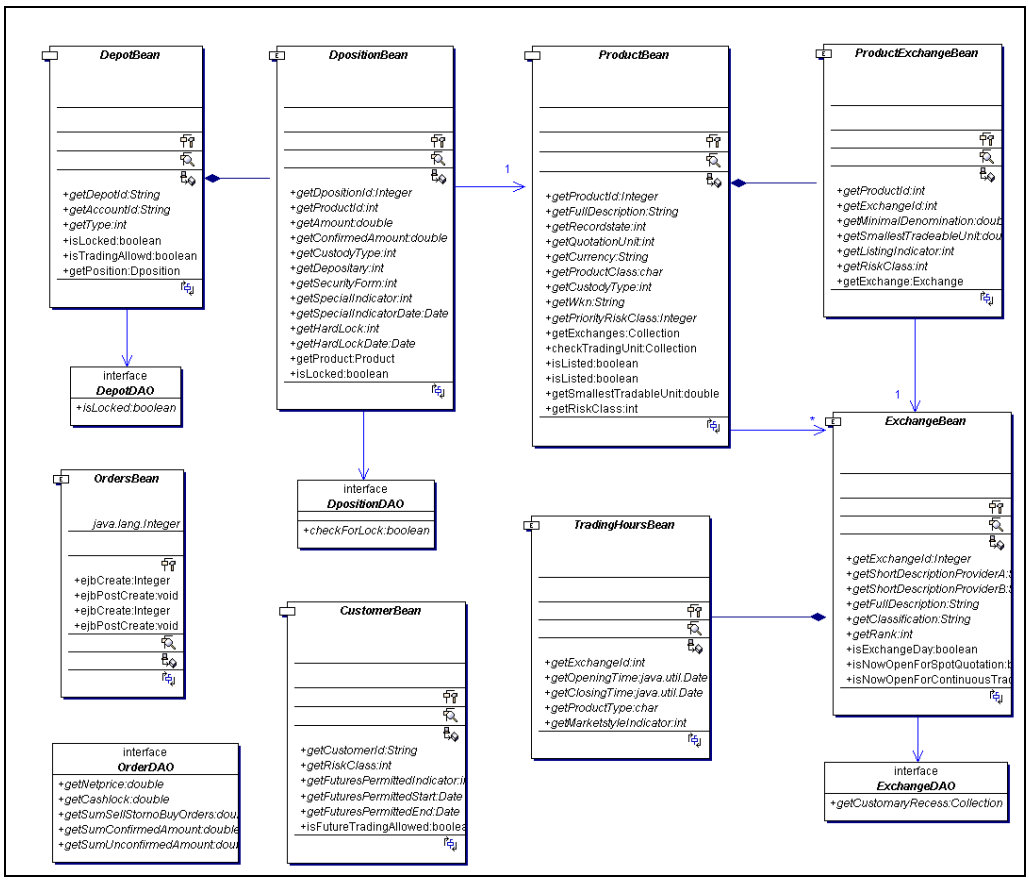

Abbildung 16: Vereinfachte Darstellung der Komponenten, die persistente Daten repräsentieren

Kundenspezifische Daten

Diese Daten umfassen die Tabellen Customer, Depot, DepotLock, Dposition und PositionLock, aus denen nur Informationen gelesen werden.
Zu der Tabelle Customer wird eine korrespondierende Entity Bean entworfen. Clients können die Daten von einer CustomerBean in Erfahrung bringen und prüfen lassen, ob ein Kunde an einem bestimmten Datum Termingeschäfte durchführen darf.

Die Geschäftslogik erfordert Zugriff auf die Daten der Tabellen Depot und Dposition. Bei den Tabellen DepotLock und PositionLock hingegen ist nur wichtig, ob dort ein Eintrag zu einem Depot bzw. zu einer Position vorhanden ist. Es wäre daher sinnvoll, den Tabellen Depot und Dposition ein weiteres Attribut Lock hinzuzufügen. Anpassungen am Datenmodell sind jedoch nicht möglich. Da es aber sinnlos ist, Daten durch Entity Beans abzubilden, die völlig ohne Belang sind, werden die benötigten Informationen analog zur Tabelle CustomaryRecess durch SQL-Abfragen verfügbar gemacht. Wäre es unter an-

deren Umständen doch sinnvoll DepotLock und PositionLock als Entity Beans abzubilden, könnten die Abfragen in home-Methoden manuell oder in select-Methoden durch den Container implementiert werden. Die Komponenten DepotBean und DpositionBean kapseln diese Information, und bieten eine entsprechende Methode an. Die Tabellen Depot und Dposition stehen in Beziehung zueinander, so dass es auch auf Komponentenebene möglich ist, Zugriff auf eine bestimmte Position zu bekommen. Der Zugriff auf Positionen wird nicht durch die Komponente DepotBean gekapselt, weil sonst für jeden Zugriff eine Positions-Id an DepotBean zu übergeben wäre, um die Position zu finden. Eine Position bezieht sich immer auf ein bestimmtes Produkt, so dass die Komponente DpositionBean über eine Methode verfügt, um Zugriff auf die entsprechende ProductBean-Komponente zu erhalten. Es wäre möglich, den Zugriff auf ProductBean zu kapseln, davon wird jedoch abgesehen, da nur bei Verkäufen über eine Position auf das zugehörige Produkt zugegriffen wird. Ein Teil der Geschäftslogik wäre dann doppelt zu implementieren.

Orderdaten

Die Orderdaten umfassen die Tabellen Orders und Execution. Aus beiden müssen Werte gelesen, außerdem in Orders auch neue Einträge gemacht werden.

Nach verschiedenen Kriterien müssen Summen über einzelne Felder gebildet werden, wofür die Tabellen teilweise miteinander zu verknüpft sind.

Die Abfragesprache EJB QL unterstützt noch keine Aggregatoperationen, so dass der Einsatz von select-Methoden keine gute Lösung darstellt. Mit EJB QL kann nur eine Sammlung von Komponenten erzeugt werden, die dann in einer Schleife durchlaufen wird, um die benötigt Summe zu bilden. Da der Code für die Datenbankzugriffe nicht vom Container generiert werden kann, könnten die Zugriffe stattdessen innerhalb von home-Methoden manuell implementiert werden. Gegen dieses Vorgehen spricht aber, dass es nicht sinnvoll ist, Tabelle Execution als Entity Bean mit einer Beziehung zur zugehörigen Komponente der Tabelle Orders abzubilden, da von ihr nie konkrete Instanzen erzeugt werden. Um home-Methoden auszuführen, wird immer eine zur Zeit nicht verwendete Instanz aus dem Pool benutzt. Der Ansatz mit home-Methoden zu arbeiten, ist nur in einem Fall sinnvoll, in dem nur auf die Tabelle Orders zugegriffen wird. Zugunsten eines einheitlichen Ansatzes werden keine home-

Methoden verwendet, sondern die benötigten Methoden in einer Schnittstelle definiert. Die SQL-Abfragen werden dann in einer Implementierungsklasse durchgeführt.

Neue Aufträge werden in die Tabelle Orders eingefügt. Hierfür werden die in der Spezifikation vorgesehen Entity Beans verwendet. Die Tabelle Orders steht in Beziehung zu den Tabellen Product und Depot. Diese werden auf Bean-Ebene nicht modelliert, da sie für den Prototyp nicht benötigt werden.

Die Eigenschaften von Entity Beans passen nicht zu den Bedürfnissen des Prototypen. Es müssen sehr viele Daten gelesen aber nur sehr wenige geschrieben werden. Entity Beans einzusetzen, würde zu vielen Datenbankzugriffen führen und somit zu langsamen Antwortzeiten und hohem Datentransfer zwischen Datenbank und Application Server beitragen. Die Read-Only Beans des WebLogic Servers passen viel besser zu den Anforderungen. Durch sie kann die Anzahl der nötigen Datenbankzugriffe und der Netzwerkverkehr zwischen Datenbank und Application Server stark reduziert werden. Datenbankzugriffe entfallen, wenn über das Primary Key-Objekt auf im Cache befindliche Daten zugegriffen wird. Da die Entity Beans auf alle Server eines Clusters verteilt werden, kann auch in einem Servercluster immer von zwischengespeicherten Daten profitiert werden. Das Verhalten des Caches kann dadurch beeinflusst werden, dass für jede Komponente festgelegt werden kann, wie viele Instanzen sich maximal im Pool befinden dürfen und nach welcher Zeitspanne Instanzen ungültig werden.

Daten im Cache zu halten bedeutet jedoch gesteigerte Hardwareanforderungen und ist nur dann sinnvoll, wenn häufig von dem Cache profitiert werden kann. Daten, auf die nur lesend zugegriffen wird, sind Börsen- und Produktdaten sowie kundenspezifischen Daten.

Die Menge der Börsendaten und damit der vorzuhaltenden Daten ist gering, da der Handel nur an den neun inländischen Börsenplätzen möglich ist. Da sich diese Daten praktisch nie ändern, kann die Zeitspanne für das ungültig werden auf einen großen Wert gesetzt werden. Da beim Vorbereiten und Ausführen einer Order geprüft werden muss, ob der Handel an einer bestimmten Börse an einem bestimmten Datum möglich ist, und da der Zugriff immer über ein

Primary Key-Objekt möglich ist, kann von den vorgehaltenen Daten auch sehr häufig profitiert werden.

Bei den Produktdaten stellt sich die Frage, ob ein Caching angesichts einer sehr großen Anzahl von Produkten überhaupt sinnvoll ist. Dies kann bejaht werden. Eine Untersuchung hat ergeben, dass knapp 80 % aller Transaktionen mit nur 100 Wertpapieren durchgeführt werden. Es ist also sehr sinnvoll ein Caching durchzuführen, da häufig dieselben Daten benötigt werden. Die vorzuhaltende Datenmenge ist in einer vertretbaren Größenordnung. Leider kann nur selten durch ein Primary Key-Objekt auf die Daten im Cache zugegriffen werden. Das Primary Key-Objekt der Produkte entspricht dem Attribut ProductId in der Tabelle Product. Beim Kauf müssen Produkte aber über die WKN gesucht werden. An dieser Stelle zeigt sich die Diskrepanz zwischen Datenmodellierung und Entity Bean-Design. Bei der Erstellung des Datenmodells wurde Wert darauf gelegt, unabhängig von fachlichen Aspekten zu sein. Daraus resultieren Nachteile für den Einsatz von Entity Beans. Auf die börsenplatzabhängigen Daten kann auch nur in bestimmten Situationen über Primary Key-Objekte zugegriffen werden. Die Daten dieser Gruppe können sich nur durch einen Import ändern, der einmal täglich durchgeführt wird. Es ist jedoch nicht sinnvoll, die Beans nach einem Tag ungültig werden zu lassen. Zum einen werden beim Import nur relativ wenige Daten geändert und zum anderen kann dann das Problem auftreten, dass eine Bean gültig ist, deren Daten in der Datenbank aber durch den Import geändert wurden. Um diese Probleme zu umgehen, sollten beim Import die Primary Key-Objekte der geänderten Daten gesammelt und an eine spezielle Management-Bean übergeben werden. Diese kann dafür sorgen, dass die entsprechenden Beans ungültig gemacht werden. Dies ist jedoch nicht mehr Bestandteil des Prototypen.

