INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

EJB E-BUSINESS APPLICATION DESIGN – CONCORD ONLINE DVD CENTER (CODC)

Hong Chen

A MAJOR REPORT

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

JUNE 2002

ABSTRACT

EJB E-BUSINESS APPLICATION DESIGN

-- CONCORD ONLINE DVD CENTER (CODC)

Hong Chen

The Enterprise JavaBeans (EJB) is an important technology in enterprise computing. Organizations that are exploring or implementing mission-critical, Web-based, and distributed systems, especially e-business applications, can get the development and deployment advantages from EJB within J2EE technology.

"Concord DVD Center" (CODC), is an EJB based multi-tier e-business web application design as an online DVD shopping center. It is designed and implemented by EJB technology with multi-tier e-business architecture.

In this report, I will introduce the concepts of e-business and EJB (Enterprise Java Bean) technology which is applied in the project, and its advantage in the current development of distributed component methodology, especially in e-business application design and implementation.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Professor Peter Grogono, thanks for his great help for this project and report.

CONTENT

List of Figures:

1 Introduction

An online shopping center is an alternative of traditional shopping method, it provides people conveniences and free choices from shopping time and locations. It allows users to view merchandises and carry out shopping activities on the Internet.

This report describes an EJB based multi-tier e-business web application design as an online DVD shopping center – "Concord DVD Center"(CODC).

With EJB, people can write applications that solve real-world problems, rather than on all overhead that goes with distributed server-side systems. EJB adds legacy integration flexibility. One can rewrite the components from scratch to be EJB-compliant, or wrap an existing component, and no need to build the enterprise system from the ground up. With the "plug-and-play" enterprise feature, people barely need to know anything at all about middleware to construct components designed to run in a scalable, multi-tier architecture (like CORBA). The components gain middleware services implicitly and transparently from EJB server. Besides, EJB standard is also a cross-platform, cross-vendor nature that can mostly benefit users. This is extremely important for today's e-business competition.

For my major report, I choose EJB e-business application design as my project – "Concord DVD Center"(CODC). It's an Internet based online DVD shopping system, I used Java technology (such as J2EE, EJB, JavaBean, JSP) for implementation instead of traditional CGI technology.

1.1 Project Object

"Concord DVD Center" (CODC) is an e-business system that provides online DVD shopping services. The customer of this application can be DVD vendors or retailers. The application faces to general public, the specific users can be the DVD Vendors and retailers, DVD suppliers, online business owners, DVD movie fans, and other community groups, etc.

General users can be easily to search the DVD products according to their needs and interests from the system, the system can provide related and consistent functional tasks and senarios for the users such as:

1) **User authentication.** Registered users would first log in to the Web site to access the complete catalog.
2) **An online product catalog.** Users should be able to browse her complete product line on the Web and view details of each product.
3) **Online quote generation.** While browsing the catalog, a user should be able to pick and choose the products she wants. The user should then be able to view the current shopping cart (it's called a quote). The user can change quantities of items he already picked out.
4) **Order generation.** Once the user is happy with the selections and has committed to ordering the products, a permanent order should be generated. A separate fulfillment application will use the data in the orders to manufacture the products and ship them. The user would be able to return to the Web site later to view the status of current orders.

5) **Billing functionality**. Once the user has placed the order, a bill will go to her, if the user does not have enough funds to pay then the order will be cancelled.

6) **Service functionality.** After sales, the user can contact the online shopping center for customer service like return merchandises and get refund, and information inquiry.

1.2 System Architecture

The EJB e-business application system – "Concord DVD Center"(CODC) is a 4-tier architecture, thus can clarify the business logic in the entire system.

Tier 1: Web Browser

This part will be genearated by HTML and Java script.

This part provide the user a browser which present the system "Concord DVD Center"(CODC), the user can have a test to choose the task and senario which provide the consequent page and result. For example, in the order page, user can make purchase order for the chosen merchandise.

Tier 2: Web Server (Presentation Tier)

This part provides a web server of the system "Concord DVD Center"(CODC). This tier displays the requested information in HTML to the end user, it also interprets the users selections and makes invocations to the application server tier (business logic tier)'s enterprise beans. I will use JSP (Java Server Pages) is used for implementation, thus networked objects that know how to interact with a user on a specific protocol.

Tier 3: Application Server (Business Logic Tier)

This part will consist of multiple Enterprise JavaBeans, running under the hood of an EJB container/server. These will be reusable components that are independent of any user

interface logic. It will be made up of persistent entity bean objects that represent data being modified, as well as session beans to provide application rules for interacting with that data.

Tier 4: Database Server

This part is where the permanent data stores reside. The databases aggregate all persistent information related to the e-commerce site. I used Microsoft Access for the database server for this project.

2 Background of e-business and EJB technology

2.1 Overview of e-business application

Today, e-business application is a bright star that is rising so quickly. The Application Framework for e-business gives you a comprehensive foundation for building, running, and managing e-business applications: It prescribes tools, application servers, and infrastructure, all linked by standards-based, common-architectural principles and technologies, so that you can use to build and deploy state-of-the-art e-business solutions. These solutions can help transform your business and leverage the opportunities of the open-networked world.

Though e-business application is developing widely, it also brings design problems that many developers have to face. Among the new factors developers must now consider are:

- Web browser clients, and other new technologies and paradigms
- The distributed, multi-tier nature of e-business applications
- Direct support of customers

- Unique security problems of the open Internet

The following is a figure that shows the structure of a typical e-business application. Many e-business applications have all the parts shown in this figure, but even when an e-business application is simpler, the basic structure still applies.

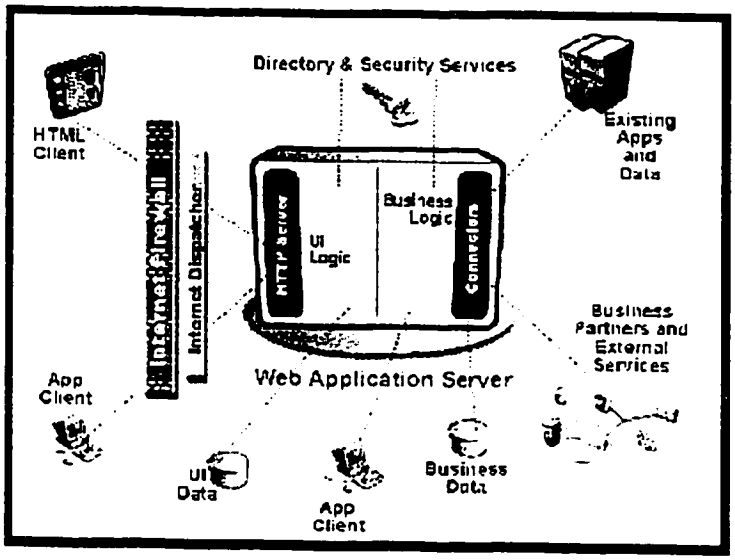


Figure 1 E-business System Structure

There are four elements of e-business applications:

a) Client

Client could be specified into to kinds: one is HTML client, it communicates with the Web application via the HTTP protocol and receives HTML responses to display. Another one is Application client, it may communicate over a number of protocols including HTTP, IIOP, or MQ, and can receive data responses that must be formatted in the client for display.

b) Web application server

Inside the web application server, it supports the HTTP protocol and basic Web functions such as page retrieval. Web application specific also determines the particular type of client and style of user interface as the UI logic. The business logic is defined as independent of the client type and user interface style. The web application server has various connectors such as: Libraries, components, and runtime support for communicating with external applications, data, and services. Another part is Data, it is used by the application through the server.

c) Infrastructure services

Inside the e-business application, the directory is provided as a shared repository for service location, configuration, and user identity information. The security function provides credential recognition and mapping. The firewall can limits communications to specified protocols, communication ports, and between specified parties. The dispatcher can distribute web requests (usually involving only HTTP) across a number of Web servers in a manner that is transparent to the client.

d) External applications, data, and services

For the existing applications and data that are already part of a company's computing resources, it may provide much (or even most) of the business logic needed for new applications, but generally it will be necessary to provide logic in the Web application server to integrate these applications. For the business partners and external services, the external applications can be integrated into the e-business application. Generally these are accessible through an electronic data interchange (EDI) or message-based interface.

Therefore, the Application Framework for e-business should provide the features as: simplifies application development and deployment; supports heterogeneous client and server platforms; leverages existing skills and assets; delivers a secure, scalable, reliable, and manageable environment, and provides the freedom to implement using many vendor's products.

Consequently, Java technology is the ideal and comprehensive developing tool for e-business that contains the following in its platform: Servlet and JSP support by system configuration, Enterprise JavaBean (EJB) support by system configuration, Database access services by the Java Database Connection (JDBC) and Java SQL (SQL-J), Directory access by the Java Naming and Directory Interface (JNDI), Security services by Java SSL, Java Cryptography Extensions, Java Authentication services.

2.2 E-business application architecture (server-side)

The idea of software components is very useful and powerful today, especially in the e-business application design. A company can purchase a well-defined module that solves a problem and combine it with other components to solve larger problems. For example, consider a software component that computes the price of goods, this can a *pricing component* in the e-business application. When handling the pricing component information about a set of products, and it figures out the total price of the order.

2.2.1 Server-side Component Architectures

A software component is code that implements a set of well-defined interfaces. It is a manageable, discrete chunk of logic. Components can't run alone, they can be used as

integrated pieces to solve some larger problem. So, customers are seeking a well-defined module that solves a problem and combine it with other components to solve larger problems, such as reusable components.

Reusable components are quite enticing because components promote rapid application development. It has such approaches to build, manage and maintain component:

1) Tools for developing components:

   Example: IDE (integrated development environment), such as Symantec's *visual café*, IBM's *VisualAge for Java*, or Inprise's *JBuilder 2*, assists Java developers in rapidly building and debugging components.

2) A container that manages deployed components.

   Provides a runtime environment for the components to play in, and a set of common services that most components will need. For example: the container could automatically instantiate new components as necessary, thus removing that burden from the component developer.

3) Tools for deploying and maintaining components

   For example: there should be a way to customize the components for a particular environment.

2.2.2 Multi-tier Architectures of e-business Application

A well-written server-side deployment has a logical software partitioning into layers. Each layer has a different responsibility in the overall deployment and has one or more components within the layer. The typical e-business application can have such layers:

A presentation layer: contains components dealing with user interfaces and user interaction. The part of stand-along application can be written in VB, HTML, Java script, etc; the web-based deployment can be Java servlet, JSP (Java server pages), Java applets.

A business logic layer: contains components that work together to solve business problems. These components can be high-performance engines, such as catalog engines or pricing engines. Typically, these components are written in Java or C++.

A data layer: used by business logic layer to persist state permanently. Central to the data layer is one or more databases that house the stored state.

2.2.3 Traditional Two-Tier Architectures

Traditionally, most high-end deployments have been two-tiered. Two tier deployments combine the business logic layer with one of the other two layers. There are two possibilities for this kind of architecture: combining the presentation layer with the business logic layer; or pushing some of the business logic layer into the data layer.

The two-tier architectures have the following disadvantage characteristics:

Deployment costs are high. Database drivers must be installed and configured on each of the client tiers, which may mean hundreds or thousands of machines.

Database driver switching costs are high. To switching one database driver with another requires a reinstallation of every client machine's database driver, maintenance costs high for that.

Database schema switching costs are high. The fat clients directly access the database via DBC, SQL/J, or ODBC. The clients are dealing directly with the underlying database

schema, if need to migrate the schema to handle new business processes, redeploying each client is a necessary.

Database type switching costs are high. The fat clients are bounded to a database API, such as a relational database API or an object database API. If need to switch between database types (such as from a relational database to an object database), the client code must be changed to suit the new database type.

Database connection costs are high. Every database client needs to establish its own database connection. These connections are limited in number and costly to establish. When clients are not using the database, the connection is often still held and cannot be used by others.

Business logic migration costs are high. Changing the business logic layer involves recompiling and redeploying the client tier.

Network performance suffers. Each time when business logic performs a database operation, need to make a number of roundtrips across the physical boundary separating the business logic layer and the data layer. If this barrier is a network boundary, it could severely hamper the total time a database operations takes – and it could clog the network, reducing the amount of bandwidth other users have.

2.2.4 N-tier Architectures

N-tier architecture has more benefits than traditional two-tier in e-business application design. A concrete example is a three-tier web-based deployment application, separated as the presentation layer, business logic layer and data layer into respective physical tier.

Characteristics of N-tier architectures can be concluded as following:

Deployment costs are low. Database drivers are installed and configured on the server-side, rather than on client machines. It is much cheaper to deploy and configure software in a controlled server-side environment than to deploy software on thousands of (for example) end user terminals.

Database switching costs are low. Clients no longer access the database directly, but rather go through the middle tier for data access. This enables to migrate database schemas, change to different database drivers, or even change your persistent storage type without re-deploying clients.

Business logic migration costs are low. Changing the business logic layer may not necessitate recompiling and redeploying the client tier.

Can secure parts of the deployment with firewalls. Place a firewall in between the presentation and business logic tiers can protect the business data while do not hamper a deployed application.

Resources can be efficiently pooled and re-used. Connections to external resources can be managed very efficiently. Resource pooling exploits that clients are often doing other things besides using resources, such as rendering a graphical user interface. The resources can be pooled and re-used for different client requests. The resulting set of database connections required is often far less than the total number of components in a deployed system. Connections to resources do not need to be re-established continuously, improving application performance. Resource pooling can be applied to other resources as well, such as socket connections and threads.

Each tier can vary independently. For example, database images can be added while minimizing changes and recompilations for the other tiers.

Performance slowdown is localized. If one tier is overloaded, the other tiers can still function properly. In a web deployment, users many able to view the front page even though the application server is overburdened.

Errors are localized. If a critical error occurs, it is localized to a single tier. The other tiers can still function properly, dealing with the situation. For instance, if an application server crashes, the Web server can report a "site down" page to client browsers.

Communication performance suffers. Since the tiers are physically separated, they must communicate across process boundaries, across machine boundaries, or across enterprise domain boundaries. This results in high communication overhead. Only by designing your distributed object application properly can you have an efficient deployment.

Maintenance costs are high. As deploying in three or more physically separate tiers, so that software installation / upgrade costs, redeployment costs and administration costs increase significantly.

Therefore, N-tier is the optimized choice for the current e-business development. For the e-business application design, there are many server-side design tools, like ASP, JSP, etc. Currently, the most ideal tool is using EJB technology as design tool.

2.3 Overview of Java and EJB technology

2.3.1 About Java and J2EE

Java is an ideal language for the component architectures of e-business application design. As an OOP design language, Java separates the *interface* of a component from its *implementation*. Thus, in a business, the component vendor can publish the contract or rules for calling the component, and the client code can adhere to these rules. When the vender releases new versions of the component, the customers can upgrade.

