M.S. (C.S) Directed Study Research Course

Research Paper Title:

Openness and Ease of Use:

Developing J2EE Components on Sybase EAServer and IBM WebSphere

By,

Kashif Ahmed

kasahmed@hotmail.com

Maharishi University of Management

Professor,

Dr. Ralph Bunker

rbunker@mum.edu

Table of Content

- Footnotes will be on each page wherever is required.

Introduction and Summary of Related Research

The following is a summary of research already performed on developing J2EE component and comparison on different application servers. This section also describes the overview, purpose and aim of this research.

INTRODUCTION

This research is being done for the final course of my M.S. (Computer Science) program at Maharishi University of Management under the supervision of Prof. Dr. Ralph Bunker. The purpose of this research course is to experience the kind of thinking and study that is necessary to write a research paper. The focus of my paper is on labs that students learning EAServer and WebSphere will do. I believe that by analyzing these labs, I can determine the ease of use of EJB development on each of J2EE application servers.
The J2EE application servers which are chosen for this research are IBM WebSphere v.5 and Sybase EAServer 4.2. Both application servers provide full J2EE platform support for EJB application development. The difference is in cost, efficiency, performance, benchmark, supported user friendly IDEs, etc. In my research I'm going to determine which application server provides the best overall "ease of use" and rapid EJB development based on the "do less and accomplish more" analogy[1].
Much research has already been done in this field to monitor the performance and scalability of EJB applications. For example:

- Performance and scalability of EJB applications: Emmanuel Cecchet, Julie Marguerite, Willy Zwaenepoel – ACM Special Interest Group on Programming Languages – 2002.
  http://doi.acm.org/10.1145/582419.582443
- Dynamo vs. iPlanet Application Server Comparison Matrix ver. 0.6– CORVIA Inc. 2002.
  http://www.ugcs.caltech.edu/~werdna/nnh/technology/ Dynamo-iPlanetMatrix.pdf
- The ServerSide Application Server Matrix
  http://www.theserverside.com/reviews/matrix.jsp
- Flashline's Open Source Frameworks Matrix
  http://www.flashline.com/components/appservermatrix.jsp?sid=1046444527500-3637481644-51

Unlike the above examples, my research will focus on the practical implementation and development of EJBs in the real world. This research consists of a lab experiment where two groups of students are supposed to learn to use each application server and then perform the same lab experiment on each. Based on those results, as well as the students' performance and feedback, we are going to analyze the ease of use and openness of developing J2EE components on each server.
(Note: For this research, all the experiments have been performed by me on each server and results have been derived based on my experience and analysis.

[1] Maharishi (the founder of M.U.M) told us the analogy of the Science of Creative Intelligence, that how one can do less and accomplish more by practicing higher levels of consciousness.