Es scheint zunächst nicht sinnvoll, Read-Only Beans für kundenspezifische Daten einzusetzen, da viele verschiedene Kunden das System nutzen. In die Betrachtung muss jedoch mit einbezogen werden, dass die kundenspezifischen Daten sowohl beim Vorbereiten als auch beim Ausführen einer Order überprüft werden müssen. Die Performance lässt sich für das Ausführen durch Read-Only Beans steigern, da auf diese Daten immer über Primary Key-Objekte zugegriffen werden kann. Der Cache für diese Daten muss so groß sein, dass die

Daten aller gleichzeitig im System angemeldeten Kunden darin gehalten werden können. Die kundenspezifischen Entity Beans brauchen auch nur für einen vergleichsweise kurzen Zeitraum im Cache zu sein. Leider liegen keine Informationen über das Kundenverhalten vor, um näher auf die Größe des Cache und die Verfallzeit der Entity Beans eingehen zu können. Auch die Kundendaten werden durch Importe verändert, so dass auch hier eine Management-Bean benötigt wird, um Inkonsistenzen zu vermeiden. Bei Änderungen an den Tabellen DepotLock und PositionLock müssen die zugehörigen Komponenten DepotBean und DpositionBean ungültig gemacht werden.

Um einen durchgängigen Ansatz zu verfolgen, werden Read-Only Beans in allen drei Bereichen eingesetzt. Bei den Produkten und den börsenplatzabhängigen Produktdaten kann derzeit nur bedingt vom Caching profitiert werden, aber in zukünftigeren Versionen des WebLogic Servers wird sich das nach Angaben von BEA ändern. Eine weitere Eigenschaft der Read-Only Beans ist, dass sie nicht an Transaktionen teilnehmen können. Das hat zur Folge, das sie nicht an CMR beteiligt sein können. Da die Beziehungen zwischen den Read-Only Beans nicht durch den Container verwaltet werden können, müssen sie manuell implementiert werden. Die Verwaltung ist aber relativ leicht möglich, da nur Daten gelesen werden und nicht über die Beziehung manipuliert wird.

CMR ist wesentlicher Bestandteil in EJB 2.0, so dass unter dem Aspekt, möglichst viele Neuerungen von EJB 2.0 einzusetzen, auf Read-Only Beans verzichtet werden sollte. Andererseits sollen aber auch die Fähigkeiten des WebLogic Servers mit einbezogen werden. Wie oben diskutiert, ist durch den Einsatz von Read-Only Beans eine wesentlich bessere Performance zu erwaten. Eine Aussage über die derzeit technischen Möglichkeiten treffen zu können, ist demnach nur mit Read-Only Beans möglich.

Da die persistenten Daten entweder durch Entity Beans abgebildet oder durch SQL-Abfragen ermittelt werden, kann die gesamte Geschäftslogik des Geschäftsvorfalls „Order Aufgeben“ innerhalb der Geschäftsschicht abgebildet werden. Der Prototyp ist also konform zum Schichtenmodel der J2EE-Architektur. Die einzelnen Schichten sind unabhängig voneinander und austauschbar.

Durch den Einsatz von Read-Only Beans ist der Prototyp nicht zur EJB-Spezifikation 2.0 kompatibel und CMR kann nicht eingesetzt werden. Es entsteht aber dennoch keine Insellösung, da auch andere Application Serverhersteller wie beispielsweise IBM Read-Only Beans unterstützen. Außerdem wird in der Spezifikation angekündigt, dass für zukünftige Versionen geplant ist, Read-Only Beans für CMP zu berücksichtigen. Standard Entity Beans zu verwenden würde Nachteile mit sich bringen, die man im bestehenden System versucht hat zu vermeiden. Nur durch Read-Only Beans können diese vermieden werden. Da der Prototyp dazu dient, die technischen Möglichkeiten aufzuzeigen, ist der Einsatz von Read-Only Beans gerechtfertigt. Es bleibt jedoch abzuwarten, ob die hier angeführten Argumente für Read-Only Beans weiterhin gelten, wenn deren Verhalten in der Spezifikation verbindlich festgelegt ist und nicht nur eine proprietäre Lösung darstellt. Der Nachteil, Beziehungen zwischen Beans selbst verwalten zu müssen, ist nicht so bedeutend, da nur Daten gelesen werden.

7.3 Implementierung

Die Umsetzung des Entwurfs erfolgt, wie in den Anforderungen vorgegeben, auf Basis der EJB-Spezifikation 2.0 und den erweiternden Möglichkeiten des WebLogic Servers 6.1.

7.3.1 Entwurfsmuster

Bei der Implementierung des Prototypen werden verschiedene Entwurfsmuster eingesetzt. Um bereits zum Zeitpunk des Kompilierens sicherzustellen, dass es keine Inkonsistenzen zwischen den Methoden des Local bzw. Remote Interface und denen der Bean-Klasse gibt, wird das Business Interface-Muster[20] bei allen Komponenten angewendet.

Durch das DataAccessObject-Muster werden Geschäft- und Datenzugriffslogik voneinander getrennt.[21] Das Muster wird für alle Datenbankzugriffe eingesetzt, die selbst zu implementieren sind. Wie in den Blueprints von Sun empfohlen, wird die Fabrik mit einer Klassenmethode realisiert. Dies wäre auch mit einem Singleton möglich, die Spezifikation verbietet aber aus Konsistenzgründen die

[20] Vgl. Marinescu, F., EJB Design Patterns, 2002, S. 40 ff.
[21] Vgl. Sun Java Center J2EE™ Patterns – Data Access Object

Synchronisation, da sie nicht mehr funktionieren kann, wenn ein Container mehrere JVM's erzeugt.[22]

Das Sequence Blocks-Muster ist eine Strategie zum Erzeugen von integer-basierten Primärschlüsseln.[23] Beim Prototypen wird das Muster eingesetzt, um neue Aufträge eindeutig identifizieren zu können. Die Nachteile des Musters, dass evtl. nicht alle möglichen Primärschlüsselwerte benutzt werden, und dass die Primärschlüsselwerte möglicherweise nicht sortiert sind, können ohne weiteres hingenommen werden und bedeuten keine Einschränkung. Anstelle dieses Musters hätte auch auf Oracle Sequences[24] zurückgegriffen werden können. Das hätte jedoch zu einer Abhängigkeit von der Datenbank geführt, die so vermieden werden kann.

Jonathan Weedon von Borland hat eine Implementierung veröffentlicht, die unabhängig von Application Server und Datenbank eingesetzt werden kann. Diese Implementierung wird auch für den Prototypen verwendet. Anpassungen werden nur für Logging und Fehlermeldungen sowie die herstellerspezifischen Einträge im Deployment Descriptor vorgenommen.

Um die Performanceanalyse durchführen zu können, muss der Clientzugriff simuliert werden. Die Kommunikation mit dem Application Server wird durch Delegates und eine EJB Home Factory durchgeführt. Durch diese Muster wird die Komplexität der Muster EJB-API verborgen[25] und die Stubs von EJBHome-Objekten können zwischengespeichert werden [26]. Um Portabilität und Wiederverwendbarkeit zu erreichen, werden die benötigten Informationen über den Application Server in einer Properties-Datei abgelegt.

Für die Bean-Klassen der Komponenten wird für jeweils Session Beans und Entity Beans eine gemeinsame Oberklasse entwickelt. Sie enthält Standardimplementierungen für die Methoden, die durch die EJB-Spezifikation vorgegeben sind. So kann unnötiges Programmieren vermeiden werden. In der verfügbaren Literatur wird dieser Ansatz nicht als Muster beschrieben. Er wird aber trotz-

---

[22] Vgl. Enterprise Java Beans Specification, S. 494
[23] Vgl. Marinescu, F., EJB Design Patterns, 2002,S. 106 ff.
[24] eine Funktion der Oracle Datenbank, die das Generieren eines eindeutigen, integerbasierten Primärschlüssels ermöglicht
[25] Vgl. Marinescu, F., EJB Design Patterns, 2002,S. 98 ff.
[26] Vgl. Marinescu, F., EJB Design Patterns, 2002,S. 92 ff.

dem an dieser Stelle aufgeführt, da er den Charakter eines Entwurfsmusters hat.