A component's interface defines the component's contract with the code that calls it. A component's implementation is the core programming logic that an object provides. Java supports interface/implementation separation at a syntactic level via the *interface* keyword and *class* keyword.

The Java 2 Platform, Enterprise Edition is a robust middleware services for server-side application development. The technologies included with J2EE are as following:

Enterprise JavaBeans (EJB): it's a component standard, defines the overall server-side architecture and how server-side components are written and provides a standard contract between components and the application servers that manage them. Also defines the interfaces between software vendors and clients, ties together with other APIs. It promotes the spawning of a component marketplace, where vendors can sell reusable components that can be purchased to help solve business problems.

Java Remote Method Invocation (RMI) and /RMI-IIOP: RMI allows for interprocess communication and provides other communication-related services. RMI-IIOP is a portable extension of RMI that can use the Internet Inter-ORB Protocol (IIOP) and can be used for CORBA integration.

Java Naming and Directory Interface (JNDI): identifies the locations of components or other resources across the network.

Java Database Connectivity (JDBC): is a relational database bridge that allows for relatively portable database operations.

Java Transaction API (JTA) and Java Transaction Service (JTS): allow components to be bolstered with reliable transaction support.

Java Messaging Service (JMS): allows for asynchronous distributed object communications. It is not integrated with EJB by Sun Microsystems now.

Java Servlets and Java Server Pages (JSPs): request / response oriented network components, such as interacting with clients over HTTP.

Java IDL: is Sun Microsystem's Java-based implementation of CORBA, allows integration with other languages. And distributed objects to leverage CORBA's services.

JavaMail: allows sending email messages in a platform-independent, protocol-independent manner from the Java Programs. For example: a confirm email for purchasing from an online shopping center.

Connectors: provides access to existing enterprise information systems and integrates with mainframe systems running high-end transactions (such as those deployed with IBM's CICS), as well as Enterprise Resource Planning systems.

The Extensible Markup Language (XML): several J2EE techonoligies such as EJB and JSP depend on XML as a meta-markup language for describing contents.

JavaBean and EJB: JavaBean is a developable component but not deployable, while EJB standard defines a component architecture for deployable components called *enterprise beans*. Enterprise beans are very similar to applets and servlets. Applets can be deployed in a Web page, where the browser's applet viewer provides a runtime container for the applets. Servlets can be deployed in a Web server, where the Web server's servlet engine provides a runtime container for the servlets. Enterprise beans are deployed in an application server, where the application server provides a runtime container for the Enterprise JavaBeans.

Both applets and servlets handle client-side operations, such as rendering graphical user interfaces (GUIs), performing other presentation-related logic and lightweight business logic operations. The client side could be a Web browser, or a Web server, the components can be dealing with the end user directly.

Enterprise beans are server-side components and performing server-side operations, such as executing complex algorithms or high-volume business transactions. It always runs in a highly available, fault-tolerant, transactional and multi-user secure environment. An application server provides this high-end server-side environment.

2.3.2 Enterprise JavaBeans (EJB)

Enterprise JavaBeans (EJB) is developed by Sun Microsystems Inc. It is an integral part within Sun's *Java 2 Platform, Enterprise Edition* (J2EE), which is a collection of

enterprise technologies. EJB is a server-side component architecture that enables and simplifies the process of building enterprise-class distributed object applications in Java. By using EJB, we can write scalable, reliable and secure applications without writing our own complex distributed object framework. EJB is about rapid application development for the server side, we can quickly and easily construct server-side components in Java by leveraging a prewritten distributed infrastructure provided by the customer (industry). EJB is designed to support application portability across any enterprise middleware services.

By understanding and using J2EE properly, then we can build portable, object-oriented, enterprise-class applications by EJB -- ***Enterprise Java Beans*** technology-based services, which including such features as: Transactional components; Distributed Transactions; Messaging and asynchronous task management; Multiple company service providers with multi-site servers and databases; Interfaces to, and use of, legacy applications; Secure transmissions and role-based authentication; Object persistence and transient state, etc.

2.3.3 EJB and CORBA

CORBA (Common Object Request Broker Architecture) is a unifying standard for writing distributed object systems. CORBA and IIOP (Inter-ORB Protocol) are standards that promote portable distributed objects. CORBA is language-independent, it provides optional value-added services, but it is slow-moving, and has a steep learning curve. The products developed under CORBA may have incompatible features.

CORBA is the basis for EJB, offers a much broader suite of middleware features with which to work. To use CORBA services, need program a complex middleware APIs, which increase the learning curve for CORBA programming, therefore, EJB and J2EE are much more suitable for rapid application development than CORBA. On the other hand, as EJB is officially being supported by the industry, it will give out a much wider variety of tools to work with in the long run.

EJB and CORBA are very related, they share much functionality – many of the qualities of service that EJB offers are also on CORBA. In fact, EJB can be considered as CORBA plus standards for how the components should be written and managed, increasing productivity. The EJB/CORBA mapping specification, along with RMI-IIOP, lifts the restriction that EJB must be solely Java-based. CORBA's persistent Interoperable Object References (IORs) make a great foundation for EJB object handles. In the future, CORBA and EJB will hopefully be integrated to guarantee on-the-wire transaction interoperability.

2.4 EJB e-business application system architecture

The foundations of EJB are the contracts and roles that each party must follow. Going with the divide-and-conquer theme, EJB partitions the responsibility of an EJB deployment to up to six different parties:

The bean provider: provides reusable business components that companies can purchase and use to help solve business problems. Beans are not complete applications but deployable components that can be assembled into complete solutions.

The container provider: supplies the low-level runtime execution environment needed to run EJB applications.

The server provider: supplies the application server logic to contain, manage, and deploy components. Currently there is no distinction between EJB container providers and EJB server providers. For example, EJB container/server products can be BEA's Weblogic, Sun's NetDynamics, Allaire's JRun , Oracle's 8i....

The application assembler: is the overall application architect, perhaps for a specific deployment. She is responsible for understanding how various components fit together and writes the applications that use components.

The deployer: takes prewritten components, and applies her deployment expertise to install the components in one or more application servers

The system administrator: oversees the well-being of a deployed system.

Here is the diagram for EJB six parties:

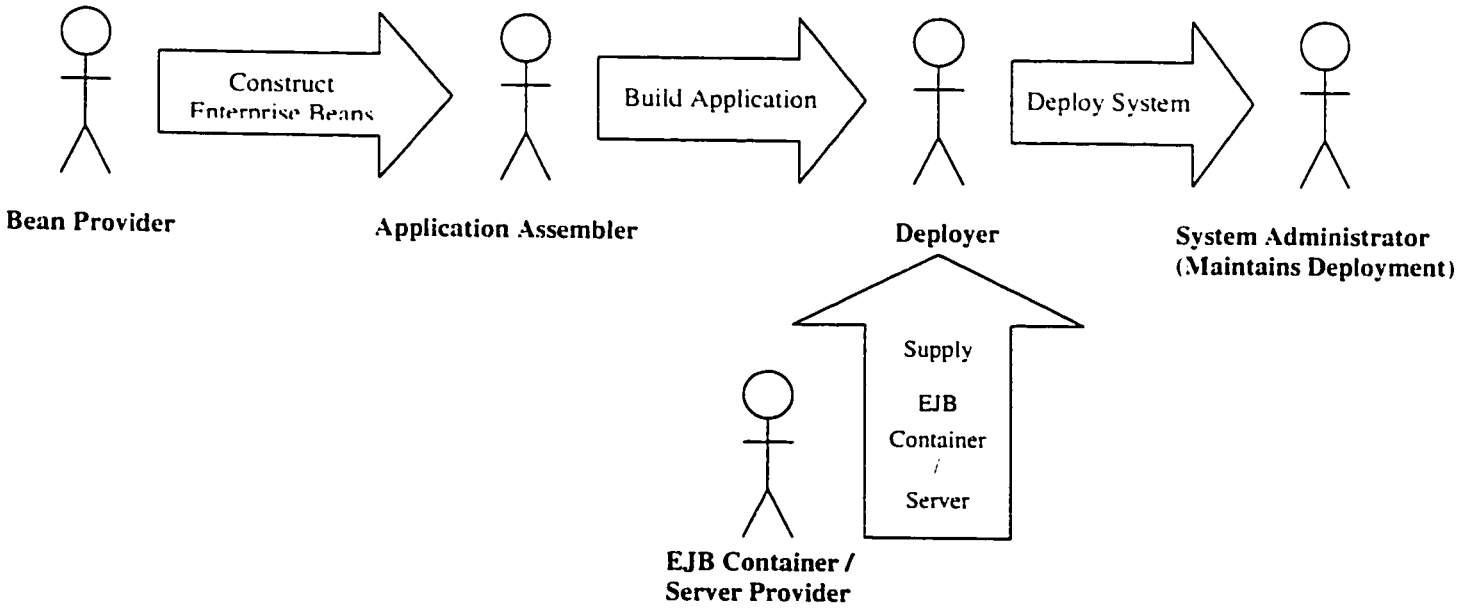


Figure 2 The six parties of EJB

EJB e-business application architecture also has the multi-tier as most other technologies that I already introduced in *2.2.2*: Presentation tier; Business logic tier (use EJB application server); Data tier.

Enterprise beans can be partitioned across multiple tiers, transactional, multi-user secure, and deployed in any EJB compatible container/server product. It has two flavors: session beans (represent a business process) and entity beans (represent permanent business data)

2.5 Tools for developing CODC

JSP (Java Server Pages)

JSP is used for the presentation tier - web server, it can be combined with Java Script to provide GUI as better effects.

EJB (Enterprise Java Bean)

EJB is used for the business logic tier in the EJB application server, for its advantage on development and deployment,

XML (Extensible Markup Language)

XML is used between business logic tier and data tier – Database server, to exchange business data which can complements EJB well. EJB current specification uses XML as a document format for deployment descriptors. XML is destined to become the de-facto standard for structuring document content. It is simple, easy to use, open internet-standard, self-describing and human-readable. XML uses Unicode, rather than ASCII, and allows for the use for URLs.

3 Requirement Specification

3.1 Functional Tasks and Senarios

- **An online catalog**

  Users should be able to browse her complete product line on the web and view details of each product.

- **User authentication**

  New customers can register in the website by email address and passwords then sign in, and returning customers can sign in by email address and passwords. Registered users can purchase from the online DVD shopping center.

- **Online quote generation**

  While browsing the catalog, a user should be able to pick and choose the products he or she wants. The user should then be able to view the current shopping basket (called as "order"). The user should be able to, for example, cancel or change quantities of items he or she has already picked out.

- **Order generation**

  Once the user is happy with his or her selections and has committed to ordering the products, a permanent order should be generated. A separate fulfillment application would use the data in the orders to manufacture the products and ship them. The user would be able to view the status of the current orders, and able to continue shopping.

- **Billing functionality**

  Once the user has placed the order, there should be a bill to him or her. If the user can't provide a valid credit card information, then the order will be cancelled.

- **After purchasing email notice functionality**

Once the purchasing activity is successfully finished, user will receive a thanks-shopping letter from CODC service center to show the order number and the purchasing details.

3.2 Technical Requirement (About EJB)

3.2.1 In J2EE e-business world, EJB could be designed into two bean types: Session bean and Entity bean. In this project, I used Entity Bean according to the application's needs and for its reliable and stable features.

- ***Session Bean***

Session bean represents work being performed for client code that is calling it, as business process objects. They implement business logic business rules, and workflow. For example, a session bean could perform price quoting, order entry, video compression, banking transactions, stock trades, database operations, complex calculations and more. It's reusable components that contain logic for business processes. It lives for about as long as the session (or lifetime) of the client code.

- ***Entity Bean***

Another fundamental part of a business is the permanent data that the business processes use. The business process components are manipulating data in some underlying data storage, such as a relational database. An entity bean is a component that represents such persistent data. In this project, Entity beans model User account, Order, Category, EJBCart, Product. Entity beans represent real data objects, such as customers, products, categories, and so on. It provides an object-oriented in-memory view of data in an underlying data store, and can last long. As it represents data in a permanent, fault-

tolerant underlying storage, so it can survive critical failures such as application servers crashing. Here is a User EJB sample diagram to show how Entity beans are viewed into an underlying data store.

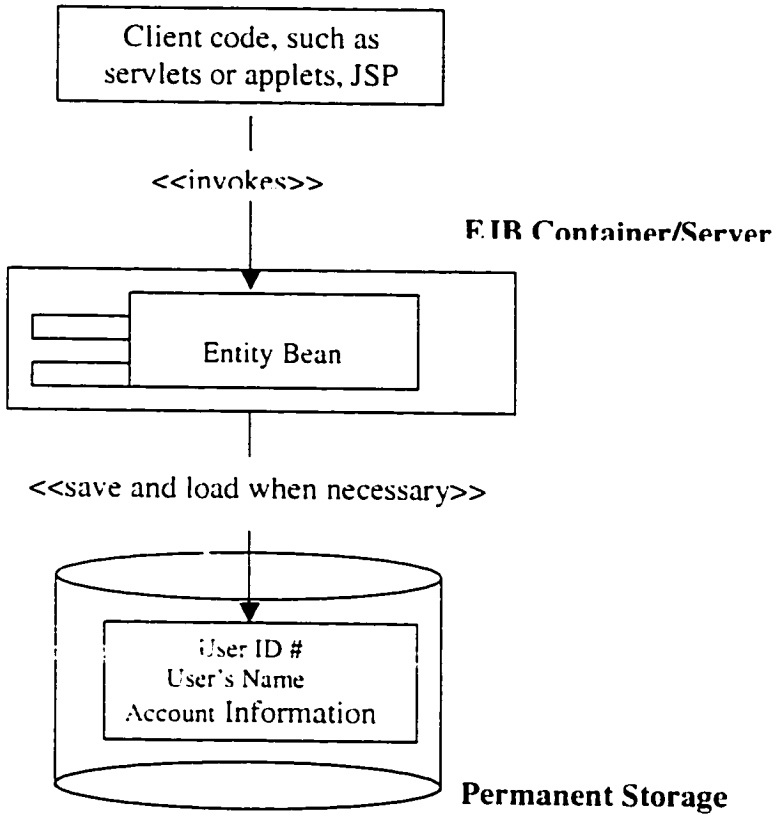


Figure 3 Entity beans are a view into an underlying data store

3.2.2 EJB objects must clone every business method that bean classes expose to invoke bean clients methods. There is a special interface that a bean provider writes, it can duplicates all the business logic methods that the corresponding bean class exposes. This interface is called the remote interface. Remote interfaces must comply with special rules that the EJB specification defines. For example, all remote interfaces must derive from a common interface that is supplied by Sun

Microsystems. This interface is called javax.ejb.EJBObject and here is a diagram to show the architecture.

