iSeries

iDrink and Cola Connections SMB scenario overview

Experience Report

iSeries

iDrink and Cola Connections SMB scenario overview

Contents

Chapter 1. iDrink and Cola Connections SMB scenario overview

This report documents the IBM(R) eServer(TM) i5 Customer Solution Test team's experience of implementing a small to medium business (SMB) scenario using WebSphere(R) Application Server - Express and Lotus(R) Domino(R). In this scenario, WebSphere Application Server - Express for iSeries(TM) is used to establish an initial Web presence through dynamic Web pages, database access, and directory server access. A Lotus Domino application meanwhile provides internal dynamic Web pages for iDrink employees.

The iDrink SMB scenario simulates a beverage distribution company. This factitious company has less than two hundred employees and is considered a small to medium size business. The beverage distributor works with beverage suppliers to obtain the inventory that it sells. iDrink works with customers who purchase large quantities of beverages to sell in a variety of stores. The employees work with both the suppliers and customers to ensure that the business is a success. The iDrink company handles customer, supplier, and product information via Java(TM) servlets and JavaServer Pages (JSPs). Most of the customer, supplier, and product information is stored in a DB2 Universal Database(TM) with employee information and passwords, customer passwords, and supplier passwords maintained in an Lightweight Directory Access Protocol (LDAP) directory.

In addition, the iDrink employees have access to a classified ads Web site called Cola Connections that they can use to advertise items for sale. This application is implemented in Lotus Domino.

iDrink overview

The iDrink company allows employees, customers, and suppliers access to information appropriate to their role.

Scenario design decisions

The following describes the scenario design decisions that were initially made:

- Use WebSphere(R) Application Server - Express for application development as it provides a means to quickly establish a Web presence for customers, employees, and suppliers to interface.
- Use the eServer(TM) i5 LDAP directory for authentication for the WebSphere application to provide security.

iDrink application

The iDrink application consists of a WebSphere Application Server - Express environment using JavaServer Pages, JavaBeans(TM), and Java(TM) servlets to allow customers, employees, and suppliers to perform the following functions:

- Customers
  - Register or login
  - View product information
  - Order products
  - Update customer information
  - View order history
- Employees
  - Update supplier information
  - Order products from suppliers

  - Modify product pricing
  - Track customer orders
  - Track product inventory
  - Create, update, and view classified ads
- Suppliers
  - Update product information

Figure 1 shows the flow of the iDrink application.

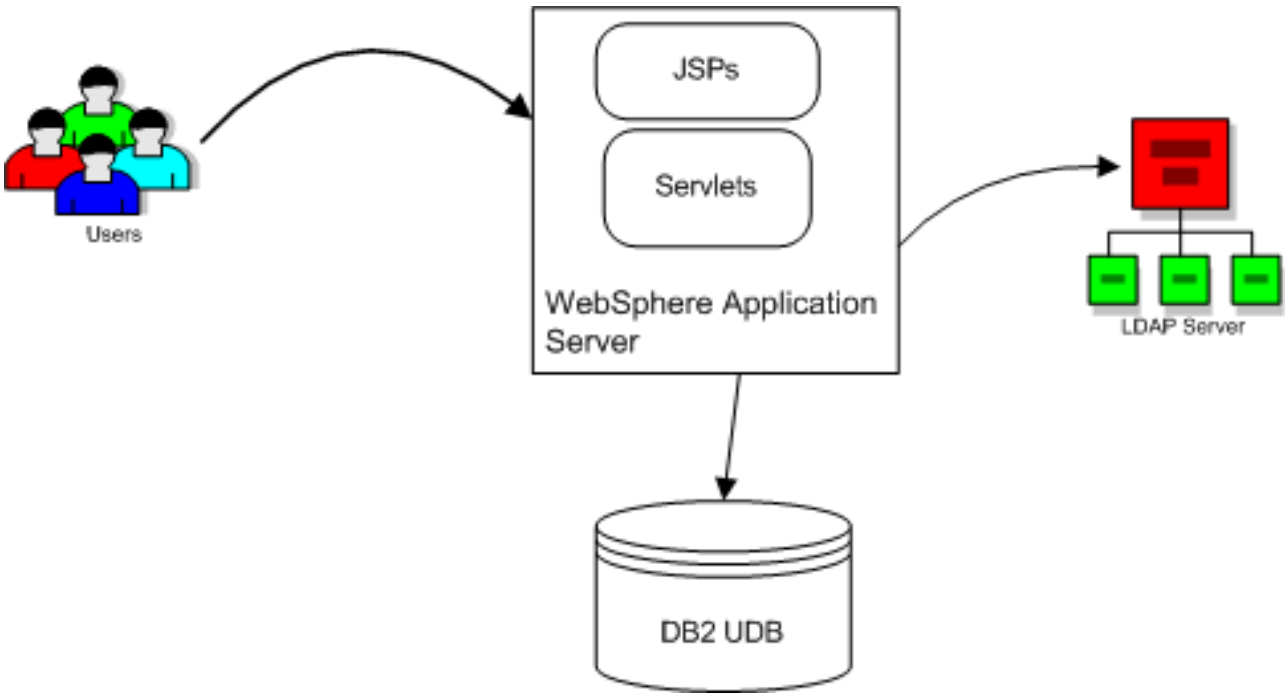

Figure 1: iDrink application flow

Cola Connections overview

The Cola Connections application provides employees with a location to list classified ads to other employees of Drink.

Application design decisions

The following describes the design decisions that were initially made:

- Use Lotus(R) Domino(R) for application development as it provides a means to establish a quick Web presence.
- Use the eServer(TM) i5 LDAP directory for authentication for the Lotus Domino application to provide security.

Cola Connections application

The Cola Connections application consists of a Lotus Domino database, agents, views, and forms that gives iDrink employees the ability to create, update, or view classified ads. Figure 2 shows the flow of the Cola Connections application.

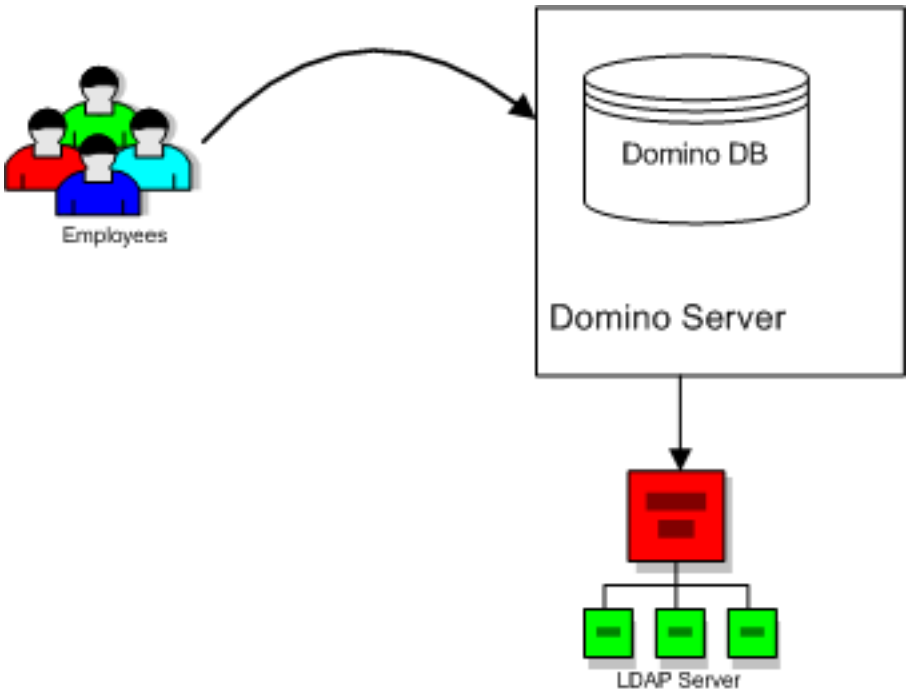

Figure 2: Cola Connections application flow

Security using Domino Directory Assistance

The Domino Directory provides support for securing application and data access. The security mechanism determines and verifies user identity and allows access to protected resources only by designated users. In Cola Connections, only employees are allowed to access the classified ads. Domino uses an access control model that stores entries in the Access Control List (ACL) for each database. Security is implemented by configuring Domino Directory Assistance to use the existing i5/OS(TM) LDAP directory. Details for setting up security can be found in the Chapter 3, "Lotus Domino environment," on page 33 section.

Environment overview

The iDrink scenario SMB environment setup is described in the following topics: run-time environment, system hardware, and the key products that were used.

Run time environment

The iDrink company is considered a small to medium size business, so they have invested in an affordable, entry-level eServer(TM) i5 designed for the demand of their challenges. This small server allows iDrink to run a variety of applications simultaneously providing security and availability plus the power and capacity to run their business applications and their e-business solution.

System hardware

One eServer i5 system is used in this scenario implementation. This eServer i5 system contains a 1-Way processor, 1 GB of memory, and 175 GB of main storage running i5/OS(TM) V5R3.

Key products

The key software products used in this scenario are:

- **WebSphere(R) Application Server - Express for iSeries(TM)** is a tightly integrated development tool and application server that provides an easily affordable entry point to e-business for companies creating dynamic Web sites. WebSphere Application Server - Express supports the specifications for JavaServer Pages (JSPs), Java(TM) servlets and Web services. WebSphere Application Server - Express allows the building of static and dynamic Web sites by accessing information in databases and performing simple updates while also providing the ability to create and use Web Services. It addresses the needs of midmarket companies by being a low cost, easy to use, out-of-the-box solution.

  Detailed information on IBM(R) WebSphere Application Server - Express for iSeries is available on the Web at: http://www.ibm.com/servers/eserver/iseries/software/websphere/wsappserver/express/indexexp51.html.
- **Lotus(R) Domino(R)** provides a foundation for collaboration and e-business, driving solutions from corporate messaging to Web based transactions and everything in between. This enterprise-class messaging and collaboration system is built to maximize productivity by unleashing the experience and expertise of individuals, teams, and extended communities.

  Detailed information on Lotus Domino for iSeries is available on the Web at: http://www.ibm.com/servers/eserver/iseries/domino/
- **Lightweight Directory Access Protocol (LDAP)** is a directory service protocol that uses TCP/IP. The LDAP directory service follows a client/server model. One or more LDAP servers contain the directory data. An LDAP client connects to an LDAP server and makes a request. The server responds with a reply, or with a pointer (a referral) to another LDAP server. Because LDAP is a directory service rather than a database, the information in an LDAP directory is usually describable, attribute-based information. LDAP generally reads the information in the directory much more often than it changes it. Updates are typically simple, all-or-nothing changes. Common uses of LDAP directories include:
  - online telephone directories
  - e-mail directories
  - authorization
  - authentication

  Detailed information on LDAP is available on the Web at: http://publib.boulder.ibm.com/pubs/html/as400/infocenter.htm
- **IBM HTTP Server powered by Apache for iSeries** is a Web server that provides tools to quickly and easily establish a Web presence and get started on the road to working the Web for business.

  Detailed information on HTTP Server powered by Apache for iSeries is available on the Web at: http://www.ibm.com/servers/eserver/iseries/software/http/
- **DB2 Universal Database(TM) (UDB) for iSeries** is an advanced, 64-bit Relational Database Management System that leverages the On-Demand features of IBM's eServer i5. A member of IBM's leading edge family of DB2(R) products, DB2 UDB for iSeries supports a broad range of applications and development environments at a lower cost of ownership, due to its unique autonomic computing (self-managing) features.

  Detailed information on DB2 UDB is available on the Web at: http://www.ibm.com/servers/eserver/iseries/db2

SMB discoveries

The following is a list of key discoveries that were uncovered while creating the iDrink SMB solution:

- If you decide to store information in session data for data communication between servlets and JSPs, make sure that the objects you store are as small as possible. Any objects stored in the session become a part of the Java(TM) heap and require memory usage. On a small box with little memory, there is not a lot of room to store objects. If you go beyond the memory allocation on the system, anything in memory will be paged to disk causing a decrease in overall system performance.

