Developing an e-business Application (INTERNET BANKING) for the IBM WebSphere Application Server using J2EE technologies

Dr. Katsuji Tsukamoto
Kogakuin University
1-24-2 NishiShinjuku,Shinjuku-ku
Tokyo 163-8677, JAPAN
tsukamoto@tsukaken.jp

Kandasamy Kandavel
Kogakuin University
1-24-2 NishiShinjuku,Shinjuku-ku
Tokyo 163-8677, JAPAN
KANDAVEL@jp.ibm.com

ABSTRACT:
This article explores the several Java 2 Enterprise Edition (J2EE) technologies adopted in developing the web based e-business application called INTERNET BANKING for IBM WebSphere Application Server. The implementation part demonstrates various approaches and elaborates the best approach followed in developing the Internet Banking e-business application.

Keyword: Java 2 Enterprise Edition(J2EE), Enterprise Java Beans(EJB), Servlet, Java ServerPage(JSP), HTML, IBM WebSphere Studio Application Developer(WSAD), IBM WebSphere Application Server(WSAS), Visual Age for Java(VAJ), Internet Banking(IBANK), IBM DB2 Universal Database(UDB), E-Business Application Framework(EBAF), Model-View-Controller(MVC), EJB Universal Test Client (UTC)

I. INTRODUCTION:

The explosive growth of the World Wide Web over the last few years continues unabated. The Web has evolved from sites that serve static HTML pages to a global arena for recreation, information, collaboration, and business transactions. This article explores the process of developing an e-business application: Internet Banking. The development process uses the Java technologies: Java Beans, Enterprise Java Beans, Servlets and Java Server Pages. The process of building servlet-based systems is maturing. Building system using JSP and servlet is a fairly new way of building Web applications, and there are several design and implementation approaches. The article explores the best approach selected in developing the Internet Banking. The design approach is based on the IBM Application Framework for e-business, commonly referred to as EBAF, in which the application development follows the model-view-controller (MVC) design pattern. The Model/View/Controller (MVC) paradigm that is popular in object-oriented programs contains the following application parts:

Model: Encapsulates all the business logic and rules and does the business processing. This is implemented either using Java Beans or EJB.
View: Takes the result of the business processing and constructs the response that is presented to the user. This is implemented using JSP.
Controller: The component that manages and controls all the interaction between the model and view and the flow of application. This is implemented by using servlet that receives the user request and passes all the input parameters to the model in which the actual work is done. Finally the JSP is called to return the output.

II. DEVELOPMENT ENVIRONMENT:

The development was carried out on Windows platform using IBM software tools: IBM WebSphere Studio Application Developer and Visual Age for Java. IBM WSAD is compliant with J2EE specification, equips to build and deploy state-of-the-art server-side applications meeting Servlets 2.2, JSP 1.1 and EJB 1.1 specifications. The deployment and publishing of application was done by using IBM WebSphere

Application Server. The persistent data store was implemented by using IBM DB2 Universal Database. The Java 2 Runtime Environment used was JVM 1.3.1.

III. JAVA 2 PLATFORM ENTERPRISE EDITION (J2EE):

An end-to-end technical platform that enables developing, deploying and managing multi-tier server-centric applications. Some of the main J2EE components that were used for IBANK development are Java servlets and JavaServer Pages, Enterprise Java Beans, Java Remote Method Invocation and RMI-IIOP, Java Naming and Directory Interface (JNDI) and Java Database Connectivity (JDBC).

IV. IBANK APPLICATION DESIGN AND IMPLEMENTATION:

IBANK design process is based on the following categories:

a) Application Requirements:

Customers can avail the following features of IBANK system:

- List of all types of account (Savings Account, Checking Account, Payee Account etc)
- Account wise current balance
- Account wise history for a period (History gives account transaction detail)
- Transfer funds between accounts
- Pay bills from account
- Manage the list of payees for bill payment
- Change of login password

b) System Requirements:

IBANK implements the following systems:

- Web server with application server integration to publish the IBANK application
- Digital certificate recognized by the client browsers

For customers to access IBANK system, the following are the pre-requisites -

- The login user ID & transaction ID with initial password from IBANK
- An Internet browser that supports the Secure Sockets Layer (SSL)
- Access to the Internet

c) Analysis Object Model:

The IBANK object model contains four main objects: Bank, Customer, BankAccount, and TransactionRecord. The Bank entity contains many customers who operate one or more accounts that may contain zero or more transactions. The E-R diagram of IBANK model is given as below:

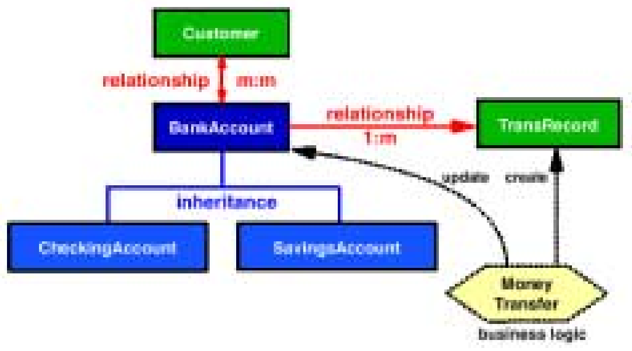

Figure 1

d) Subsystem Design:

The IBANK consists of eight subsystems that are listed as below:

- Application Manager: Initializes the bank and provides session management with logout function
- Login: Authenticates the user
- Account Information: Provides account balance and history
- Pay bill: Enables the user to pay bills to corporations (Payees)
- Payee Setup: Manages the user's list of payees
- Transfer funds: Enables the user to transfer money between bank accounts

e) Security Model:

The IBANK security design is considered as below:

- Access to the web server is controlled through password
- Web server serves application pages using SSL protocol
- Two level password checking for user logon and transaction authentication

f) Architecture and Design:

The architecture of IBANK is defined by the scope of the project: to build an Internet Banking

Application using IBM tools that runs on the IBM WebSphere Application Server. Three major design goals that were considered are as below:

Access to the business model: A domain firewall layer to separate the web application from the business model implementation. The domain firewall is an API that abstracts the object model for the client that is implemented using Java interfaces.