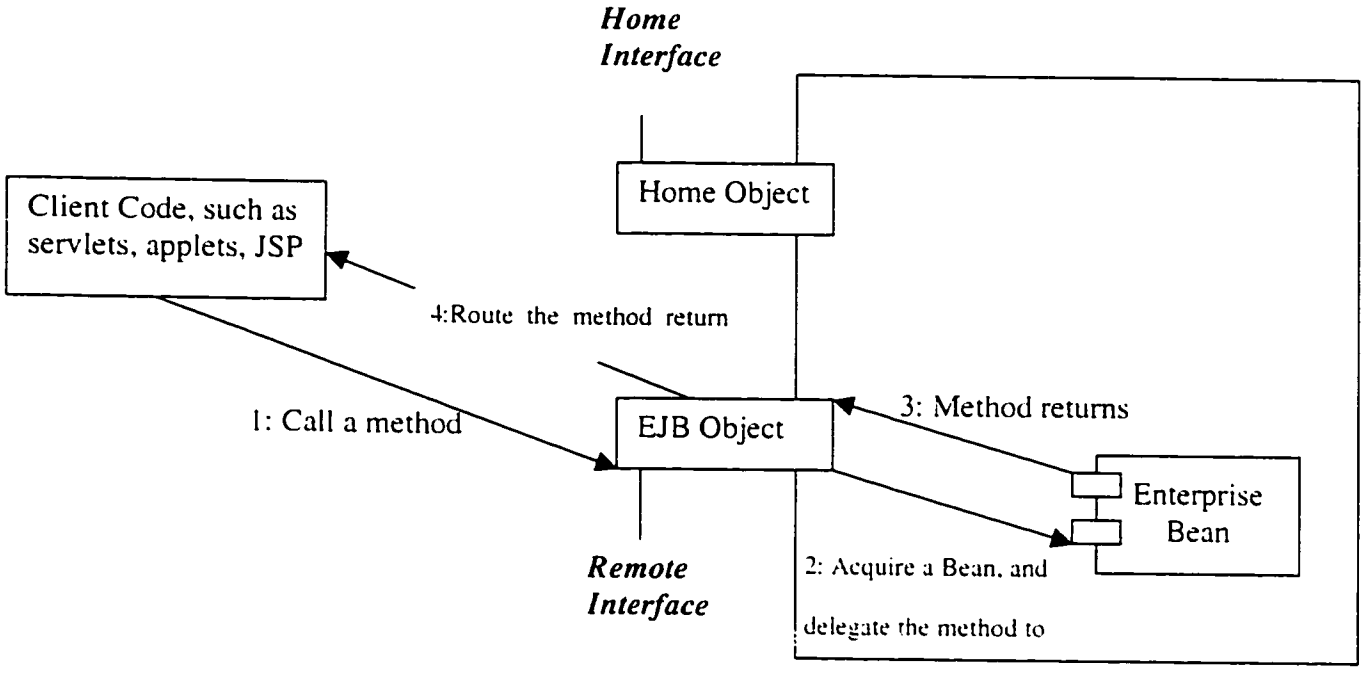


Figure 4 EJB Objects

3.3 System Support

CODC is a Web-based online shopping center application that is used to provide DVD online shopping facilities for general users, and provide a online e-business challenges to the specified users as DVD vendors, suppliers, and some other IT management or consultant who will be in charge of webmaster for online help and customer service.

3.4 System Requirement

The system should be implemented under the configuration of the following required components:

- Web Browser: Netscape or MS Internet Explore.
  - Installed Netscape 4.7 or above and MS IE 5.0 and above
- Web Server: Java Sever Pages
- Application Server: J2EE, EJB application server
  - Downloaded and installed Java 2 SDK Enterprise Edition V1.3 properly which also provides Java 2 Runtime Environment for EJB application server, from www.java.sun.com.
  - Downloaded and installed Java 2 SDK Standard Edition V1.3.1 for standard Java file environment.
  - Downloaded and installed JRun Application Server from www.macromedia.com
  - The only one I can choose is JRun Server Develop Edition for Windows English, this one is free but can only be accessed by 3 users in the same time.
  - URL: http: www.macromedia.com software jrun trial
- Database server: Microsoft Access – exists in Windows.

To achieve a better performance, the system gave out a testing for the application which provides on an IBM PC or compatible running on Windows 95/98, Windows NT / 2000:

- 36 MB RAM or more memory for Windows 95/98, Windows Me
- 32 MB for Windows NT/2000
- 10 MB of disk space for program files
- JDK 1.2.2 or higher environment
- Java 2 Run Time Environment

4 System Design

4.1 Architecture

CODC deployment is partitioned into three tiers:

The business logic tier consists of multiple Enterprise JavaBeans, running under the hood of an EJB container/server. These reusable components are independent of any user interface logic. The business tier is made up of persistent entity bean objects that represent data being modified, as well as provide application rules for interacting with that data.

The presentation tier involves one Web server that is responsible for interacting with the end user. It displays the requested information in HTML to the end user; it also reads in and interprets the user's selections and makes invocations to the business tier's enterprise beans. Here use JSP and Java Script for implementation.

The data tier is where permanent data stores inside. I used MS Access for database server. The databases aggregate all persistent information related to the e-commerce site - CODC.

4.2 System Overview

4.2.1 Object Model of the application server tier (Business Logic)

To fulfill CODC's requirements, here is the abstractions in the business's object model:

Products:

- The unique product ID

- The product name
- A description of the product
- The base price of the product (indicating the price of the product, with no taxes applied)

Customers (In the project, it is called User):

- The customer's ID number
- The customer's name
- The customer's email address (also used as the user's login name for the web site)
- The customer's billing address and shipping address
- The customer's password (used to verify the user's identify)
- The customer's payment method

EJB Cart (shopping action):

- The customer (entity bean) who is authenticated at the login screen. Need to store customer information for bill, and where to ship the manufactured products.
- The products and quantities that the customer currently has selected. This data is best represented in its own separate bean.
- The subtotal for the quote, taking into account all the prices of the products the user wants, as well as any discounts the user gets.

Product Line:

- The ID of the line item
- The product (entity bean) that the user wants to buy
- The quantity of the product

Prices: will take a quote as input and compute the subtotal of that quote.

Orders

- The ID of this order which the user can check on order status
- The Customer (entity bean) for which this order is generated, it's used for shipping address information.
- The products and quantities that should be ordered. (as with quotes, contained in Order line items)
- The subtotal on the order
- The date the order was placed

Order Line Items

- The ID of this order line item
- The product that this order line item represents (used by manufacturing to reveal which product to make)
- The quantity of the product that should be prepared
- The discount that the customer received on this product

Credit Card Accounts

- The credit card number
- The name of the credit card holder
- The expired date

4.2.2 Object Model of the web server tier (Presentation Logic)

To design the system's presentation tier, which displays the graphical user interface to the end user, I used Java Server Pages to interact with a client over HTTP.

Here is a state diagram for the presentation tier Object Model of CDOC system:

This diagram completes the object model design for the presentation tier. The flow of control is showing out, while the product base object is used behind the scenes to store references to Product entity beans.

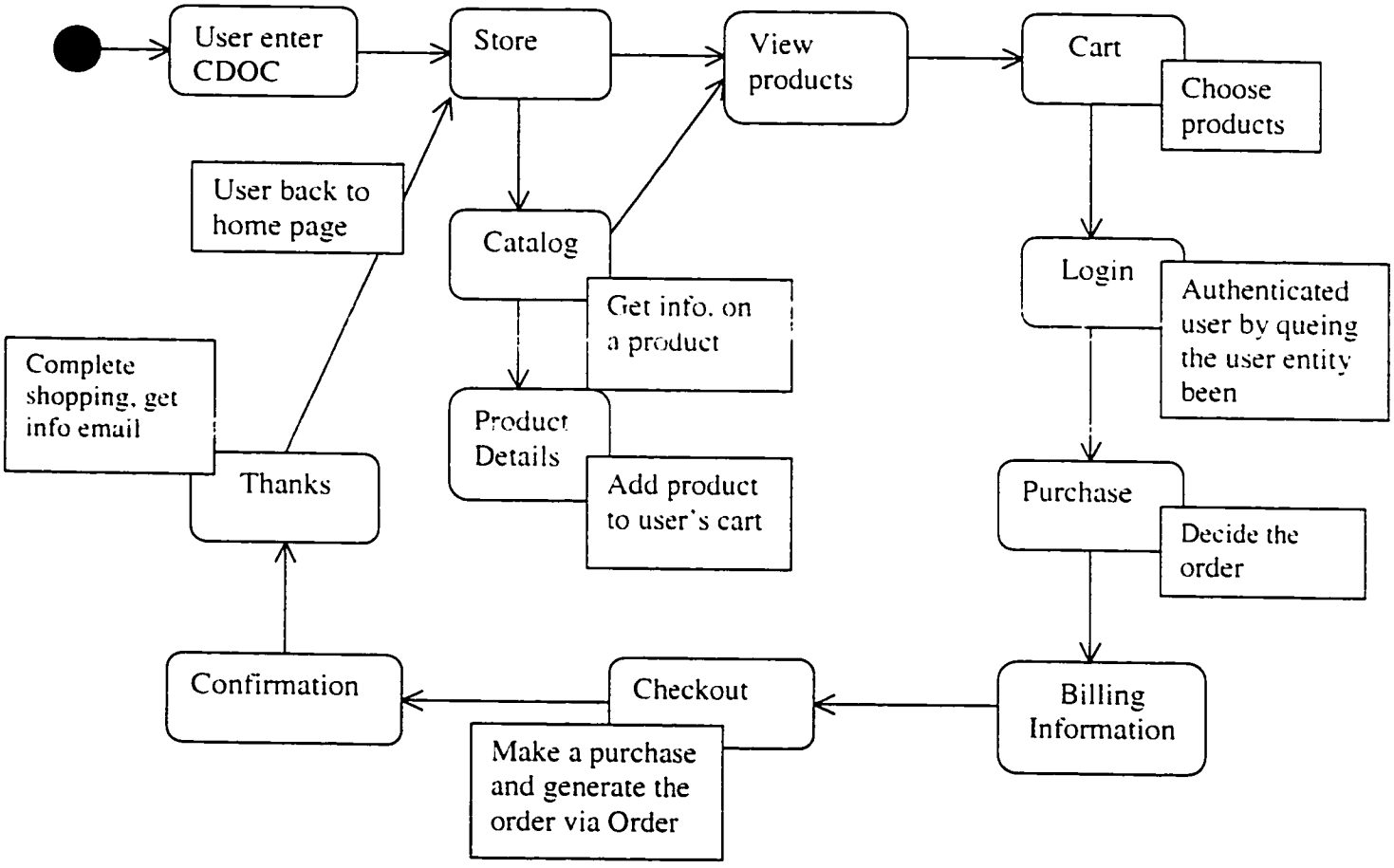


Figure 5 Object Model of CDOC System

4.2.3 Object Model of the database server tier (Data Tier)

Here is the Relational Schema for the Database of CODC:

- **Product:** (PID: number, CID: number, PName: string, Price: number, Actor: string, Director: string, Rated: string, Format: string, Length: number, Year: number, Company: string, Description: string, Image: string, PQty: number, Scount: number)

  Primary Key: PID ***Foreign Key:*** CID, Reference Category

- **Category:** (CID: number, CName: string) ***Primary Key:*** CID

- **Order:** (UID: number, OID: number, Date: date)

  Primary Key: UID, OID ***Foreign Key:*** OID, Reference User

- **User:** (UID: number, LoginName: string, Password: string, Email: string, FName: string, LName:string, Address: string, Phone: number, City: string, Province: string, Country: string, Postcode: string, CCType: string, CCNumber, ExpiredDate: date)

  Primary Key: UID

- **Cart:** (UID: number, OID: number, PID: number, CQty: number)

  Primary Key: UID, OID, PID ***Foreign Key:*** UID, OID, PID,

  Reference: User, Order, Product

Here is the diagram for the ER of database:

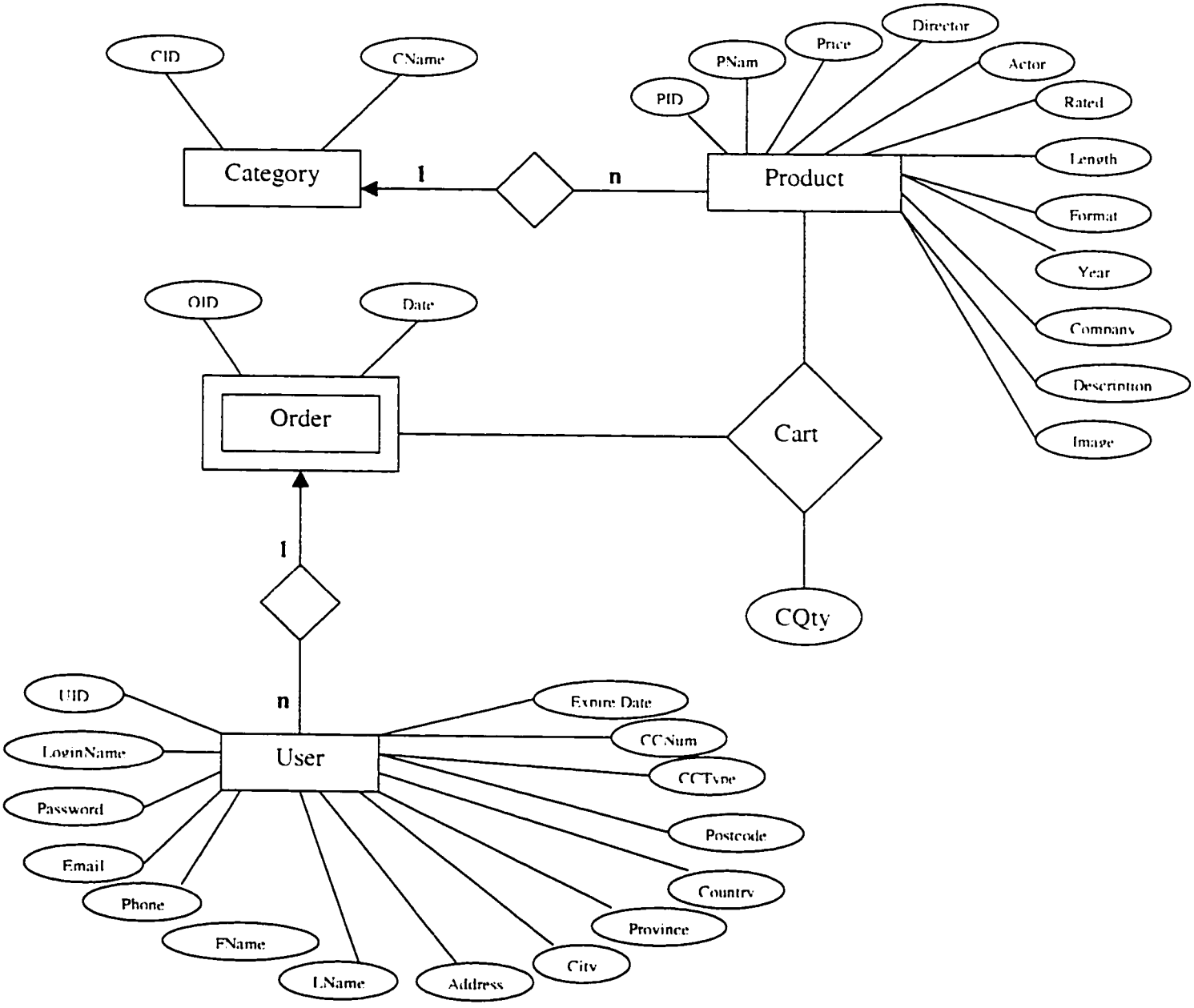


Figure 6 Diagram for the ER of database

4.3 User Interface Prototype

JSP is used for the web server tier (presentation logic tier), as the following:

- DVDdigital.jsp (home)
- Header.jsp (including header in every page)
- Footer.jsp (including footer in every page)
- Catalog.jsp (browse product category)
- Product.jsp (browse product)
- ProductDetail.jsp (browse specific product detail informations)
- Cart.jsp (shopping cart)
- Checkout.jsp (4 steps of checking out precession)
- EjbCart.jsp (shopping action through EJB deployment)
- Login.jsp (User login in)
- Update.jsp (User update the shopping quantity)
- RemoveFromCart.jsp (User cancel the shopping action)
- Thank.jsp (Final greeting for shopping with a detail email for user's information)

For the UI design, header.jsp and footer.jst can include the header and footer part in each page, this gives the UI part an efficient result to browse out the friendly-like GUI.