Brief Intro to J2EE and EJBs

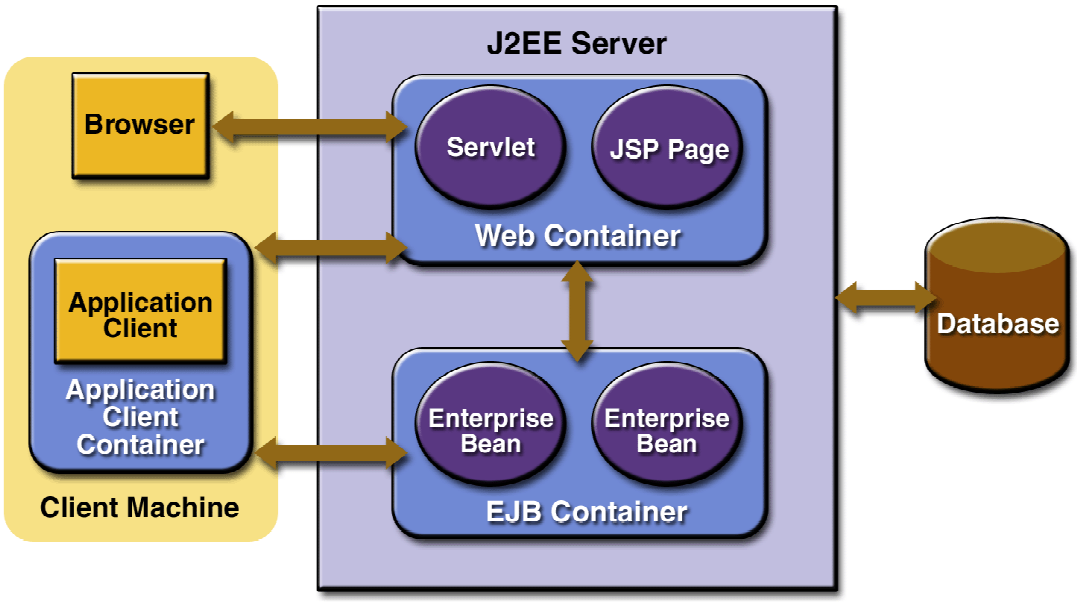
This section gives you brief introduction to Java 2 Enterprise Edition (J2EE) platform and Enterprise Java Beans (EJBs), just to have an overview of them. As, our both experiments are done using EJBs therefore all students must have knowledge about J2EE and EJBs.[2]

J2EE Server and Container Overview

The J2EE platform is essentially a distributed application server environment – a Java environment that provides the following:

- A runtime infrastructure for hosting applications
- A set of Java extension APIs to build applications.

Following picture describe the graphical view of J2EE environment .



Enterprise Java Bean (EJB)

Enterprise JavaBeans is an important part of J2EE. It provides an environment in which components from several manufacturers can be assembled into a working application. The application assembler, with deep knowledge of the requirements of the business, can choose the component that best matches the task at hand.
Enterprise beans are specialized components that can encapsulate session information, workflow, and persistent data. A bean client is an application that uses enterprise beans to access database data or an enterprise bean that relies on the functionality provided by another enterprise bean. An enterprise bean has three parts:

[2] The related URLs to study more about EJBs have been mentioned reference section # 10.

- **The home interface** is used by the client to create and discard beans.
- **The remote interface** is used by the client to execute the bean's business methods.
- **The implementation or bean class** is where the bean's business methods and callback methods are implemented. The client never invokes these methods directly; they are invoked by the bean container.

Types of Java Beans

Summary of Enterprise Bean Types	
Enterprise Bean Type	**Purpose**
Session	Performs a task for a client
Entity	Represents a business entity object that exists in persistent storage
Message-Driven	Acts as a listener for the Java Message Service API, processing messages asynchronously

The two major types of enterprise beans which are mostly used are: session beans and entity beans.

- **Session beans** come in two flavors: ***stateful*** and ***stateless***. Stateful session beans maintain state between method invocations; stateless session beans do not.

  Stateless session beans are useful for grouping related methods in one place. Stateful session beans can be used to encapsulate workflow. In most cases it's more efficient for client applications to use session beans (stateless or stateful) to accomplish their tasks than to use entity beans directly because network traffic is reduced.

- **Entity Beans** are similar to enterprise objects (the objects that represent an instance of a data entity in Enterprise Objects). They encapsulate access to data stored in databases and other types of data stores.

An enterprise-bean developer can focus on the high-level business logic needed to implement the services that a bean provides instead of on low-level system or data- store calls (those functions can be left to the container).
One of the most important functions of the container is transaction management and access control.

General Introduction to Application Servers

[3] Product Summary

EAServer 4.2

Summary:

EAServer has open standards-based support for multiple component models and innovative new performance features and tuning options. Provides various tools and utilities for an easy setup, deployment, and configuration. Its administration capabilities are not wizard-based but still powerful.

Strengths	Weaknesses
It provides robust performance and scalability support; full J2EE platform support; it provides GUI and CLI tools for deployment and configuration; provides better EJB/Servlet/JSP support; Easily configured and managed clustering.	Limited monitoring capabilities; lack of wide third party tools support; missing some features for RAD and wizard-based deployment and configuration; Bigger footprint; Documentation needs improvement.