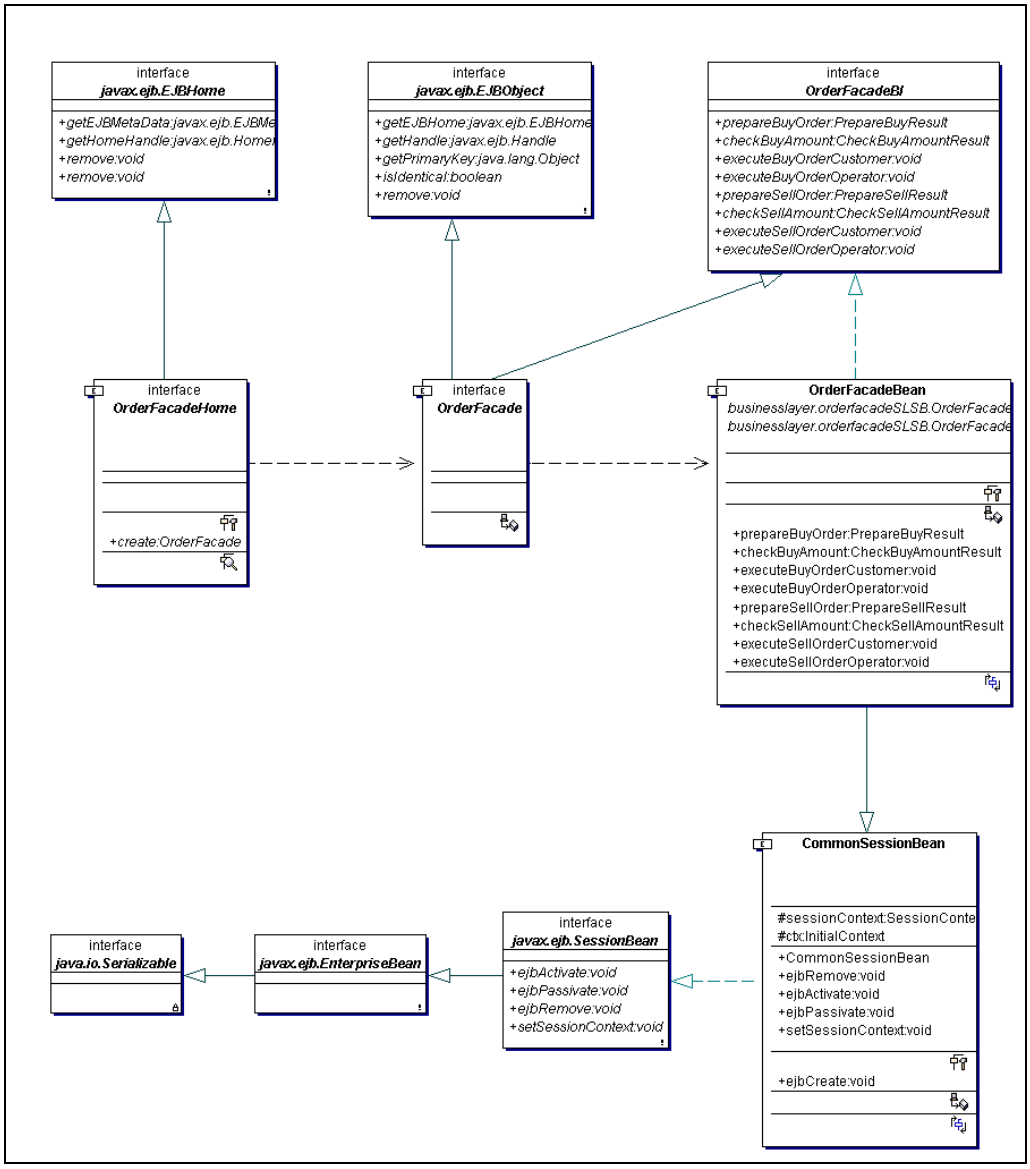

Abbildung 17: Bestandteile einer Komponente am Beispiel von OrderFacadeBean

7.3.2 Sperrmechanismus

Beim Aufgeben einer Order wird eine Dispositionsprüfung durchgeführt. Wenn ein Kunde versucht mehrere Aufträge gleichzeitig aufzugeben, dürfen möglicherweise nicht alle ausgeführt werden, weil das Guthaben nicht ausreicht oder nicht die nötige Menge im Depot vorhanden ist. Liegen jedoch gleichzeitig mehrere Aufträge zur Prüfung vor, werden die Auswirkung auf das Guthaben bzw. den Bestand bei parallel durchgeführten Prüfungen nicht berücksichtigt. In diesem Fall können möglicherweise alle Aufträge aufgegeben

werden. Aus diesem Grund muss durch einen Sperrmechanismus verhindert werden, dass mehrere Aufträge gleichzeitig aufgegeben werden können.

Ein Kunde kann für mehrere Depots dasselbe Verrechnungskonto benutzen. Für die Dispositionsprüfung beim Kaufen ist das Guthaben auf dem Verrechnungskonto ausschlaggebend, so dass das Konto zeitweilig gesperrt werden muss. Für die Dispositionsprüfung beim Verkaufen wird nicht nur der Bestand der Position betrachtet, von der verkauft werden soll, sondern alle Positionen eines Depots mit dem zu handelnden Wertpapier. Das ist nötig, da auch negative Bestände möglich sind, die mit den positiven verrechnet werden müssen. Beim Verkauf muss also das gesamte Depot gesperrt werden.

Eine Anforderung an den Prototypen besteht darin, dass er clusterfähig sein soll. Das bedeutet, dass der Sperrmechanismus nur an einer zentralen Stelle durchgeführt werden kann. Dafür kommen der JNDI-Baum und die Datenbank in Frage. Eine Lösung könnte sein, ein Objekt in den globalen JNDI-Baum des Clusters einzufügen. Das Objekt könnte eine Verwaltung implementieren, um Konten und Depots zu sperren. Es besteht die Gefahr, dass gleichzeitig auf das Objekt zugegriffen wird, so dass die Methoden synchronisiert werden müssen. Dieser Ansatz kann aber nicht verfolgt werden, da, wie bereits oben angeführt, die Spezifikation eine Synchronisation verbietet. Bleibt nur der Weg über die Datenbank. Es werden zwei Tabellen angelegt, eine um Konten und eine um Depots zu sperren. Mit dem Primärschlüssel verfügen die Tabellen jeweils nur über ein Attribut für die Konto- bzw. Depotnummer. Zu den Tabellen werden die zugehörigen Entity Beans TradingLockBuy und TradingLockSell bereitgestellt. Wird eine Order ausgeführt, erzeugt die Anwendungsfallsteuerungskomponente eine neue Entity Bean um ein Konto beim Kauf oder ein Depot beim Verkauf zu sperren. Kann keine Entity Bean erzeugt werden, weil das Konto bzw. das Depot bereits gesperrt ist, wird der Vorgang mit einer entsprechenden Fehlermeldung abgebrochen. Ist die Order erfolgreich aufgegeben worden, oder ist eine Exception während der Bearbeitung aufgetreten, wird die zuvor erzeugte Entity Bean wieder gelöscht, um das Konto bzw. das Depot wieder freizugeben .

Für den Produktionsbetrieb muss dieser Mechanismus noch erweitert werden. Wenn z.B. der Application Server abstürzt, bleiben die Sperren bestehen, so

dass auch über andere Server des Clusters nicht gehandelt werden kann. Eine Lösung könnte sein, die Sperren mit einem Zeitstempel zu versehen und beim Start des Application Servers ältere Einträge zu löschen.

7.3.3 Datentypen

Die Datentypen der Eigenschaften der Entity Beans richten sich nach den Oracle Datentypen der entsprechenden Tabellen. Bei der Umwandlung von Oracle-Datentyp nach Java-Datentyp wird folgendermaßen verfahren. Für Ganzzahlen und Nachkommazahlen werden in Java standardmäßig die Datentypen int und double verwendet[27]. Die Typen short und byte sind für besondere Situationen vorgesehen. Der Datentyp float verfügt mit nur sieben signifikanten Stellen nur über eine sehr begrenzte Genauigkeit im Vergleich zur doppelten Genauigkeit (15 signifikante Stellen) von double. Bei dem Attribut prioriskclass der Tabelle Product muss jedoch anders verfahren werden. Es ist von entscheidender Bedeutung, ob das Attribut einen Wert hat oder nicht. Wenn die Umwandlung in den Typ int erfolgt kann nicht mehr unterschieden werden, ob das Attribut den Wert Null oder gar keinen Wert hat. Derzeit erlauben die Geschäftsregeln den Wert Null nicht, problematisch wird es aber, wenn sich die Geschäftsregeln ändern und Null ein zulässiger Wert wird. Aus diesem Grund wird das Attribut prioriskclass als java.lang.Integer abgebildet. Wenn kein Wert in der Datenbank vorhanden ist, dann ist die entsprechende Eigenschaft der Entity Bean „null“. So kann eindeutig festgestellt werden, ob ein Wert vorhanden ist oder nicht. Die Oracle-Datentypen für Zeichenketten CHAR und VARCHAR werden als java.lang.String dargestellt. Eine Ausnahme bilden Attribute deren Zeichenketten nur aus einem Zeichen bestehen, hierfür bietet Java den Typ char. Der Oracle-Datentyp DATE wird in java.util.Date umgewandelt.

Vom Client werden die Kundenangeben an die Anwendungsfallsteuerung übergeben. Für die Kundenangeben werden folgende Datentypen verwendet:

Die WKN wird als java.lang.String angegeben. Die WKN besteht zwar nur aus Ziffern, aber die zukünftig geltende ISIN enthält auch Buchstaben.

---

[27] Vgl. Horstamnn, C. S./Cornell, G., Core Java 2, 1999, S. 70 ff.

Depotpositionen werden in der Datenbank durch ganze Zahlen identifiziert. Der entsprechende Parameter muss darum als int angegeben werden.

Die Menge ist als Nachkommazahl dazustellen. Der Java- Standarddatentyp dafür ist double.

Das Datum muss vom Benutzer im Format „TT.MM.JJJJ“ angegeben werden. Da es nicht als trivial angesehen wird, einen String in ein Datum umzuwandeln, muss ein String in dem entsprechenden Format übergeben werden.

Bei einem limitierten Auftrag wird der Limitwert als Nachkommazahl angegeben. Da Aufträge auch unlimitiert möglich sind, muss unterschieden werden können, ob kein Wert angegeben wurde, oder die Angabe ungültig ist. Darum muss der Limitwert als Datentyp Double übergeben werden. Für unlimitierte Aufträge wird dann „null“ übergeben.

Um Limittyp und Orderzusatz festzulegen, sind jeweils nur bestimmte Werte möglich. Da die Werte auch in der Datenbank gespeichert werden, sind die möglichen Werte in Schnittstellen als global gültige Konstanten definiert. Der Limittyp muss als String und der Orderzusatz als int angegeben werden.

Die TAN-Überprüfung wird vom Buchungssystem des Betreibers durchgeführt, das im Prototypen durch die Komponente OLSWrapperBean angebunden ist. Da OLSWrapperBean TAN-Angaben als String erwartet, müssen diese auch der Anwendungsfallsteuerungskomponente als String übergeben werden.

Da im Prototypen die Sicherheits- und Sessionmechanismen des bestehenden Systems eingesetzt werden, muss vom Client immer ein NLAuthentication-Objekt übergeben werden.

Bei Berechnungen mit Werten vom Typ double kann es zum Verlust der Genauigkeit kommen. Um sicherzustellen, dass dieser Fall nicht eintreten kann, werden für Berechnungen Objekte vom Typ java.math.BigDecimal verwendet. Sie ermöglichen Berechnungen mit beliebiger Genauigkeit.

Die Klasse java.util.GregorianCalendar implementiert die Regeln des Gregorianischen Kalenders. Für Überprüfung und Anpassung des Ausführungsdatums werden darum Objekte dieses Typs benutzt.

7.3.4 Sonstige Detaills

Im Prototyp werden einige Komponenten und Klassen aus dem bestehenden Brokerage-System übernommen. Diese Komponenten und Klassen benutzen eine Klasse von Netlife, um Logging durchzuführen. Damit alle Log-Ausgaben an einer Stelle stehen, benutzt der Prototyp ebenfalls diese Klasse.

Der Prototyp soll unabhängig davon sein, ob das Logging durch Klassen von Netlife oder über ein Produkt wie IBM's Log4J durchgeführt wird. Darum verfügt er über die Klasse Log. Sie kapselt die eigentliche Logging-API und leitet Aufrufe an diese weiter.

Alle Fehlerwerte und Nachrichten, die an den Client geschickt werden, sind als String-Konstanten in der Schnittstelle ErrorCodes definiert. Es werden Strings verwendet, da sie einerseits als Objekte von Containern aufgenommen werden können und andererseits, da sie in Log-Files geschrieben werden können und ihre Bedeutung dann immer noch klar und nicht durch einen Zahlencode verschlüsselt ist. Die Fehlerwerte könnten von einer weiteren Klasse aufgelöst werden, welche die Schnittstelle implementiert.