Diagrams will be shown in Chapter 6.

5 System Implementation of CODC

5.1 Graphic User Interface Design of CODC

In CODC, I used Java technology to develop the user interface as a standard online shopping GUI, combined with HTML, JSP, Java Script, I set events and handler to implement the different functional components in good effective and user-friendly way, and tried to achieve the following characteristics for GUI design:

- Use MVC (Model-View-Controller) structure to de sign the graphic user interface architecture. One Model controls multiple Views.
- Use JSP and Java Script technology to implement the GUI elements in a friendly Look-and-Feel mood.
- Set event and handling to implement the component functions dynamically and efficiently.

Architecture for interactive applications Introduced by Smalltalk developers as:

1) Separate out presentation (View), user input handling (Controller) and "semantics" (Model) which does the work; 2) Partitions application so that it is scalable and maintainable: Program a new model, and then re-use existing views and controllers; Multiple, different kinds of views on same model.

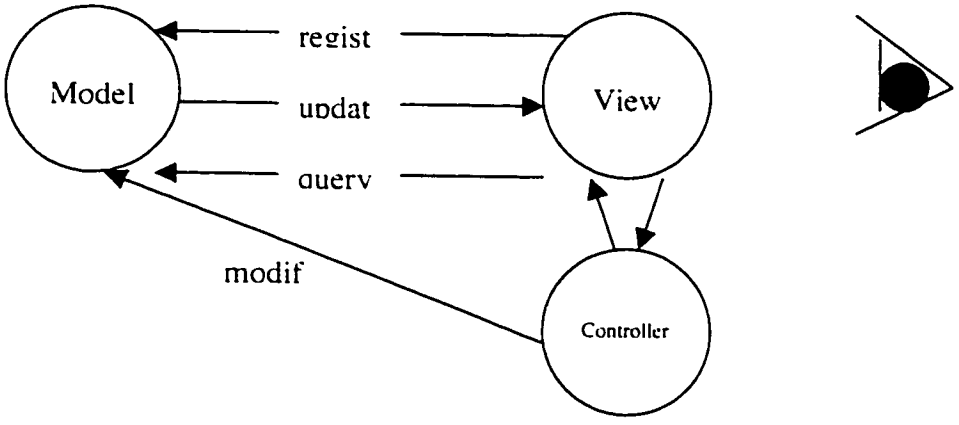


Figure 7 MVC Illustration Diagram

5.1.1 The Introduction of GUI of CODC :

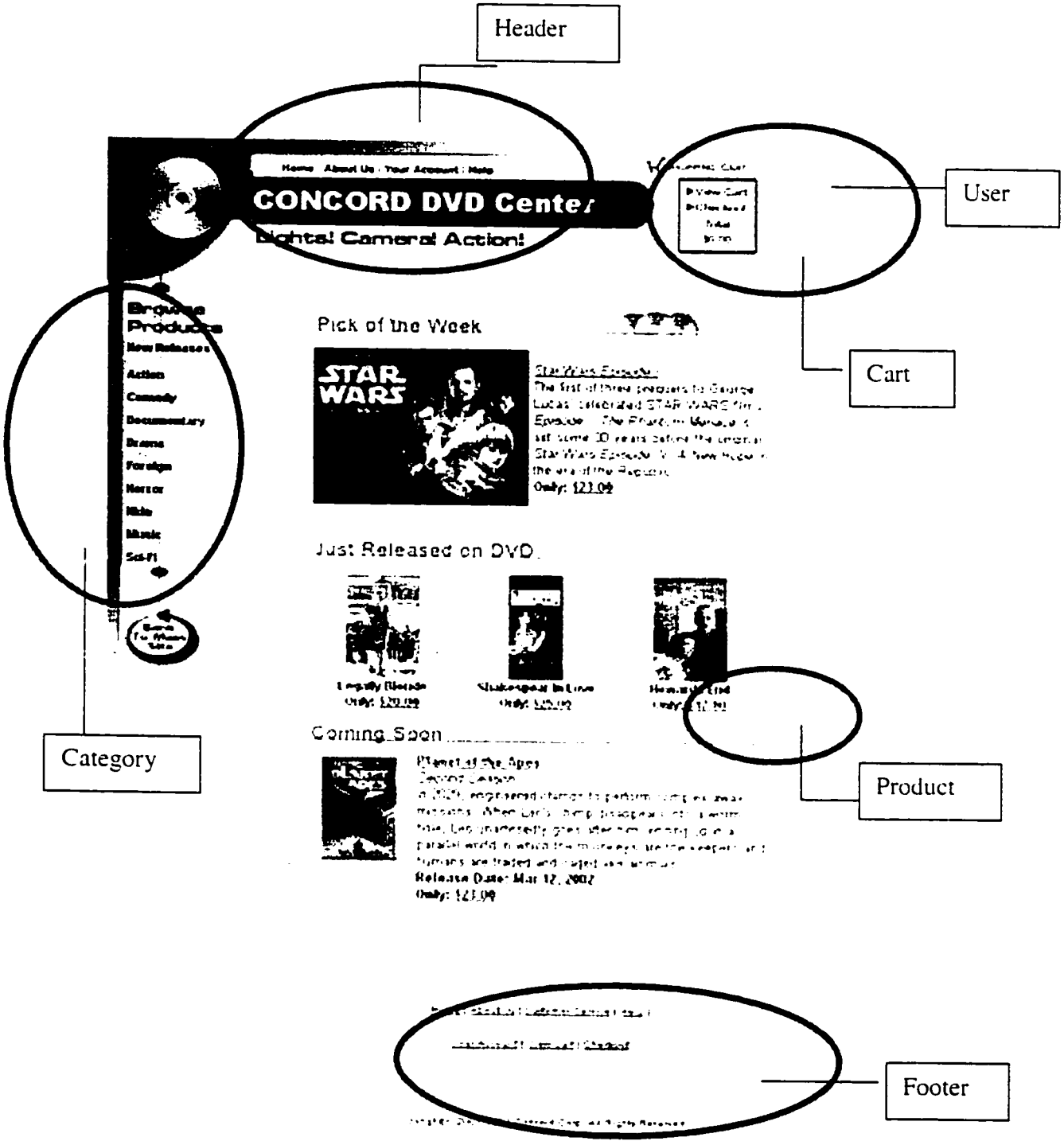


Figure 8 GUI of CODC System

5.2 CODC Server-side API

On the server-side, Entity Bean is chosen to be the object-oriented application logic components of the different objects in the multi-tier deployment, by JNDI (Java Naming and Directory Interface), such as: Category, User, Product, EJBCart, Order. Here is the object model for the product line of CODC to show the container-Managed Persistence concepts of the entity bean.

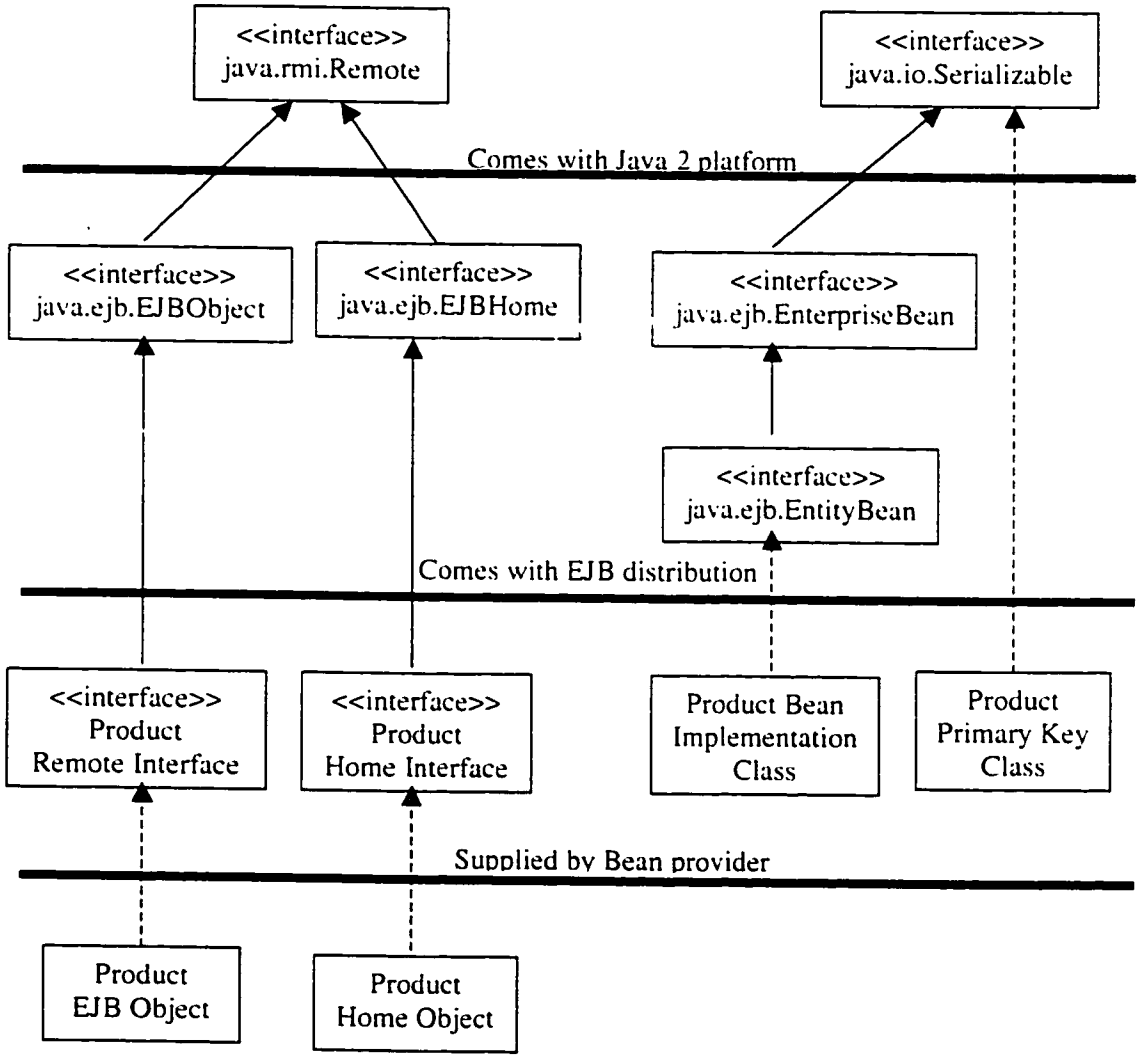


Figure 9 Object model for the product line in CODC system

5.2.1 Entity Bean Contents

Entity Beans can be filed as the following:

- **The entity bean class** is a Java class that models persistent data, it maps to an entity definition in a database schema.
- **The entity bean's remote interface** is the interface to the beans on which clients invoke, it has each entity bean's business method signatures.
- **The entity bean's home interface** is the interface clients use to crate, find and destroy entity bean EJB objects. It's the factory for EJB objects. Clients can find the home object by performing a JNDI lookup.
- **The entity bean's primary key class** is a unique identifier, it makes every entity bean different and it's an object that may contain any number of attributes.
- **The entity bean's deployment descriptor** contains a list of properties that should be used by the container on deployment.

5.2.2 CODC Server-side API description

1) The Category Entity Bean:

- Category.java
- CategoryBean.java
- CategoryHome.java
- CategoryKey.java

Function Description: Establish the category of CODC by category ID Number:CID() and category Name: CName(). Call from DB by CategoryKey.

Signature Information: Argument of Category CID and CName refer to the category ID number and name, the return value of the method is true if the category is found out from DB successfully, otherwise it will be false.

Throwing Exception: throws RemoteException; throws FinderException; throws CreateException; throws RemoveException

2) The User Entity Bean

- User.java
- UserBean.java
- UserHome.java
- UserKey.java

Function Description: Establish the user's information by: UID(), LoginName(), Password(), Email(), FirstName(), LastName(), Address(),City(), Province(), Country(), Postcode(), AreaCode(), Phone(), and Credit Card Information : CCType(), CCNumber(), ExpireDate(). The Key is UID().Users are created by direct database inserts.

Signature Information: Argument of User refers to the user's information including: User ID (this is the Key), login name, password, email address...etc., these are created by direct database inserts, a simple maintenance utility could easily be written to add new User via the GUI as well.

Throwing Exception: throws RemoteException; throws CreateException; throws FinderException

3) The Product Entity Bean

- Product.java

- ProductBean.java
- ProductHome.java
- ProductKey.java

Function Description: Establish the Product by: PID(); CID();PName(); PDescription(); PPrice();PDirector();PLength();PRated();PActor();PFormat();PYear();PImage(). The Key is PID() and CID(). These functions carry out the product details and can be called from Database as they are inserted. The return value of the method is true if the category is found out from DB successfully, otherwise it will be false.