WebSphere 5.0 Server

Summary:

WebSphere provides a comprehensive array of offerings for the J2EE platform including complete tools support. Also provides sophisticated management tools (extensive integration with Tivoli line), includes powerful features for self-optimizing, self-configuring, self-protecting and RAS/Troubleshooting.

Strengths	Weaknesses
WebSphere is a Java centric; Strong contribution to J2EE efforts; It is one of the leading application servers which provides server management tools/APIs, transaction support, security, and messaging infrastructure. It provides custom development and deployment IDE and tools; also provides integration with complimentary third party products and other IBM products.	Performance features are not as advanced; Setup is complex and management of server also includes complexities.

[3] This information has been researched by Sybase and includes my personal experience and being used with their approval.

Brief Intro to EJB development tools and database that would be used during experiments.

There are so many Java development tools for EJB development. We will be using Boland JBuilder for our lab experiments. I chose JBuilder for its GUI based (Viusal) development of EJBs. It provides a GUI wizard to create the skeleton for you and it also provides plug-in for most application servers to deploy the EJBs to the servers. The following is a brief introduction to the tool.

INTRODUCTION TO JBUILDER®[4]

JBuilder® is the cross-platform environment for building industrial-strength enterprise Java applications. JBuilder Enterprise Offer speeds EJB, ™ Web client, XML, Web Services, mobile, and database application development with two-way visual designers and rapid deployment to leading J2EE™ platform application servers. It also has UML™ code visualization, refactoring, code formatting, HotSwap debugging, enterprise unit testing, and support for multiple version control systems.

FEATURES AND BENEFITS

- JBuilder provides full support for building EJB 1.1- and EJB 2.0- compliant applications. It allows you to rapidly create reusable Enterprise JavaBeans™ (EJB™) including entity, session, and message-driven beans with a two-way visual EJB designer.
- JBuilder includes tight integration with Borland® Enterprise Server, BEA WebLogic Server, ™ IBM® WebSphere,® Oracle9i™ Application Server, Sybase® EAServer, and Sun® ONE Application Server. Run and debug Enterprise JavaBeans, locally and remotely. Hot deploy EJBs instantly - without shutting down the application server.
- It helps you develop, find, consume, and deploy Web Services quickly and simply. JBuilder supports the latest Web Services technologies, including Apache™ Axis 1.1, SOAP 2.3, WSDL, UDDI, and WSIL.

INTRODUCTION TO Sybase Adaptive Server AnyWhere®

I used Sybase Adaptive Server AnyWhere™ (ASA) database in my experiments. ASA is, free for evaluation and the evaluation version include all database features. It is a small database with all features of DBMS included.

Adaptive Server Anywhere is the relational database at the core of the product is a transaction-based SQL database designed for personal and workgroup use. Adaptive Server Anywhere runs on a wide range of operating systems, including many flavors of Windows and UNIX, as well as on Novell NetWare. It runs on hardware ranging from multiple-CPU workgroup servers to the most modest PCs, as well as on Windows CE devices.

[4] This information has been taken from Borland official JBuilder Web site and JBuilder's documentation.

The Experiments

The same experiment has been done on each server to see the difference in development of EJB components on each of them. The first experiment was designed to develop a Session Bean and test the ease of use and openness of each server. The second experiment was designed to develop an Entity Bean using Sybase ASA database. The following is a description and the results of each experiment on both servers.