- Increase the size of the WebSphere(R) data source connection pool to handle the number of requests that are expected on the site. Also set the Aged Timeout value so that connections can be discarded if they are not being utilized allowing memory usage to remain low. To set these values in the WebSphere Administration Console:
  - Expand the plus next to **Resources** in the left hand frame, then click on the **JDBC Providers** link. In the main frame, select the **Server Scope** radio button then press the **Apply** button
  - In the main frame, click on the JDBC provider link used by the application. Towards the bottom of the JDBC Providers page under additional properties press the **Data Sources** link
  - In the main frame, click on the data source link being used by the application. Towards the bottom of the data source page click on the **Connection Pool** link
  - On the connection pool page set the minimum and maximum connections as well as the aged timeout
  - Click the **Apply** button, followed by the **Save** link in the upper left corner of the screen. On the next page, click the **Save** button
  - Restart WebSphere Application Server
- Increase the number of threads on the Web Container inside the WebSphere Application Server instance to match the maximum number of users. Also click to allow thread allocation beyond maximum thread size. To set these values in the WebSphere Administration Console:
  - Expand the plus next to **Servers** in the left hand frame, then click on the **Application Servers** link. In the main frame, click on the application server link where the application is running
  - On the application servers page, click on the**Web Container** link
  - On the web container page, click on the **Thread Pool** link
  - On this page set the minimum and maximum size as well as clicking the radio box for **Is Growable**
  - Click the **Apply** button, followed by the **Save** link in the upper left corner of the screen. On the next page, click the **Save** button
  - Restart WebSphere Application Server
- Increase the number of threads to process requests on the HTTP server to match the maximum number of users. To set these values in the IBM(R) tasks page:
  - Click on the **Manage** tab then click on the **HTTP Servers** tab
  - Select the HTTP server associated to the WebSphere Application Server from the drop down box
  - In the left hand frame select the **System Resources** link
  - In the main frame, click on the **Advanced** tab
  - Set the number of threads to process requests
  - Press the **Apply** button then press the **OK** button
  - Restart the HTTP server
- Increase the system-wide TCP send buffer and receive buffer sizes depending upon the nature of the applications running on a partition. To change these settings in a 5250 session:
  - **CHGTCPA**
  - Change the TCP receive buffer and send buffer sizes
  - Press **Enter**

Chapter 2. WebSphere Application Server - Express environment

IBM(R) WebSphere Application Server - Express for iSeries(TM), Version 5.1 is a combination of both development tools and an application server that provides an integrated package for Web-based applications. WebSphere Application Server - Express contains the following:

- An application server containing:
  - Servlet 2.3 and JSP 1.2 support
  - Embedded Web server
  - Web container
  - Web services support
  - XML and XSL support
  - JDBC 2.0 support
  - Connection pooling
  - Simple WebSphere Authentication Method (SWAM)
  - Servlet 2.3 and JSP 1.2 support
- IBM WebSphere Development Studio Client for iSeries, Version 5.1, a development tool containing:
  - Servlet 2.3 specification
  - JSP 1.2 specification
  - HTML
  - JavaScript(TM) (client-side and server-side)
  - Dynamic HTML (DHTML)
  - XML and XHTML
  - Web services use and creation
  - Team development using CVS
  - JDBC 2.0
  - Support for remote server configuration and operation
  - Customer tag libraries
  - Struts

The iDrink application is contained in an Enterprise Archive (EAR) file. An EAR file is a compressed JAR file that contains a J2EE application. J2EE applications contain elements such as servlets, JSPs, JavaBeans(TM), and XML configuration files. The EAR file can be created with WebSphere Development Studio Client, WebSphere Studio Application Developer, or WebSphere Studio Application Site Developer.

WebSphere(R) Application Server Overview

The WebSphere Application Server - Express environment was set up during the eServer(TM) i5 Customer Solution Test iDrink scenario work. The goal was to quickly establish a Web presence so that the iDrink company could provide an interface for its customers and suppliers to perform necessary business functions. The company also uses the WebSphere Application Server - Express environment to provide an interface for employees to perform their work. The iDrink interface was built using JavaServer Pages (JSPs), JavaBeans(TM), and servlets.

Application flow

There are a number of JSPs used within the iDrink application. These JSPs are listed and described in the User, Customer, Employee, and Supplier sections.

The following diagram illustrates the high-level flow within the iDrink application:

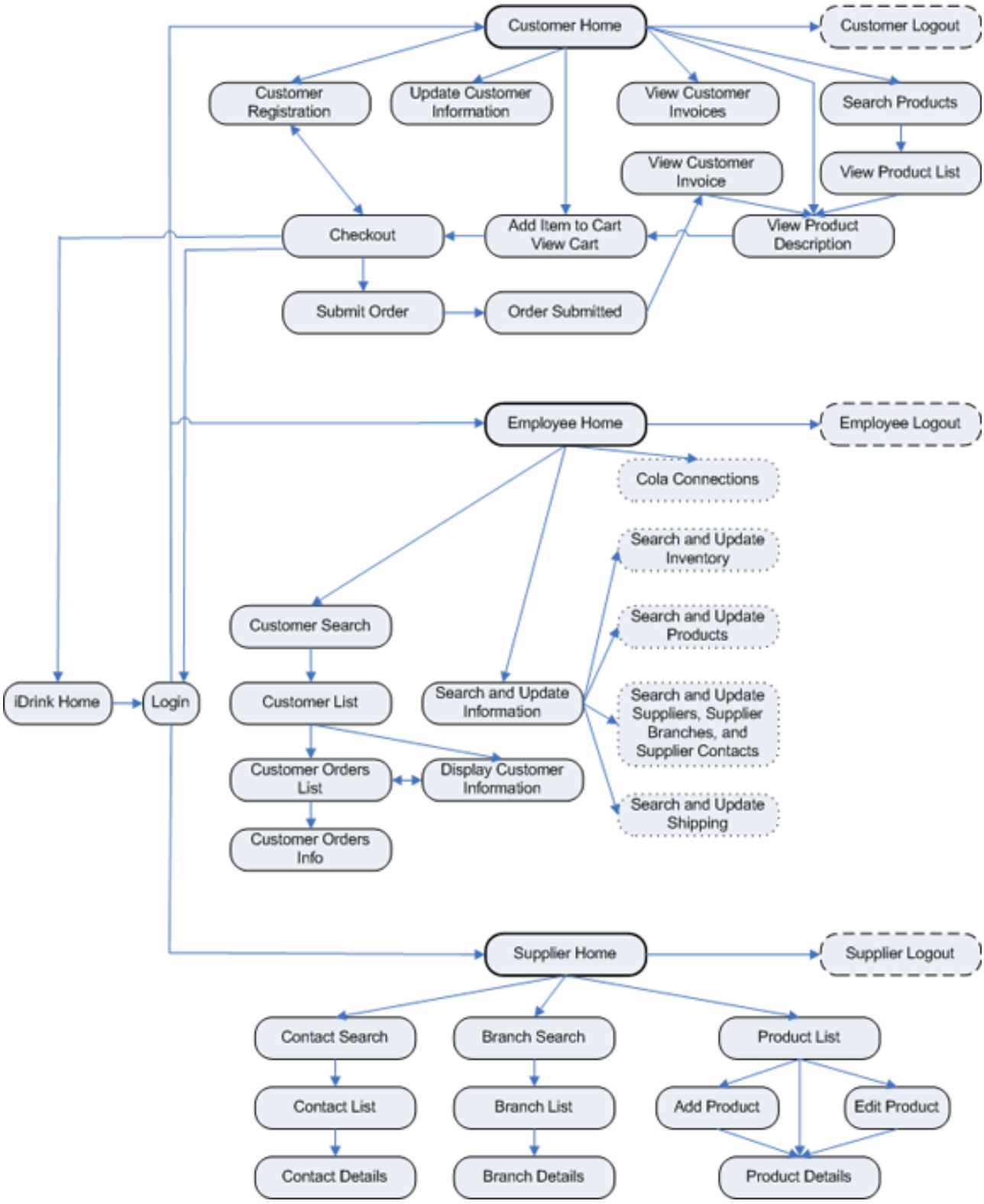

Figure 3: iDrink Application Flow

Development environment

The iDrink team chose IBM(R) WebSphere(R) Studio Application Developer (WSAD) version 5.0 to develop the servlets, JavaServer Pages (JSPs) and JavaBeans(TM) that together form the iDrink application. WSAD provides an Eclipse-based development environment that allows developers to easily create, compile and package Java(TM) 2 Platform, Enterprise Edition (J2EE) applications for deployment on a production system. Once an application is coded and compiled, it can be run in a test environment in WSAD or it can be deployed to an IBM WebSphere Application Server instance and configured via the WebSphere administration console. Another tool that can be used for development of Web applications is WebSphere Development Studio Client (WDSC).

Application details

The iDrink application is an enterprise application that is composed of Java(TM) servlets, JavaBeans(TM) and JavaServer Pages (JSPs). There are four servlets in the iDrink application. These include the UserServlet, the CustomerServlet, the EmployeeServlet and the SupplierServlet. The UserServlet is responsible for handling all login, logout and password change requests from all users. The CustomerServlet is responsible for handling all requests from customers and from all users who have not logged into iDrink. The EmployeeServlet handles all requests from iDrink employees who have logged in and the SupplierServlet handles all requests from iDrink suppliers who have logged in.

When each iDrink servlet receives a request from a user, it parses the contents of the request to determine which action to take. For example, the servlet may need to access or create a Lightweight Directory Access Protocol (LDAP) entry or a record in a database file. If data has been retrieved, an appropriate JavaBean is created and the data is stored within the JavaBean. The JavaBean is then added to the request object or the session object, and the appropriate JSP is displayed. Figure 4 illustrates the way these components work together.

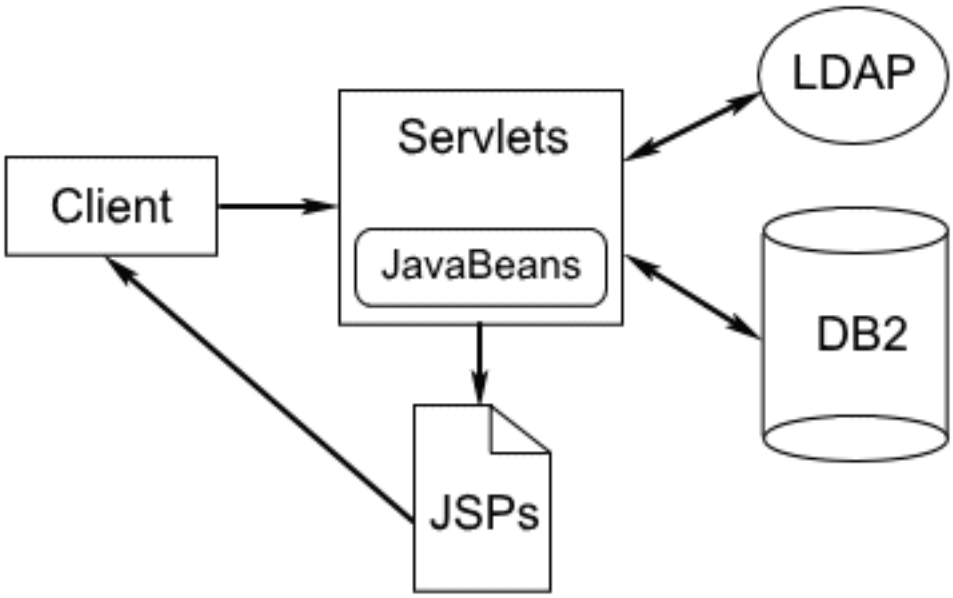

Figure 4: iDrink application components

This section contains detailed information about the methods in the UserServlet, the CustomerServlet, the EmployeeServlet and the SupplierServlet, in addition to the JSPs that are called by these servlets. This section also contains a description of the application design points that the servlets follow, as well as the steps that are taken to install an enterprise application, such as iDrink, to an IBM(R) WebSphere(R) Application Server - Express server.

Application design points

Before designing the application, consideration is taken into account on how the LDAP server and database schema is designed. This section contains the information on these considerations as well as the JavaBeans(TM) and Java(TM) servlet designs for the iDrink scenario.

Lightweight Directory Access Protocol (LDAP) design

A LDAP directory is used within the iDrink and Cola Connections applications as an authentication mechanism. Figure 5 shows how the LDAP directory is configured.

