# 11

# Case Study: Overview of Securities Trading System

## CERTIFICATION OBJECTIVES

11.01 Example of SCEA Part 2

This chapter presents a case study that will help you to prepare for and complete Part 2 and 3 of the SCEA exam. You should also review Chapter 3 on object-oriented design. Parts 2 and 3 do not require any code. Part 2 is an architecture and design project. Part 3 is a set of essay questions related to your project. The essays we include in other chapters in the book should help you with this part. The nature of the Part 2 assignment requires that you architect a solution for a small business system. To keep the amount of work involved to a reasonable level, the programs you create will be restricted in capability, and simpler than anything you would actually create for a "real-world" client. You will be graded on correctly solving the technical and performance requirements.

According to the Sun site, your project will be evaluated on a large number of objective criteria that fall into three categories:

1. **Class Diagram** This category covers how well your class diagram(s) address the object model needed to satisfy the requirements.
2. **Component Diagram** This category covers how well your component diagram(s) convey the structure of the architecture in satisfying the requirements.
3. **Sequence/Collaboration Diagrams** This category covers how well your sequence or collaboration diagrams satisfy the requirements of the assignment.

Additionally, each category is evaluated on UML compliance.

The maximum number of possible points is 100. The minimum passing grade is 70. The maximum points per category are

| | |
|---|---|
| Class Diagram(s) | 44 maximum points |
| Component Diagram(s) | 44 maximum points |
| Sequence/Collaboration Diagrams | 12 maximum points |

In this chapter, we present you with a securities trading application that includes use cases, a domain object model, and additional requirements. Most systems in the real world start off with requirements, and the exam assignment chooses to define the requirements in use cases and a domain model. As the architect for the application, you must develop the class diagram(s), component diagram(s), and sequence or collaboration diagrams to describe your architecture.

This particular case study application was developed for a Wall Street clearing firm, which interacted with hundreds of correspondent trade brokers and their customers. It was intended to help the firm by facilitating the customers' ability to trade securities on the web. To be successful, a trading firm (broker-dealers) must

transition their firm to a customer-centric, web-based environment. The Internet was seen as a way to maintain and grow new and existing distribution channels, customers, and strategic partnerships. Basically, the challenge was to determine how this trading firm could develop a winning strategy to compete for customers and brokers who wanted the ease of trading securities via the web.

The solution is the same for all businesses competing in the new e-commerce world—a three-step approach to successful enterprise development:

1. Spend the time to understand what the client needs—in this case, browser-based trading functionality.
2. Choose the right technology—in this case, Java and Java 2 Enterprise Edition (JEE).
3. Develop a team and teamwork atmosphere to implement the technology—choose architects and developers skilled in trade processing and JEE design and development.

# The Case Study Infrastructure

As with many large organizations, Bank of New Amsterdam (BNA) has several different operating systems running many kinds of software systems. The primary platform for the production business data is the IBM Mainframe S/390 (the mainframe). In addition, Microsoft NT boxes use Internet Information Services (IIS) and other Microsoft software to provide reporting capabilities, using a SQL Server shadow copy of the business data. BNA also has an investment in Oracle on the S/390. Several preexisting legacy applications provide trade-processing links to the major securities exchanges. These applications work well and have been developed and maintained for the past two decades, during which time a great deal of time and money has been spent. The applications do not need modification; instead, they need a new web front end to make them look good. The development environment is primarily Windows-based.

The advantages of using the mainframe include its reliability, scalability, flexibility, and security. The mainframe has been running continuously for years in BNA without major problems. The mainframe can easily be amended to add hardware resources, such as CPU, memory, disk, or networking hardware, to increase capacity without changing the operating system or application systems. In addition, the CPU, memory, and disk space can be redistributed as application requirements change.

Traditionally, the downside to the mainframe has been the user interface—the 3270 dumb terminal (the green screen), which is not user friendly. Prior to the commencement of BNAs Java project, another group spent a few months trying to develop Windows Active Server Pages (ASPs) to talk to the mainframe. The only solution providing ASP reports, a portfolio management system, solved the interface problem, but it was difficult to connect to the mainframe using ASP for trading.

BNA has experienced difficulty getting the order messages to the mainframe. Along with performance problems, it seemed that every ASP order transaction required multiple dedicated connections to the SQL Server database. Fortunately, Java Database Connectivity (JDBC) connection pooling and Java's platform independence provided the performance and scalability that was needed. Moreover, it allowed us to take advantage of the mainframe for deployment and Windows for development. (Because budget is a limiting factor in an economic downturn, where every developer is competing for business, it is critical that you deliver a solution quickly that will integrate with an organization's existing infrastructure.)

The IBM HTTP server, WebSphere application server, and Oracle 9*i* Enterprise relational database management system (RDBMS) are all in existence on the company's legacy IBM mainframe enterprise server.

Table 11-1 describes the software components of the web front end.

Figure 11-1 illustrates the enterprise architecture as a diagram, and Figure 11-2 shows a Unified Modeling Language (UML) sequence diagram.

**TABLE 11-1** Web Front End Software

| Type of Component | Vendor/Component Name/Version | Description | Software/Hardware Required for Support |
|---|---|---|---|
| Application server | WebSphere Application Server 5 | Serves up JSPs and servlets, runs EJBs, and provides JDBC connection pools to user data | IBM S/390, Windows XP SP2 |
| DBMS | Oracle 10.0*i* | Database required for application data (orderdb) | Oracle 10*i* |
| JavaScript | Netscape/JavaScript | ECMA JavaScript | IE 6 and above or Navigator 5 and above |
| Browser | Microsoft or Mozilla Firefox | Web browser required to support JavaScript | IE 6 and above or Navigator 5 and above |
| XML Tool | IBM/XML Parser for Java/3.1.1 | A library for parsing and generating XML documents | Windows XP SP2 |

**FIGURE 11-1**

Production architecture components

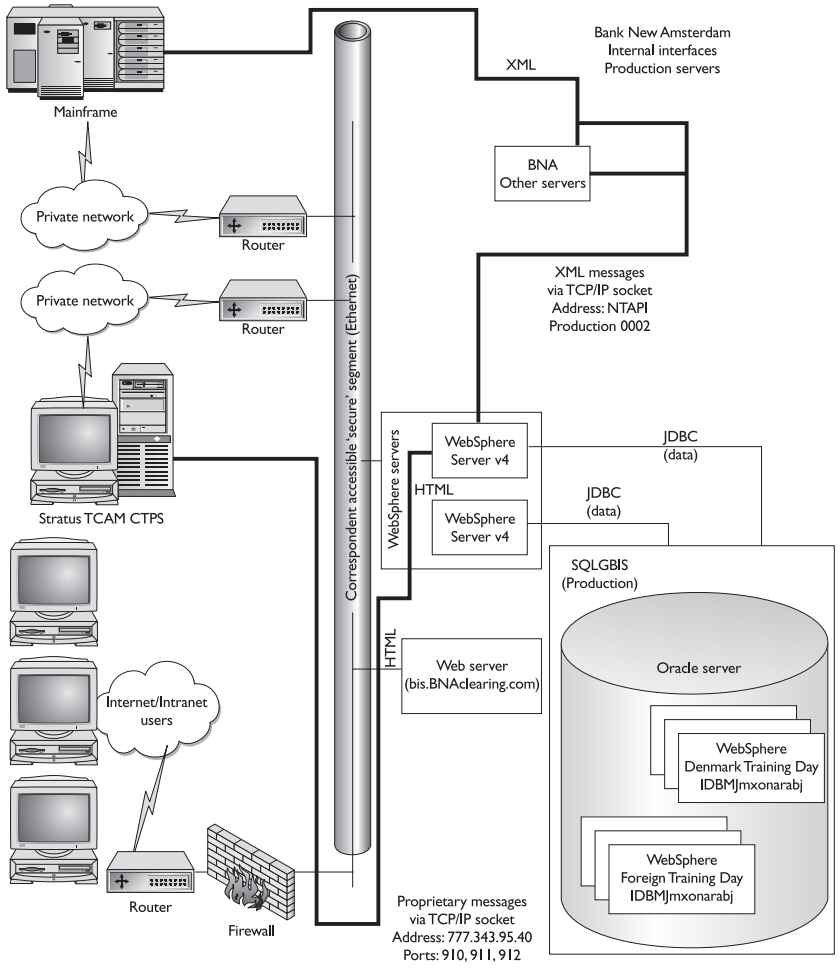
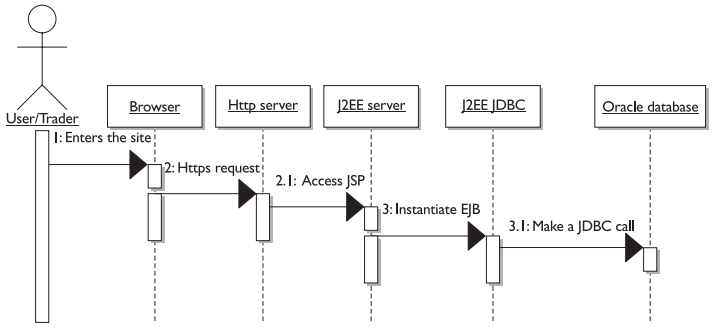



**FIGURE 11-2**

Production architecture as a UML sequence diagram



### WebSphere Application Server

The WebSphere partition is also connected via TCP/IP sockets to the BNACS API. This API is used to communicate with and retrieve information from the mainframe system via an XML-based message format. Finally, the WebSphere partition is connected to the Stratus TCAM CTPS (the continuous trade processing system from TCAM) application. The WebSphere application sends and maintains orders, and it can look up order status by communicating with CTPS via a proprietary message format.