Konstanten, die für die gesamte Anwendung einheitlich gültig sind, werden jeweils in einer Schnittstelle fachlich zusammengefasst. Diese Schnittstellen werden von mehreren Komponenten und Klassen implementiert.

Die Exception-Klasse OrderCheckerException verfügt über ein zusätzliches Attribut, um bei Ausnahmen in Application-Level und System-Level zu unterscheiden. Die Stufe Application-Level zeigt an, dass während der Prüfung gegen die Geschäftslogik verstoßen wurde, die Stufe System-Level zeigt an, dass eine Kritische Situation (z. B ein Netzwerk- oder Datenbankfehler) eingetreten ist, so dass die Prüfungen nicht durchgeführt werden können.

Beim Prototypen werden die Börsenfeiertage der nächsten drei Monate geladen. Nach dem Neuladen einer ExchangeBean werden auch die Börsenfeiertage neu geladen, so dass der relevante Zeitraum immer abgedeckt ist. Der Zeitraum ist über den Deployment Descriptor konfigurierbar.

Die Börsenplätze verfügen über das eindeutige Attribut rank, um sie in der vorgegeben Reihenfolge sortieren zu können. Die Komponenten werden in einer verketteten Liste gesammelt und durch die Klasse ExchangeComparator

sortiert. Anstelle einer Liste könnte auch ein Baum verwendet werden. Da die Anzahl der zu sortierenden Elemente aber sehr klein ist, ist es aufwendiger die einzelnen Elemente in einen Baum einzufügen. Das gilt insbesondere dann, wenn die Elemente bereits in der richtigen Reihenfolge eingefügt werden. Die Klasse ExchangeComparator implementiert die Schnittstelle Comparator, über dessen Methoden der Sortiervorgang erfolgt. Der Einsatz der Schnittstelle Comparable wäre ebenfalls möglich gewesen. Mit ihr wird eine natürliche Sortierreihenfolge für die Objekte der Klasse festgelegt, welche die Schnittstelle implementiert. Wenn sich die Geschäftslogik ändert, und es erforderlich ist, nach verschiedenen Kriterien zu sortieren, kann die Schnittstelle Comparable folglich nicht mehr verwendet werden. Anders bei Schnittstelle Comparator. Hier kann für jedes Sortierkriterium eine spezielle Klasse bereitgestellt werden. Mit Hinblick auf eine größere Flexibilität wurde darum der Ansatz mit der Schnittstelle Comparator gewählt.

7.3.5 Packagestruktur und Namenskonventionen

Auf eine eindeutige Identifizierung der Klassen wird verzichtet. Zugunsten einer flachen Packagestruktur werden keine Domainnamen mit in die Hierarchie aufgenommen.

Für jede Komponente gibt es ein separates Package, wobei die Packagenamen den Komponentennamen und die Bean-Art enthalten. Stateless Session Beans werden durch das Suffix SLSB und Container Managed Persistence Entity Beans durch CMPEB gekennzeichnet. Die Klassen und Schnittstellen der Komponenten erhalten folgende Suffixe um sie kenntlich zu machen:

- Locale und Remote Interface (kein Suffix)
- Locale Home und Remote Home Interface Home
- Bean-Klasse Bean
- Business Interface BI

Alle Komponenten, die Geschäftslogik implementieren sowie die Order- und Datentransferobjekte befinden sich im Package businesslayer.

Das Package datalayer beinhaltet Komponenten und Klassen, die Daten repräsentieren und Datenbankzugriffe durchführen. Data Access Objects werden jeweils in eigene Packages mit dem Namen dao zusammengefasst.

Die Komponente QuoteFeederBean bindet Hostsysteme für die Kursversorgung an und befindet sich im Package interfacelayer.

Das Package constants enthält Schnittstellen, die für die gesamte Anwendung eindeutige Konstanten definieren.

Das Package util beinhaltet die die Klasse Log und die Schnittstelle ErrorCodes.

Die Komponenten, durch die das Sequence Bocks-Muster implementiert wird, sind im Package sequencegenerator enthalten.

Das Package delegate beinhaltet Klassen, durch die das Business Delegate- und das EJB Home Factory-Muster implementiert werden.

Die Klassen, mit denen der Lasttest für die Performanceanalyse durchgeführt wird, sind in dem Package loadtest enthalten.

8 Performanceanalyse

An Anwendungen werden viele verschiedene Anforderungen gestellt. Für die Benutzer ist wichtig, dass mit dem System vernünftig gearbeitet werden kann. Eine ganz wesentliche Rolle kommt dabei der Ausführungsgeschwindigkeit zu. Langsame Antwortzeiten z. B. werden dazu führen, dass das System von den Anwendern nicht akzeptiert wird. Um das Bedürfnis nach einer schnellen Ausführungsgeschwindigkeit zu befriedigen, sind im bestehenden Brokerage-System in einer mehrschichtigen Anwendungsarchitektur zusätzlich Stored Procedures eingesetzt worden. Dieser Ansatz bringt die in den vorangegangenen Kapiteln aufgezeigt Nachteile mit sich. Die Weiterentwicklung der EJB-Spezifikation und der Application Server hat es ermöglicht, das System mit einem neuen Design zu versehen. Dabei sind die Nachteile der bestehenden Lösung beseitigt worden und die dort verfolgten Ziele, die Anzahl der Datenbankzugriffe und die zu übertragende Datenmenge gering zu halten, wurden trotzdem berücksichtigt. In einer Performanceanalyse soll die Leistung der beiden Systeme miteinander verglichen werden.

8.1 Bewertungsmethode

Bei einer Performanceanalyse werden normalerweise mehrere Systeme über einen Benchmark miteinander verglichen. Dabei werden Leistungsfähigkeit und Kosten der Systeme beurteilt und als Preis-Leistungsverhältnis dargestellt. Benchmarks definieren Vorgehensweisen, um Ergebnisse zu ermitteln und zu bewerten und werden von verschiedenen Gremien wie SPEC (System Performance Evaluation Cooperative) und TPC (Transaction Processing Performance Council) für verschiedene Fragestellungen entwickelt. So gibt es z.B. Benchmarks für wissenschaftliche und hardwarespezifische Bereiche sowie für Datenbanken und Transaktionsabwicklung. Universell einsetzbare Benchmarks gibt es nicht, da die Anforderungen in verschiedenen Bereichen sehr stark variieren.[28]

Da es weder allgemeine Benchmarks noch spezielle für Brokerage-Systeme gibt, muss eine andere Bewertungsgrundlage entwickelt werden. Weil das Ziel der Analyse darin besteht, die Leistungsfähigkeit beider Systeme miteinander

[28] Vgl. Gran, J., Benchmark

zu vergleichen, werden die Kostenaspekte nicht berücksichtigt. An einen Benchmark wird insbesondere die Anforderung gestellt, dass die maximale Leistungsfähigkeit beim Durchführen einer typischen Operation ermittelt wird.[29] Eine Operation wird auch als Transaktion bezeichnet und kann aus mehreren Schritten bestehen. Um die Leistung zu beurteilen, wird der Durchsatz betrachtet. Der Durchsatz ist eine Kennzahl die angibt, wie viele Transaktionen pro Sekunde durchgeführt werden können. Die Kennzahl wird errechnet, indem die Anzahl der Transaktionen durch die Summe der Ausführungszeiten geteilt wird.

Im Prototyp wurde der zentrale Geschäftsvorfall „Order aufgeben“ umgesetzt. Die Anforderung, einen typischen Vorgang für die Bewertung zu berücksichtigen, kann also erfüllt werden. Wertpapiere handeln zu können bedingt jedoch, dass dies für einen bestimmten Kunden durchgeführt wird. Darum wird festgelegt, dass eine Transaktion aus folgenden Schritten bestehen soll: Anmelden eines Kunden, Vorbereiten einer Order, Ausführen einer Order und Abmelden des Kunden. Messungen werden sowohl für den Kauf als auch für den Verkauf durchgeführt. Beide Fälle haben die gleiche Bedeutung für das System, erfordern aber verschiedene Prüfungen und belasten das System unterschiedlich. Da das An- und Abmelden nicht Bestandteil des Prototypen ist, werden die entsprechenden Teile aus dem bestehenden System übernommen.

Um Aussagen über die Leistungsfähigkeit machen zu können, werden Lasttests durchgeführt.[30] Dabei wird eine Transaktion parallel von mehreren Clients mehrfach hintereinander ausgeführt. Üblich sind drei bis zehn Wiederholungen.[31] Die Anzahl der Clients wird für jeden Durchlauf erhöht. Für jeden Testlauf werden die durchschnittliche Antwortzeit und der Durchsatz ermittelt. Anhand der Ergebnisse kann beurteilt werden, wie sich das System bei unterschiedlicher Last verhält. Dabei werden die Transaktion als Ganzes sowie die Maßgeblichen Schritte, Vorbereiten und Ausführen einer Order, gesondert betrachtet.

Caching kann die Performance des Prototypen stark beeinflussen, so dass zusätzlich einzelne Testfälle durchgeführt werden müssen, bei denen verschieden

[29] Vgl. Gran, J., Benchmark
[30] Vgl. Gómez, P./Zadrozny, P., J2EE with BEA WebLogic Server, 2000, S. 365
[31] Vgl. Gómez, P./Zadrozny, P., J2EE with BEA WebLogic Server, 2000, S. 366

häufig vom Cache profitiert werden kann. Das Caching muss auch bei der Durchführung der Lasttests berücksichtigt werden, da sonst verfälschte Ergebnisse entstehen. Die Daten müssen so gewählt werden, dass eine möglichst realistische Ausnutzung des Cache erreicht wird.

Das Caching wird für Börsen-, Produkt- und Kundendaten durchgeführt. Der Handel ist nur an wenigen Börsenplätzen möglich und die entsprechende Datenmenge ist so gering, dass sie komplett im Speicher gehalten werden kann. Darum kann voraussichtlich immer vom Cache profitiert werden. Aus diesem Grund können die Kundenangaben, die sich auf die Börsendaten beziehen, für alle Tests unverändert bleiben. Bei den Produkt- und Kundendaten hingegen kann nur ein kleiner Teil der Daten vorgehalten werden. Wie bereits angeführt wurde, werden etwa 80 % aller Transaktionen mit 100 Wertpapieren durchgeführt. Über das Kundenverhalten liegen leider keine Informationen vor, so dass hierfür der ungünstigste Fall angenommen wird. Messungen müssen automatisierbar sein, damit sie in annehmbarer Zeit durchgeführt werden können. Es ist schwierig eine bestimmte Trefferquote durch Testdaten sicherzustellen. Darum werden die Testläufe immer mit den gleichen Wertpapieren aber immer für unterschiedliche Kunden durchgeführt, wobei ein Client in jeder Wiederholung immer dasselbe Wertpapier für einen anderen Kunden handelt. Da beim ersten Durchlauf noch keine Daten im Speicher vorhanden sind, wird es einen gewissen Anteil geben, bei dem das Produkt geladen werden muss. Wie groß der ist, hängt von der Anzahl der Wiederholungen ab.