Signature Information: Argument of Product methods refers to the Product ID (combined with Category ID), Product Name, Product Description, Product Price, Product Director, Product Length, Product Rated, Product Format, Product Year, Product Image…, these are created by direct database inserts, a simple maintenance utility could easily be written to add new User via the GUI as well.

Throwing Exception: throws CreateException; throws FinderException; throws RemoveException.

4) The EJBCart Entity Bean

- EJBCart.java
- EJBCartBean.java
- EJBCartHome.java
- EJBCartKey.java

Function Description: Create EJBCart object by OID, PID, CQty (Quantity in Cart) and will establish a Cart ID for the object when a shopping activity is successfully processed. The methods: key.getCartID();ejbFindByOID; ejbActivate(); ejbPassivate(); ejbLoad();

ejbStore(); getOID(); getPID(); getCQty(). The return value of the methods "getOID(); getPID(); getCQty()" is true if the shopping act is processed successfully, otherwise it will be false.

Signature Information: Arguments of the EJBCart methods refer to the shopping cart activities. Argument of "getOID(); getPID(); getCQty()" refer to get the product into the shopping cart by the Order ID Number, Product ID Number, in Cart Quantity. The return value of the methods is true if the shopping cart with products successfully with a Cart ID number, or the return value is false.

Throwing Exception: throws CreateException; throws FinderException; throws RemoveException.

5) The Order Entity Bean

- Order.java
- OrderBean.java
- OrderHome.java
- OrderKey.java

Function Description: Create Order object by OID(), UID(), ODate() , ODate() is the key for this object. It will establish a Order ID for the object when an order activity is successfully processed. The methods: key.getODate();ejbFindByOID; ejbActivate(); ejbPassivate(); ejbLoad(); ejbStore(); getOID(); getUID(); getODate(). The return value of the methods "getOID(); getUID(); getODate()" is true if the order activity is processed successfully, otherwise it will be false.

Signature Information: Arguments of the Order methods refer to the Order activities. Argument of "getOID(); getUID(); getODate()" refer to get the order activity reference

by the Order ID Number, User ID Number, Order Date. The return value of the methods is true if the order activity with products is successfully processed with a Order ID number, or the return value is false.

Throwing Exception: throws CreateException; throws FinderException; throws RemoveException.

5.3 Data Base tier

To connect the DB with the EJB objects, the XML file must firstly use the EJBKey methods to call the different EJB object and then connect it to the DB by these Key attributes. For example, in EJB Order object, XML use UID, OID, ODate as the Key. As MS Access doesn't provide the combination of primary key / foreign key feature functions, so it is defined in EJB key files such as: CategoryKey.java, EJBCartKey.java, OrderKey.java, UserKey.java, ProductKey.java. The key files can be called in Database with deployment by calling XML files and it can be generated as ejb-jar.category.xml, ejb-jar.EJBCart.xml, ejb-jar.order.xml, ejb-user.xml, ejb-jar.product.xml.

To connect Database with application, should connect ODBC JDBC as bridge, and set the DB property in the application server. In this project, the connection procedure is:

- Go to win2000 ODBC Data Source Administrator to add project database file "ejb_chen" into System DSN and configure it, and choose correct DB driver.
- Go to JRun Administration Server, through JRun Default Server > JDBC Data Source, add the project database file ejb_chen, use sun.jdbc.odbc.JdbcOdbcdriver as the connection driver, the URL for internal deployment is jdbc:odbc:ejb_chen.

6 Experimental Result

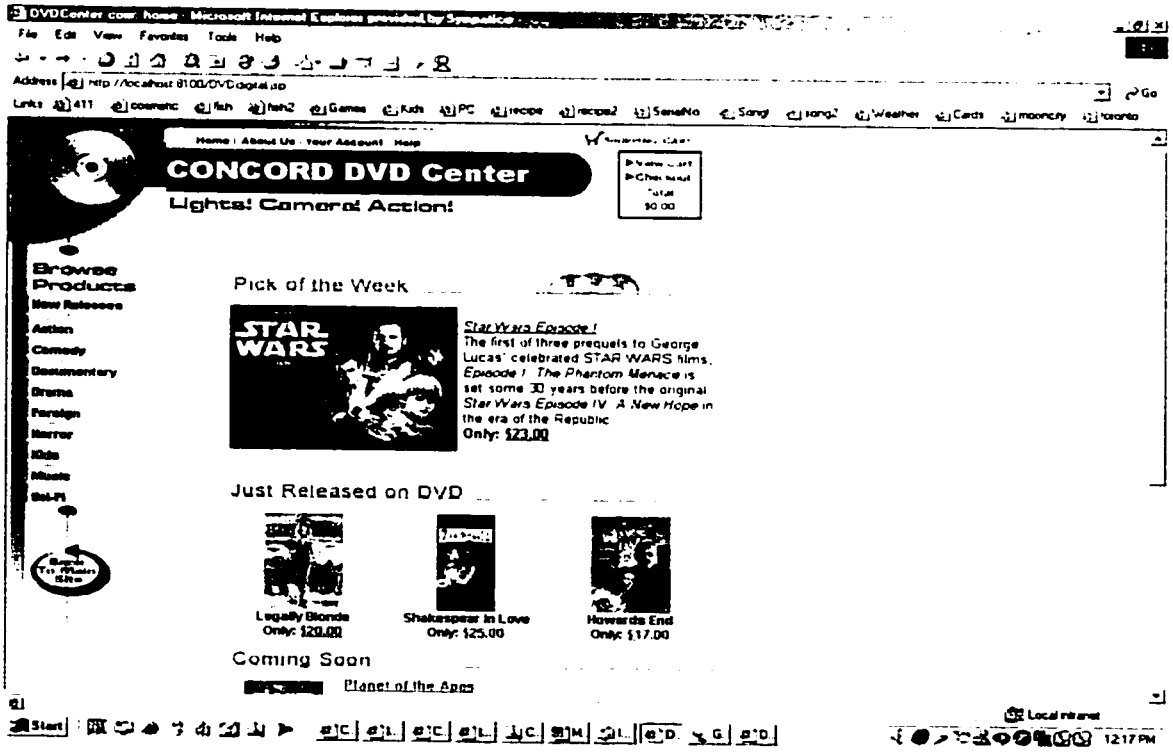
The CODC (Concord Online DVD Center) Test Program is intended to test functionality of CODC by mapping each method in CODC to the corresponding internal business logical transactions in the CODC test program. The test program is located in the directory "bin" has a name of "Project". the testing has been taken on the functions of CODC, the main functions can work correctly according the e-business logic.

Testing procedure is as the following:

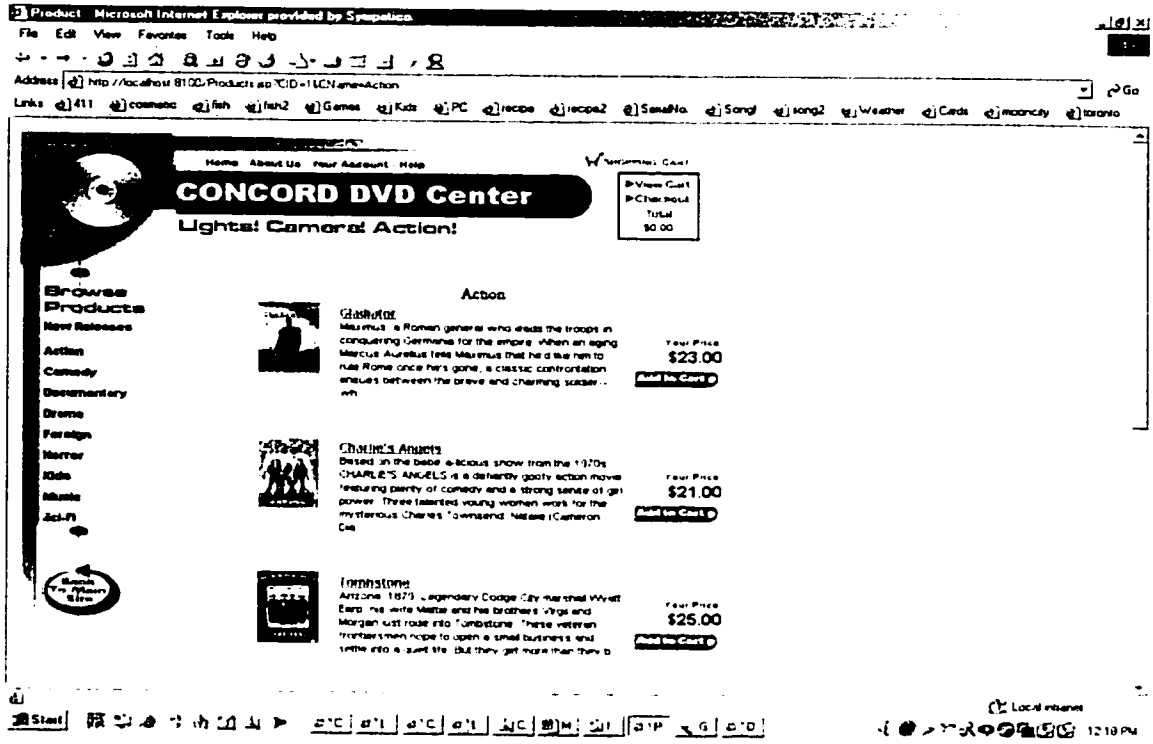
1) Open home page by the URL: http://65.94.182.188:8100/DVDdititat.jsp

Then browse the Concord Online DVD Center (CODC).

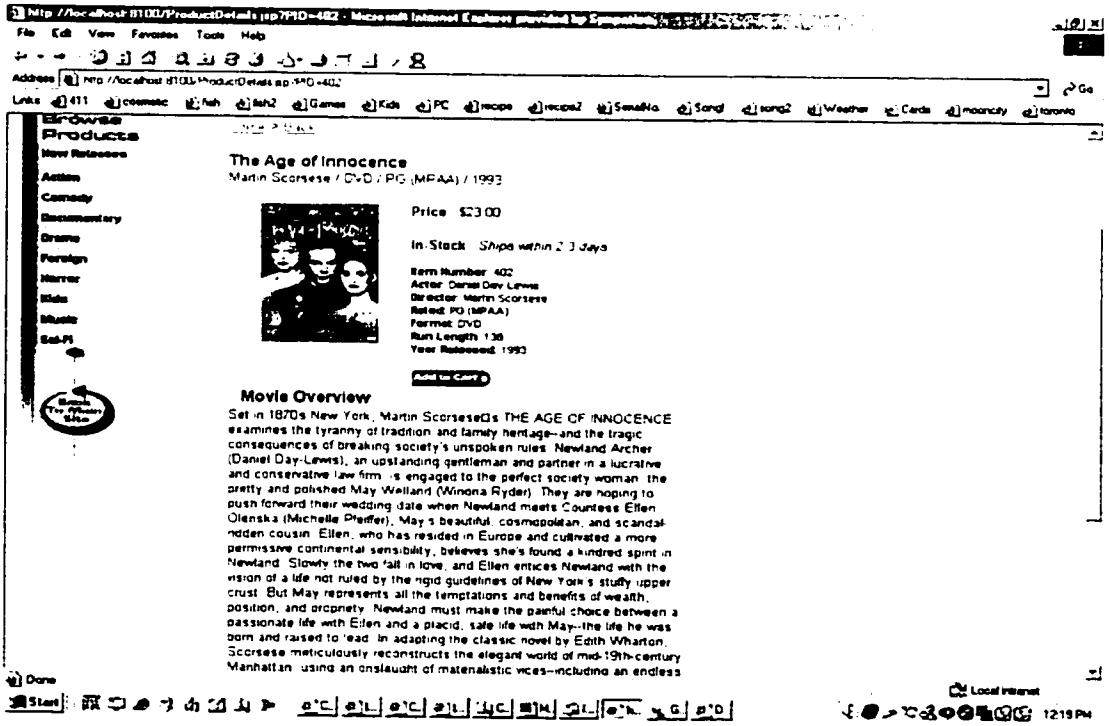
Please note: as the application IP address is provided by the local internet service provider, so that the IP address is different when link to the local network in different time.



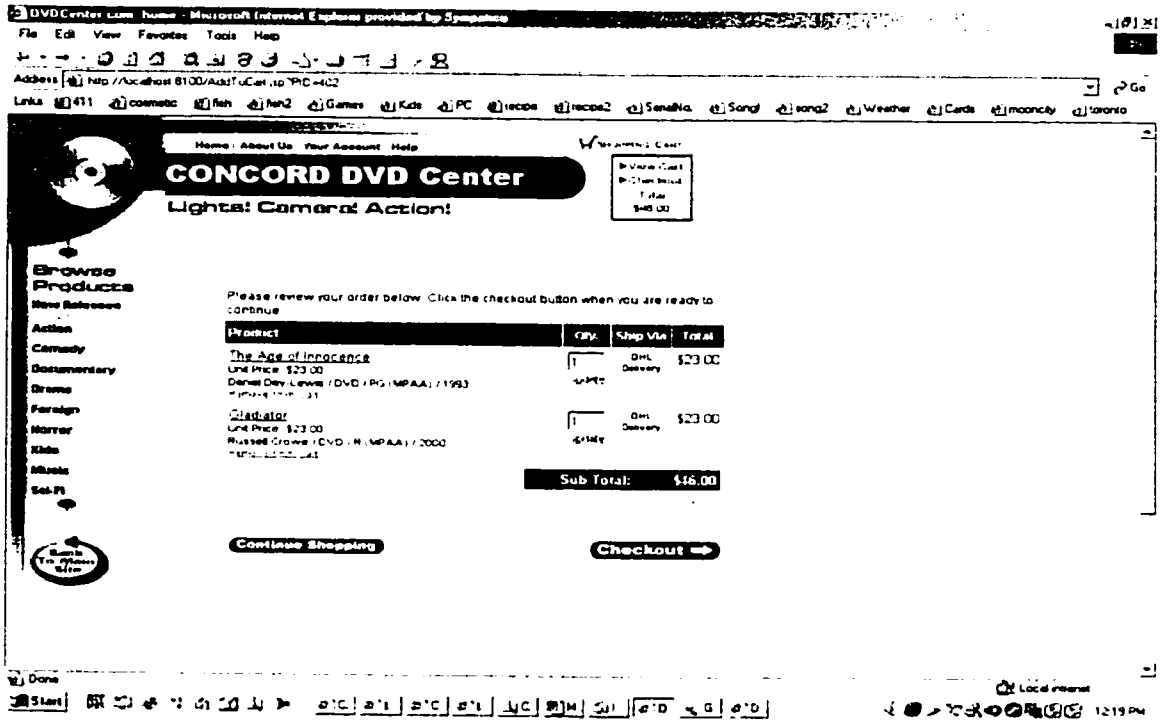
2) Go to category, click on "Action", show up the product page of Action category, click on "Gladiator", show up the product detail page of "Gladiator", then add it in cart.