### Continuous Trade Processing

The CTPS system is an order-routing system that is connected to several exchanges and market makers (firms that stand ready to buy and sell a particular stock on a regular and continuous basis at a publicly quoted price). The system receives orders either by direct entry into its terminals or via an in-house-built TCP/IP socket server (sometimes known as the Stratus Gateway Interface). These orders are routed to the appropriate exchange or market maker according to a set of correspondent-defined rules. Executions are then passed back from the exchange or market maker to CTPS, which updates the order file and forwards the result of the execution to the mainframe.

### SQLBIS Database Server

The SQLBIS database server was created to service BNAs trading web site, Brokerage Information System (BIS). On the database server are several databases (or data *marts*) that are used by BIS and the BIS Trading Area to look up account and application access as well as cross-references and other information. The ORDERDB database was created exclusively to support the WebSphere applications. In development, the tables and views were created by developers and migrated to production by the database administration (DBA) group.

## Model and Develop the Case Study

This section describes the case study trading application using text and diagrams (mostly UML diagrams). This task is not unlike what you as a SCEA Part 2 test taker must accomplish. As you saw in Chapter 3, UML diagrams and use cases

can replace what was formerly called the *functional requirements* in an application development scenario. Moreover, UML is an adopted and widely accepted standard used to describe business processing. UML provides benefits to architects and enterprises by facilitating the construction of robust and maintainable business models, which can support the entire software development life cycle (SDLC).

The *use case model* describes the target functionality of a new application. A *use case* represents a unit of interaction between an actor and some function. A *use case diagram* describes an interaction between an actor and the system. It presents a collection of use cases and actors and typically specifies or characterizes the functionality and behavior of an enterprise application interacting with one or more external actors. The users and any system that may interact with the system are the *actors*. Actors help delimit the system and give a clearer picture of what it is supposed to do.

Use cases are developed on the basis of the actors' needs. This ensures that the system will turn out to be what the users expected. Use case diagrams contain stick figure icons that represent actors, association relationships, generalized relationships, packages, and use cases. A top-level use case diagram shows the context of a system and the boundaries of the system's behavior. One or more use case diagrams can be drawn to describe a part of an application system. Use cases can include other use cases as part of their behavior. A use case diagram shows the set of external actors and the system use cases in which the actors participate.

After the use case model is completed and signed off by the business managers, development begins in earnest. For the remainder of the chapter, we will mix some of the UML modeling techniques with the actual development product to illustrate the case study.

A use case model typically comprises the following interrelated components:

- Actor definition
- Business process model
- Sequence diagrams
- Class descriptions
- Class diagrams
- State transition (life cycle) diagrams

## Actor Definition

The people involved in the business process are described as a series of actors, who may represent existing jobs or roles in the organization or may be completely new jobs or roles. Table 11-2 shows the various actors involved in a business process and their roles.

TABLE 11-2

Actors and Their Roles

| Actor | Description |
|---|---|
| Customer | Trades with the application according to limits |
| Trader | Trades with the application without limits |
| Continuous trade processing system (CTPS) | Routes orders to the mainframe trading system |
| Mainframe Trading System | Holds BNACS back office books and records |
| SQL (Oracle) | Contains the database reference data |

## Business Process Model

A number of *scenarios*, specific examples of performing the task, are identified for each task that is carried out in the business process.

The *business process*, or task model, describes how the business processes will perform the necessary tasks with or without a computer application. It represents an important aspect of the business requirements, since it describes from a user and business perspective what work is done. The model provides the basis for designing the functionality of the computer application.

A *business process model* is a model of one or more business processes. Each process has a process owner and process goals (such as cycle time, defect rate, and cost) and consists of a set of business activities (in sequence and/or parallel). Figure 11-3 shows an example of a securities trade order and the processing steps it goes through from submission to completion.

### Development Environment and Database Design

Before we begin the physical construction of the application components, we must make certain that prerequisite physical items such as infrastructure, development environment, and so on are in place for use by the development team. In addition to an adequate workstation and the appropriate server(s), the JEE project libraries and the development GUI presentation tool such as JBuilder are accessible to developers with the appropriate permissions in place. The RDBMS application database, in this case Oracle, with current maintenance and whatever third-party or in-house JEE development software, is ready for use from each workstation. Developers have been availed of the guidelines and naming standards that the project team agreed to use to develop both the database and the application.

Important for development is the preliminary physical database design used for the application. The database design at this point can differ from that of the final

**FIGURE 11-3** A trade as a business process

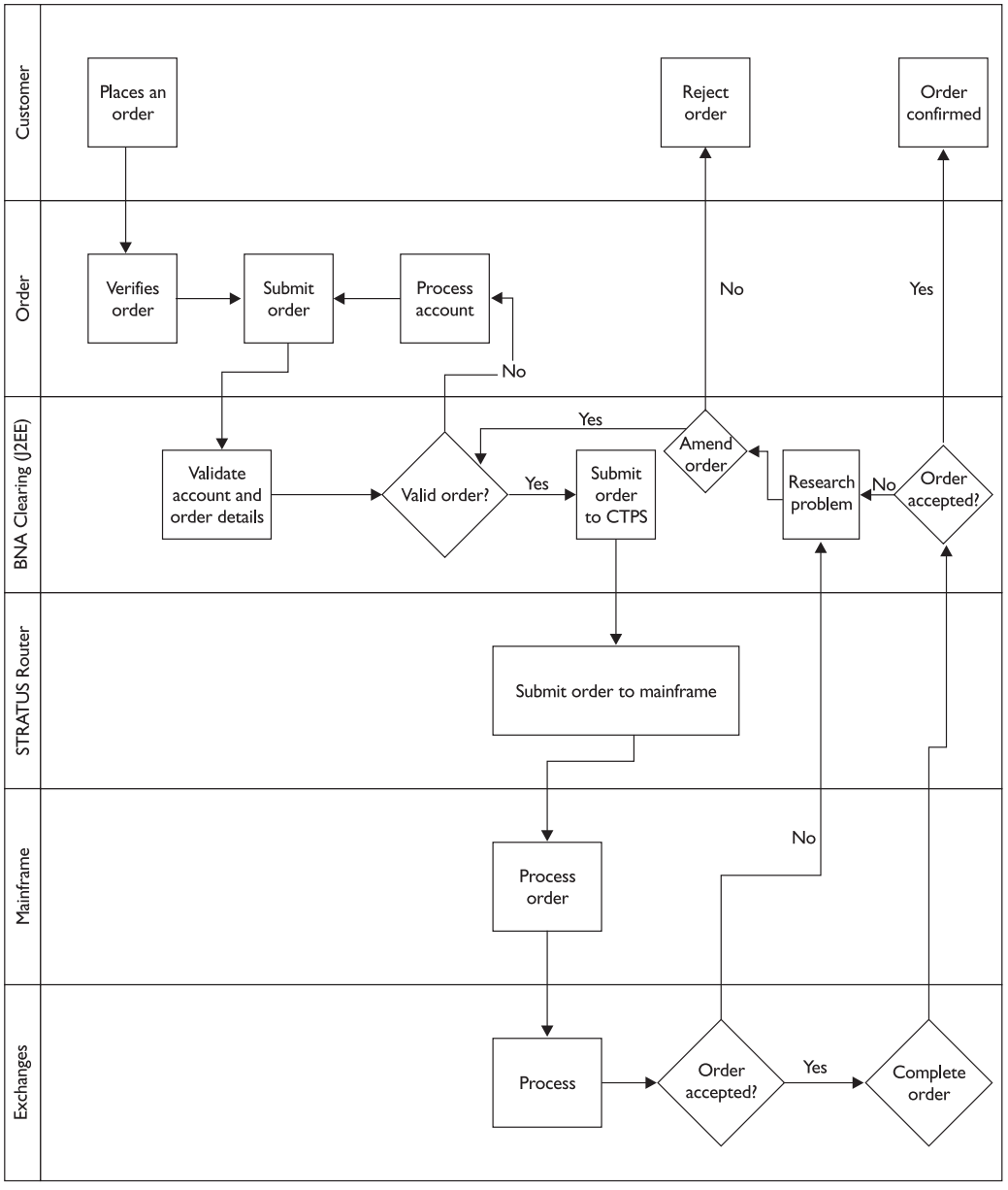
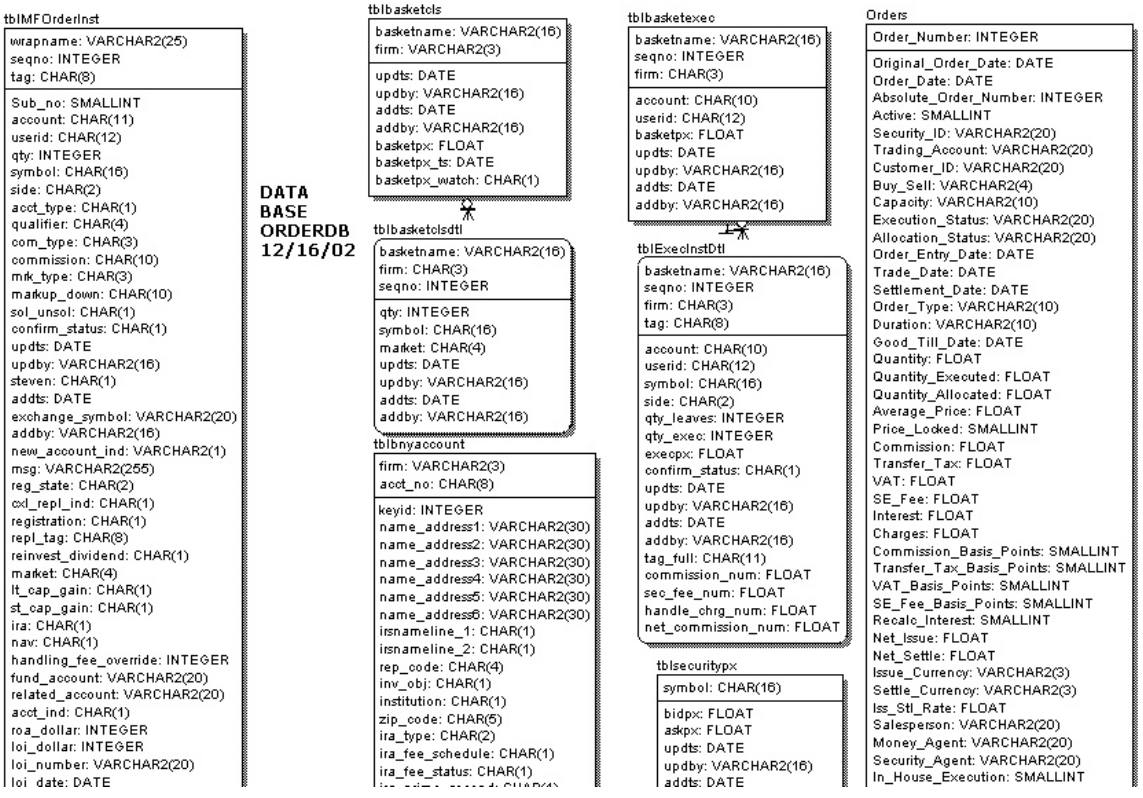