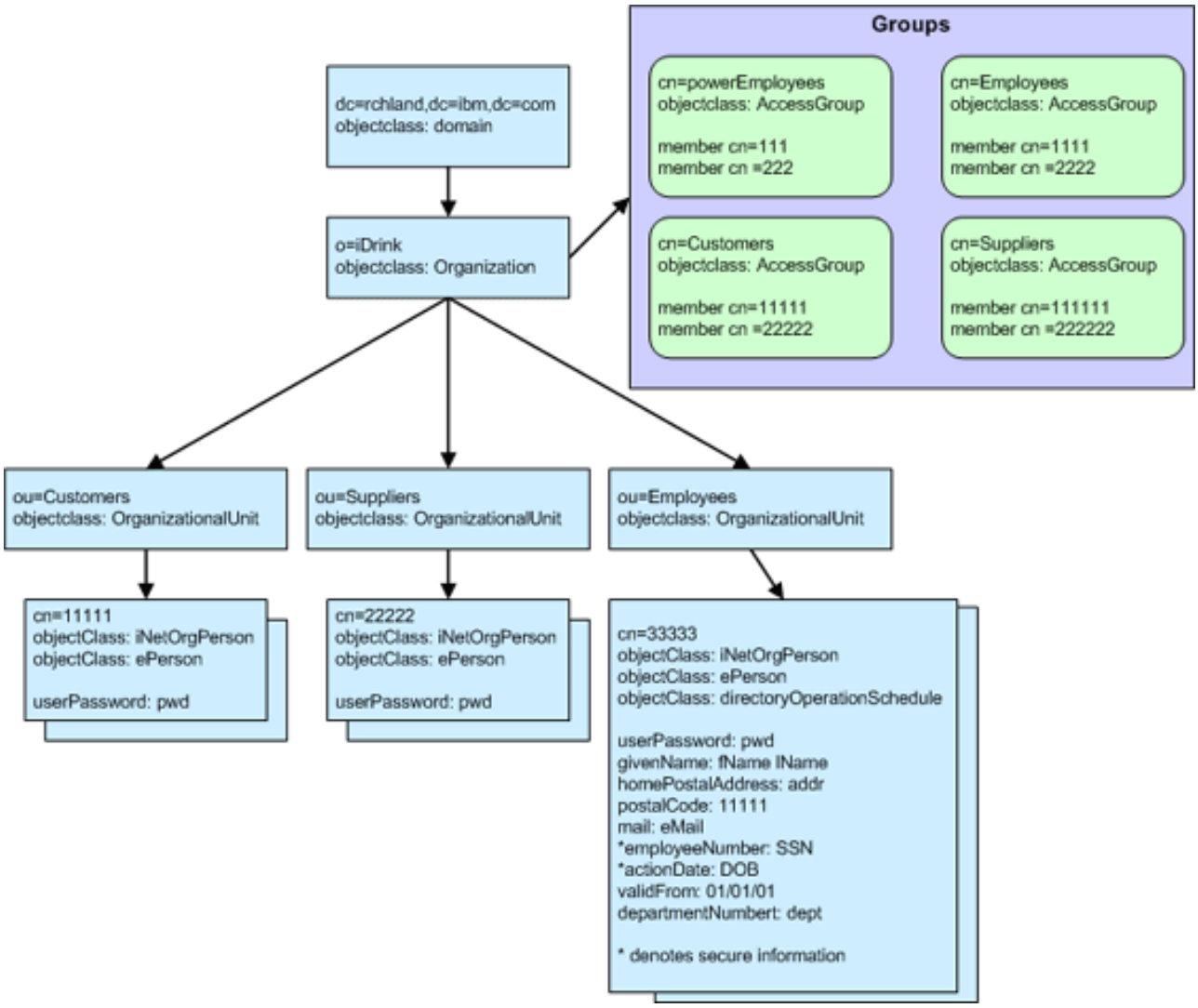

Figure 5: LDAP directory

When configuring the LDAP directory, the following choices for object classes and ACLs are used:

- Customers and Suppliers use the object classes inetOrgPerson and ePerson. All customers are members of a group called cn=customers and all suppliers are members of a group called cn=suppliers.
- Employees use the object classes inetOrgPerson and ePerson. All employees are members of a group called cn=employees. A handful of employees are members of a group called cn=powerEmployees. The powerEmployees group is allowed to do certain actions to the LDAP entries that regular employees are not allowed to do.
- Acess Control List (ACL) filters were added to the employee records to secure the data contained in the actionDate (DateOfBirth) and the employeeNumber (Social Security Number). The ACL filters that are used allow employees to read the record, however, only powerEmployees can view the employeeNumber and actionDate values.
  - aclentry: group:cn=employees,o=iDrink,dc=*domainName*,dc=*domainSuffix*:normal:grant:rsc
  - aclentry: group:cn=employees,o=iDrink,dc=*domainName*,dc=*domainSuffix:*
    at.employeeNumber:deny:rsc:at.actionDate:deny:rsc
  - aclentry: group:cn=powerEmployees,o=iDrink,dc=*domainName*,dc=*domainSuffix.*:
    at.employeeNumber:grant:rsc:at.actionDate:grant:rsc
- When populating the LDAP direcotry, all date fields have a format of YYYYMMDDHHMMSS[.|,fraction][(+|-HHMM)|Z]

Database design

The following diagram illustrates iDrink's database design:

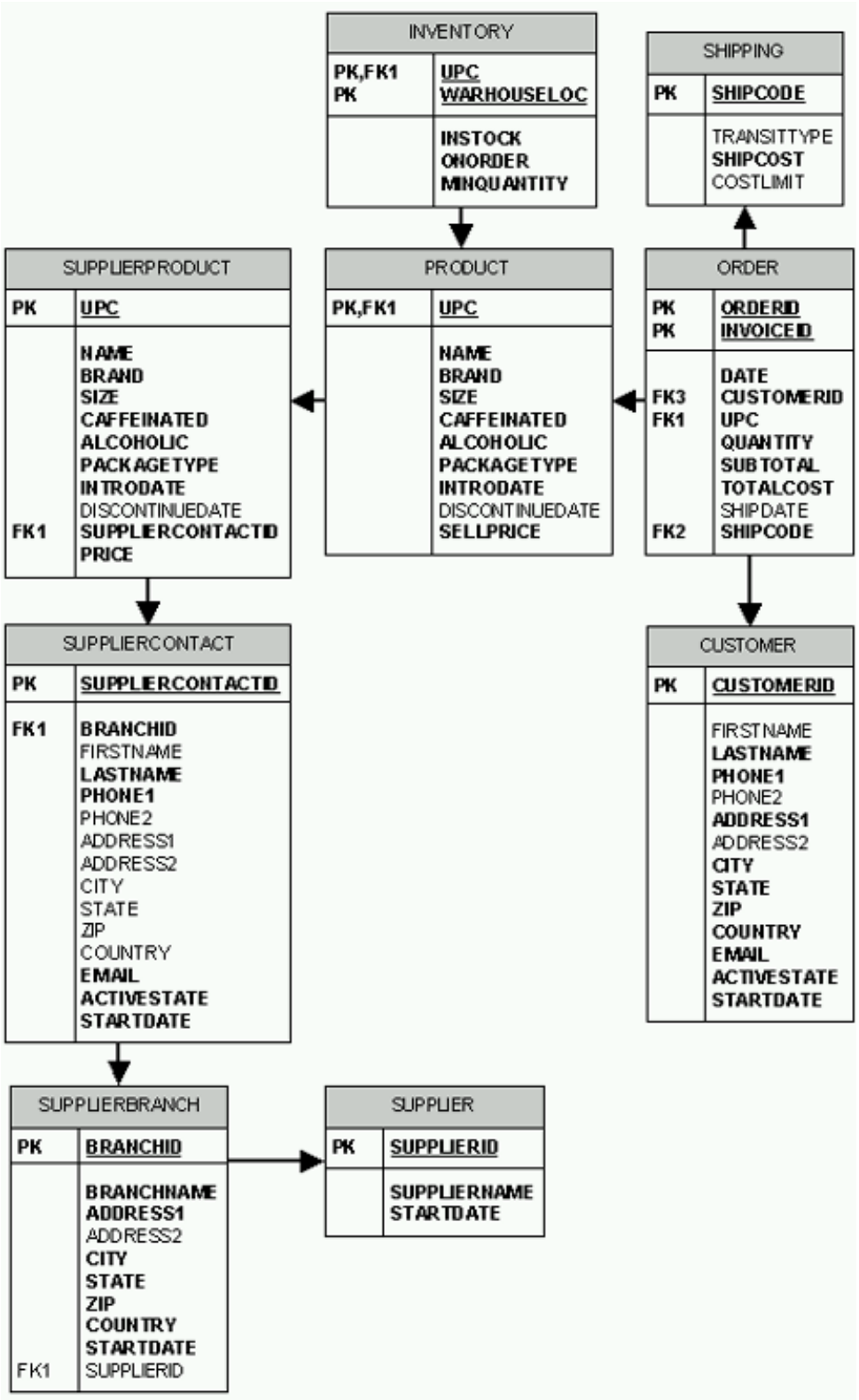

Figure 6: iDrink database design

The following is a brief summary of each table:

- SUPPLIER: Contains information about iDrink's suppliers.

- SUPPLIERBRANCH: Contains information about iDrink's supplier branches.
- SUPPLIERCONTACT: Contains information about iDrink's supplier contacts.
- SUPPLIERPRODUCT: Contains information about products supplied by iDrink's suppliers. Some of the products in this database are not sold by iDrink.
- PRODUCT: Contains information about products supplied by iDrink. The values in the PRODUCT table are a subset of the values in the SUPPLIERPRODUCT table.
- INVENTORY: Contains information regarding the quantity of products that iDrink currently has, and where these products are located.
- CUSTOMER: Contains customers that purchase products from iDrink.
- ORDER: Contains information for orders placed by iDrink customers.
- SHIPPING: Contains the shipping types used by iDrink to ship orders.

JavaBean Design

JavaBeans were created to store data retrieved by servlets. In the iDrink application, the following JavaBeans were created:

- **CustomerBean** - The CustomerBean stores information from the CUSTOMER database.
- **CustomerInvoiceBean** - The CustomerInvoiceBean stores information from the ORDER database.
- **InventoryBean** - The InventoryBean stores information from the INVENTORY database.
- **InvoiceItemBean** The InvoiceItemBean stores information pertaining to a single item in a particular order in the ORDER database.
- **ProductBean** - The ProductBean stores information from the PRODUCT database.
- **ShippingBean** - The ShippingBean stores information from the SHIPPING database.
- **SupplierBean** - The SupplierBean stores information from the SUPPLIER database.
- **SupplierBranchBean** - The SupplierBranchBean stores information from the SUPPLIERBRANCH database.
- **SupplierContactBean** - The SupplierContactBean stores information from the SUPPLIERCONTACT database.
- **SupplierProductBean** - The SupplierProductBean stores information from the SUPPLIERPRODUCT database.
- **UserBean** - The UserBean stores user information found in LDAP.

Servlet Design

Four servlets were used in the iDrink application. The following briefly describes each servlet:

- **CustomerServlet** - The CustomerServlet contains methods used in the customer application flow.
- **EmployeeServlet** - The EmployeeServlet contains methods used in the employee application flow. Some methods are also used in the supplier application flow.
- **SupplierServlet** - The SupplierServlet contains methods used in the supplier application flow.
- **UserServlet** - The UserServlet contains methods used throughout the entire application flow to ensure that only users with the proper authority can logon and access the appropriate pages (for example, it prevents customers from viewing the employee pages).

UserServlet

The iDrink UserServlet provides the functionality that any iDrink user needs to login to iDrink, to change their password and to logout of iDrink. The servlet utilizes an LDAP directory to accomplish these tasks.

Design Points

The iDrink application needed a mechanism to determine if a user was logged in. If a user was logged in, the application also needed to know what type of user that user was. To accomplish these tasks, the application relies on the UserBean class. A UserBean object contains the user ID of the user and an integer value that is assigned one of three constant values that are associated with each of the iDrink user types. When a user logs into iDrink, the UserServlet creates a new UserBean object and sets the user ID and user type values, then adds the object to the session. The object remains in the session, until the user logs out. Thus, once a user logs in, any servlet or JSP that is accessed by the user can easily determine the user's user ID and user type.

Methods

The following is a list of methods that are found in the UserServlet. The doPost method receives all incoming requests from the customer, and calls the appropriate method from this list of methods.

- **assertUserType** - The assertUserType method retrieves the UserBean from the request. If the UserBean is null or is not of a valid user type, the method displays the UserLogin JSP.
- **changePassword** - The changePassword method accepts a user ID, an old password and a new password. The LDAP entry for the specified user ID is found, and the userPassword attribute in that entry is updated with the new password. If this is successful, the method returns a true boolean value. If any exceptions are thrown during this process, the method returns a false boolean value.
- **handlePasswdRequest** - The handlePasswdRequest method verifies the new password matches the confirmation password, then calls the changePassword method to change the existing password. The method then displays the UserPasswdResults JSP.
- **loginUser** - The loginUser method retrieves the user ID and the password from the request, and calls the verifyUser method to verify that the specified password is correct for the user ID. If the password is correct, a UserBean is created and added to the session object. The appropriate JSP is then displayed, based on the type of user that is logging on. A customer would view the iDrinkHome JSP, an employee would view the EmployeeHome JSP and a supplier would view the SupplierHome JSP.
- **logoutUser** - The logoutUser method removes the UserBean object from the session object, then displays the iDrinkHome JSP.
- **verifyUser** - The verifyUser method accepts a user ID and password, and verifies that the password is correct for the specified user ID. If the password is correct, the method returns an integer which represents the type of iDrink user. If the password is incorrect, the method returns zero.