8.2 Umgebung

Um die Tests durchzuführen, steht eine Workstation mit Sparc-Prozessor, einem GB RAM und Sun Solaris 2.6 als Betriebsystem zur Verfügung. Darauf sind eine Oracle 8.1.7-Instanz, ein WebLogic Server 5.1 und ein WebLogic 6.1 installiert. Um auf die Datenbank zuzugreifen wird der JDBC-Treiber von Oracle eingesetzt. Die Connection-Pools sind so dimensioniert, dass während der Testläufe keine weiteren Verbindungen erzeugt werden müssen. Eine weitere Workstation wird eingesetzt, um Clients zu simulieren. Beide Workstations befinden sich im Netzwerk von Netlife. Um den Lasttest durchzuführen wird das Tool Grinder eingesetzt.

Die Testmöglichkeiten sind durch die zur Verfügung stehende Hardware eingeschränkt. Das System, das als Server fungiert, bietet nur eine geringe Leistungsfähigkeit. Insbesondere die Datenbank ist dadurch stark eingeschränkt. Datenbank und Application Server laufen auf einem Rechner mit wenig Ressourcen und behindern sie sich dadurch gegenseitig besonders stark. Da die Rechner sich nicht in einem separaten Netzwerk befinden, können die Ergebnisse zusätzlich durch nicht kontrollierbaren Netzwerkverkehr beeinflusst werden. Nur wenn die einzelnen Bestandteile des Systems ihre optimale Leistung liefern können und keinen Hardwarebeschränkungen unterliegen, kann die Leistung des Gesamtsystems wirklich beurteilt werden. Optimale Testergebnisse lassen sich daher nur erreichen, wenn für Client, Application Server und Datenbank jeweils eigene leistungsstarke Rechner eingesetzt werden, die sich in einem separaten Netzwerk befinden. Nur so ist es möglich, die gesamte Umgebung zu kontrollieren, vor Fremdeinwirkungen zu schützen und Aussagen über die Performance einzelner Teile sowie des gesamten Systems zu treffen.

Das bestehende System ist nur in einer Umgebung mit WebLogic Server 5.1 lauffähig. Eine Migration des Brokerage-Systems auf den WebLogic Server 6.1 ist sehr aufwendig und wird deshalb nicht durchgeführt. Bei der Auswertung muss berücksichtigt werden, dass beide Systeme nicht unter den gleichen Voraussetzungen ausgeführt werden können.

Für die Durchführung der Lasttests wird das Tool Grinder eingesetzt. Es wurde von den Autoren des Werkes „Professional Java 2 Enterprise Edition with BEA WebLogic Server“ entwickelt und steht frei zur Verfügung. Das Tool ist in der Lage, simultane Clients für Web- und EJB-Komponenten sowie für Java-Programme mit grafischer Oberfläche zu simulieren. Für die zu testende Anwendung muss eine spezielle Testklasse bereitgestellt werden, die in den Grinder über eine Schnittstelle eingebunden wird.

Die Anzahl der Clients und der Wiederholungen sowie die aufzurufenden Methoden, werden über eine Konfigurationsdatei festgelegt. Zusätzlich kann angegeben werden, dass bei Testbeginn die Clients mit einer bestimmten zufälligen Verzögerung gestartet werden, und dass zwischen den einzelnen Aufrufen eine Pause gemacht wird. Diese Einstellungen ermöglichen es, ein realistischeres Benutzerverhalten zu simulieren. Zugriffe im selben Moment sind unrealis-

tisch und die Interaktion zwischen Benutzer und System wird berücksichtigt. Neben den zu testenden Vorgängen muss die Testklasse auch spezielle Methoden beinhalten um Initialisierungs- und Aufräumtätigkeiten durchzuführen.

Der Grinder führt sowohl die einzelnen Methoden als auch für den gesamten Lauf statistische Auswertungen durch. So wird in beiden Fällen die durchschnittliche Antwortzeit und für jede Methode der Durchsatz ermittelt.

8.3 Durchführung

8.3.1 Auswirkung des Caching ohne Last

Es hat sich gezeigt, dass unter gleichen Bedingungen die Ausführungsgeschwindigkeit desselben Testlaufs bei mehrmaliger Wiederholung sehr stark variiert, obwohl nur geringe Unterschiede hätten auftreten dürfen. Dieses Verhalten kann nur auf die zu leistungsschwache Hardware zurückgeführt werden. Darum werden alle Testläufe mehrmals wiederholt und ein Mittelwert von den Ergebnissen gebildet.

Im Datenbankserver findet ebenfalls ein Caching statt, durch welches die Performance des bestehenden Systems beeinflusst wird. Werden für Abfragen immer dieselben Daten verwendet, können die Ergebnisse extrem schnell ermittelt werden. Werden jedoch unterschiedliche Daten verwendet, können dadurch erheblich längere Bearbeitungszeiten resultieren. Es ist problematisch, dies in den Testläufen zu berücksichtigen, da keine Möglichkeit besteht, direkt auf den Cache Einfluss zu nehmen.

Eine Untersuchung für zwei verschiedene Szenarien hat gezeigt, dass vom Datenbank-Caching gar nicht profitiert werden kann. Sowohl für den Kauf als auch für den Verkauf wurden zehn mal dieselbe Transaktion und zehn Transaktionen mit unterschiedlichen Kundendaten durchgeführt. Es hat sich gezeigt, dass die Ausführungsgeschwindigkeit der einzelnen Wiederholungen stark variiert, und dass es keinen positiven Einfluss hat, wenn immer dieselben Daten verwendet werden.

	Kauf				Verkauf			
	Anmelden	Vorbereiten	Ausführen	Abmelden	Anmelden	Vorbereiten	Ausführen	Abmelden
alle Angaben identisch	470	387	498	85	532	484	490	84
	434	390	486	82	414	503	387	78
	479	363	471	81	426	449	428	79
	453	383	725	79	467	491	409	77
	446	488	460	82	411	351	399	86
	527	441	490	87	419	360	396	91
	428	372	532	80	372	404	380	92
	421	384	516	82	414	328	378	81
	459	394	493	88	415	341	414	101
	451	377	511	80	409	361	427	97
∅	456,8	397,9	518,2	82,6	427,9	407,2	410,8	86,6
Kundendaten variieren	432	527	577	84	486	398	422	87
	429	371	493	85	424	350	419	81
	415	390	465	80	415	347	423	111
	414	369	452	77	443	340	408	82
	518	378	546	81	413	348	387	78
	431	357	491	88	454	399	482	88
	410	561	468	80	431	364	394	80
	414	358	457	86	487	392	487	93
	475	476	494	84	488	342	625	84
	411	366	483	81	469	403	636	90
∅	434,9	415,3	492,6	82,6	451	368,3	468,3	87,4

Abbildung 18: Antwortzeiten für zwei Szenarien jeweils für Kauf und Verkauf

Für Vergleiche mit dem Prototyp werden darum für die Mittelwerte aus allen Wiederholungen beider Szenarien benutzt.

- Vorbereiten einer Kauforder: 406,6 Millisekunden
- Ausführen einer Kauforder: 505,4 Millisekunden
- Vorbereiten einer Verkauforder: 387,75 Millisekunden
- Ausführen einer Verkauforder : 439,55 Millisekunden

Es fällt auf, dass es beim Kaufen länger dauert einen Auftrag vorzubereiten und auszuführen. Die Ursache dafür ist nicht klar ersichtlich. Ein Aspekt ist aber mit Sicherheit, dass beim Kaufen nicht direkt in der Tabelle Product nach einem Wertpapier gesucht wird, sondern die Tabelle ProductName benutzt wird, um den entsprechende Eintrag in der Tabelle Product zu finden. Beim Verkauf hingegen ist der Umweg nicht nötig, da über den Primärschlüssel der Tabelle Product gesucht werden kann.

Das Ausführen dauert sowohl beim Kaufen als auch beim Verkaufen länger als das Vorbereiten. Beim Ausführen werden fast die gleichen Prüfungen durchge-

führt wie beim Vorbereiten. Zusätzlich sind aber noch weitere Datenbankzugriffe nötig, um einen Auftrag persistent zu machen und um dafür zu sorgen, dass ein Kunde nur eine Order zur Zeit aufgeben kann.

Um die Ausführungszeiten des Prototypen zu untersuchen, werden vier verschiedene Szenarien jeweils für Kauf und Verkauf konstruiert:

- Fall 1: im Objekt-Cache sind keine Daten
- Fall 2: die Börsendaten befinden sich im Cache
- Fall 3: die Börsen- und Produktdaten befinden sich im Cache
- Fall 4: die Börsen-, Produkt- und Kundendaten sind im Cache

Beim Application Server kann durch das Entfernen der Komponenten sichergestellt werden, dass der Objekt-Cache leer ist. Durch gezieltes Ausführen von Aufträgen kann die gewünschte Situation erzeugt werden. Der dritte Fall, bei dem sich die Börsen- und Produktdaten im Cache befinden, ist der Praxis voraussichtlich am häufigsten auftretende Fall und wird auch im Lasttest untersucht.

Die Ergebnisse zeigen, dass sowohl beim Kauf als auch beim Verkauf die Performance deutlich steigt, je häufiger vom Cache profitiert werden kann und dass das Ausführen von Orders in allen Fällen etwa gleich schnell geschieht. Diese Ergebnisse wurden auch erwartet. Insbesondere das Laden der Börsenplatzdaten ist sehr aufwendig, da für jede Börse ein Datenbankzugriff nötig ist und ein weiterer Zugriff um Handelszeiten zu laden. Die Antwortzeiten in den Fällen zwei bis vier unterscheiden sich nicht mehr so stark, da für die Produkt- und Kundendaten jeweils nur zwei Datenbankzugriffe durchgeführt werden. Da durch das Vorbereiten eines Auftrags bereits alle benötigten Daten bereits im Cache vorhanden sind, findet das Ausführen eines Auftrages in allen vier Fällen unter den gleichen Voraussetzungen statt. Die Schwankungen können nur durch die Hardwareeinschränkungen erklärt werden.