**FIGURE 11-4** The ORDERDB database




application database design, but it is eventually reconciled in terms of function back to the overall design.

This design of the trade system ORDERDB, shown in Figure 11-4, meets the constraints of the DBMS, and since it is derived directly from a composite data model, it also satisfies the system requirements. The rules ensure that the physical design is valid and that it follows good practice. During development, no attempt is made to achieve good performance. Rather, the design provides a sound starting point for physical tuning when the design of both database and Java classes is adjusted to achieve performance objectives. The database characteristics can vary depending on the DBMS being used.

The key characteristics of many of the popular DBMS engines, such as Oracle, are built according to the following rules:

1. **Entities** Most entities on the composite data model become tables. The key-only entities may be paired with other key-only entities to form junction tables, which can speed up joins.
2. **Primary Keys** The primary key of each entity becomes the primary key of the corresponding table. Specify a UNIQUE index for the entire primary key.
3. **Alternate Keys** Each alternate key becomes a UNIQUE secondary index.
4. **Foreign Keys** Indexing each entity foreign key becomes a secondary NON-UNIQUE index.
5. **Referential Integrity** Make each entity foreign key a FOREIGN KEY for the table, referencing the master of the supported relationship.
6. **Other Non-Unique Keys** Each other non-unique key becomes a NON-UNIQUE index on the corresponding column(s).

. **Exclusive Relationships** If a detail entity has two or more mutually exclusive masters:

- Provide foreign key indexes to support each relationship as defined in Rule 4.
- The foreign key columns for the relationships should all be defined as NULLS ALLOWED.
- Maintenance of exclusivity must be handled by program.

## Developing the Trading System

The trading system is a browser-based user interface that provides trading functionality—that is, the ability to send trade orders (even baskets of stocks) and view customer account and trade order requests to BNA. Written in Java with JEE, it provides the customer with an integrated, platform-independent method for accessing account information and submitting and viewing orders via the Internet or a private network. All trading functionality is accessible via a single menu page, which uses a frameset with a header, footer, and a navigation frame on the left to expose the functionality in the right side mainframe (no pun intended). Figure 11-5 shows the main trading frameset.

On the left side of the trading page is a frame that exposes the functionality that is currently available for trading—stocks, bonds, mutual funds, as well as baskets and multiple orders—along with some basic operational functionality used to maintain accounts and other external information. The main trading application page and operations page can be depicted as use case diagrams, as shown in Figures 11-6 and 11-7.

**FIGURE 11-5**

Trading page main menu frameset

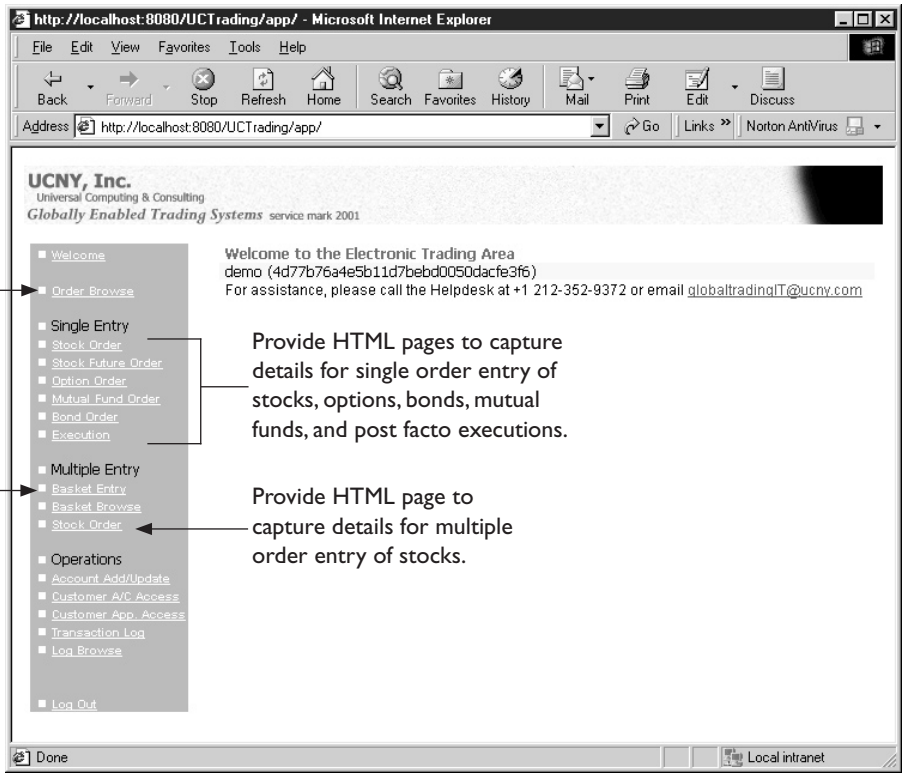



**FIGURE 11-6**

Trading Page functionality as a use case

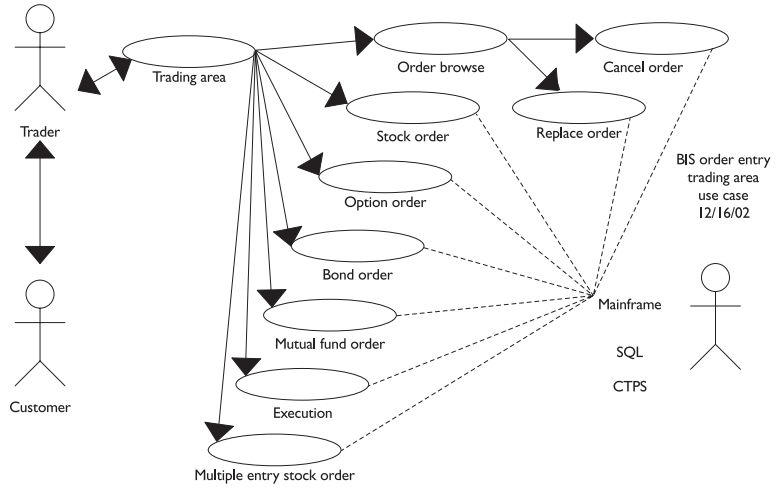


**FIGURE 11-7**

Operations page functionality as a use case

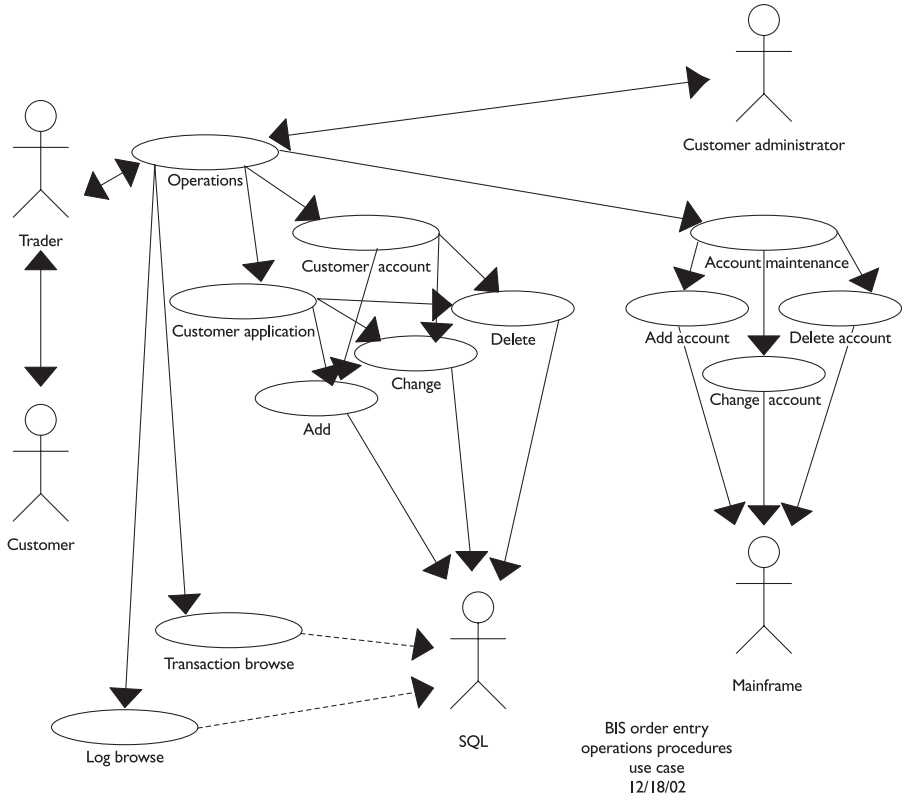



Table 11-3 describes the use case task goals and scenarios.