Controlling the Interaction between the Client and Server: Servlets are used as controller that adds another layer to the application beyond JSP.

User Interface design: Implemented using JSP in which the programming logic resides in view beans that are created by the servlet and placed in the HttpSession or the HttpRequest object. Bean tags are used to request information from view beans.

g) Error Handling:

The error handling is carried out in the following ways:

User errors: If user enters an incorrect value, an appropriate message is displayed either on a separate page or the page on which the entry was made.

Application Errors: If the application detects an error, for example, the bank or account is not available, the callErrorPage method is called that sends the user to a generic error page and lists the error.

h) Implementation:

The IBANK implementation consists of the following subsystems:

➢ *Business model:*

The business logic is defined in the EJB model in which both CMP entity bean and session bean are used. The CMP entity beans are used to embody permanent business data and provide methods to manipulate that data. The EJB to RDB mapping was done using bottom-up approach. The entity bean access is implemented in the following ways:

a) Using Access Beans to entity beans

The direct access to entity beans was not considered mainly due to the impact on the performance of the application and to avoid EJB programming in the client side. Compared to copy-helper and Java wrapper access bean methods, the rowset access bean method that contains all the characteristic of both copy-helper and Java Wrapper was found suitable for the finder method (findAccountByCustID) that returns multiple result beans. This gives very good performance particularly when the result contains minimum rows. The code that implements is as below:

```
try {
AccountAccessBeanTable acctrows = new
AccountAccessBeanTable();
AccountAccessBean acct = new
AccountAccessBean();

acctrows.setAccountAccessBean(acct.findAcc
ountByCustId(new CustomerKey(custId));

try {
for (int i=0; i < acctrows.numberOfRows();
i++){
acct = acctrows.getAccountAccessBean(i);
System.out.println("AccountNo: " +
acct.getAccountNo() +
" AccountType: " + acct.getAccountType() +
" AccountBalance: " +
acct.getAccountBalance());
}
} catch (IndexOutOfBoundsException e){
System.out.println("End of list");
} catch (Exception e){
e.printStackTrace();
}
}
```

b) Using Session Façade to entity beans

This approach is used to implement most of the business logic. The entity beans are accessed via a stateless session bean that caches the home and provides access to all the properties of the bean in a single method. The performance is very good here, as the number of the remote method calls is limited and all the method calls between the façade bean and the entity bean are made locally (through the local pipes). Session beans are created to access set of entity beans that are accessed in one application. The stateless session bean code sample is as below:

```
Object objHome = initialContext.lookup("BankxactHome");
BankxactHome bankxactHome = (BankxactHome)
javax.rmi.PortableRemoteObject.narrow(objHome, BankAccountHome.class);
Bankxact trans;
java.util.Enumeration enum = bankxactHome.findTransByAccountId(new AccountKey(accId) );
while(enum.hasMoreElements()){
trans = (Bankxact) javax.rmi.PortableRemoteObject.narrow
(enum.nextElement(),Bankxact.class);
// From trans object the data items such as TransId, TransType, TransAmount...
}
```

- *Access to business model (Domain firewall):*

This layer wraps up and instantiate all the available session beans using Java Interfaces and classes. The approach makes the client code simple by avoiding EJB programming and code reuse. The layer provides access to all the functionality of the bank implementation as well as providing initial finder methods to locate and instantiate bank implementation.

- *Web application (Servlet and JSP implementation):*

The client programming to access session beans (through domain firewall) is coded using servlets and the result is passed to JSP through view beans. This approach separated the web page presentation and generation logic and avoids the JSP scriptlet coding.

V. TESTING:

Initially the bean testing was carried out using EJB UTC plug-in test tool of WSAD in the order of entity bean finder methods first followed by the session bean methods to access entity beans. This was followed with the testing of domain firewall code and servlet. Finally, the complete integration testing was done using the HTML and JSP pages. As part of testing, performance comparison between access bean and session bean to entity bean was carried out to access bulk transactions of a particular account of a customer. Both methods yield the same performance.

VI. DEPLOYMENT:

The EJB development is packaged in one JAR file (IBANKEJB.JAR) and the web development is packaged in one WAR file (IBANKWEB.WAR). Both EJB project and WEB project were assembled in one single Enterprise application project (IBANKEAR) that was deployed successfully to Web Server and Application Server for publishing and opening the IBANK application for e-business.

VII. CONCLUDING REMARKS:

Both Access beans and Session beans give very good performance especially when the unit of work contains more transactions. Servelet based controller is found to be better when compared to JSP as view and controller code are logically separated.

VIII. REFERENCES:

[1] IBM Redbooks: EJB Development with Visual Age for Java for Web Sphere Application Server

[2] IBM Redbooks: Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server

[3] IBM Redbooks: Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java

[4] IBM Redbooks: VisualAge for Java and WebSphere Studio