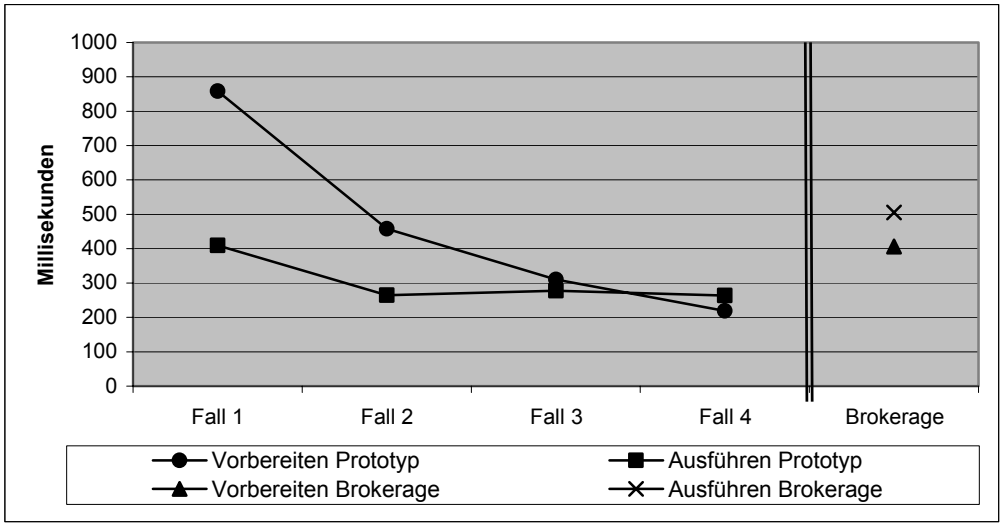

Abbildung 19: Durchschnittliche Antwortzeiten beim Vorbereiten und Ausführen von Kaufaufträgen

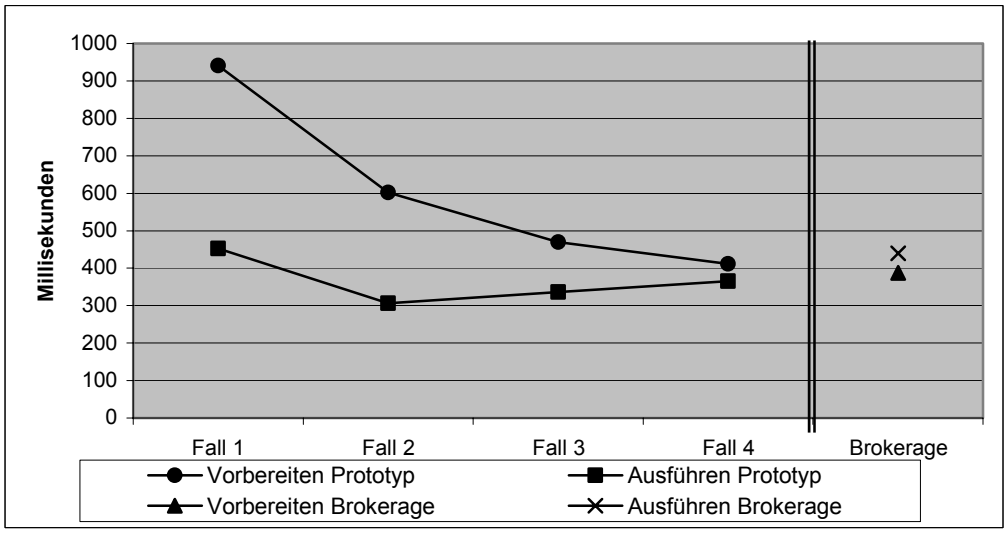

Abbildung 20: Durchschnittliche Antwortzeiten beim Vorbereiten und Ausführen von Verkaufaufträgen

Im Fall vier, bei dem sich alle Daten im Cache befinden, dauert das Ausführen einer Kauforder länger als das Vorbereiten. Wie beim bestehenden System ist die Ursache dafür, dass beim Ausführen Datenbankzugriffe erforderlich sind, um einen Auftrag zu speichern und um sicherzustellen, dass ein Kunde nur einen Auftrag zur Zeit aufgeben kann. Beim Verkauf hingegen dauert das Vorbereiten länger als Ausführen. Dafür kann jedoch keine logische Begründung gefunden werden, so dass nur die geringe Leistungsfähigkeit des Systems als Ursache infrage kommt.

Die insgesamt etwas längeren Ausführungszeiten beim Verkauf können dadurch erklärt werden, dass dort die Prüfungen umfangreicher sind, und dass für die Orderdaten immer ein Datenbankzugriff mehr durchgeführt werden muss als bei Kauf.

Ein Vergleich beider Systeme zeigt, dass die Performance des Prototypen teilweise besser und teilweise schlechter ist als die des bestehenden Systems.

Die durchschnittliche Antwortzeit beim Ausführen einer Verkauforder ist mit 439,55 Millisekunden beim bestehenden System gegenüber 365 Millisekunden beim Prototypen um ca. 20% langsamer, das Ausführen einer Kauforder dauert sogar um etwa 66% länger.

Beim Vorbereiten einer Order hat der Cache entscheidenden Einfluss auf die Performance des Prototypen. Befinden sich noch keine Daten im Cache, benötigt der Prototyp mehr als doppelt so lange wie das bestehende System. Diese Situation wird im Betrieb aber voraussichtlich nur in sehr seltenen Fällen eintreten. Das Vorbereiten einer Verkauforder dauert beim Prototypen immer länger als beim bestehenden System. Die in der Praxis voraussichtlich am häufigsten eintretenden Situation (Fall drei) beträgt der Unterschied immerhin 21%, und selbst im günstigen Fall noch 6%. Beim Vorbereiten einer Kauforder ist die Performance des Prototypen im wahrscheinlich eintretenden Fall um ca. 30% und wenn sich alle Daten im Cache befinden sogar um etwa 85% besser. Wenn sich nur die Börsendaten im Cache befinden, ist der Prototyp ca. 12% langsamer als das bestehende System.

8.3.2 Lasttest

Bei Lasttests werden die Durchläufe normalerweise mit zwischen drei und zehn Wiederholungen durchgeführt.[32] Da die Antwortzeiten der Systeme wegen der geringen Leistungsfähigkeit der Hardware stark schwanken, werden alle Durchläufe mit zehn Wiederholungen durchgeführt. Die zu erzeugende Last muss an die Möglichkeiten der Hardware angepasst werden und kann darum nur in bestimmten Grenzen liegen. Die Last wird, ausgehend von fünf parallelen Clients, jeweils um fünf weitere gesteigert bis die maximal Anzahl von 30 konkurrierenden Clients erreicht ist.

---

[32] Vgl. Gómez, P./Zadrozny, P., J2EE with BEA WebLogic Server, 2000, S.366

Damit alle Aufträge erfolgreich ausgeführt werden können, müssen das System entsprechend konfiguriert und benötigte Testfälle vorbereitet werden. Zu einem Wertpapier werden jeweils zehn verschiedene Depots ausgewählt. Es werden immer 100 Stücke eines Wertpapiers an der Frankfurter Börse zur fortlaufenden Notierung gehandelt. Als Ausführungsdatum wird das aktuelle Datum genommen. Das System wird in einem Modus ausgeführt, bei dem keine Kontostände beim Buchungssystem angefragt werden, sondern immer 1.000.000,01 € als Guthaben angenommen wird. Dadurch ist sichergestellt, dass Käufe immer ausgeführt werden können. Die Produkte werden so ausgewählt, dass der Handel von 100 Stücken an der Frankfurter Börse möglich ist. Die Depots sind nicht gesperrt und ihnen wird jeweils eine Position mit einem Bestand von 1000 Stücken für ein Wertpapier hinzugefügt, so dass auch Verkäufe immer möglich sind. Zwischen den Durchläufen wird der Datenbestand immer in den Ausgangszustand zurück versetzt. Die Zuordnungen von Depots zu einem Wertpapier werden in einer Properties-Datei abgelegt. Auf diese Datei wird aus den Testklassen für den Grinder zugegriffen. Die Properties-Datei wird während der Initialisierungsphase des Grinders verarbeitet. Dadurch ist sichergestellt, dass für beide Systeme für eine bestimmte zu simulierende Last immer dieselben Testfälle verwendet werden. Der Grinder wird so konfiguriert, dass die erste Methodenausführung zu einem zufällig Zeitpunkt innerhalb der ersten Sekunde nach Start des Testlaufs durchgeführt wird, und dass zwischen zwei Methodenaufrufen eine Sekunde gewartet wird.

Werden alle vier Schritte (Anmelden, Vorbereiten einer Order, Ausführen einer Order und Abmelden) betrachtet, zeigt sich, dass sowohl beim Kauf als auch beim Verkauf der Prototyp über eine bessere Performance verfügt als das bestehende System. Lediglich bei niedriger Last ist der Prototyp unterlegen. Die Ursache für die rapide Leistungsabnahme vom bestehenden System gegenüber dem Prototypen liegt vermutlich in der Konfiguration der Datenbank, die durch die Beschränkung der Hardware beeinflusst wird. Je größer die Last wird, um so mehr nähert sich der Durchsatz beider System einander an. Anhand der Trendlinien kann man erkennen, dass der Prototyp beim Verkauf besser skaliert als das bestehende System. Beim Kauf hingegen nimmt die Leistungsfähigkeit beider Systeme annähernd gleichermaßen ab.

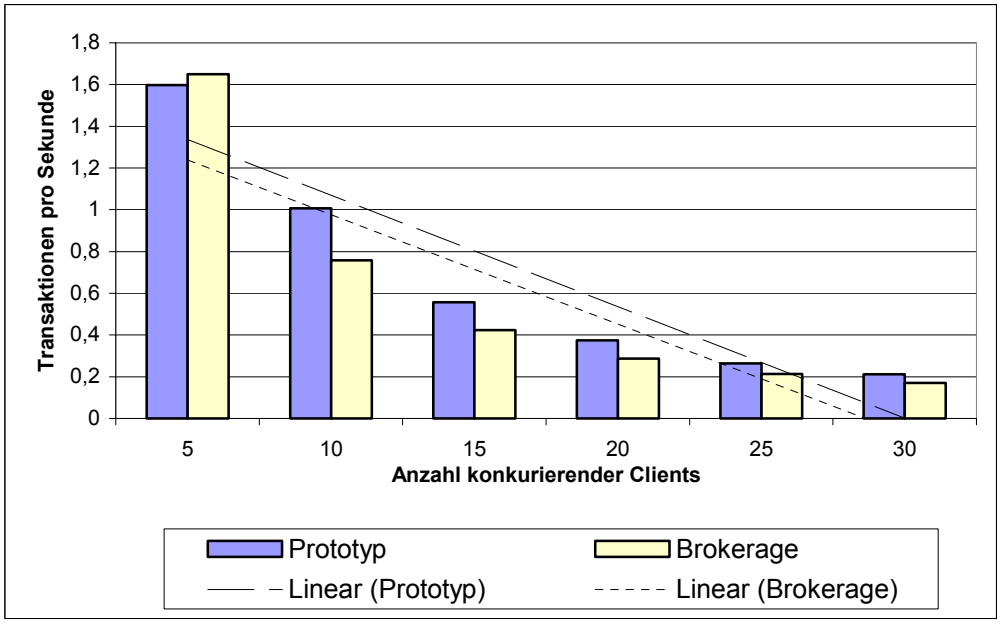

Abbildung 21: Durchsatz beim Aufgeben vom Kaufaufträgen bei zehn Wiederholungen pro Client