## Sequence Diagrams

Sequence diagrams are models of business processes that represent the different interactions between actors and objects in the system. Each process has a process owner and goals (such as cycle time, defect rate, and cost) and consists of a set of business activities (in sequence and/or in parallel). Figure 11-8 depicts the sequence of a trade order being placed by a customer as it moves through the JEE application server for verification and onto the trade process router to the mainframe and then to the securities exchange, where the actual trade is executed. After a confirmed execution, each of the front-end processors is notified, and ultimately the customer is availed of the completed order and price.

**TABLE 11-3** Tasks and Scenarios

| Task Name | Task Goal | Task Scenarios |
|---|---|---|
| All forms of order entry: equity, bonds, mutual funds, baskets | Send order to CTPS routed to the mainframe trading system | Order information is keyed in, and information is edited and routed to CTPS and ultimately to mainframe trading system for processing. |
| Order browse | Review/cancel/replace orders that have been previously sent to CTPS-mainframe trading system | Previously entered order information is edited and sometimes replaced; information is edited and routed to CTPS and ultimately to mainframe trading system for processing. |
| Account maintenance | Add/update/delete retail customer accounts | Account information is edited and sometimes replaced; information is edited and routed to mainframe trading system for processing. |
| Customer application maintenance | Add/update/delete customer application entitlement | Application information is edited and sometimes replaced; information is edited and routed to SQL for processing. |
| Customer account maintenance | Add/update/delete customer account/user entitlement | Account/user information is edited and sometimes replaced; information is edited and routed to SQL for processing. |

## Class Descriptions

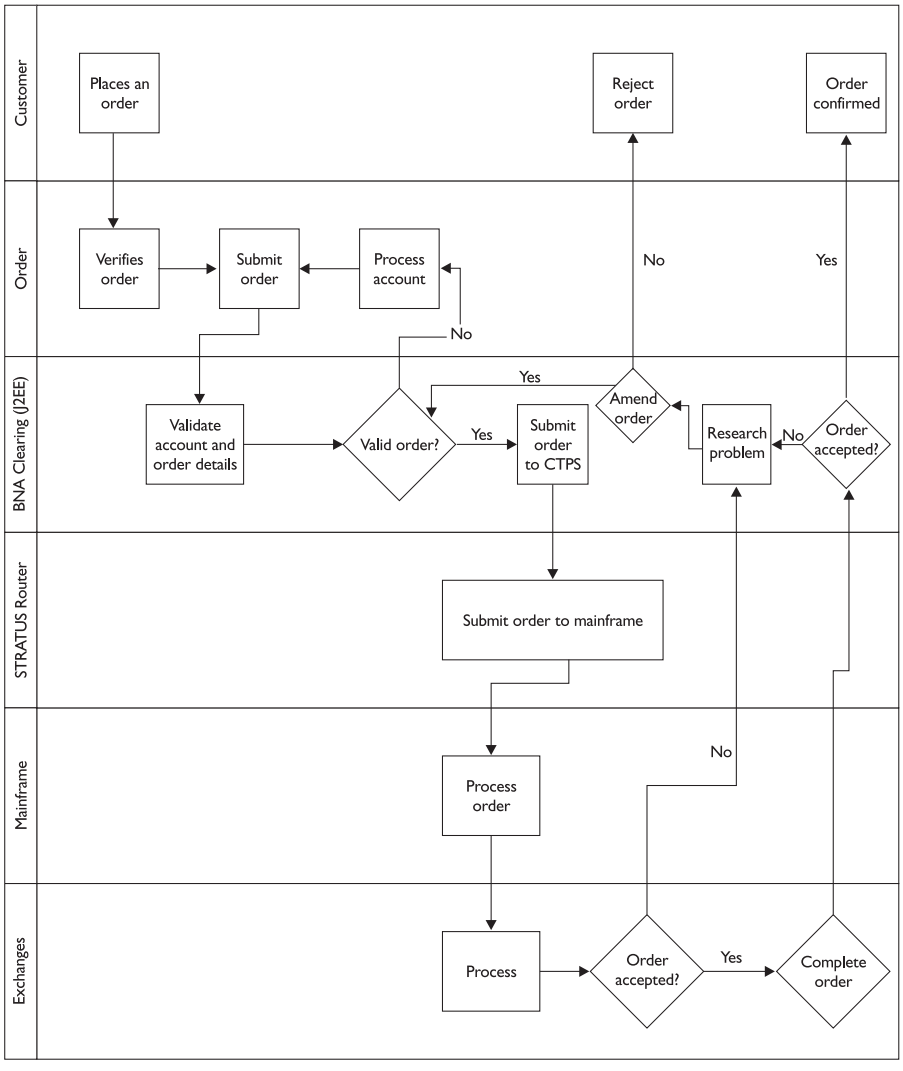
Table 11-4 describes the business classes included in the business process sequence diagram in Figure 11-8.

## Class Diagrams

Using the initial list of business classes, you develop class diagrams by identifying and defining the relationships among the classes. This is best done in an interactive development workshop with business partners. It is also useful to keep these diagrams on display on a whiteboard or other medium, and to develop it gradually as the project progresses. The diagrams can also be stored on a UML tool to provide access to all team members and other interested parties.

**FIGURE 11-8**

Sequence of trade order




The class diagrams are also used to show relationships among classes. This aspect of the diagrams will tend to emerge later in the design process, as "lower level" classes are identified. The class diagrams will improve the definition of the classes, which in turn may require changes to the sequence diagrams and, when developed, the state transition diagrams. These other diagrams will also have an impact on the class diagrams.

**TABLE 11-4**

Business Classes and Descriptions

| Class | Description |
|---|---|
| Customer | A person who orders securities |
| Potential Customer | A member of the general public who makes inquiries about our publications to potentially order publications from the company |
| Order | A request for a security |
| Order Inquiry | An inquiry from a customer concerning an order that has been placed |
| Delivery | The security that is purchased is sent to the customer portfolio |
| Payment | The payment by a customer for securities ordered and/or received |
| Company | BNACS |
| Order Browse | A request to view all outstanding security orders |
| Order Cancel | A request to cancel all outstanding security orders |
| Order Replace | A request to replace all outstanding security orders |

Two important classes in terms of the back-end processing are the Enterprise JavaBeans (EJB) session beans that process orders: the *AccessOrderBean* will send and track orders, and the *AccessDataBean* will provide associated data pertaining to the customer and the associated order(s). Figures 11-9 and 11-10 are UML class diagrams illustrating the methods and associations for each of these classes.

## State Transition (Life cycle) Diagrams

It is useful to trace what happens to a class through the execution of a business process, or through the computer system that is developed to support the business process. The state transition diagrams show the various states in which a class can exist and the way in which the class changes from one state to another. Figure 11-11 shows a state transition of trade order processing.