Experiment # 1

Experiment # 1 on Sybase EAServer

Tasks list of Lab Experiment # 1:

1. Run JBuilder and EAServer on their machine.
2. Get familiar with JBuilder Environment
3. Create EJB Session Bean using JBuilder.
4. Start and Configure Sybase EAServer
5. Deploy EJB Session Bean to Sybase EAServer.
6. Write Java Client to access EJB Session Bean, deployed on EAServer.
7. Test the Session Bean

Lab Description:

- Create the EJB Module in JBuilder called TestSesMod and EJB version as EJB 2.0 compliant.
- Create a Stateful Session Bean called TestSes.
- Use JBuilder wizard to add getMessage() Method in session bean and interface type as remote.
- Code your getMessage() method with default greetings.
- Deploy and Run Session Bean
  - Create Run time configuration in JBuilder for EAServer.
  - Build your project that will create a .jar file and Sybase specific deployment descriptor.
  - Deploy EJB JAR file to Sybase EAServer.
- Test Session Bean
  - Create an EJB Test client using EJB Test Client Wizard in JBuilder.
  - Code your main method to create your EJB Component and call getMessage() method.
  - Run the Test Client.
- Debug the Session Bean
  - Assign Breakpoints in your bean.
  - Start EAServer in Debug mode.
  - Run Test Client.

Experiment # 1 on IBM WebSphere

Tasks list of Lab Experiment # 1:

1. Run IBM WebSphere on the machine.
2. Get Familiar with WebSphere Studio development environment.
3. Read the help section for EJB Development and deployment.
4. Generate the EJB Jar file from JBuilder.
5. Create EJB Project in WebSphere for Session EJB and import jar file.
6. Deploy EJB Session Bean from generated jar file to WebSphere Server.
7. Run Test Client to test the EJB.

Lab Description:

- Create the EJB Project in IBM WebSphere.
- Import the EJB Jar file and make sure all the classes have been imported.
- Generate "Deploy and RMI Code" of EJB project for WebSphere server.
- Run your EJB on Server under WebSphere Test Environment.
- Test Session Bean
    - Use the Universal Test Client to perform the following tasks:
        - Finding an enterprise bean
        - Calling an EJB method
        - Removing an enterprise bean
        - Viewing fields
        - Viewing methods
        - Verifying the results.

Interpretation of Experiment # 1 Results

JBuilder offers a user-friendly environment to create an EJB via wizard format. It helps by creating the whole skeleton and automatically creating Remote and Home objects for you. Total Steps required to create a Session Bean are:

- Create a Project
- Make EAServer as a default target Server
- Create EJB Module
- Visually create a Session Bean

Deployment and Testing Session Bean on EAServer:

I was able to deploy my session EJB to EAServer directly from JBuilder using the plug-in for EAServer. It deploys the EJB to EAServer and installs it on the server it also creates the stubs and skeleton. If you don't like to deploy your EJBs from JBuilder then it is also easy to deploy the ejb.jar file to EAServer and generate Stubs and Skeletons.
To deploy Session Bean from JBuilder to EAServer you need to create a runtime configuration for applications. Following are the steps to set the runtime configuration.

From the JBuilder main menubar, click Run à Configurations à The Project Properties window shows up with Run tab selected- à Click New to create a new runtime configuration à In the Runtime Configuration Properties window, enter the values for Name, Build Target, and Type à Click OK à Click OK.

It also provides run time monitoring at component level through which you can monitor how many components are active or pooled and other information.

Importing JAR to EAServer

Start Jaguar Server à Start Jaguar Manager and connect to the Jaguar server. To import a JAR file, right-click **Packages**. Select Deploy à EJB Jar à Use the Browse button to select <your ejb>**.jar** à Click Next à Click **FINISH.**

After importing the JAR files you must go to **Server** à **Jaguar** and add this new package to the installed packages. Right click **Installed Packages** à **Install Package** à **Install an Existing Package**. Choose the existing **TestSesMod** package and click **OK**.
To see your EJB component properties, you can right click on the component and select component properties. It shows you all information related to your EJB; like home interface name, JNDI name etc

Testing EJB on EAServer

EAServer doesn't provide any testing tool to test the deployed EJB. In order to test the EJB I had to use JBuilder to create a Test Client. JBuilder provides a Wizard to create a Test Client for you (you can write the Test Class on any notepad too). Once the Test Client is created, I coded the main method to call the session bean method.