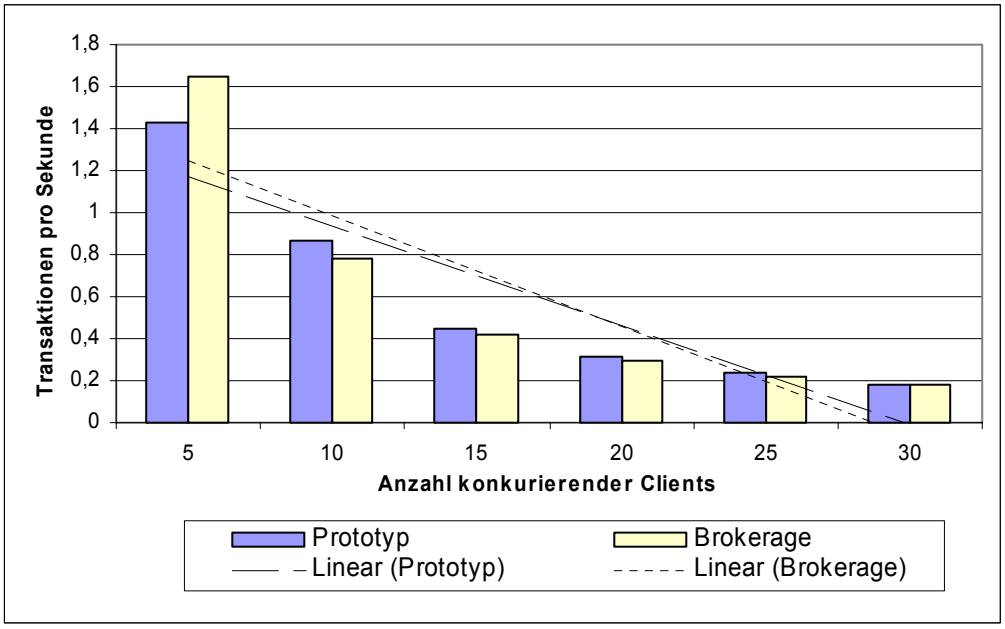

Abbildung 22: Durchsatz beim Aufgeben vom Verkaufaufträgen bei zehn Wiederholungen pro Client

Um die Leistungsfähigkeit des Prototypen besser beurteilen zu können, werden nur die Schritte Vorbereiten und Ausführen einer Order betrachtet, da für das An- und Abmelden dieselben Komponenten verwendet werden.

Der Prototyp verfügt beim Kauf über eine bessere Performance als das bestehende System. Insbesondere das Ausführen von Kaufaufträgen ist im bestehenden System weniger performant. Durch die Trendlinie wird deutlich, dass

die Leistungsfähigkeit des Prototypen bei steigender Last insbesondere beim Ausführen stärker fällt als beim bestehenden System.

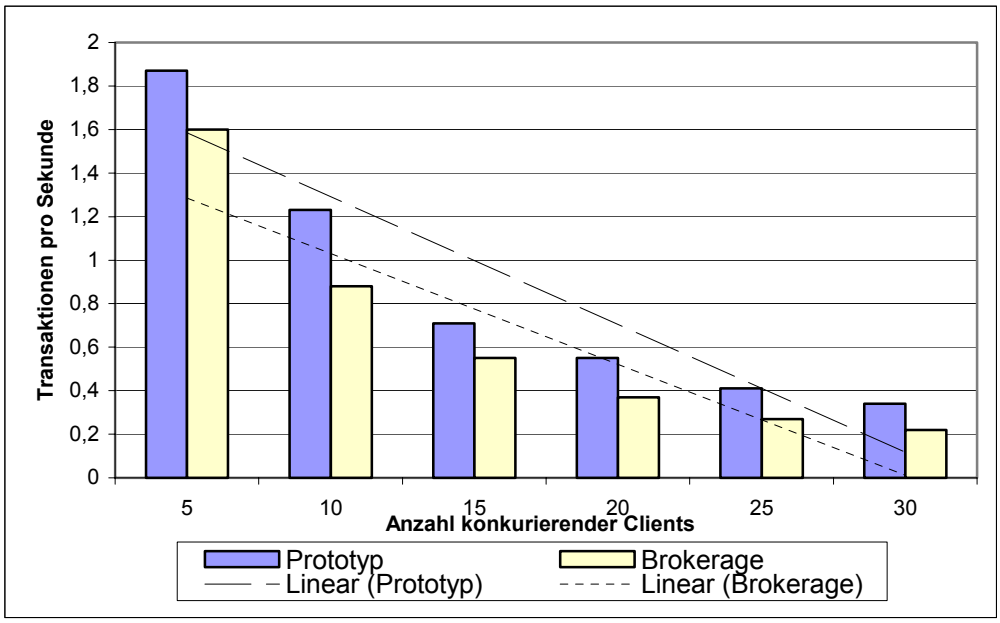

Abbildung 23: Durchsatz beim Vorbereiten einer Kauforder bei zehn Wiederholungen pro Client

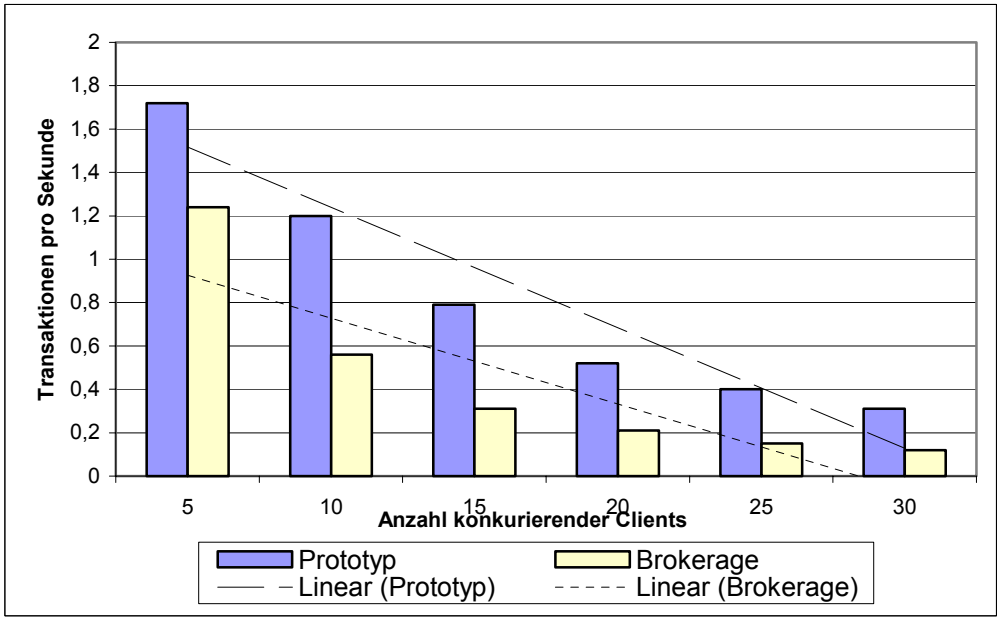

Abbildung 24: Durchsatz beim Ausführen einer Kauforder bei zehn Wiederholungen pro Client

Beim Verkaufen hingegen ist das bestehende Brokerage-System beim Vorbereiten und beim Ausführen mit geringer Last performanter als der Prototyp. Bei höherer Last kann der Prototyp beim Ausführen mehr Transaktionen pro Sekunde durchführen. Beim Vorbereiten skaliert der Prototyp deutlich besser; die

Trendlinien schneiden sich sogar. Beim Ausführen nimmt die Leistungsfähigkeit in beiden Systemen gleichstark ab.

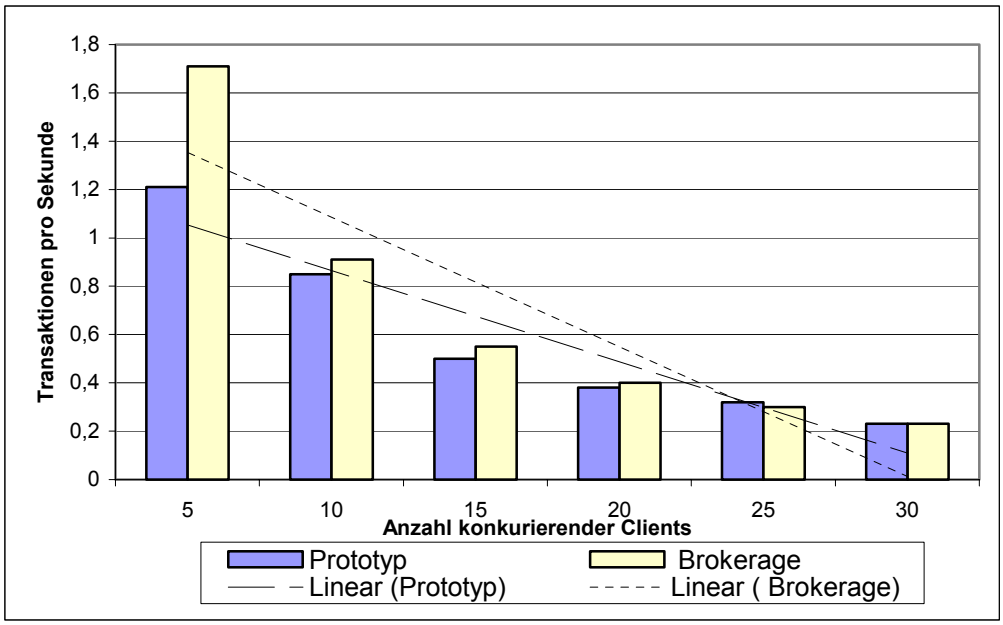

Abbildung 25: Durchsatz beim Vorbereiten einer Verkauforder bei zehn Wiederholungen pro Client

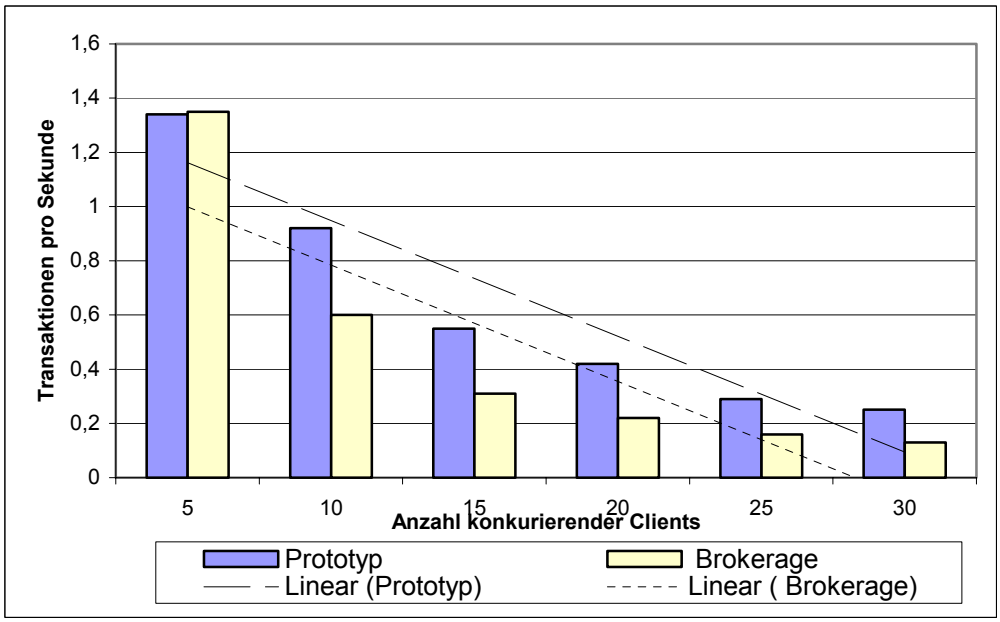

Abbildung 26: Durchsatz beim Ausführen einer Verkauforder bei zehn Wiederholungen pro Client