**FIGURE 11-9** Class diagram for access order session bean

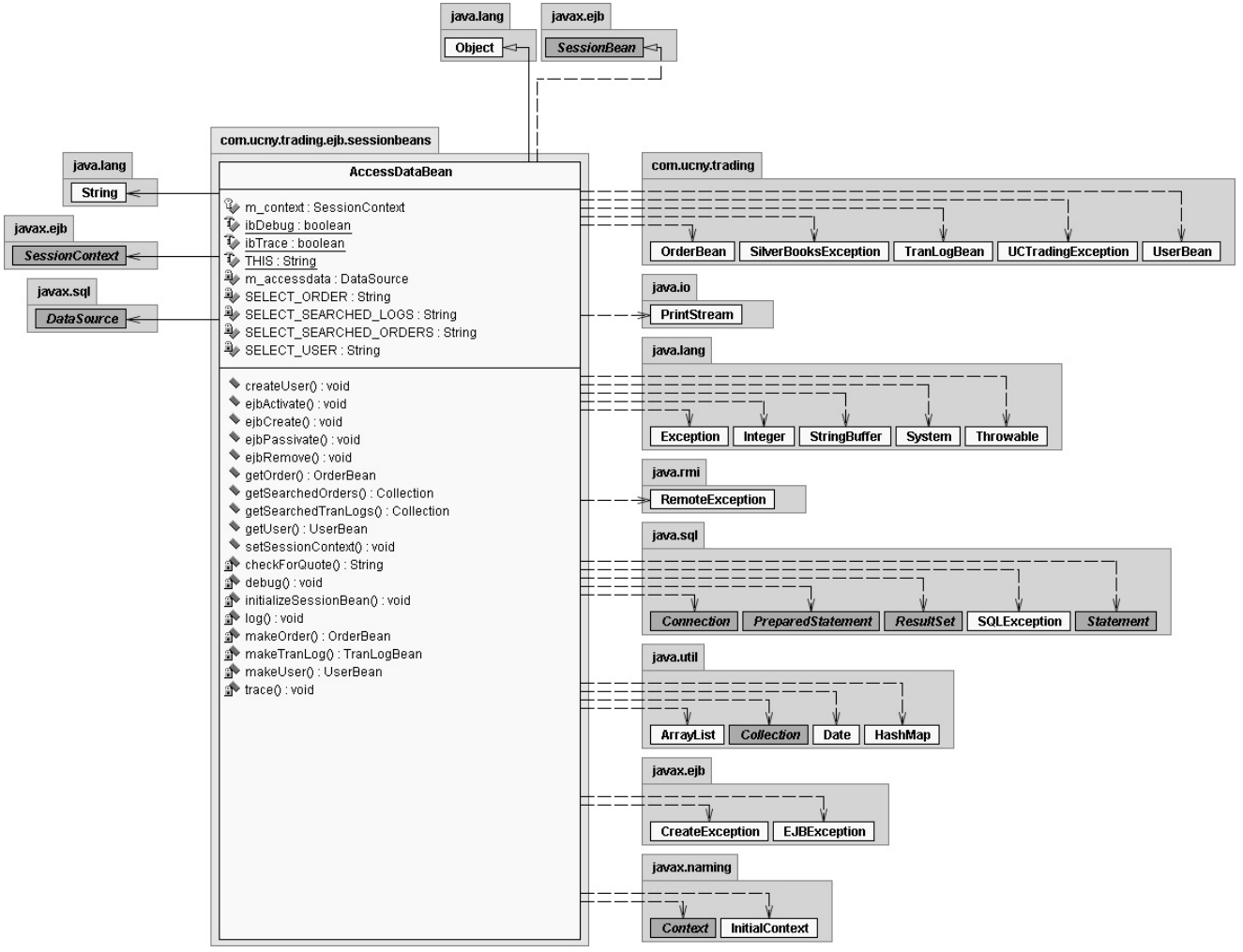


**FIGURE 11-10** Class diagram for access data session bean

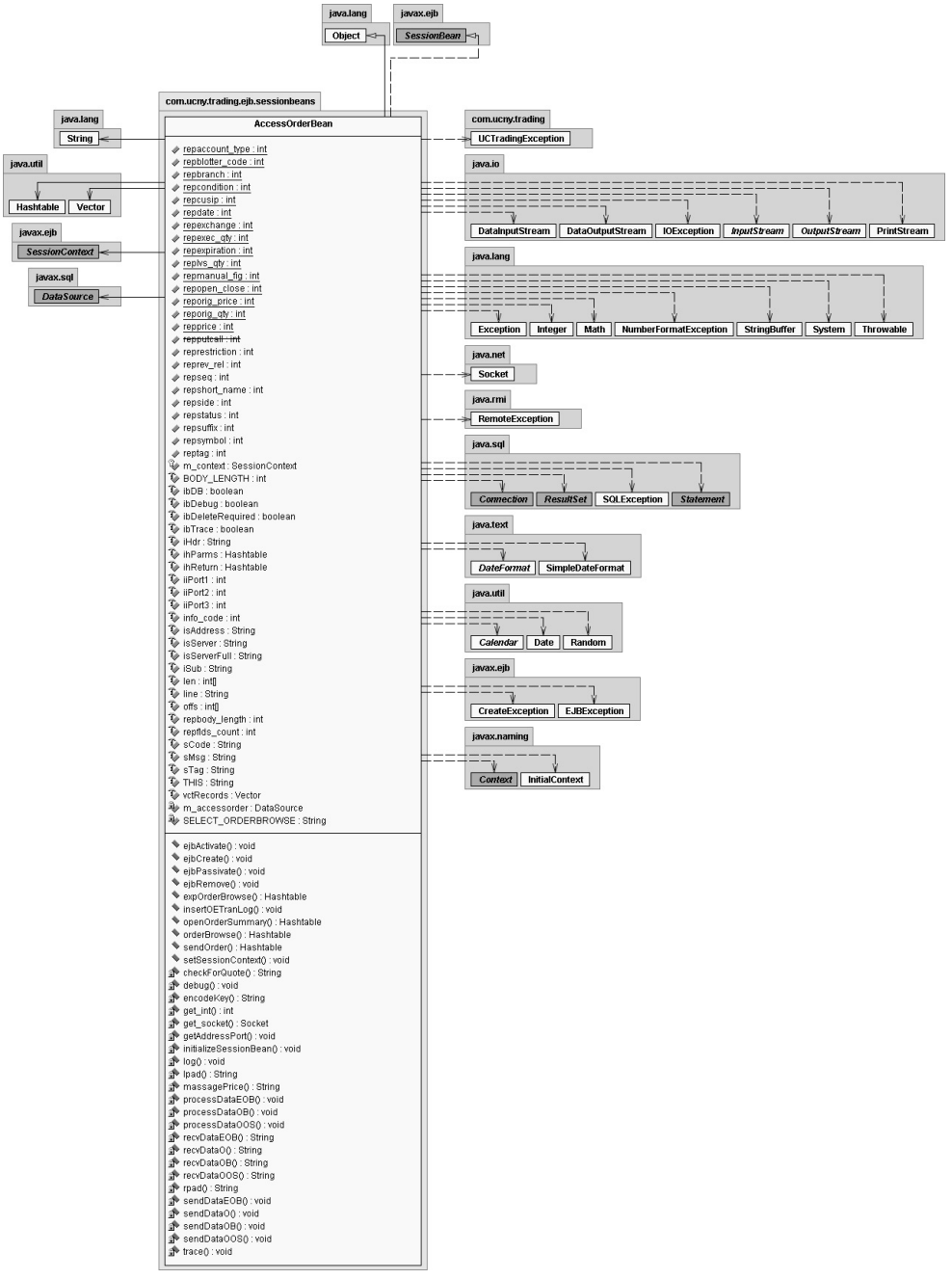
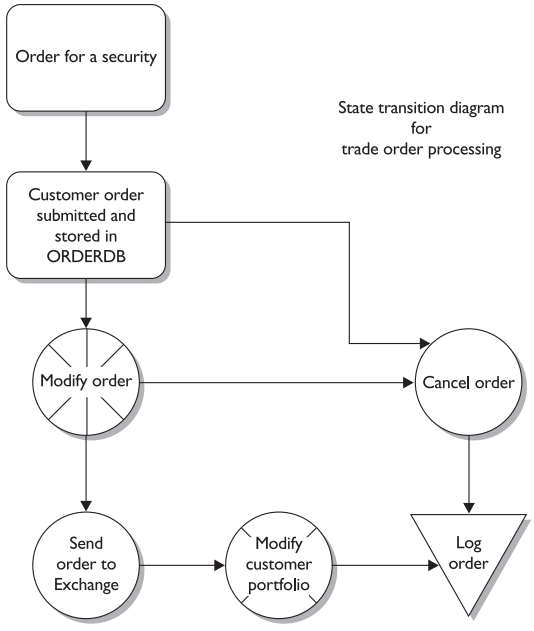


**FIGURE 11-11**

State transition (life cycle) diagrams




# Trade System Design and Implementation

This section describes the user-interface layout, class diagrams, controls, actions, and navigational aspects for the trading application. It is a comprehensive description of how the application works and affects the underlying data elements.

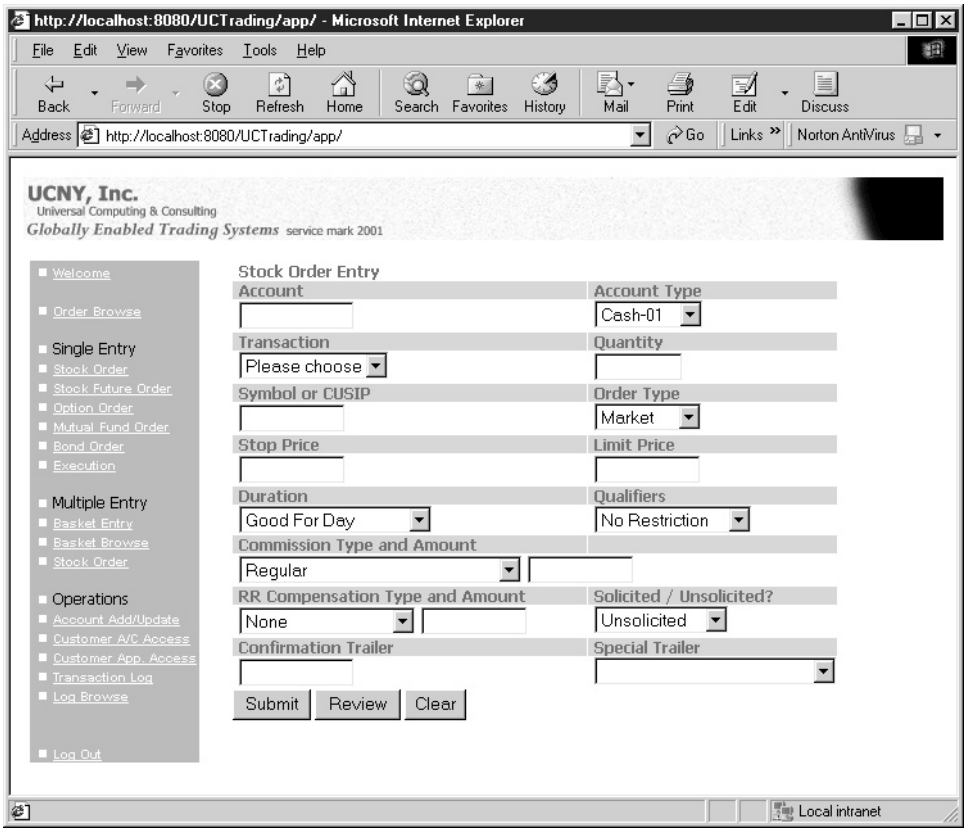
## Stock Order Entry Screen

Figure 11-12 shows the stock order entry screen, in which a buyer clicks Stock Order and enters information in all the fields.