JavaServer Pages

The following is a list of JSPs that are called by the UserServlet:

- **CustomerProceedToCheckout** - The CustomerProceedToCheckout JSP displays a link that will take a customer to the CustomerCheckout JSP. This JSP is used as an intermediate step when a customer logs in during the check out process.
- **EmployeeHome** - The EmployeeHome JSP is displayed after an employee logs in.
- **iDrinkHome -** The iDrinkHome JSP is displayed after a customer logs in. It is also displayed after a user logs out.
- **SupplierHome** - The SupplierHome JSP is displayed after a supplier logs in.
- **UserLogin -** The UserLogin JSP is displayed at the start of the login process. It displays text fields for a user ID and a password.
- **UserPasswdResults** - The UserPasswdResults page is displayed after a user changes their password. It informs the user that their password change request has succeeded or failed.

CustomerServlet

The iDrink CustomerServlet provides the functionality that an iDrink customer needs in order to do business with iDrink. It gives a customer the ability to register with iDrink, update their customer information, search for products, view product descriptions, add items to their shopping cart, place orders, and view their order history. The CustomerServlet and the JavaServer Pages (JSPs) that it calls work with the UserServlet to provide functionality that allows a user to login and logout of iDrink and to change their password. By default, the CustomerServlet handles all of the requests of an iDrink user, until they log in as an iDrink employee or as an iDrink supplier. If an iDrink user logs in as an iDrink customer, the CustomerServlet continues to handle the requests from that user.

Design Points

During a customer's shopping experience with iDrink, the CustomerServlet uses an iDrinkCart object to store the items that a customer adds to their shopping cart. The iDrinkCart object contains a java.util.Hashtable, which stores iDrinkCartItems that have been added to the iDrinkCart. Each iDrinkCartItem contains information about the item in the cart, which includes the UPC number, quantity and price. The key value for each iDrinkCartItem in the hashtable is that item's UPC number, which allows for easy retrieval of items from the hashtable.

When a new customer registers with iDrink, a new, unique customer ID must be generated and assigned to the new customer's account. There is only one record in the customer table for each customer, so an identity column is used to generate this customer ID value. An identity column provides an easy way of automatically generating an unique, primary key value for every row in a table. This eliminates the concurrency and performance problems that can occur when an application must generate it's own unique values, which are usually based on the values that are already in the table. Since the customer ID is an identity column in the customer table, the CustomerServlet does not have to try to determine what the next customer ID should be. It simply lets the functionality of the identity column handle this work automatically.

When a customer places an order with iDrink, a new, unique invoice ID must be generated and assigned to that order. When the CustomerServlet receives the order information from the customer, it inserts one record into the order table for each unique item in the order. Each of these records must contain the same invoice ID. Thus, there can be any number of records in the order table that have the same invoice ID. Because of this, an identity column could not be used to generate an unique invoice ID, since an identity column would generate an unique value for each item in the order, instead of one unique value for the entire order itself. To generate an unique invoice ID, the CustomerServlet relies on a DB2(R) sequence object. A DB2 sequence object is an object that generates sequential values. Whenever a value is requested from a sequence object, it returns the next value in a sequence that was defined when the sequence object was created. The sequence defined in the invoice ID sequence object is simply a sequence of integers incremented by one, in ascending order. When the CustomerServlet places an order for a customer, it accesses the sequence object to retrieve the next invoice ID. This value is then used in all SQL insert statements that are executed for the order that is being placed.

A customer is not forced to login before they use the iDrink website. Thus, it was necessary to add logic to the CustomerServlet, the UserServlet and various JSPs to handle the case in which a customer adds items to their cart and does not login or register until they begin the checkout process. If a customer has not logged in before they press the checkout button, an additional parameter is sent in the HTTP request to the CustomerServlet (for customer registration) or to the UserServlet (for user login). If either of these servlets finds this parameter in a registration or login request, the servlet will process the request, then redirect the user to an intermediate checkout page that displays the customer ID and a link to continue with the checkout process. If a customer initiates the login or registration process from any other point in the iDrink website, they are redirected to a welcome screen, once the registration or login request successfully completes. In this case, the intermediate page is not needed, and is not displayed.

Application flow

The iDrink customer interface does not force a customer down a predetermined path. Thus, a customer can easily jump from one task to another at any time. As a result of this flexibility, it is would be very difficult to illustrate all possible potential paths that a customer could take through the customer interface while shopping at iDrink. Figure 7 illustrates the paths most commonly used by iDrink customers.

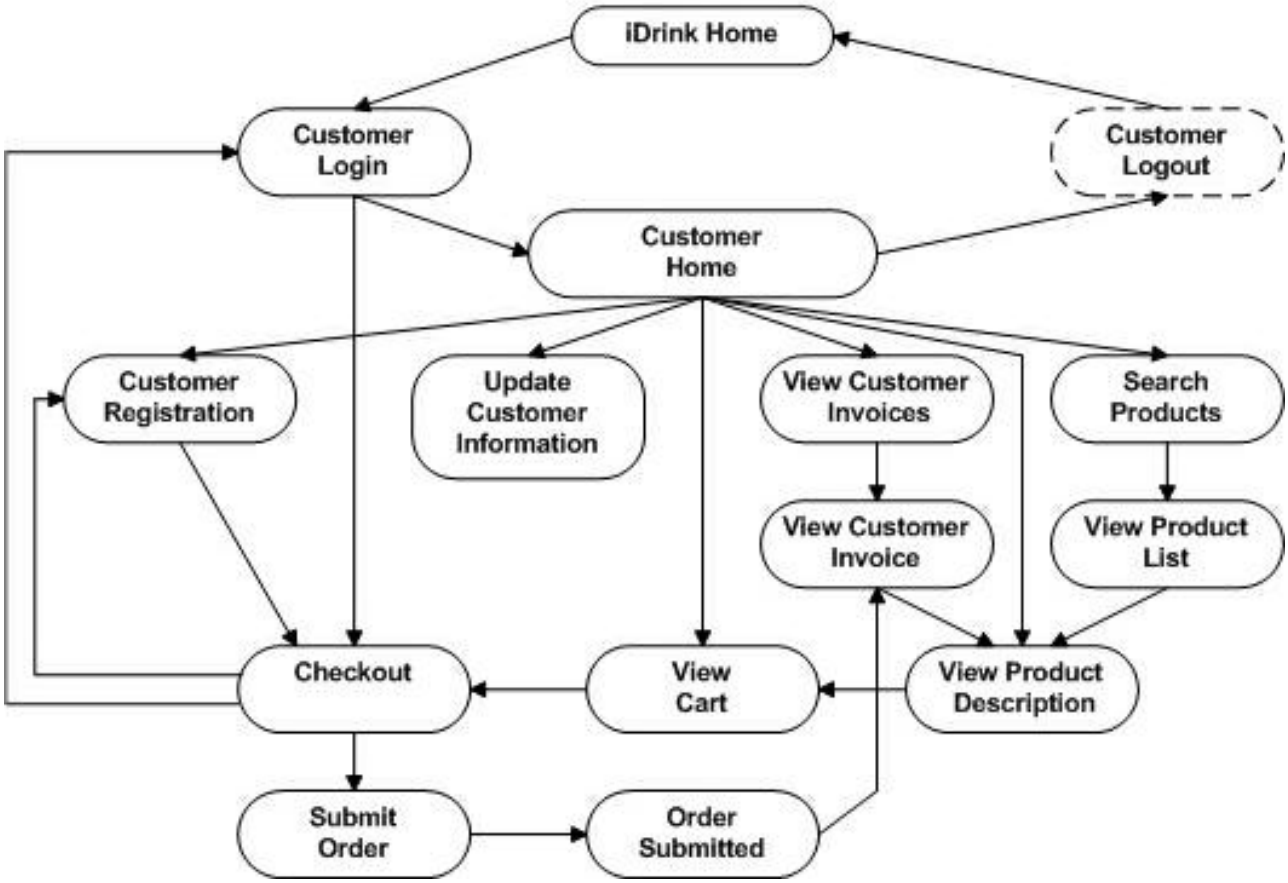

Figure 7: iDrink customer application flow

Methods

The following is a list of methods that are found in the CustomerServlet. The doPost method receives all incoming requests from the customer, and calls the appropriate method from this list of methods.

- **addItemToCart** - The addItemToCart method adds a specified quantity of a specified product to the customer's iDrinkCart. The updated cart contents are displayed in the CustomerViewCart JSP.
- **checkout** - The checkout method retrieves the valid shipping codes for a customer's order, based on the value of the items in their iDrinkCart. These shipping codes are displayed for the customer's selection in the CustomerCheckout JSP.
- **displayMatchingProducts** - The displayMatchingProducts method retrieves basic information about products that match search criteria specified by a customer. This information is displayed in the CustomerMatchingProducts JSP.
- **getNextInvoiceID** - The getNextInvoiceID method returns the next invoice ID, which is supplied by a DB2 sequence object.
- **registerNewCustomer** - The registerNewCustomer method inserts a new record in the iDrink customer table. This record contains the customer's information. It also creates a new LDAP entry for the customer. This entry contains the customer's password.

- **submitOrder** - The submitOrder method retrieves the new invoice ID from the getNextInvoiceID method. It then inserts one record into the iDrink order table for each item in the customer's iDrinkCart.
- **updateCustomer** - The updateCustomer method updates the customer information of the current customer with values provided by the customer.
- **updateItemsInCart** - The updateItemsInCart method updates the quantity of each item in the customer's iDrinkCart, based on values provided by the customer. The updated contents are displayed in the CustomerViewCart JSP.
- **viewInvoices** - The viewInvoices method retrieves basic information about each order that has been placed by the current customer. This information is displayed in the CustomerInvoices JSP.
- **viewInvoiceInformation** - The viewInvoiceInformation method retrieves detailed information about a specified customer order. This information is displayed in the CustomerInvoiceInfo JSP.
- **viewProduct** - The viewProduct method retrieves detailed information about a product with a specified UPC. This information is displayed in the CustomerProductInfo JSP.
- **viewUpdateCustomerScreen** - The viewUpdateCustomerScreen method retrieves the customer information for the current customer. This information is displayed in the CustomerUpdate JSP.

JavaServer Pages

The following is a list of JSPs that are called by the CustomerServlet:

- **CustomerCheckout -** The CustomerCheckout JSP displays a list of available shipping options that a customer may select for their order.
- **CustomerDisplayCart** - The CustomerDisplayCart JSP displays the contents of the customer's iDrinkCart.
- **CustomerInvoiceInfo** - The CustomerInvoiceInfo JSP displays detailed information about a specific customer order.
- **CustomerInvoices** - The CustomerInvoices JSP displays a list of all orders that have been placed by a customer.
- **CustomerMatchingProdcuts** - The CustomerMatchingProducts JSP displays all products that match the search criteria specified by a customer.
- **CustomerOrderSubmitted** - The CustomerOrderSubmitted JSP informs a user that their order has been placed, and displays the invoice number.
- **CustomerProceedToCheckout** - The CustomerProceedToCheckout JSP displays a link that will take a customer to the CustomerCheckout JSP. This JSP is used as an intermediate step when a customer registers or logs in during the check out process.
- **CustomerProductInfo** - The CustomerProductInfo JSP displays detailed information about a specific iDrink item.
- **CustomerRegistration** - The CustomerRegistration JSP provides input fields which allow a new customer to enter and submit their customer information.
- **CustomerSearchProducts** - The CustomerSearchProducts JSP provides input fields which allow a customer to enter and submit a set of product search criteria.
- **CustomerUpdate** - The CustomerUpdate JSP displays current customer information in text fields, which allow a user to modify and submit updated customer information.

EmployeeServlet

The iDrink EmployeeServlet provides the functionality that an iDrink employee needs in order to work with all entities of the company, including customers, suppliers, orders, products, and inventory. It gives employees the ability to:

- view customer orders
- add, view, update, and delete products that iDrink offers

- add, view, update, and delete products from inventory
- create, view, update, and delete suppliers, supplier branches, and supplier contacts
- create, view, update, and delete shipping types
- access the Cola Connections site

The EmployeeServlet and the JSPs that it calls work with the UserServlet to provide functionality to ensure that only employees can log on to and access the pages associated with the EmployeeServlet.