Insgesamt ist festzustellen, dass der Prototyp so stark vom Caching profitieren kann, dass in den meisten Fällen mehr Durchsatz erzielt werden kann als beim bestehenden System. Nur in wenigen Situationen ist das bestehende System überlegen. Beim Kaufen ist das bestehende Brokerage-System skalierbarer,

wohingegen der Prototyp beim Verkaufen besser skaliert, so dass keine Aussage zugunsten eines der Systeme möglich ist. Leider war es nicht möglich, für beide Systeme den neueren WebLogic Server 6.1 einzusetzen. Möglicherweise hätte sich dadurch die Performance des bestehenden Systems noch etwas verbessert. Eine deutliche Veränderung ist aber angesichts der Tatsache, dass innerhalb der Geschäftsschicht kaum Geschäftslogik ausgeführt wird, nicht zu erwarten. Durch die stufenweise Steigerung der Belastung sind keine Unregelmäßigkeiten aufgefallen.

Die Ergebnisse lassen sich nicht verallgemeinern, da keine optimale Umgebung für die Durchführung der Analyse zur Verfügung stand. Rückschlüsse auf den Produktionsbetrieb können deswegen ebenfalls nicht gezogen wegen. Hinzu kommt noch, dass im Produktionsbetrieb durch Clustering gezielt auf die Skalierbarkeit Einfluss genommen wird.

Bezieht man in die Betrachtung mit ein, dass in der Version 6.1 des WebLogic Servers nur in bestimmten Fällen von den vorgehaltenen Daten profitiert werden kann, ist für zukünftige Versionen noch eine Leistungssteigerung zu erwarten. Andererseits muss auch berücksichtigt werden, dass die eingesetzten Read-Only Beans noch nicht Bestandteil der EJB-Spezifikation sind. Es ist unklar was Read-Only Beans für Möglichkeiten bieten können, wenn sie in die Spezifikation aufgenommen sind.

9 Schlusswort

9.1 Rückblick

Im Rahmen dieser Diplomarbeit wurde der Prototyp eines Brokerage-Systems erstellt. Im Kern stand die Geschäftslogik des Geschäftsvorfalls „Order aufgeben“. Als besondere Herausforderung haben sich dabei die Komplexität und der Umfang der Geschäftslogik erwiesen. Von einem ersten Entwurf ausgehend, haben nach und nach neue Erkenntnisse und Erfahrungen im Umgang mit der Technologie zu einer kontinuierlichen Verbesserung des Prototypen geführt.

Neben der Geschäftslogik gibt es noch viele weitere Aspekte, die nicht berücksichtigt werden konnten. Die Sessiondaten werden ausschließlich gelesen, so dass auch hier alternative Ansätze, z.B. mit Read-Only Beans, vorstellbar sind. Wesentliche Bestandteile des bestehenden Systems sind der Datenimport in die Schattendatenbank, der mit Stored Procedures durchgeführt wird, und die Kommunikation mit dem Hostsystem der Wertpapiersammelbank. Erst anhand von weiteren Untersuchungen kann entschieden werden, ob das gesamte Brokerage-System mit den Möglichkeiten von EJB 2.0 und des WebLogic Servers umgesetzt werden kann.

In der jetzigen Form kann der Prototyp den Geschäftsvorfall „Order aufgeben“ vollständig abdecken und ist auch für den Einsatz in einem Produktivsystem bereit. Es ist aber ratsam, das Verhalten des Systems beim Aufgeben einer Order gezielt durch Transaktionen zu beeinflussen. So kann erreicht werden, dass auch in kritischen Situationen, die durch Serverabstürze oder Netzwerkausfall entstehen, Aufträge nur einmal aufgegeben werden. Bei der Entwicklung des Prototypen wurde auf diese Absicherung verzichtet.

9.2 Bewertung

Der Prototyp hat gezeigt, dass es auf der Basis von EJB 2.0 und dem WebLogic Server 6.1 möglich ist die Anforderungen zu erfüllen. Die Neuerungen der Spezifikation haben es ermöglicht, fein-granulare Entity Beans mit CMP einzusetzen. Clients kommunizieren nur mit der Steuerungskomponente OrderFacadeBean. Die gesamte Geschäftslogik wird innerhalb der Geschäftsschicht abgebildet. Es gibt für den betrachteten Geschäftsvorfall keine Abhängigkeiten

mehr von der Datenbank, und der Prototyp ist konform zum Schichtenmodell der J2EE-Architektur.

Beim Design wurde insbesondere auf die Leistungsfähigkeit geachtet, so dass keine Stateful Session Beans eingesetzt wurden und zugunsten von Read-Only Beans auf CMR verzichtet wurde. CMR ist eine wichtige Neuerung in CMP 2.0, so dass deren Einsatz wünschenswert war. Die Performanceanalyse hat jedoch bewiesen, dass zu Recht auf CMR verzichtet wurde.

Die Leistungsfähigkeit des Prototpen zu beurteilen ist schwierig. Nach den Testergebnissen zu Urteilen, ist der Prototyp durchaus konkurrenzfähig zum bestehenden Brokerage-System, so dass es sinnvoll ist den gewählten Ansatz weiter zu verfolgen. Die Testmöglichkeiten sind durch die zur Verfügung stehende Hardware eingeschränkt worden. Darum sind in jedem Fall weitere Test erforderlich, um eine endgültige Aussage treffen zu können.

Maßgeblichen Anteil an der Leistungsfähigkeit haben die nicht spezifizierten Fähigkeiten des WebLogic Servers. Sofern in zukünftigen Versionen der EJB-Spezifikation Read-Only Beans aufgenommen werden, bleibt abzuwarten, welche Möglichkeiten diese bieten. Möglicherweise werden auch dann die herstellerspezifischen Fähigkeiten entscheidend die Performance beeinflussen.

Literaturverzeichnis

Enterprise Java Beans Specification, Version 2.0, 2001

Gamma, Erich/Helm, Richard/Johnson, Ralph/Vlissides, John [Entwurfsmuster, 1996]: Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software; Bonn: Addison-Wesley, 1996

Gray, Jim [Benchmark]: http://www.benchmarkresources.com/handbook/1-1.asp

Grill, Wolfgang/Perczynski, Hans [Wirtschaftslehre, 1998]: Wirtschaftslehre des Kreditwesens, 32. Auflage, Bad Homburg vor der Höhe: Gehlen, Juni 1998

Gómez, Paco/Zadrozny, Peter [J2EE with BEA WebLogic Server, 2000]: Professional Java 2 Enterprise Edition with BEA WebLogic Server, Birmingham: Wrox 2000

Hagenmüller, Karl Friedrich/Diepen, Gerhard (Hrsg.): Der Bankbetrieb: Lehrbuch und Aufgabensammlung, 12. Aufl., Wiesbanden: Gabler 1989

Horstamnn, Cay S./Cornell, Gary: Core Java 2 – Grundlagen, Band 1, München: Prentice Hall, 1999

[BluePrints] Java™ BluePrints – Guidelines, patterns, and code for end-to-end Java applications, http://java.sun.com/blueprints/

Java™ 2 Platform, Enterprise Edition (J2EE™) Specification, Version 1.3

Marinescu, Floyd [EJB Design Patterns, 2002]: EJB Design Patterns – Advanced Patterns, Processes, and Idioms, New York: Wiley, 2002

Maron, Jonathan/Pavlik, Greg: EJB 2.0 Impacts Next-Generation J2EE Servers, http://www.java-pro.com/upload/free/features/javapro/2001/02feb01/jm0102/jm0102p.asp, Februar 2001

Mougin, Philip: EJBs from a critical perspective, http://www.techmetrix.com/trendmarkers/tmk1299/tmk1299-2.php3, Dezember 1999

Oestereich, Bernd [Objektorientierte Softwareentwicklung, 2001]: Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language, 5. völlig überarbeitete Aufl.., Oldenbourg, 2001

Pawlan, Monica: Introduction to the J2EE Plattform, http://developer.java.sun.com/developer/technicalArticles/J2EE/Intro/Index.html, 23.3.2001

Programming WebLogic Enterprise JavaBeans, 30.7.2001

Roman, Ed [Mastering Enterprise JavaBeans, 1999]: Mastering Enterprise JavaBeans and the Java 2 Plattform Enterprise Edition, New York: Wiley, 1999

Roman, Ed/Ambler, Scott/Jewell, Tylor: Mastering Enterprise JavaBeans Second Edition, New York: Wiley, 2002

Shah, Apu: A Peek at EJB 2.0, Part 1, http://softwaredev.earthweb.com/java/article/0,,12082_887081,00.html, 18.9.2001

Shah, Apu: A Peek at EJB 2.0, Part 2, http://softwaredev.earthweb.com/java/article/0,,12082_891431,00.html, 25.9.2001

Shah, Apu: A Peek at EJB 2.0, Part 3, http://softwaredev.earthweb.com/java/article/0,,12082_896011,00.html, 2.10.2001

Sun Java Center J2EE™ Patterns – Aggregate Entity, http://developer.java.sun.com/developer/restricted/patterns/AggregateEntity.html

Sun Java Center J2EE™ Patterns – Data Access Object, http://developer.java.sun.com/developer/restricted/patterns/DataAccessObject.html

Sun Java Center J2EE™ Patterns – Value Object, http://developer.java.sun.com/developer/restricted/patterns/AggregateEntity.html

Using WebLogic Server Clusters, 30.7.2001

Wertpapier-Mitteilungen Datenservice: ISIN-Einführung (Stand 10/2001), Oktober 2001

Anhang

Im Anhang dieser Arbeit befinden sich ein ausgedrucktes Klassendiagramm und eine CD. Das Klassendiagramm enthält fast alle Klassen und Schnittstellen des Prototypen. Lediglich die Exception-Klassen und der Inhalt des Pakets util sind nicht mit abgebildet. Auf der CD befinden sich alle Quelldateien und Deployment Descriptoren sowie die Konfigurationsdateien des Grinders.