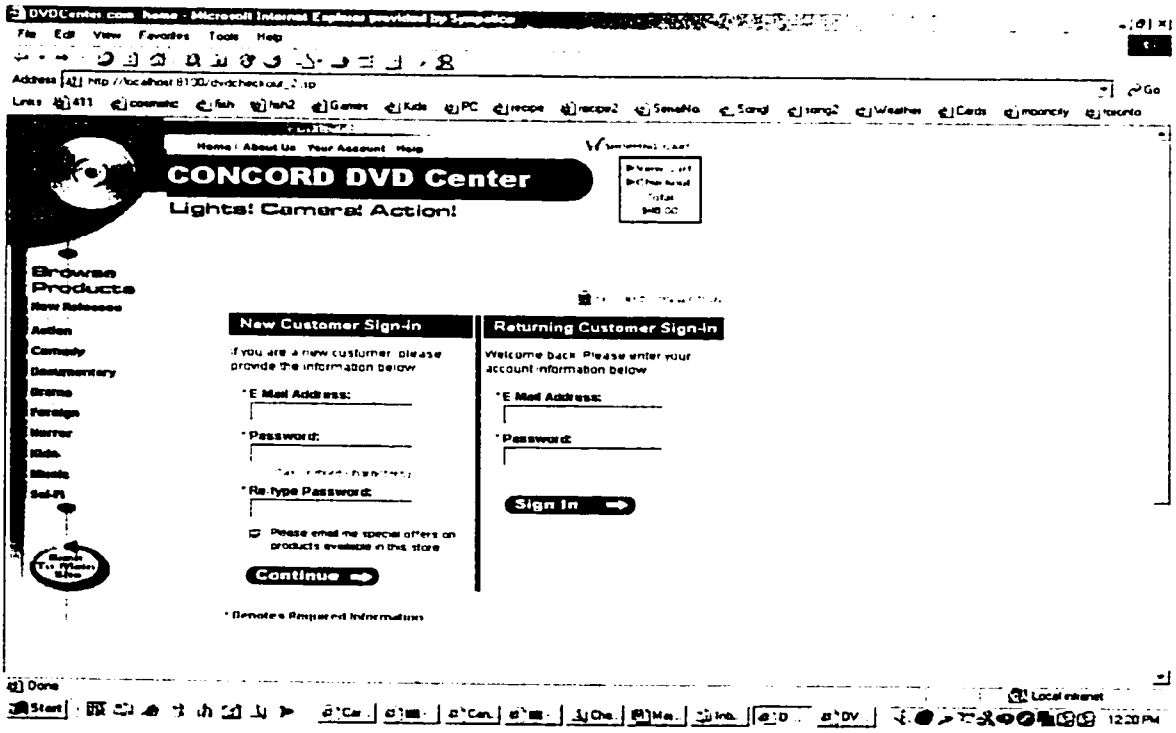
3) Go to category, click on "Drama", click the product page of "The age of innocent".



4) Open "shopping cart" page, two items are in the cart, click "Back" to the home page.



5) Click "View Cart" to "shopping cart" page, then "continue" to "customer sign-in".



6) Back to homepage, click "checkout", it also goes to "customer sign-in" page. (same picture as above)

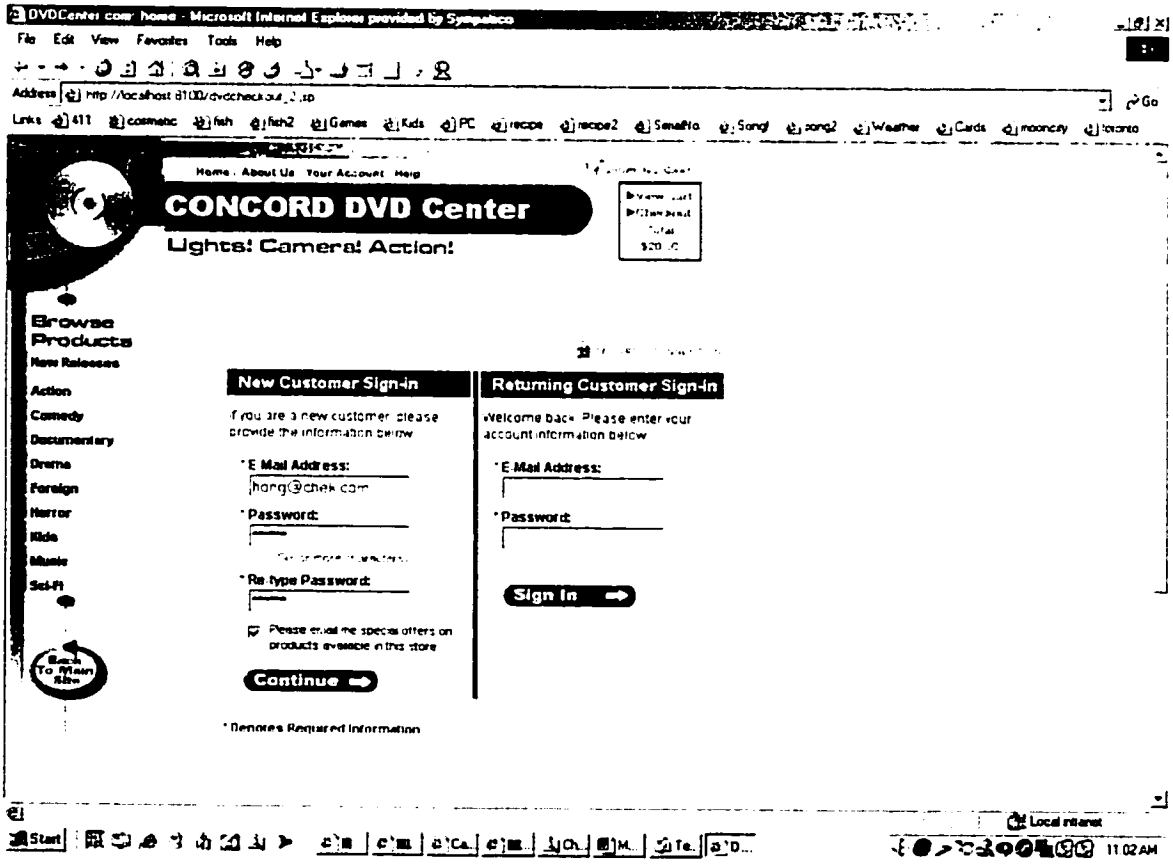
7) In "customer sign-in" page, enter new customer information:

email address: hong@chek.com

password: hhhhhh

re-type password: hhhhhh

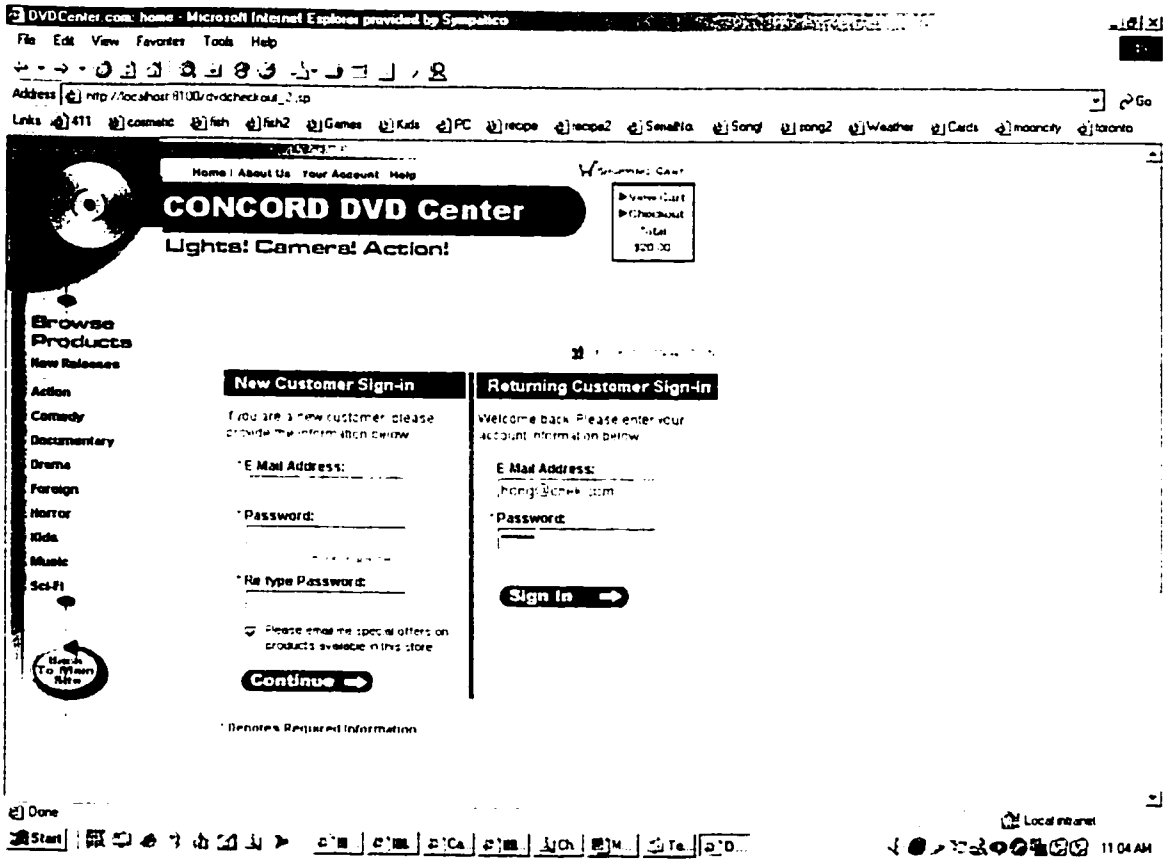
Then click "continue", it goes to Billing Information page.



8) Back to "sign in:", even if I didn't shopping, the above account is already set up and stored in database, so I can use Returning Customer Sign-in for the above user account.

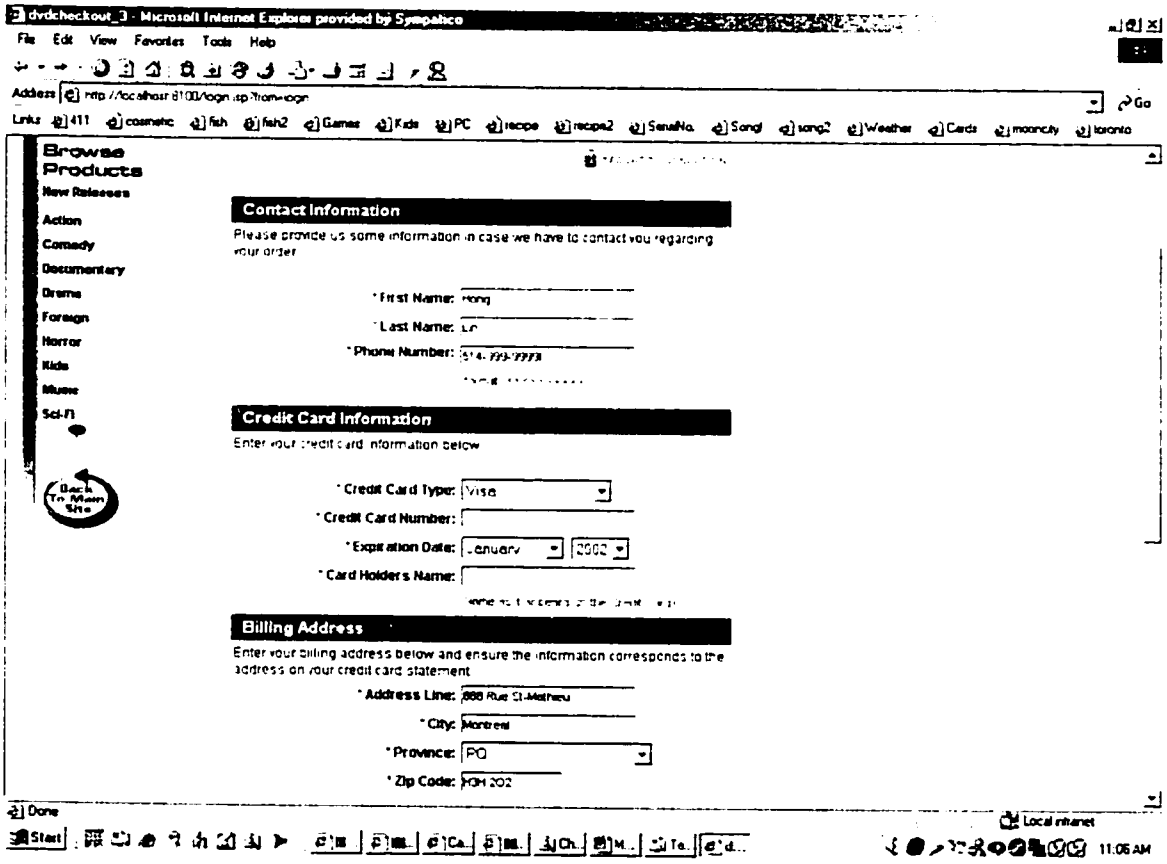
Email address: hong@chek.com

Password: hhhhhh

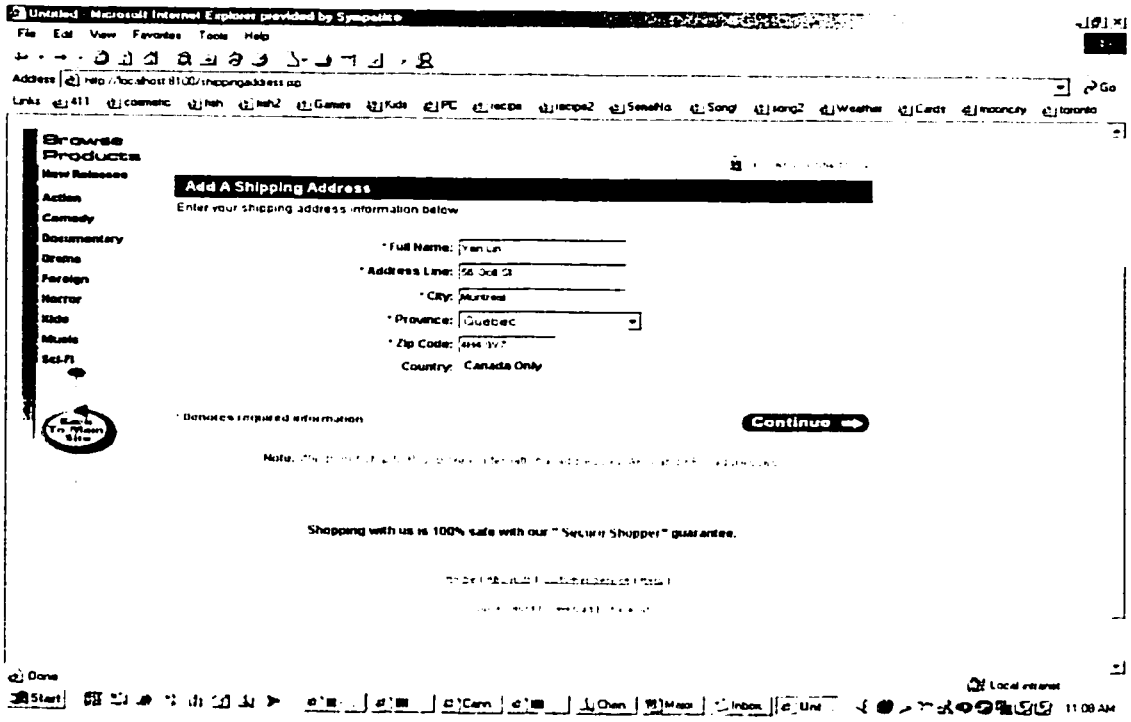


9) Now, go to the second step of checkout – billing information page.