Design Points

- Along with using identity columns in the design of the customer application flow, identity columns are used in the employee application flow as well. Identity columns are used in these tables to allow for easy creation for primary key values:
  - SHIPPING
  - SUPPLIER
  - SUPPLIERBRANCH
  - SUPPLIERCONTACT
- Upon comparison of the employee and supplier application flows, it was discovered that these actions were similar:
  - searching and viewing supplier information
  - searching and viewing supplier branch information
  - searching and viewing supplier contact information

  Because these actions were identical, it was decided to only write one set of methods for these actions in the EmployeeServlet instead of writing methods in both the EmployeeServlet and the SupplierServlet.

  Also, only one set of JSP pages was created to search and view the information. Because only one set of JSPs was used, it was important to specify that both suppliers and employees were allowed access to these pages. This was done by adding the supplier check condition to the search and view JSPs, as this code demonstrates:

```
UserServlet.assertUserType
   (request, response, UserBean.USER_TYPE_SUPPLIER | UserBean.USER_TYPE_EMPLOYEE, "UserLogin.jsp");
```

  Since the employee application flow required links to update and delete supplier, supplier branch, and supplier contact information from these pages, it was important to check the user type before displaying these links. If the user was of type supplier, the links to update and delete were not displayed. This JSP code gives an example:

```
<%
  if(userbean.getUserType() == UserBean.USER_TYPE_EMPLOYEE)
  {
  %>

  <A HREF="EmployeeServlet?action=viewSupplierContactToUpdate&contactID=<%=
     supplierContactBean.getContactID()%>">Update Supplier Contact</A>
  <br /><BR>
  <A HREF="EmployeeServlet?action=viewSupplierContactToDelete&contactID=<%=
     supplierContactBean.getContactID()%>">Delete Supplier Contact</A>
  <br /><BR>

  <%
  }
```

  When an employee deletes a supplier, supplier contact, or inventory, that entity is deleted from the database. However, deletion of products and ship codes have a different meaning. Since the information needs to be left in the database for historical reasons, the items are left in the database, and a particular field is set to represent that the item is deleted.
  - To delete a product, the DISCONTINUEDATE is set. (A product with a null DISCONTINUEDATE means the product has not been deleted).

– To delete a ship code, the COSTLIMIT is set to 0. (A ship code with a COSTLIMIT greater than 0 is an active ship code that can be used for orders).

Also, certain items have conditions that must be met in order to delete the item:

– To delete a supplier, the supplier must not have any supplier branches associated with the supplier.
– To delete a supplier branch, the supplier branch must not have any supplier contacts associated with the supplier branch.
– To delete a supplier contact, the supplier contact must not have any supplier products associated with the supplier contact and the supplier contact must not be in an active state.

- In order for iDrink employees to add a new product to their customer offerings, the employee must choose a product from the SUPPLIERPRODUCT table that is not already in the PRODUCT table. Recall that the SUPPLIERPRODUCT table contains all products that are offered by iDrink's suppliers. The PRODUCT table contains products that iDrink offers to its customers. The products in the PRODUCT table are a subset of the products in the SUPPLIERPRODUCT table. In order to display the products in the SUPPLIERPROUDCT table that are not in the PRODUCT table, a subselect was performed on the SUPPLIERPRODUCT table, as the following example demonstrates:

```
SELECT *
FROM IDRINK.SUPPLIERPRODUCT
WHERE UPC NOT IN (SELECT UPC FROM
                  IDRINK.PRODUCT)
```

- When updating existing inventory, there were several options that could be implemented:
  – Overwriting the existing inventory with a new amount
  – Adding or subtracting an amount from the existing inventory

  After much consideration, it was decided to implement the first option: overwriting the existing inventory with a new amount. This provided the simplest option to implement as well as to use by employees.
- When working with shipping, it is beneficial for the employees to view both the minimum and maximum cost limits for an order for each ship code. However, in the SHIPPING table, only the maximum cost limit for the ship code is stored. The minimum cost limit is implied by the maximum cost limit for the next lowest cost limit for the ship code's shipping type (shipping types currently used by iDrink include GROUND, NEXT DAY AIR, and 2ND DAY AIR). In order for the JSPs to easily access the minimum cost limit information, a variable called costLimitMin was created in the ShippingBean to hold this information.

Application flow

The iDrink employee interface does not force employees down a predetermined path. Thus, an employee can easily jump from one task to another at any time. As a result of this flexibility, it is would be very difficult to illustrate all possible potential paths that an employee could take through the employee interface. Figure 8 illustrates the basic application flow available to iDrink employees.

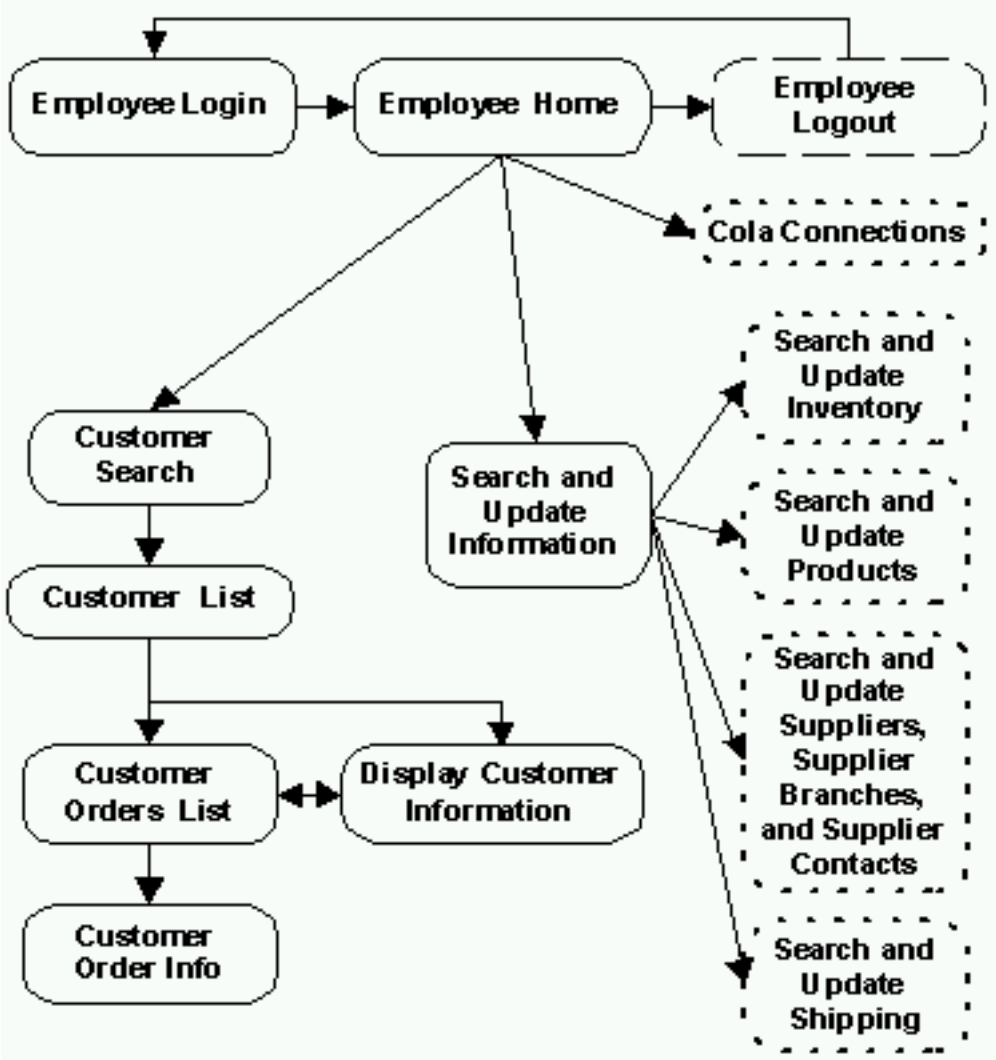

Figure 8: iDrink employee application flow

The application flow of these items are similar:

- Search and Update Inventory
- Search and Update Products
- Search and Update Suppliers, Supplier Branches, and Supplier Contacts
- Search and Update Shipping

For example, figure 9 illustrates the application flow for Search and Update Products:

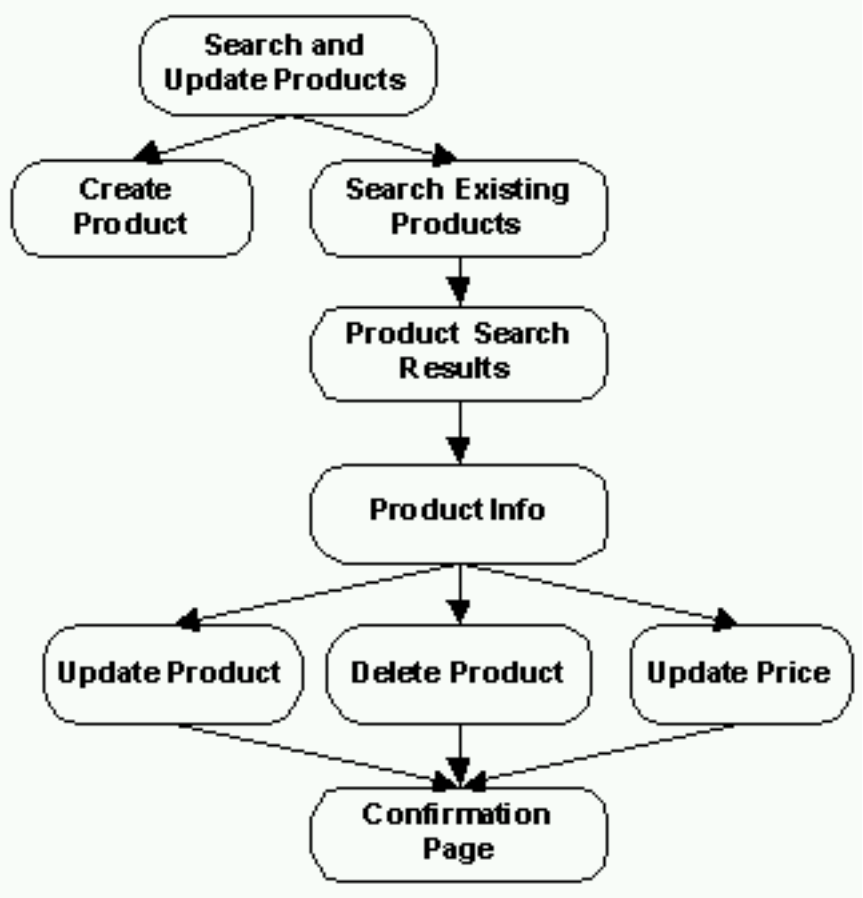

Figure 9: Search and update products application flow

With a few minor exceptions, the same application flow is used for Inventory, Shipping, Suppliers, Supplier Branches, and Supplier Contacts.

The Cola Connections application flow is described in the Lotus(R) Domino(R) Environment section.

Methods

The following tables contain the methods that are found in the EmployeeServlet. Since most iDrink entities have methods that perform similar actions, the methods are categorized by action.

The doPost method receives all incoming requests from the customer, and calls the appropriate method from this list of methods.

Entity	Method Name	Tables Used	What is called after the method completes
Inventory	createInventory	INVENTORY	Confirm.jsp
Product	createProduct	PRODUCT, SUPPLIER PRODUCT	Confirm.jsp
Shipping	createShipping	SHIPPING	Confirm.jsp
Supplier	createSupplier	SUPPLIER	Confirm.jsp
Supplier Branch	createSupplierBranch	SUPPLIER, SUPPLIERBRANCH	Confirm.jsp

Supplier Contact	createSupplierContact	SUPPLIER, SUPPLIERBRANCH, SUPPLIERCONTACT	Confirm.jsp

Table 1: Methods to create a new object and insert the object into the appropriate table.

Entity	Method Name	Tables Used	What is called after the method completes
Inventory	deleteInventory	INVENTORY	Confirm.jsp
Product	deleteProduct	PRODUCT	Confirm.jsp
Shipping	deleteShipping	SHIPPING	Confirm.jsp
Supplier	deleteSupplier	SUPPLIER	Confirm.jsp
Supplier Branch	deleteSupplierBranch	SUPPLIER, SUPPLIERBRANCH	Confirm.jsp
Supplier Contact	deleteSupplierContact	SUPPLIER, SUPPLIERBRANCH, SUPPLIERCONTACT	Confirm.jsp

Table 2: Methods to allow employees to remove objects. All methods except deleteProduct and deleteShipping will delete the object from the database, provided the appropriate conditions are met.