Figure 11-13 shows a class diagram that identifies and defines the relationships between the classes.

**FIGURE 11-12**

Equity trade order screen




In Figure 11-14, you can see that a buy order has been placed for 100 shares of IBM at the market for account 14112345678.

Table 11-5 shows the controls and description of the information entered and validity notes regarding what each control should contain.

Figure 11-15 shows the stock Order Browse screen, where you enter an account, a transaction side (i.e., B for Buy or S for Sell), or a symbol and then click Find The Orders. All of the buy orders are shown here.

Figure 11-16 shows a class diagram that identifies and defines the relationships among the classes.

**FIGURE 11-13** Equity trade order class diagram

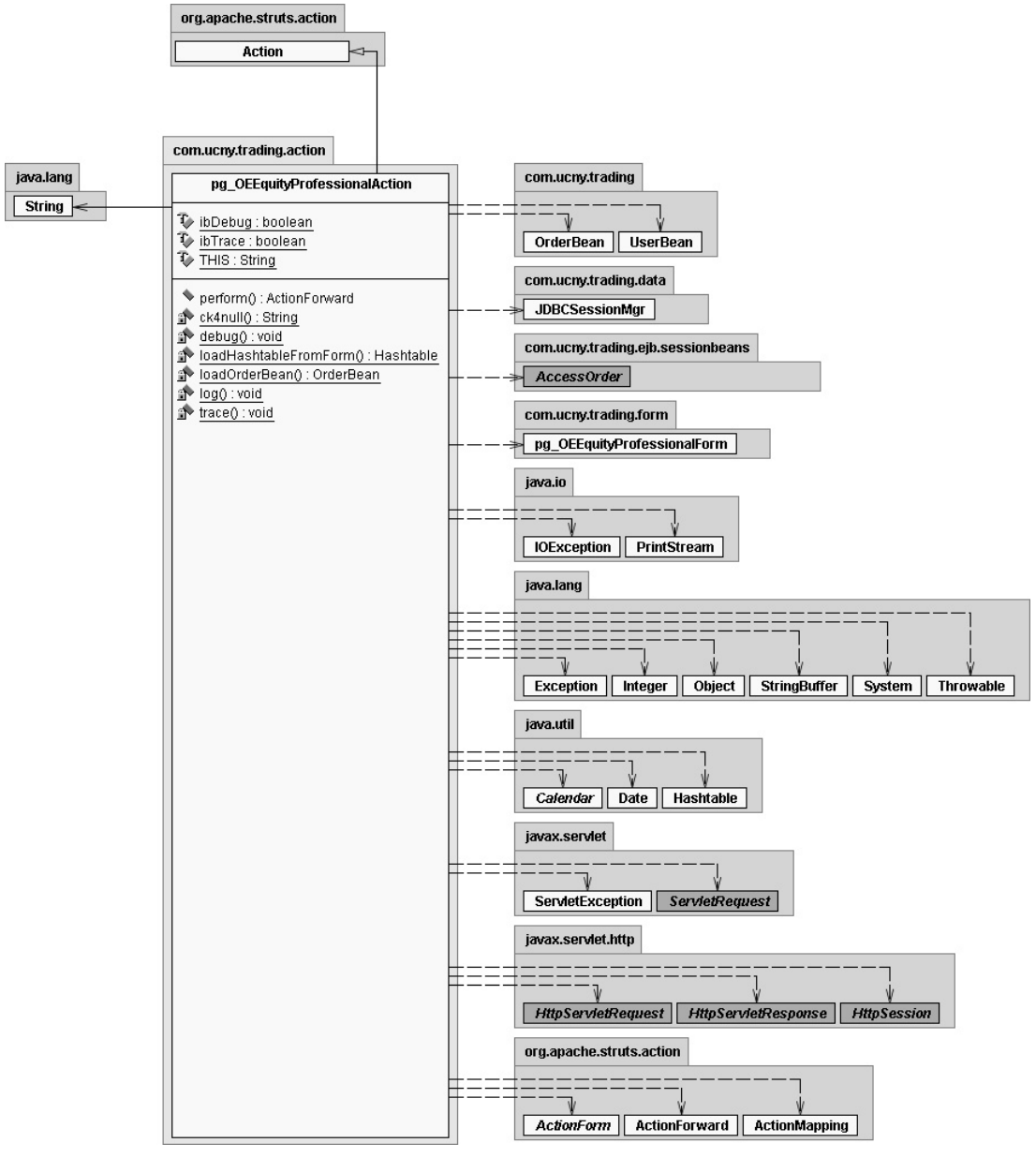


**FIGURE 11-14**

Submission and execution of trade order

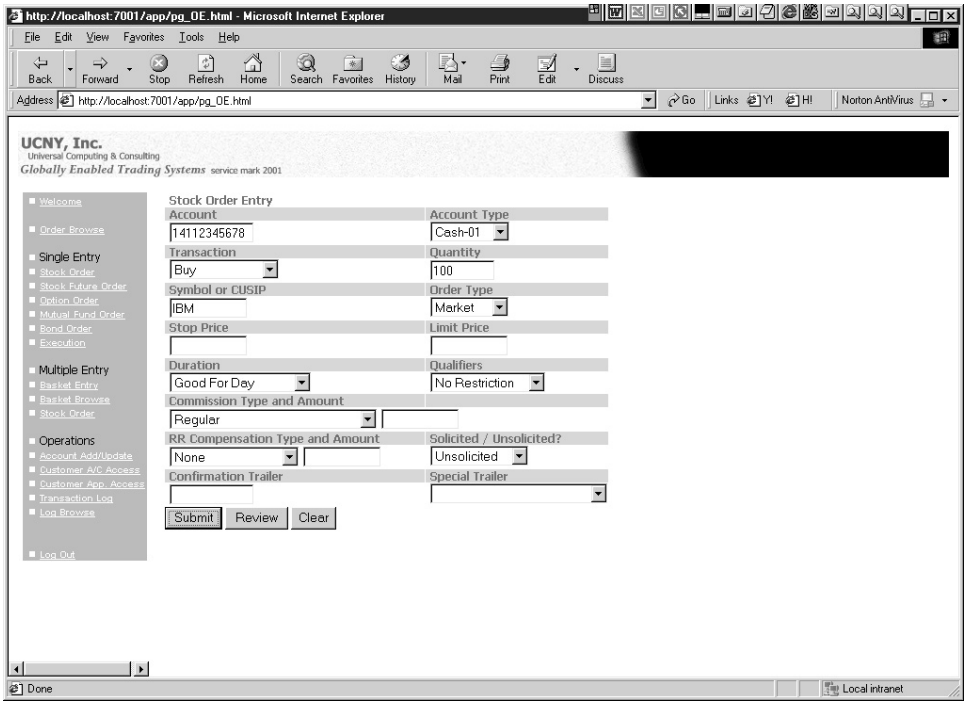


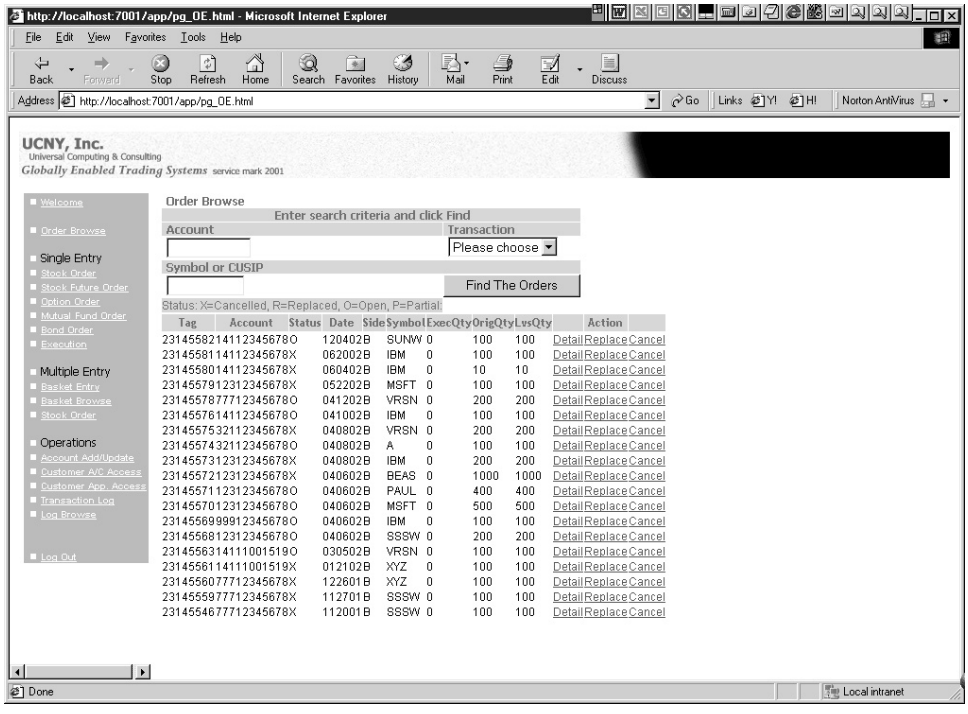

**TABLE 11-5**

Controls on the Trading Page

| Controls | Description |
|---|---|
| Account | The customer's Account number<br>Valid values: This is from the Customer Information System CIS system |
| Account Type | The valid values are: Cash/Margin/Short |
| Transaction | Side of the transaction<br>Valid values: B/SL/SS/Buy Cover |
| Quantity | How many shares of the security<br>Valid values: Numeric/integer > 0 |
| Symbol of CUSIP | Security identifier/symbol or CUSIP<br>Valid values: Symbol checked against warehouse security table |
| Messages | Green or Red, depending upon the result. Green is success; red is problematic |
| Submit button | Process the data and exit the form |
| Review button | Review and exit the form |
| Clear button | Cancel the data entered and exit the form |

**FIGURE 11-15**

Equity trade Order Browse screen




## Trade Application Packages

After all of the application components are completed and the application is ready for deployment, a package diagram(s) can be used to describe associations among the component classes. Figures 11-17, 11-18, and 11-19 show package diagrams for *com.ucny.trading*, *com.ucny.trading.ejb.sessionbeans*, *com.ucny.trading.data*, and *com.ucny.trading.action*, which ties together all of the components (JSPs, EJBs, JavaBeans, and so on).

## Trade Application Implementation Infrastructure

After the application is deployed, a component diagram(s) can be useful for describing associations among the hardware and software components and the system functionality. Figure 11-20 shows the hardware and software involved in the trade process flow.

Figure 11-21 shows the hardware and software involved in the security pricing process flow.