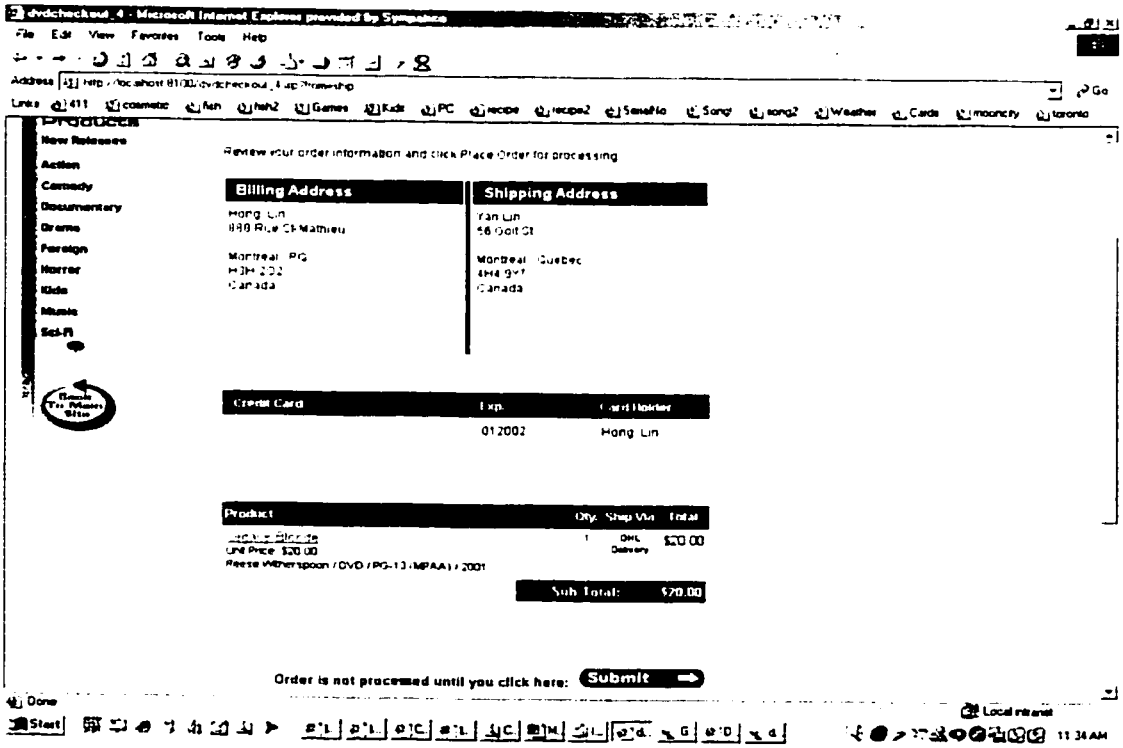
In Billing Information page, as a new customer should fulfill the null contact information, billing address and credit card information; as a returning customer, the database will show out the fulfilled contact information, billing address and contact information. Select in the check box for the shipping address, then "continue".



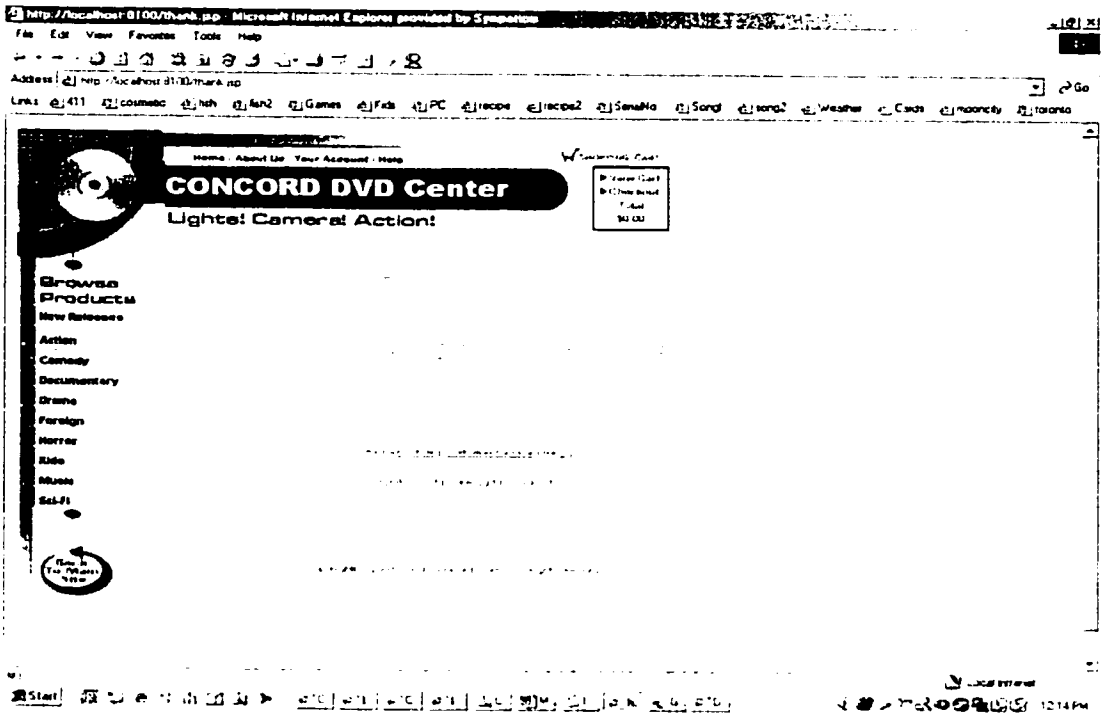
10) Go to Shipping Information page, fulfill the shipping requirement.



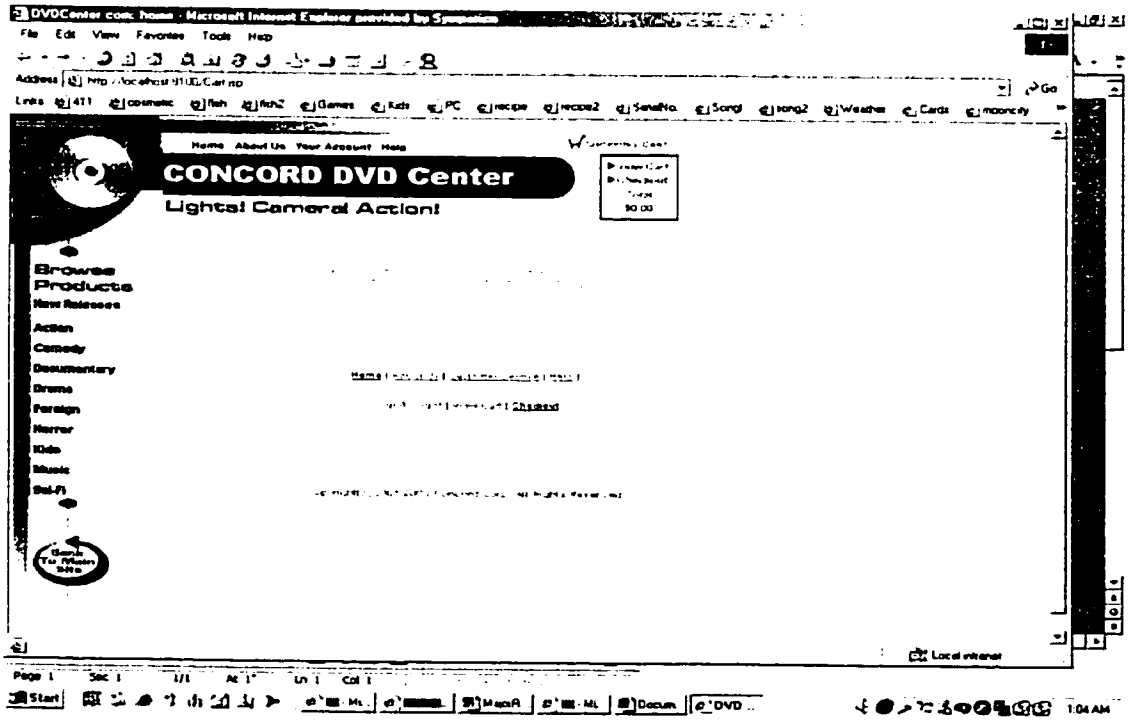
11) Then click "Continue", go to "Order Review" page



12) Click " Submit", it appears " Thank you" and notify you of your order ID number for reference. And you will get a thank you letter for your merchandise detail information.



13) Click "shopping cart" if without shopping, it appears: your cart is empty!



14) Go to Home by the "home" icon on the head.

15) Choose "Shakespear in Love" in the home page, and add it to cart

16) Check out, go to "customer sign-in" page

email address: puzzle@hotmail.com

password: aaaaaa

Click " continue", the following procedure will the same as above.

7 Conclusion

7.1 Difficulties

The most difficult of this project is to configure the required components of the system, and the design of the contents of EJB, like the components of different session beans.

EJB debug is another key difficult in the implementation.

Here are some typical difficulties on EJB application design:

IDE Integration (integrated development environment): Most EJB container/server vendors provide an easy-to-use IDE, can assist in code management, automate programming tasks and aid in debugging, but some vendors won't, so it will cause difficulties during developing and debugging.

High-Availability: server-side deployment is critical, even for EJB server. It will cause the difficulties in configuring the operating systems and hardware to support the server.

Intelligent E-Commerce Business Logic: even for the skilled developer, this could be a big difficult for e-business application design. Without the efficient business logic, the application won't be successful.

7.2 Future Work

CODC is a challenge project that I try to use Java technique such as JSP J2EE, EJB to implement an online DVD shopping center. For the future EJB technology standard, it's already raised up to 2.0, while the common reference books are all about 1.1. So the most important work for is to get updated from the newest technical version standard. On the other hand, my thoughts about the future work on CODC EJB e-business application design could be generated as the following:

- Improve the **business logic** during application design and make it much more customer oriented.
- Provide **SSL** (Secure Sockets Layers), to enhance the security protection during online shopping transaction, provides users with a safer way for online shopping.
- **Drag and drop** capability: allows users to drag and drop merchandise items directly from category into shopping cart without viewing the product detail information page, this will provide efficient service for some familiar customers.
- **GUI** design could be much more intuitive and fancy to attract potential online customers.
- **Browser URL Integration:** which provides powerful, reliable and speedy to accelerate user's browser download CODC's URL, not only accessed from IP address.
- **Find Utility:** enable users to search DVD items that match a name or the other properties such as making year, director or actors, etc, to enhance the efficient usability for customers.

8 Reference

- Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition (by Ed Roman, Wiley Computer Publishing, John Wiley % Sons, Inc)
- The Jakarta Tomcat reference implementation for JavaServer Pages and Java servlet technologies: it's free from Apache: http://jakarta.apache.org/tomcat
- Sun's J2EE FAQ: http://java.sun.com/j2ee/faq.html
- "A Beginner's Guide to Enterprise JavaBeans," Mark Johnson (*JavaWorld*, October 1998) covers EJB 1.0, but is still useful conceptually: http://www.javaworld.com/javaworld/jw-10-1998/jw-10-beans.html
- "CRN Interview: Bill Roth, Sun," Colin Browne (*ITP.net*, 2001) discusses licensing, compatibility tests, and open source: http://www.itp.net/features/97524614097034.htm
- "Develop N-tier Applications Using J2EE," Steven Gould (*JavaWorld*, December 2000) is a primer on J2EE technologies: http://www.javaworld.com/javaworld/jw-12-2000/jw-1201-weblogic.html
- Online library of www.java.sun.com, about EJB and J2EE
- Online library of http://www.macromedia.com, about JRun Application server and EJB introductions.
- EJB 2.0 specification: http://www.javasoft.com/products/ejb/docs.html
- Java experts answer your toughest Java questions in *JavaWorld*'s **Java Q&A** column: http://www.javaworld.com/javaworld/javaqa/javaqa-index.html