Entity	Method Name	Tables Used	What is called after the method completes
Customer	searchCustomerOrders	ORDER	EmployeeCustomerOrdersList.jsp
Customer	searchCustomers	CUSTOMER	EmployeeCustomerList.jsp
Inventory	searchInventory	INVENTORY, PRODUCT	InventorySearchResults.jsp
Product	searchProduct	PRODUCT	ProductSearchResults.jsp
Shipping	searchShipping	SHIPPING	ShippingSearchResults.jsp
Supplier	searchSupplier	SUPPLIER	SupplierSearchResults.jsp
Supplier Branch	searchSupplierBranch	SUPPLIER, SUPPLIERBRANCH	SupplierBranchSearchResults.jsp
Supplier Contact	searchSupplierContact	SUPPLIER, SUPPLIERBRANCH, SUPPLIERCONTACT	SupplierContactSearchResults.jsp
Supplier Product	searchSupplierProduct	PRODUCT, SUPPLIERCONTACT, SUPPLIERPRODUCT	ProductSearchResults.jsp

Table 3: Methods to allow an employee to search for an object.

Entity	Method Name	Tables Used	What is called after the method completes
Inventory	updateInventory	INVENTORY	Confirm.jsp
Product	updatePrice	PRODUCT	Confirm.jsp
Product	updateProduct	PRODUCT	Confirm.jsp
Shipping	updateShipping	SHIPPING	Confirm.jsp
Supplier	updateSupplier	SUPPLIER	Confirm.jsp

Supplier Branch	updateSupplierBranch	SUPPLIER, SUPPLIERBRANCH	Confirm.jsp
Supplier Contact	updateSupplierContact	SUPPLIER, SUPPLIERBRANCH, SUPPLIERCONTACT	Confirm.jsp

Table 4: Methods to allow an employee to update an object.

Entity	Method Name	Tables Used	What is called after the method completes
Customer	viewCustomer	CUSTOMER	EmployeeCustomerInfo.jsp
Customer	viewCustomerOrder	ORDER, PRODUCT, SHIPPING	EmployeeOrderInfo.jsp
Inventory	viewInventoryItem	INVENTORY	InventoryInfo,jsp
Product	viewPrice	PRODUCT	PriceUpdate.jsp
Product	viewProductItem	PRODUCT	ProductInfo.jsp
Shipping	viewShippingItem	SHIPPING	ShippingInfo.jsp
Supplier	viewSupplierItem	SUPPLIER	SupplierInfo.jsp
Supplier Branch	viewSupplierBranchItem	SUPPLIER, SUPPLIERBRANCH	SupplierBranchInfo.jsp
Supplier Contact	viewSupplierContactItem	SUPPLIER, SUPPLIERBRANCH, SUPPLIERCONTACT	SupplierContactInfo.jsp

Table 5: Methods to allow an employee to view the details of an object.

Entity	Method Name	Tables Used	What is called after the method completes
Inventory	inventoryInfo	INVENTORY, PRODUCT	
Inventory	viewInventoryToDelete (calls inventoryInfo)		InventoryDelete.jsp
Inventory	viewInventoryToUpdate (calls inventoryInfo)		InventoryUpdate.jsp
Product	productInfo	PRODUCT	
Product	viewProductToDelete (calls productInfo)		ProductDelete.jsp
Product	viewProductToUpdate (calls productInfo)		ProductUpdate.jsp
Shipping	fetchShippingInfo	SHIPPING	
Shipping	viewShippingToDelete (calls fetchShippingInfo)		ShippingDelete.jsp
Shipping	viewShippingToUpdate (calls fetchShippingInfo)		ShippingUpdate.jsp
Supplier	fetchSupplerInfo	SUPPLIER	
Supplier	viewSupplierToDelete (calls fetchSupplierInfo)		SupplierDelete.jsp
Supplier	viewSupplierToUpdate (calls fetchSupplierInfo)		SupplierUpdate.jsp

Supplier Branch	fetchSupplierBranchInfo	SUPPLIER, SUPPLIERBRANCH	
Supplier Branch	viewSupplierBranchToDelete (calls fetchSupplierBranchInfo)		SupplierBranchDelete.jsp
Supplier Branch	viewSupplierBranchToUpdate (calls fetchSupplierBranchInfo)		SupplierBranchUpdate.jsp
Supplier Contact	fetchSupplierContactInfo	SUPPLIER, SUPPLIERBRANCH, SUPPLIERCONTACT	
Supplier Contact	viewSupplierContactToDelete (calls fetchSupplierContactInfo)		SupplierContactDelete.jsp
Supplier Contact	viewSupplierContactToUpdate (calls fetchSupplierContactInfo)		SupplierContactUpdate.jsp

Table 6: Methods to either prepare session-associated beans with current information or to call methods that prepare session-associated beans with current information. The information is then displayed on the appropriate page.

The remaining methods are described in detail:

- **getInventoryCreateInfo** - The getInventoryCreateInfo method retrieves a list of valid UPCs from the PRODUCT table. This information is displayed in the InventoryCreate JSP.
- **getInventorySearchInfo** - The getInventorySearchInfo method retrieves a list of valid UPCs from the PRODUCT table and all valid warehouse locations from the INVENTORY table. This information is displayed in the InventorySearch JSP.
- **getProductCreateInfo** - The getProductCreateInfo method retrieves information on products that are in the SUPPLIERPRODUCT table but not in the PRODUCT table. This information is displayed in the ProductCreate JSP.
- **getProductSearchInfo** - The getProductSearchInfo method retrieves a list of the valid package types to provide default values for the search. This information is displayed in the ProductSearch JSP.
- **fetchCostLimitMin** - The fetchCostLimitMin method calculates the lowest cost limit that the shipping code can be applied to, and sets this information in the Shipping Bean. This method is called by the method fetchShippingInfo.
- **fetchSupplierIDs** - The fetchSupplierIDs method retrieves a list of valid supplier IDs and names to provide default values for the search. This information is displayed in the SupplierBranchSearch JSP.
- **preCreateSupplierBranch** - The preCreateSupplierBranch method calls the fetchSupplierIDs method to retrieve a list of valid suppliers. This information is displayed in the SupplierBranchCreate JSP.
- **preSearchSupplierBranch** - The preSearchSupplierBranch method calls the fetchSupplierIDs method to retrieve a list of valid suppliers. This information is displayed in the SupplierBranchSearch JSP.
- **fetchSupplierBranchIDs** - The fetchSupplierBranchIDs method retrieves a list of valid supplier branch IDs and names to provide default values for the search. This information is displayed in the SupplierContactSearch JSP.
- **preCreateSupplierContact** - The preCreateSupplierContact method calls the fetchSupplierBranchIDs method to retrieve a list of valid supplier branches. This information is displayed in the SupplierContactCreate JSP.
- **preSearchSupplierContact** - The preSearchSupplierContact method calls the fetchSupplierIDs method and the fetchSupplierBranchIDs to retrieve a list of valid suppliers and supplier branches. This information is displayed in the SupplierContactSearch JSP.

JavaServer Pages

The following tables contain the JSPs that are used by the EmployeeServlet. Since most iDrink entities are associated with JSPs that perform similar actions, the JSPs are categorized by the information that is displayed or requested.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Inventory	InventoryCreate.jsp		InventoryHome.jsp
Product	ProductCreate.jsp		ProductHome.jsp
Shipping	ShippingCreate.jsp		ShippingHome.jsp
Supplier	SupplierCreate.jsp		SupplierHome.jsp
Supplier Branch	SupplierBranchCreate.jsp		SupplierHome.jsp
Supplier Contact	SupplierContactCreate.jsp		SupplierHome.jsp

Table 7: JSPs that prompt for information to create a new object.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Inventory	InventoryDelete.jsp	viewInventoryToDelete	
Product	ProductDelete.jsp	viewProductToDelete	
Shipping	ShippingDelete.jsp	viewShippingToDelete	
Supplier	SupplierDelete.jsp	viewSupplierToDelete	
Supplier Branch	SupplierBranchDelete.jsp	viewSupplierBranchToDelete	
Supplier Contact	SupplierContactDelete.jsp	viewSupplierContactToDelete	

Table 8: JSPs that delete objects.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Customer	EmployeeCustomerInfo.jsp	viewCustomer	
Customer	EmployeeCustomerOrderInfo.jsp	viewCustomerOrder	
Inventory	InventoryInfo.jsp	viewInventoryItem	
Product	ProductInfo.jsp	viewProductItem	
Shipping	ShippingInfo.jsp	viewShippingItem	
Supplier	SupplierInfo.jsp	viewSupplierItem	
SupplierBranch	SupplierBranchInfo.jsp	viewSupplierBranchItem	
SupplierContact	SupplierContactInfo.jsp	viewSupplierContactItem	

Table 9: JSPs that display detailed information about an object.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Inventory	InventoryHome.jsp		EmployeeHome.jsp
Product	ProductHome.jsp		EmployeeHome.jsp
Shipping	ShippingHome.jsp		EmployeeHome.jsp

Supplier, Supplier Branch, Supplier Contact	SupplierHome.jsp		EmployeeHome.jsp

Table 10: JSPs that are home pages for various entities.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Customer	EmployeeCustomerOrdersSearch.jsp		EmployeeHome.jsp
Inventory	InventorySearch.jsp		InventoryHome.jsp
Product	ProductSearch.jsp		ProductHome.jsp
Shipping	ShippingSearch.jsp		ShippingHome.jsp
Supplier	SupplierSearch.jsp		SupplierHome.jsp
Supplier Branch	SupplierBranchSearch.jsp		SupplierHome.jsp
Suppler Contact	SupplierContactSearch.jsp		SupplierHome.jsp

Table 11: JSPs that allow employees to search for objects.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Customer	EmployeeCustomerList.jsp	searchCustomers	
Customer	EmployeeCustomerOrdersList.jsp	searchCustomerOrders	
Inventory	InventorySearchResults.jsp	searchInventory	
Product	ProductSearchResults.jsp	searchProduct	
Shipping	ShippingSearchResults.jsp	searchShipping	
Supplier	SupplierSearchResults.jsp	searchSupplier	
SupplierBranch	SupplierBranchSearchResults.jsp	searchSupplierBranch	
SupplierContact	SupplierContactSearchResults.jsp	searchSupplierContact	

Table 12: JSPs that display objects that match specified search criteria.

Entity	JSP Name	Method that calls the JSP	Page that contains a link to the JSP
Inventory	InventoryUpdate.jsp	viewInventoryToUpdate	
Product	ProductUpdate.jsp	viewProductToUpdate	
Shipping	ShippingUpdate.jsp	viewShippingToUpdate	
Supplier	SupplierUpdate.jsp	viewSupplierToUpdate	
Supplier Branch	SupplierBranchUpdate.jsp	viewSupplierBranchToUpdate	
Supplier Contact	SupplierContact.jsp	viewSupplierContactToUpdate	

Table 13: JSPs which prompt for information to update an existing object.

SupplierServlet

The iDrink SupplierServlet provides the functionality that iDrink suppliers need in order to manage products supplied to iDrink. It gives suppliers the ability to:

- view detailed information about the products they supply to iDrink
- add new products that iDrink can choose to add to their product listings
- edit product information (if iDrink is not already purchasing the product from the supplier)
- search for supplier's branches and contacts
- view detailed information about the supplier's branches and contacts

The SupplierServlet and the JSPs that it calls work with the UserServlet to provide functionality to ensure that only suppliers can log on to and access the pages associated with the SupplierServlet.

Design Points

- As noted in the EmployeeServlet section, many of the methods and JSPs that were needed between the EmployeeServlet and SupplierServlet were similar. In order to avoid duplication of code, these methods and JSPs were coded only once. The common methods were coded in the EmployeeServlet. See the EmployeeServlet section for more details.
- A supplier can only edit products that iDrink does not currently have in production. A product is in production for iDrink if the product is in the PRODUCT table. The JSP pages reflect this by checking a field in the SupplierProductBean called isEditable. This field is set to true if the product is currently in production for iDrink and false if it is not. The following code from SupplierProductListing.jsp shows how this field is checked. (In the code below, spb is a SupplierProductBean):

```
<td>
<%
if(spb.isEditable()) {
%>
<b><a href="SupplierServlet?action=productedit&upc=<%=spb.getUPC()%>">edit</a></b>
<%
} else {
%>
<span class="noteditable">in distribution</span>
<%
}
%>
</td>
```

- In order to display only products supplied by the logged in supplier and not display products supplied by other suppliers, the supplier ID was used as a check in the SQL statement:

```
String queryString = "select
idrink.product.upc as pupc,
idrink.supplierproduct.suppliercontactid,
idrink.supplierproduct.upc,
idrink.supplierproduct.brand,
idrink.supplierproduct.name,
idrink.supplierproduct.size,
idrink.supplierproduct.packagetype
from idrink.supplierproduct,
idrink.suppliercontact,
idrink.supplierbranch
left outer join idrink.product
on idrink.product.upc=idrink.supplierproduct.upc
where idrink.supplierproduct.suppliercontactid=idrink.suppliercontact.suppliercontactid
and idrink.suppliercontact.branchid=idrink.supplierbranch.branchid
and idrink.supplierbranch.supplierid=?";
```

Application flow

The iDrink supplier interface does not force suppliers down a predetermined path. Thus, a supplier can easily jump from one task to another at any time. As a result of this flexibility, it is would be very difficult to illustrate all possible potential paths that a supplier could take through the supplier interface. Figure 10 illustrates the basic application flow available to iDrink suppliers.