**FIGURE 11-16** Class diagram for equity trade Order Browse screen

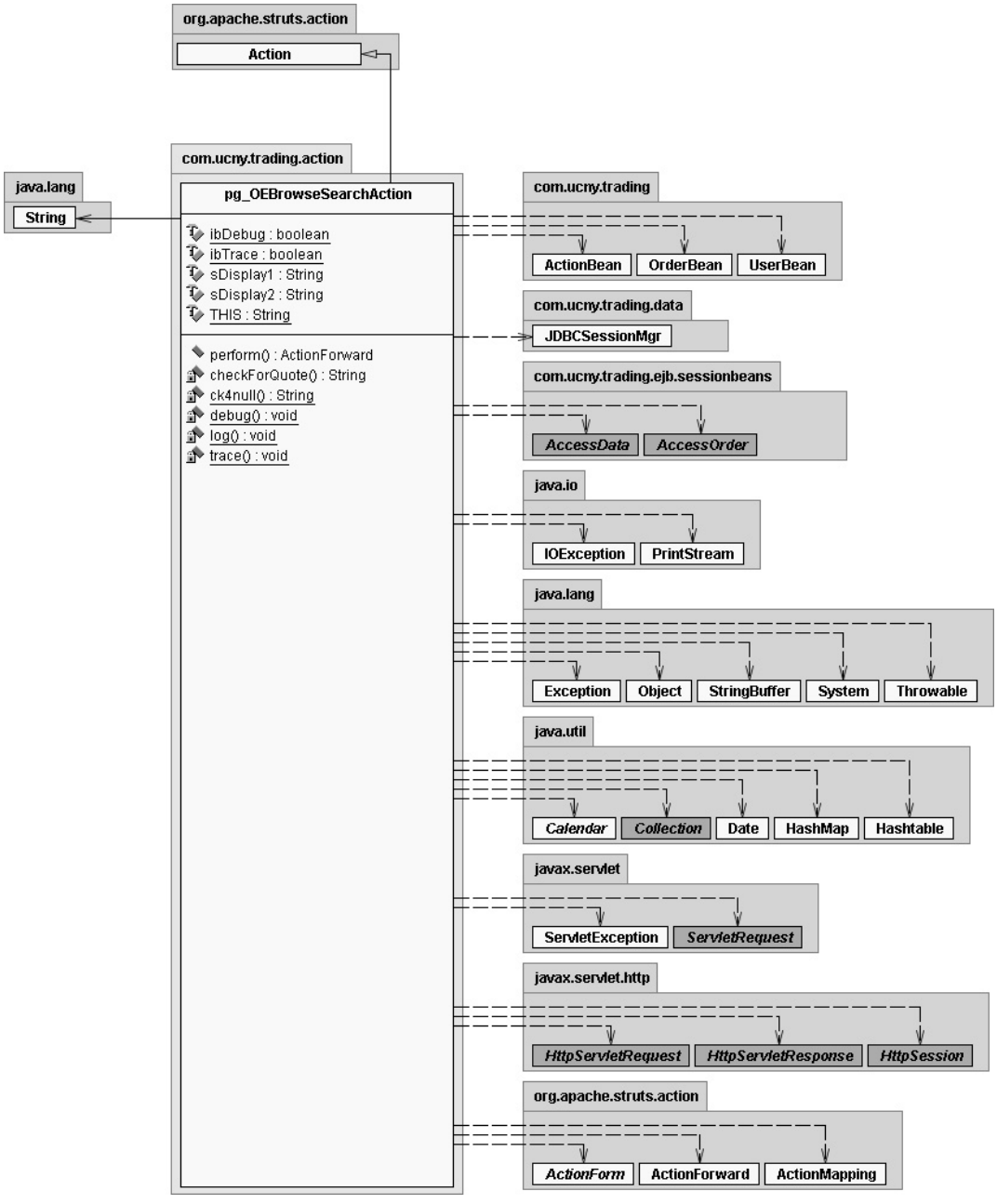


**FIGURE 11-17**

Package diagram for com.ucny .trading

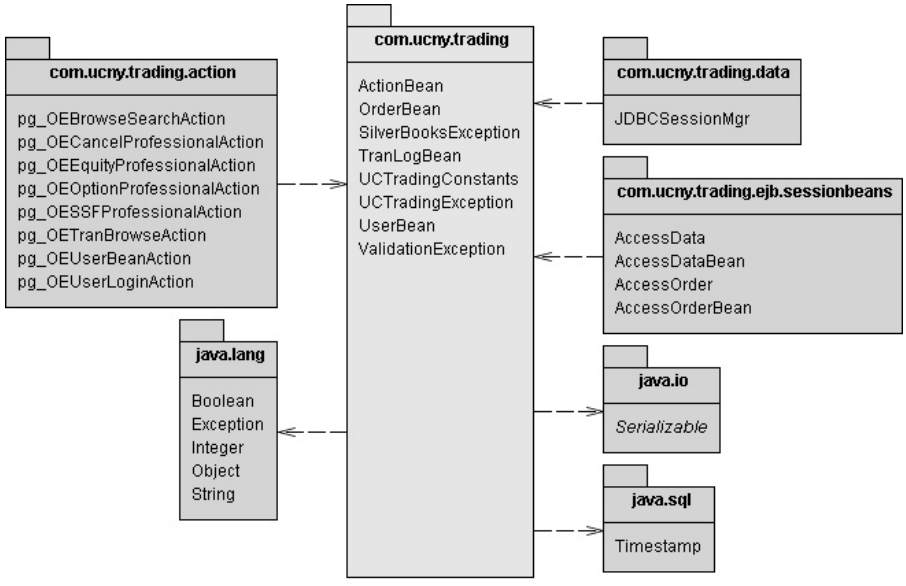


**FIGURE 11-18** Package diagram for com.ucny.trading.ejb.sessionbeans

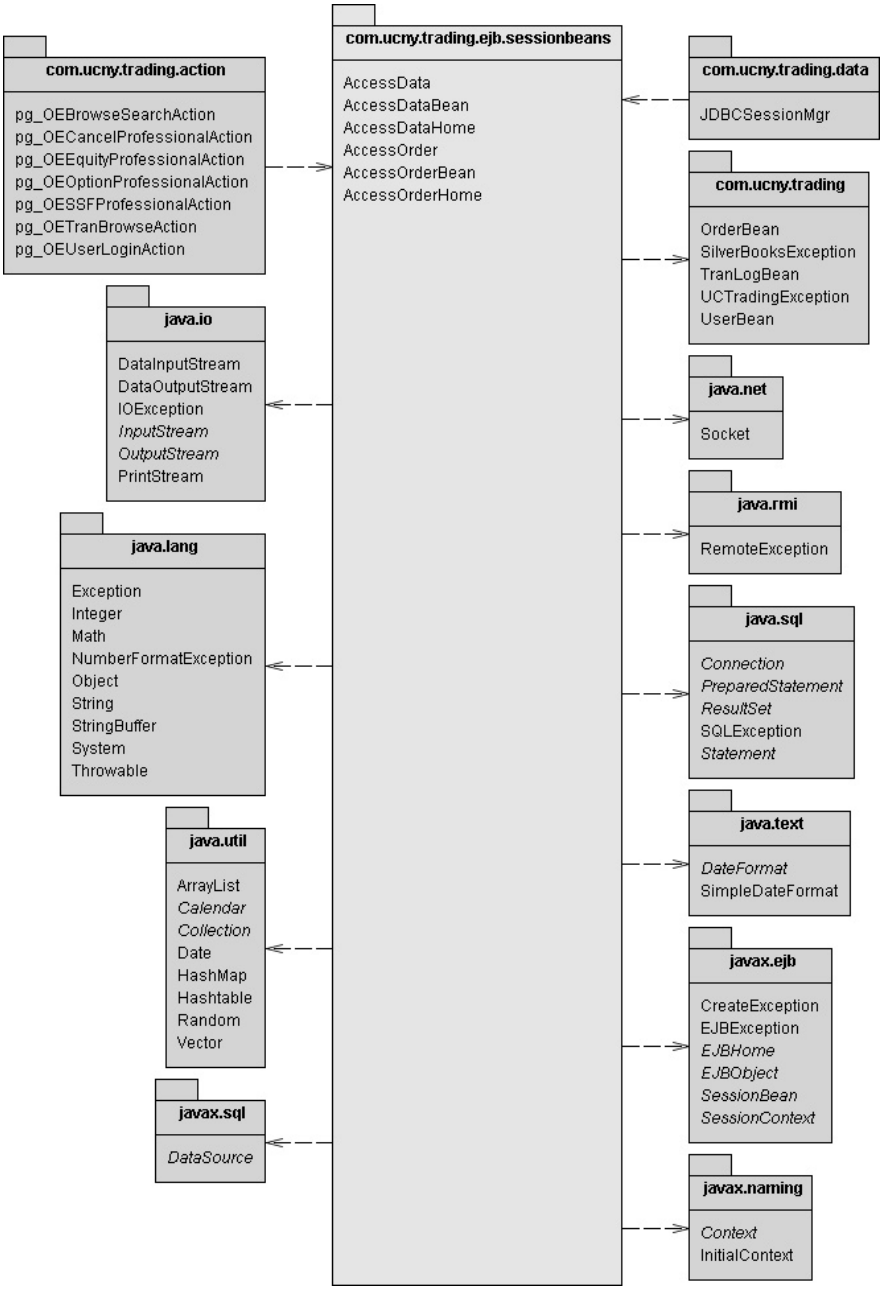


**FIGURE 11-19**

Package diagram for com.ucny .trading.data

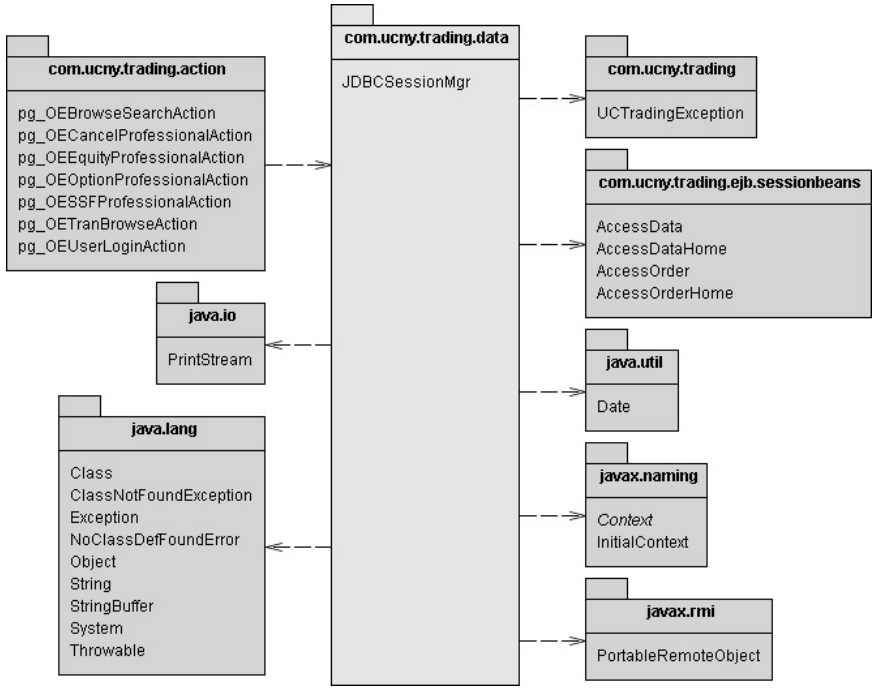



**FIGURE 11-20**

Hardware and software involved in the trade process flow

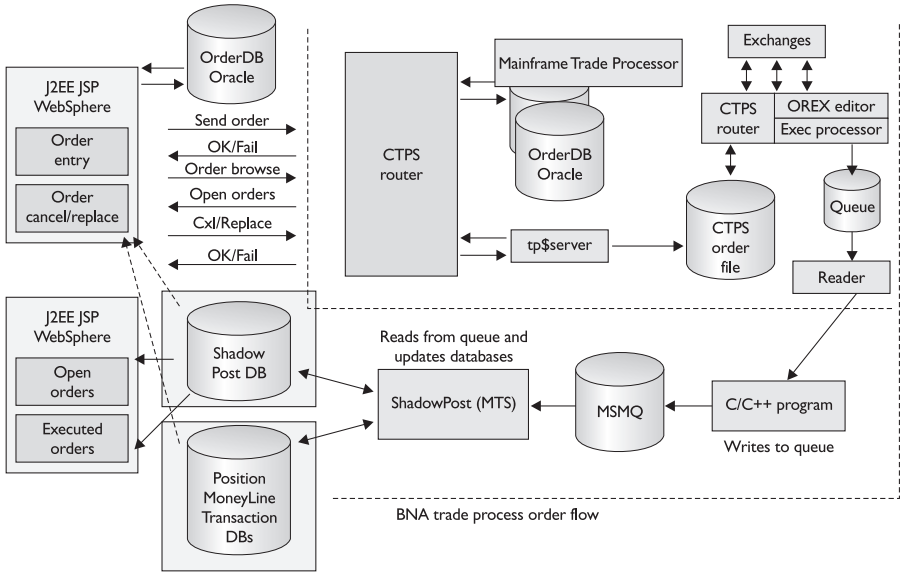
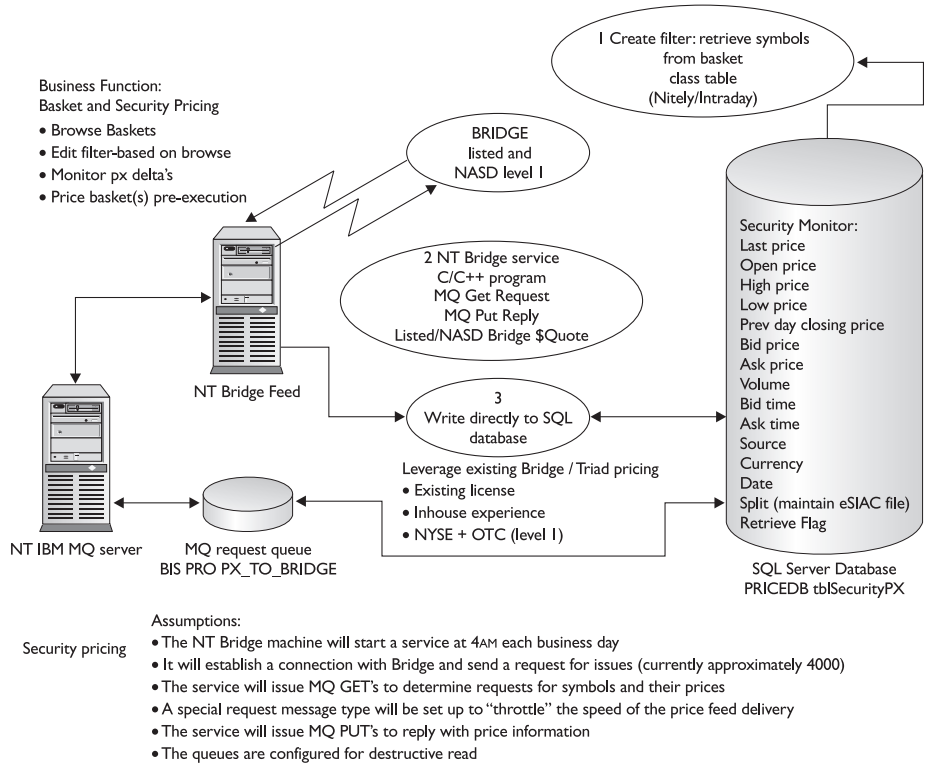


**FIGURE 11-21**

Hardware and software involved in the security pricing flow




# CERTIFICATION SUMMARY

This case study is an example of a real-world application. As architects, and for the purposes of the exam, you will create use cases, sequence diagrams, component diagrams, and other types of diagrams to provide a clear picture of the functions of an enterprise application. Its infrastructure, functionality, and deployment particulars can be illustrated using UML in conjunction with other diagramming and text descriptions to help users and others to evaluate and understand the application.