9 Appendices (Key pieces of source code)

9.1 JSP File (Home.jsp)

```
<!-- Top.jsp-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>DVDCenter.com: home</TITLE>
 <SCRIPT language=JavaScript type=text/javascript>
</SCRIPT>
</HEAD>
<BODY vLink=#000000 link=#003399 bgColor=#ffffff leftMargin=0 topMargin=0 marginwidth="0"
marginheight="0">
<jsp:include page="header.jsp" />

        <!-- Begin Main content -->
    <TABLE cellSpacing=0 cellPadding=0 width="100%" border=0>
     <TBODY>
     <TR>
      <TD width=#1 bgColor=#ffffff><IMG height=1
       src="DVDimage/spclear.gif" width=1 border=0></TD>
      <TD vAlign=top align=middle width="100%" bgColor=#ffffff>
       <TABLE cellSpacing=0 cellPadding=2 width="100%" border=0>
        <TBODY>
        <TR>
         <!-- home -->
         <TD vAlign=top align=middle width="100%" bgColor=#ffffff>
          <TABLE cellSpacing=0 cellPadding=0 width="100%" border=0>
```

```
<TBODY>
<!-- b catlayout -->
<TR>
 <TD colSpan=6>
  <!-- product Super Mod -->
  <TABLE cellSpacing=0 cellPadding=2 width=435 border=0>
   <TBODY>
   <TR>
    <TD width="100%"><IMG height=30
     src="DVDimage/hdr_bmmdvd.gif" width=435>
    <TD>
   <TR>
   <TR bgColor=#ffffff>
    <TD vAlign=top align=left bgColor=#ffffff><a
     href="ProductDetails.jsp?PID=906"><img
     height=150 hspace=5
     src="DVDimage/starwars.jpeg"
     width=200 align=left vspace=5 border=0></a> <FONT
     face=Arial color=#000000 size=2><BR><a
     href="ProductDetails.jsp?PID=906">
     <I>Star Wars Episode I</I> </a><BR>
     The first of three prequels to George Lucas' celebrated
     STAR WARS films, <I>Episode I: The Phantom Menace</I>
     is set some 30 years before the original <I>Star
     Wars Episode IV: A New Hope</I> in the era of the
     Republic.<BR>
     <FONT
     face=ARIAL size=2><B>Only: </B><a
```

```
        href="ProductDetails.jsp?PID=906"></FONT><B>$23.00<FONT
        color=#ff0000><B>< B>< FONT>< B>< A>
          <FONT face=arial color=red
        size=2><B>< b></FONT><BR>
        <BR>
        <BR>
        </FONT></TD>
      <TR>
      <TBODY>
     <TABLE>
   <TD>
  <TR>
  <TR>
   <TD colSpan=6>
    <TABLE cellSpacing=0 cellPadding=0 width=435 border=0>
     <TBODY>
     <TR>
      <TD width="100%" colSpan=3><IMG height=30
       src="DVDimage/hdr_bmmjustreleaseddvd.gif"
       width=435> </TD> < TR>
     <TR>
      <TD vAlign=top align=left colSpan=5><IMG height=10
       src="DVDimage/spclear.gif" width=1
       border=0><BR>
      </TD></TR>
     <TR>
      <TD vAlign=center align=middle width=145><A
       href="ProductDetails.jsp?PID=204"><IMG
```

```
 height=100
 src="DVDimage/LegallyBlonde.jpg"
 width=95 border=0></A><BR>
<TD>
<TD vAlign=center align=middle width=145><A
 href="ProductDetails.jsp?PID=203"><IMG
 height=100
 src="DVDimage/Shakespear.jpg"
 width=95 border=0></A><BR>
</TD>
<TD vAlign=center align=middle width=145><A
 href="ProductDetails.jsp?PID=406"><IMG
 height=100
 src="DVDimage/HowardsEnd.jpg"
 width=95 border=0></A><BR>
</TD></TR>
<TR>
<TD Align=top align=middle width=145><SPAN
    style="FONT-SIZE: 9pt; COLOR: #000000; FONT-FAMILY: Arial"><A
 href="ProductDetails.jsp?PID=204"><B></b></A><B> Legally Blonde
 </B><BR>
 <B>Only:</B> <A
 href="ProductDetails.jsp?PID=204"><B>$20.00</B></A><BR>
 <FONT
 color=#ff0000><B></B></FONT> </SPAN></TD>
<TD vAlign=top align=middle width=145><SPAN
   style="FONT-SIZE: 9pt; COLOR: #000000; FONT-FAMILY: Arial"><A
 href="ProductDetails.jsp?PID=203"><B></b></A><B>Shakespear In Love
```

```
        </B><BR>
        <B>Only:</B> <A
        href="ProductDetails.jsp?PID=203"><B>$25.00</B></A><BR>
        <FONT
        color=#ff0000><B></B></FONT> </SPAN></TD>
      <TD vAlign=top align=middle width=145><SPAN
          style="FONT-SIZE: 9pt; COLOR: #000000; FONT-FAMILY: Arial"><A
       href="ProductDetails.jsp?PID=406"><B></b></A><B>Howards End</B><BR>
        <B>Only:</B> <A
        href="ProductDetails.jsp?PID=406"><B>$17.00</B></A><BR>
        <FONT
        color=#ff0000><B></ B></ FONT> </ SPAN></ TD>
    </ TR> </ TBODY> </ TABLE>
</ TD> </ TR>
<TR>
  <TD colSpan=6>
    <!-- product Super Mod -->
    <TABLE cellSpacing=0 cellPadding=2 width=435 border=0>
      <TBODY>
      <TR>
        <TD width="100%"><IMG height=30
          src="DVDimage/hdr_bmmcomingsoon.gif"
          width=435> </TD>
      </TR>
      <TR bgColor=#ffffff>
        <TD vAlign=top align=left bgColor=#ffffff><A
          href="ProductDetails.jsp?=pid902"><IMG
          height=100 hspace=5
```

```
            src="DVDimage/PlanetApes.jpg"
            width=95 align=left vspace=5 border=0></A> <FONT
            face=Arial color=#000000 size=2><A
               href="ProductDetails.jsp?=pid902"><B>Planet of the Apes</B>< A><BR>
            <!--<a href="">-->
            <I>Second Season</I><BR>
            In 2029, engineered chimps to perform complex away missions.
               When Leo's chimp disappears into a worm hole, Leo unadvisedly
               goes after him, ending up in a parallel world in which the monkeys
               are the keepers and humans are traded and caged like animals...<BR>
               <B>                     
                   
                 Release Date: Mar 12, 2002</B>
<BR>   <FONT face=arial size=2><B>
                       
                   
                 Only: <A
            href="ProductDetails.jsp?=pid902">$23.00</A>
              <BR>
            <BR>
            </b></FONT></FONT>
              </TD> </TR>
             </TBODY>
           </TABLE>
       </TD>
    </TR>
      <!-- e catlayout -->
      </TBODY>
```

```
        </TABLE>
       </TD>
       <!-- home -->
      </TR>
      </TBODY>
     <TABLE>


          <!-- End of Main Content -->
               <!-- Begin Footer -->
               <jsp:include page="footer.jsp" />
```

9.2 EJB File (Use Category as sample here, there are four files in Category)

- **Category.java**

```
package com.ejb_chen;
import java.rmi.*;
import javax.ejb.*;

public
interface Category
extends EJBObject
{
  String getCID() throws RemoteException;
  String getCName() throws RemoteException;
}
```

➢ CategoryBean.java

```
package com.ejb_chen;

import java.rmi.*;

import java.sql.*;

import javax.ejb.*;

import allaire.ejipt.*;


public

class CategoryBean

implements EntityBean

{

    private EntityContext _context;

    private transient String _CID = "";

    private String _CName = "";

    private String _Level = "";


    public void setEntityContext(EntityContext context)

    {

      _context = context;

     }

    public void unsetEntityContext()

    {

      _context = null;

     }

    public CategoryKey ejbCreate(String id, String name, String level)

            throws CreateException, RemoteException

    {
```

```
  _CID = id;
  _CName = name;
   _Level = level;
   return null;
 }

 public void ejbPostCreate(String id, String name, String level)
        throws CreateException
 {}
 public CategoryKey ejbFindByPrimaryKey(CategoryKey key)
        throws FinderException
 {
    _CID = key.getCID();
    return null;
 }
 public String ejbFindByLevel(String level)
        throws FinderException
 {
    _Level = level;
    return null;
 }
 public void ejbRemove()
        throws RemoveException
 {}
 public void ejbActivate()
 {}
 public void ejbPassivate()
 {}
```

```
    public void ejbLoad()
    {}

    public void ejbStore()
    {}
    public String getCID()
    {
     return _CID;
     }
    public String getCName()
    {
     return _CName;
         }
  }
```

- **CategoryHome.java**

```
package com.ejb_chen;

import java.rmi.*;
import javax.ejb.*;
import java.util.*;

public
interface CategoryHome
extends EJBHome
{
```

```
    Category create(String id, String name, String level) throws CreateException,
    RemoteException;

      //boolean login(String loginName, String pw) throws RemoteException;
      public Enumeration findByLevel(String level) throws FinderException, RemoteException;
      //public Enumeration findAll(String aa) throws FinderException, RemoteException;
      Category findByPrimaryKey(CategoryKey key) throws FinderException, RemoteException;
}
```

➢ CategoryKey.java

```
package com.ejb_chen;
import java.io.*;

public
class CategoryKey
implements Serializable
{
  // the names of the fields have to match the names of the corresponding fields in the bean
  private String _CID;
  public CategoryKey()
  {}
  public CategoryKey(String id)
  {
    _CID = id;
  }
```

```
    public String getCID()
    {
      return _CID;
    }
    // WARNING: be sure to implement an equals and a hashCode method for your PK class
    // one can get surprising results without these two methods

    public boolean equals(Object object)
    {
      if (this == object)
      {
        return true;
      }

      if (object instanceof CategoryKey)
      {
        return (_CID == ((CategoryKey)object)._CID);
      }

      return false;
    }

    public int hashCode()
    {
      return _CID.hashCode();
    }
}
```

9.3 XML File

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD EnterpriseJavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
 <description>Default Authentication</description>
 <display-name>Default Authentication</display-name>
 <enterprise-beans>
  <entity>
   <display-name>CategoryBean</display-name>
   <ejb-name>CategoryHome</ejb-name>
   <home>com.ejb_chen.CategoryHome</home>
   <remote>com.ejb_chen.Category</remote>
   <ejb-class>com.ejb_chen.CategoryBean</ejb-class>
   <prim-key-class>com.ejb_chen.CategoryKey</prim-key-class>
   <persistence-type>Container</persistence-type>
   <reentrant>False</reentrant>
     <cmp-field>
      <field-name>_CID</field-name>
     </cmp-field>
     <cmp-field>
      <field-name>_CName</field-name>
     </cmp-field>
                <cmp-field>
      <field-name>_Level</field-name>
     </cmp-field>
   <env-entry>
```

```
    <env-entry-name>ejipt.maxContexts</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>unspecified</env-entry-value>
  </env-entry>

  <!--    CMP properties    -->

  <env-entry>
    <env-entry-name>ejipt.createSQL</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>INSERT INTO Category (CID, CName, Level) VALUES (?, ?, ?)< env-entry-
value>
  </env-entry>
  <env-entry>
    <env-entry-name>ejipt.createSQL.source< env-entry-name>
    <env-entry-type>java.lang.String< env-entry-type>
    <env-entry-value>ejb_chen< env-entry-value>
  </env-entry>
  <env-entry>
    <env-entry-name>ejipt.createSQL.params</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>_CID, _CName, _Level</env-entry-value>
  </env-entry>
  <env-entry>
    <env-entry-name>ejipt.createSQL.paramTypes</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>VARCHAR, VARCHAR</env-entry-value>
  </env-entry>
```

```
<env-entry>
   <env-entry-name>ejipt.loadSQL</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>SELECT CName FROM Category WHERE CID = ?</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.loadSQL.source</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>ejb_chen</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.loadSQL.params</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>_CID</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.loadSQL.paramTypes</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>VARCHAR</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.loadSQL.fields</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>_CName</env-entry-value>
</env-entry>

<env-entry>
```

```
    <env-entry-name>ejipt.storeSQL</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>UPDATE Category SET CName = ? WHERE CID = ?</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.storeSQL.source< env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>ejb_chen</env-entry-value>
< env-entry>
<env-entry>
    <env-entry-name>ejipt.storeSQL.params</env-entry-name>
    <env-entry-type>java.lang.String< env-entry-type>
    <env-entry-value>_CName, _CID</env-entry-value>
< env-entry>
<env-entry>
    <env-entry-name>ejipt.storeSQL.paramTypes< env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>VARCHAR, VARCHAR< env-entry-value>
< env-entry>

<env-entry>
    <env-entry-name>ejipt.removeSQL</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>DELETE FROM Category WHERE CID = ?</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.removeSQL.source</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
```

```
    <env-entry-value>ejb_chen</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.removeSQL.params</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>_CID</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.removeSQL.paramTypes</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>VARCHAR</env-entry-value>
</env-entry>

<env-entry>
    <env-entry-name>ejipt.findByPrimaryKeySQL</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>SELECT CID FROM Category WHERE CID = ?</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.findByPrimaryKeySQL.source</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>ejb_chen</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.findByPrimaryKeySQL.fields</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>_CID</env-entry-value>
</env-entry>
```

```
<env-entry>
   <env-entry-name>ejipt.findByPrimaryKeySQL.params</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>_CID</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.findByPrimaryKeySQL.paramTypes</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>VARCHAR</env-entry-value>
</env-entry>

      <env-entry>
   <env-entry-name>ejipt.findByLevelSQL</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>SELECT CID FROM Category WHERE Level = ?</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.findByLevelSQL.source</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>ejb_chen</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.findByLevelSQL.fields</env-entry-name>
   <env-entry-type>java.lang.String</env-entry-type>
   <env-entry-value>_CID</env-entry-value>
</env-entry>
<env-entry>
   <env-entry-name>ejipt.findByLevelSQL.params</env-entry-name>
```

```
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>_Level</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.findByLevelSQL.paramTypes</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>VARCHAR</env-entry-value>
</env-entry>

        <env-entry>
    <env-entry-name>ejipt.loadSQL</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>SELECT CName FROM Category WHERE CID = ?</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.loadSQL.source</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>ejb_chen</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.loadSQL.fields</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>_CName</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.loadSQL.params</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>_CID</env-entry-value>
```

```
</env-entry>
<env-entry>
    <env-entry-name>ejipt.loadSQL.paramTypes< env-entry-name>
    <env-entry-type>java.lang.String< env-entry-type>
    <env-entry-value>VARCHAR< env-entry-value>
</env-entry>

<env-entry>
    <env-entry-name>ejipt.storeSQL< env-entry-name>
    <env-entry-type>java.lang.String< env-entry-type>
    <env-entry-value>UPDATE Category SET CName = ? WHERE CID = ?< env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.storeSQL.source< env-entry-name>
    <env-entry-type>java.lang.String< env-entry-type>
    <env-entry-value>ejb_chen< env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.storeSQL.params</env-entry-name>
    <env-entry-type>java.lang.String< env-entry-type>
    <env-entry-value>_CName, _CID</env-entry-value>
</env-entry>
<env-entry>
    <env-entry-name>ejipt.storeSQL.paramTypes</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>VARCHAR, VARCHAR</env-entry-value>
</env-entry>
</entity>
```

```
</enterprise-beans>
<assembly-descriptor>

 <security-role>
    <role-name>all</role-name>
 </security-role>

  <method-permission>
    <role-name>all</role-name>
    <method>
       <ejb-name>CategoryHome</ejb-name>
       <method-name>getCID< method-name>
    </method>
  < method-permission>
  <method-permission>
    <role-name>all< role-name>
    <method>
       <ejb-name>CategoryHome</ejb-name>
       <method-name>getCName< method-name>
    </method>
  < method-permission>
  <method-permission>
    <role-name>all</role-name>
    <method>
       <ejb-name>CategoryHome</ejb-name>
       <method-name>create</method-name>
    </method>
  </method-permission>
```

```
            <method-permission>
        <role-name>all</role-name>
        <method>
          <ejb-name>CategoryHome</ejb-name>
          <method-name>findByPrimaryKey</method-name>
        </method>
      </method-permission>
            <method-permission>
        <role-name>all</role-name>
        <method>
          <ejb-name>CategoryHome</ejb-name>
          <method-name>findByLevel</method-name>
        </method>
      </method-permission>

  </assembly-descriptor>
</ejb-jar>
```