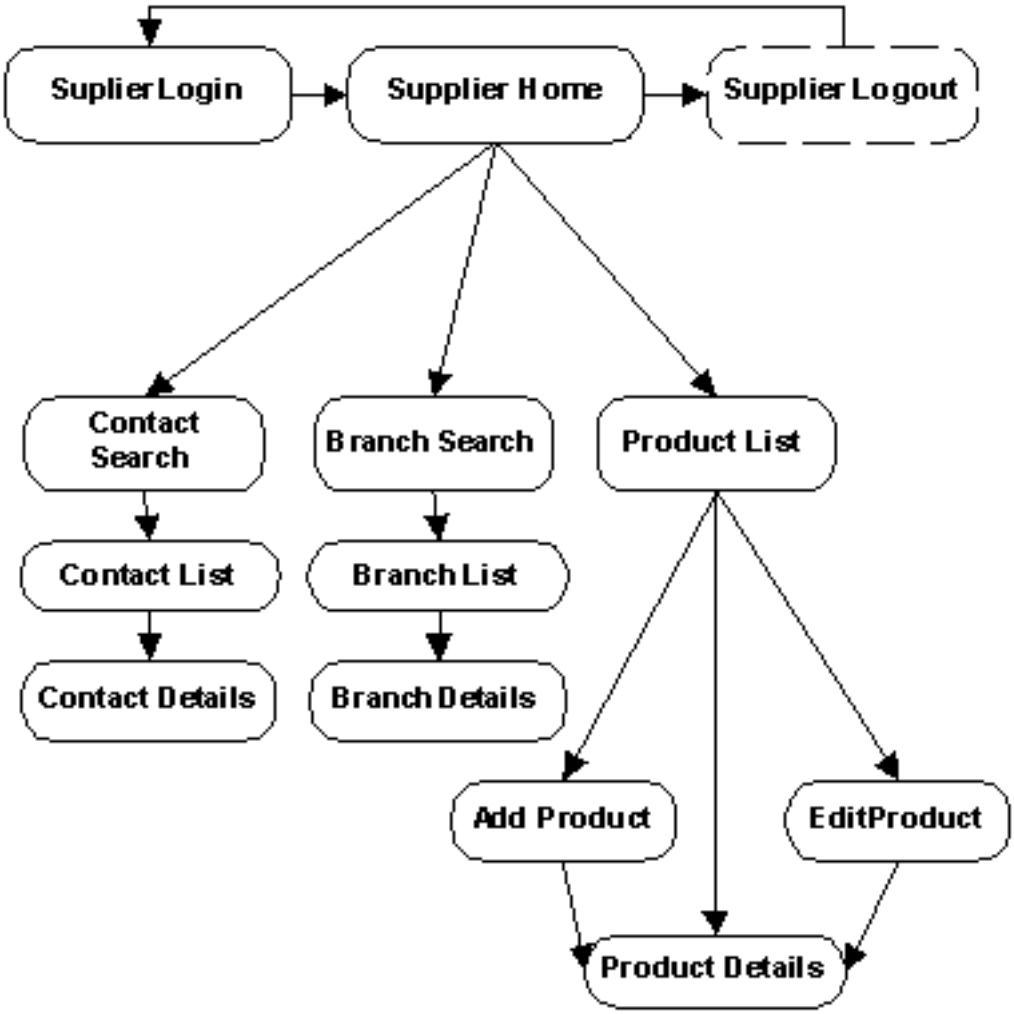

Figure 10: iDrink supplier application flow

Methods

The following is a list of methods that are found in the SupplierServlet. The doPost method receives all incoming requests from the customer, and calls the appropriate method from this list of methods.

- **addSupplierProduct** - The addSupplierProduct method inserts a new product into the SUPPLIERPRODUCT table based on the form/request data.
- **displaySupplierProductDetails** - The displaySupplierProductDetails method calls the fetchSupplierProductDetails method to prepare the session-associated SupplierProductBean with current information. This information is displayed in SupplierProductInfo.jsp.
- **editSupplierProduct** - The editSupplierProduct method updates the supplier product information with values provided by the supplier.
- **fetchSupplierProductDetails** - The fetchSupplierProductDetails method prepares the session-associated SupplierProductBean with the information needed to display the current supplier product (just in case

changes have been
made in between the time the information is displayed and when the information needs to be updated).

- **loadSupplierContacts** - The loadSupplierContacts method retrieves a list of valid supplier contacts for the current supplier.
- **prepareForSupplierProductAdd** - The prepareForSupplierProductAdd method calls the loadSupplierContacts method to obtain a list of current supplier contacts. This information is displayed in SupplierProductAdd.jsp.
- **prepareForSupplierProductEdit** - The prepareForSupplierProductEdit method calls the fetchSupplierProductDetails method to prepare the session-associated SupplierProductBean with current information. This information is displayed in SupplierProductEdit.jsp.
- **searchSupplierProducts** - The searchSupplierProducts method retrieves products supplied by the logged in supplier. This information is displayed in SupplierProductListing.jsp.

JavaServer Pages

The following is a list of JSPs that are used by suppliers (the JSPs marked with an * are also used in the employee application flow):

- ***SupplierBranchInfo** - The SupplierBranchInfo JSP displays detailed information about a specific supplier branch.
- ***SupplierBranchSearch** - The SupplierBranchSearch JSP provides input fields which allow a supplier to enter and submit a set of branch search criteria.
- ***SupplierBranchSearchResults** - The SupplierBranchSearchResults JSP displays all supplier branches that match the specified search criteria.
- ***SupplierContactInfo** - The SupplierContactInfo JSP displays detailed information about a specific supplier contact.
- ***SupplierContactSearch** - The SupplierContactSearch JSP provides input fields which allow a supplier to enter and submit a set of contact search criteria.
- ***SupplierContactSearchResults** - The SupplierContactSearchResults JSP displays all supplier contacts that match the specified search criteria.
- **SupplierHome** - The SupplierHome JSP is the home page to the supplier application flow.
- **SupplierProductAdd -** The SupplierProductAdd JSP provides input fields which allow a supplier to create a new supplier product.
- **SupplierProductEdit** - The SupplierProductEdit JSP provides input fields which allow a supplier to edit an existing supplier product.
- **SupplierProductInfo** - The SupplierProductInfo JSP displays detailed information about a specific supplier product.
- **SupplierProductListing** - The SupplierProductListing JSP displays all products that the supplier supplies.

Installation of enterprise application

The installation of an enterprise application in an IBM(R) WebSphere(R) Application Server instance is handled by wizards found in WebSphere Studio Application Developer (WSAD), the IBM WebSphere Administrative Console and the IBM Web Administration for iSeries(TM) Console. To deploy an application, the application must first be exported to a Java(TM) 2 Platform, Enterprise Edition (J2EE) Enterprise ARchive (EAR) file. The EAR file contains all class files and additional information that is required to install the application on a production system, and can be created by a wizard in WSAD. Once the EAR file has been created, the developer can use the WebSphere Administrative Console or the IBM Web Administration for iSeries Console to deploy the EAR file on the production system.

Export the application to an EAR file

Once an enterprise application has successfully compiled without any errors, a developer will export the application to an EAR file, which will be placed in the **installableApps** directory (\QIBM\UserData\WebASE51\ASE*instanceName*\installableApps) of the WebSphere Application Server instance. This process is documented in *Section 13.8.1 - Exporting an enterprise application to a file location* in the *IBM WebSphere Application Server - Express V5.0.2 Developer Handbook, SG24-6555* Redbook.

Deploy the EAR file on an IBM WebSphere Application Server instance Once the EAR file has been successfully created by the WSAD wizard, the EAR file can be deployed to a production system. A developer may choose to use the WebSphere Administrative Console or the IBM Web Administration for iSeries Console to complete this task. The process of deploying the EAR file via the WebSphere Administrative Console is documented in *Section 6.4 - Installing applications* in the *WebSphere Application Server - Express V5.0.2 Administrator Handbook, SG24-6976* Redbook. To deploy the EAR file via the IBM Web Administration for iSeries Console, the EAR file must already be on the target system. To deploy the EAR file, follow these steps:

- In a Web browser, go to the IBM Web Administration for iSeries Console on the target system. The URL is *http://mysystem:2001/HTTPAdmin*, where "mysystem" is the name of the target system. Enter a user ID and password for the target system when prompted for that information.
- Select the **Manage** tab at the top of the page, then select the **Application Servers** tab below it.
- Ensure your WebSphere Application Server server instance is selected in **Instance/Server** menu. If the server instance is stopped, click the green start button to the left of the menu to start the server instance. Once the server instance is **running**, continue to the next step to begin the installation of the application.
- In the frame on the left, click the **Install New Application** link.
- In the **Specify Application Location** screen, click the **Browse** button. In the file browser window, select the EAR file and click the **OK** button. Click the **Next** button in the **Specify Application Location** screen.
- In the **Provide Options to Perform Install** screen, enter an application name in the **Application Name** textfield and check the **Pre-compile JSPs** checkbox if JSPs should be precompiled. Click the **Next** button.
- In the **Map Virtual Hosts for Web Modules** screen, select **default_host** in the **Virtual host** menu. Click the **Next** button.
- In the **Summary** screen, verify the information is correct and click the **Finish** button. It may take several minutes for the application to install.

WebSphere discoveries

- Prior to WebSphere Application Server - Express for iSeries(TM), Version 5.1, JSPs could import classes from the default package. However, with WebSphere Application Server - Express for iSeries, Version 5.1, JSPs could no longer import classes from the default package. Initially, the iDrink servlets and JavaBeans(TM) were located in the default package. To move them to a non-default package, follow these steps in WebSphere Studio Application Developer:
  - Create a new package and move the java source files into that new package (WebSphere Studio Application Developer will add the appropriate "package" statement to all of your source files when they are moved)
  - Wherever code instantiates a bean, include the package name with the bean name, for example
    ```
    Beans.instantiate(getClass().getClassLoader(), "packageName.beanName");
    ```
  - Wherever a JSP imports a bean, include the package name with the bean name, for example
    ```
    <%@ page import="packageName.beanName" %>
    ```
  - Update the servlet list in web.xml file so that it points to servlets that include the package name. Even though WebSphere Studio Application Developer adds the appropriate package statement to each of your source files when they are moved, it does not update the web.xml file to point at the servlets in the new package. If you use the GUI editor, remove the servlets from the list and add them again. The "Servlet class" field should contain packageName.servletName.

References

- WebSphere Application Server - Express V5.0 for iSeries, IBM(R) Redpaper REDP-3624-00
- WebSphere J2EE Application Development for the IBM eServer(TM) iSeries Server, IBM Redbook SG24-6559-00
- WebSphere Development Studio Client for iSeries V5.0, IBM Redbook SG24-6961-00
- WebSphere Studio Application Developer Programming Guide, IBM Redbook SG24-6585-00

Chapter 3. Lotus Domino environment

The iDrink company wanted to provide a Web site where employees could browse and place classified ads. This Web site, Cola Connections, was created with Lotus Domino version 6.5 on eServer(TM) i5.

With Lotus Domino, the iDrink company was able to quickly create a classified ads dynamic Web site. In addition, Lotus Domino allowed the iDrink company to take advantage of workflow processing and provided interoperability with many other platforms.

The Cola Connections application uses various forms and views to create and maintain ad information. The ad information consists of classified ads that are listed for sale by the employees. To securely manage the employee data, a Lotus Domino Directory Assistance database is used to reference an i5/OS(TM) Lightweight Directory Access Protocol (LDAP) directory.

The iDrink company provides a Web interface using Lotus Domino framesets. Through the Cola Connections Web site, the iDrink employees can perform the following tasks:

- View classified ads
- Create classified ads
- Edit existing classified ads with an option to mark the classified ad as sold

Configuring Lotus Domino to use OS/400 LDAP

Because security was a requirement for the Cola Connections Web site, Lotus Domino was configured to use i5/OS(TM) LDAP services for user authentication. By default, Domino uses its own LDAP capability. In order for Domino to use the i5/OS LDAP service instead of its own LDAP capability, a Directory Assistance database must be created.

For detailed instructions on how to set up a Directory Assistance database see section *4.4.1 Creating a Directory Assistance database* in the Redbook entitled *Integrating Lotus Domino 6 and WebSphere(R) Express V5 on the IBM(R) eServer(TM) iSeries(TM) Server (SG24-6998-00).*

These tips are useful for setting up the Cola Connections Directory Assistance database while following the instructions contained in the Redbook:

- In step 8, when entering the fields in the **Basics** tab, the domain name can be specified as any arbitrary name except for the domain that the Domino server uses. For example, iDrinks' Domino server was configured in the iDrink domain. So in the **Basics** tab, iDrink could not be specified as the domain name.
- In step 10, when entering the fields in the **LDAP** tab, based on the LDAP setup for iDrink, for the field "Type of search filter to use" specify custom and under the "Customized filters" section add the following to the "Authorization Filter" field:

  ```
  (|(&(objectclass=groupofuniquenames)(UniqueMember=%*))(&(|(objectclass=groupofnames)
      (objectclass=AccessGroup))(member=%*)))
  ```

  This search filter allows Domino the capability to search LDAP groups. The Access Control List (ACL) on the Domino database is then set so only the employee group can access it. Since all employees are part of this group, they can now access the Cola Connections Application.

Continue configuring the Domino server by following the instructions in section 4.*4.2 Configuring Domino to use OS/400 LDAP* through the end of the chapter.

To configure the ACL on the Domino database, follow the instructions in section *5.2.3 Updating Domino ACLs for adding users registered in OS/400 LDAP*. For example, the Cola Connections database used the

following person group ACL from the iDrink LDAP schema:
cn=employees/o=idrink/dc=*domainName*/dc=*domainSuffix*. The slashes need to be entered in the above fashion so that Domino can find the group in the LDAP schema.

Application details

From the Cola Connections Web site, an iDrink employee can perform several tasks. An iDrink employee can create an ad by pressing the **New Ad** button. Once created, this new document can then be updated, marked as sold, or displayed.

Application design points

When designing the Lotus(R) Domino(R) application, the iDrink company had to choose between using framesets or navigators for their Web interface. The following lists the differences between framesets and navigators:

- Framesets can contain a form, folder, page, document, view, navigator, or frameset. The frame can also contain a Web page and be associated with a specific URL. Links and relationships between frames can be created with framesets.
- Navigators provide a graphical display of folders, views and design elements to make it easier for most users to find information.

Since the iDrink company wanted to provide a consistent structure throughout the website that could display different forms and views, and since frames can contain forms, folders, pages, documents and views, the frameset was the best choice.

Another task the iDrink company faced was finding and implementing the best way to secure their data. Cola Connections security requirements included:

- A robust architecture providing reliability, speed of deployment, and ease of administration
- A solution that integrated easily with existing applications
- A solution that could be integrated with other software packages

These requirements led the iDrink company to implement a Directory Assistance database on the Domino Server that references an i5/OS(TM) LDAP directory.

Since the iDrink company has several employees with skills in Java(TM) development, background agents for the application were written in Java. This proved to be valuable in terms of utilizing existing skills and resources.

Figure 11 shows the flow of the Cola Connections application.

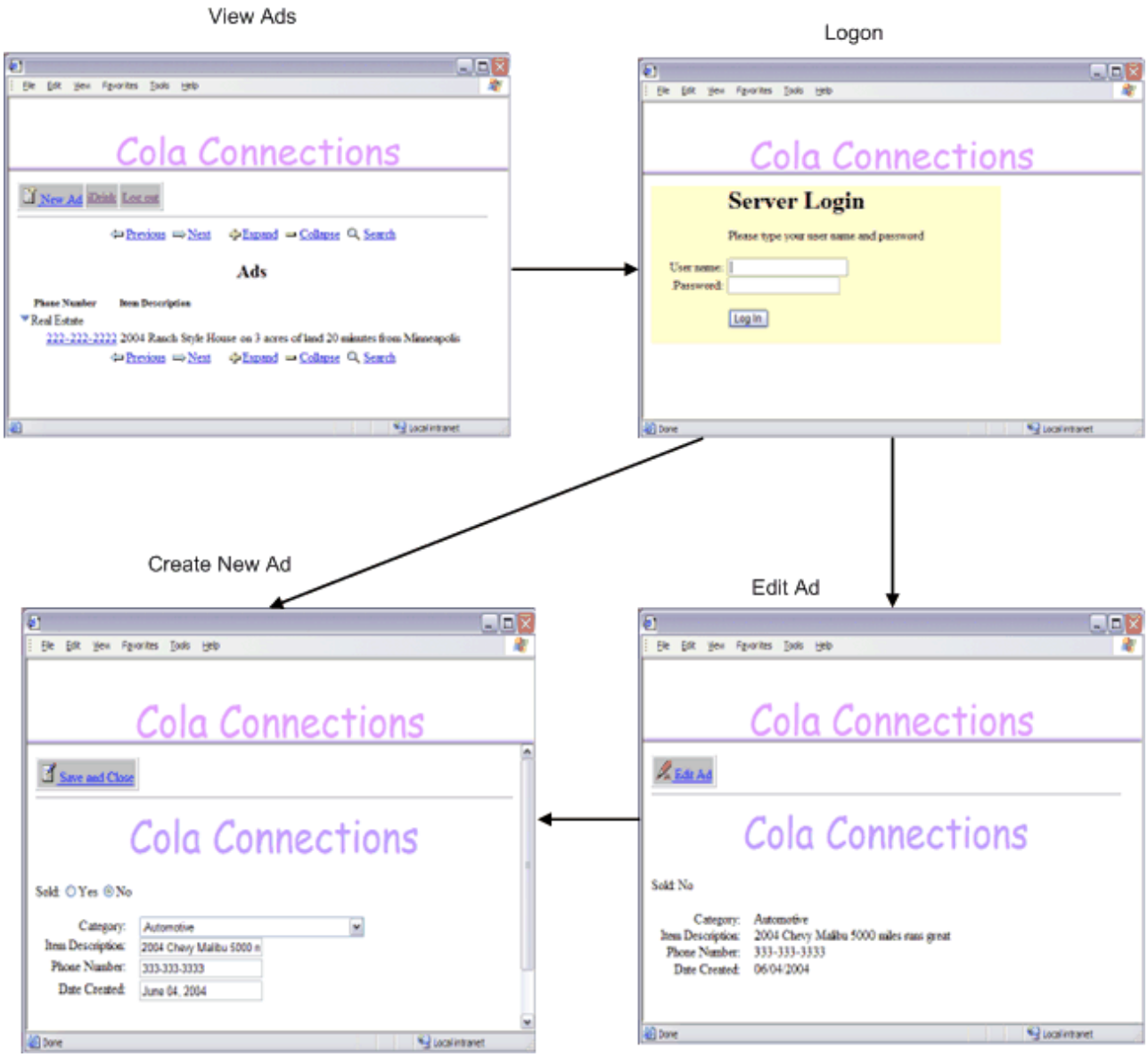

Figure 11: Cola Connections application flow

Application setup

Lotus(R) Domino(R) Designer(R) was used as the development tool for the Lotus Domino implementation work. Lotus Domino designer is a Lotus Notes(R) client application which can be used to quickly create and modify Lotus Domino applications. It provides the application building blocks for all database elements, including forms, views, and agents. Forms were used in iDrink to create new documents in a database and display current documents. Views provide a flexible and intuitive way for documents to be organized. Users can easily create, sort, view, and edit documents.

Lotus Domino forms and views

The iDrink company employees created the Ad Information form and the Ads and All Ads views for their Lotus Domino application.

The Ad Information view contains the list of all available ads that have been created by iDrink employees. The Ad Information form is used to create a new ad or to display an existing ad.

The Ads and All Ads views display the various ads and are sorted by categories such as automotive, computer/home office, and real estate.

Table 13 shows the fields in the Ad Information form.

Name	Field Name	Type	Description
Category	Category	Dialog List	The category of the Ad. The possible categories are: Automotive, Misc Auto, Boats/RVs/Snowmobiles & Other Vehicles, Clothing, Computer/Home Office, Household, Infant, Misc, MotorCycle, Musical, Photo/Stereo/TV, Sporting Goods, Real Estate, Trade, Wanted, Give Away
Item Description	ItemDescription	Text	The description of the item for the ad. Limited to 100 characters
Phone Number	PhoneNumber	Text	The phone number of the employee selling the item
Date Created	DateCreate	Date/Time	Automatically generated field that indicates the date the Ad was created
Sold	Sold	Radio Button	Indicates if the item has been sold. Possible values are “Yes” or “No”

Table 13: Fields in the Ad Information form.

Lotus Domino agents

Lotus Domino agents are design elements added to a Lotus Domino database to automate tasks. Agents can be initiated by a user action or run on a scheduled basis. Agents are commonly used to update or create documents, or to access databases. Lotus Domino agents can be written in Java(TM), LotusScript, or Formula Language.

The creation of agents requires Lotus Domino Designer(R). Decisions that must be made when creating an agent include when the agent should run, what language the agent should be written in, and what documents the agent should run under. After the agent has been written and compiled, it is automatically scheduled to run at the specified time. The built-in debugging capabilities of Lotus Domino Designer are helpful when writing agents.

The Lotus Domino Cola Connections application contains one agent, Delete Ads. This Java Agent runs on a scheduled basis. It will delete any ad documents that have a sold status of yes or that are older than one month.

A code snippet for the Delete Ads agent is shown below:

```
private void deleteAd(AgentContext agentContext, Session session) {

  Vector adDateVector;
```

```
DateTime adDateTime = null;
Date adDate = null;
Date currentDate = null;
int timeBetween = 0;
float timePassed = 0;
try {
  Database db = agentContext.getCurrentDatabase();
  DateTime currentDateTime = session.createDateTime("Today");            // get the current date
  currentDateTime.setNow();         // get all documents where the Sold Status is Yes from All Ads
  DocumentCollection adInformationDC = db.search("SELECT ((Form = \"Ad Information\"))");
  Document adDoc = adInformationDC.getFirstDocument();    // get first ad document
  while (adDoc != null)              // while there are ad documents
    String adStatus = adDoc.getItemValueString("Sold");     // obtain the value of the Sold variable
    adDateVector = adDoc.getItemValue("DateCreate");    // obtain value of create date for ad
    adDateTime = (DateTime)adDateVector.elementAt(0);   // obtain value of create date for ad
    timeBetween = currentDateTime.timeDifference( adDateTime);    // returns difference in seconds
    timePassed = timeBetween/2629744; //divide by seconds in a month
    Document adDoc1 = adInformationDC.getNextDocument();    // get the next ad document
    if ((adStatus.equals("Yes")) || (timePassed > 1))    // if item is Sold, remove the document
      adDoc.remove(true);
    adDoc = adDoc1;        // get the next ad to process
  } // end while adDoc is not null  }
catch(Exception e) {
  e.printStackTrace();
}} // end delete ads
```

Lotus Domino discoveries

Following is a list of key discoveries that were uncovered while creating the Cola Connections scenario:

- After the initial setup, iDrink employees were not being prompted to log in when creating a new ad. At times, the log in prompt did not appear until the Save and Close button was pressed. In order to require an employee to log in when pressing the New Ad button, the following formula language command was set on the action New Ad:

  ```
  @URLOpen(@WebDbName + "/Ad+Information?OpenForm&login")
  ```

- The iDrink company wanted to implement a way for employees to log out of the Cola Connections application. Consequently, a log out feature was incorporated. However, when the log out button was pressed, the employees were not being redirected to the Cola Connections home page. To enable redirection to the home page, the following formula language command was set on the action Log out:

  ```
  @SetTargetFrame("_top");
  @URLOpen("http://nenotes:2500/ColaConn.nsf?logout&RedirectTo=http://NotesServerName:2500
         /ColaConn.nsf/Cola%20Connections?OpenFrameSet")
  ```

References

- IBM(R) Lotus Domino for iSeries(TM) - OS/400(R) Web site
  http://www.ibm.com/servers/eserver/iseries/domino
- IBM Lotus Domino for iSeries (PatnerWorld for Developers)
  http://www.ibm.com/servers/eserver/iseries/developer
- Lotus Web site
  http://www.lotus.com

Printed in USA