The test client runs within JBuilder by creating another tab, TestSesTestClient, in the message pane. You can see the successful execution of the Test Client from the last line ***Return Value from getMessage (): Testing Successful. EJB Session Bean.***

Here is the code:

```
package com.borland.demo.ejb;

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

public class TestSesTestClient1 {
 private static final String ERROR_NULL_REMOTE = "Remote interface reference is null.  It must be created by calling one of the
Home interface methods first.";
 private static final int MAX_OUTPUT_LINE_LENGTH = 100;
 private boolean logging = true;
 private TestSesHome testSesHome = null;
 private TestSes testSes = null;

 //Construct the EJB test client
 public TestSesTestClient1() {
   long startTime = 0;
   if (logging) {
     log("Initializing bean access.");
     startTime = System.currentTimeMillis();
   }

   try {
     //get naming context
     Context ctx = getInitialContext();

     //look up jndi name
     Object ref = ctx.lookup("TestSesMod/TestSes");

     //cast to Home interface
     testSesHome = (TestSesHome) PortableRemoteObject.narrow(ref, TestSesHome.class);
     if (logging) {
       long endTime = System.currentTimeMillis();
       log("Succeeded initializing bean access.");
       log("Execution time: " + (endTime - startTime) + " ms.");
```

```
    }
  }
  catch(Exception e) {
    if (logging) {
      log("Failed initializing bean access.");
    }
    e.printStackTrace();
  }
}

private Context getInitialContext() throws Exception {
  String url = "iiop://power2:9000";
  String user ="jagadmin";
  String password ="powernow";
  Properties properties = null;
  try {
    properties = new Properties();
    properties.put(Context.INITIAL_CONTEXT_FACTORY, "com.sybase.ejb.InitialContextFactory");
    properties.put(Context.PROVIDER_URL, url);
    properties.put(Context.SECURITY_PRINCIPAL, user);
    properties.put(Context.SECURITY_CREDENTIALS, password);

    return new InitialContext(properties);
  }
  catch(Exception e) {
    log("Unable to connect to EAServer at " + url);
    log("Please make sure that the server is running.");
    throw e;
  }
}

//-------------------------------------------------------------------------
// Methods that use Home interface methods to generate a Remote interface reference
//-------------------------------------------------------------------------

public TestSes create() {
  long startTime = 0;
  if (logging) {
    log("Calling create()");
    startTime = System.currentTimeMillis();
  }
  try {
    testSes = testSesHome.create();
    if (logging) {
      long endTime = System.currentTimeMillis();
      log("Succeeded: create()");
      log("Execution time: " + (endTime - startTime) + " ms.");
    }
  }
  catch(Exception e) {
    if (logging) {
      log("Failed: create()");
    }
    e.printStackTrace();
  }

  if (logging) {
    log("Return value from create(): " + testSes + ".");
  }
  return testSes;
}

//-------------------------------------------------------------------------
// Methods that use Remote interface methods to access data through the bean
//-------------------------------------------------------------------------

public String getMessage() {
  String returnValue = "";
  if (testSes == null) {
    System.out.println("Error in getMessage(): " + ERROR_NULL_REMOTE);
```

```
    return returnValue;
  }
  long startTime = 0;
  if (logging) {
    log("Calling getMessage()");
    startTime = System.currentTimeMillis();
  }

  try {
    returnValue = testSes.getMessage();
    if (logging) {
      long endTime = System.currentTimeMillis();
      log("Succeeded: getMessage()");
      log("Execution time: " + (endTime - startTime) + " ms.");
    }
  }
  catch(Exception e) {
    if (logging) {
      log("Failed: getMessage()");
    }
    e.printStackTrace();
  }

  if (logging) {
    log("Return value from getMessage(): " + returnValue + ".");
  }
  return returnValue;
}

//--------------------------------------------------------------------------
// Utility Methods
//--------------------------------------------------------------------------

private void log(String message) {
  if (message == null) {
    System.out.println("-- null");
    return ;
  }
  if (message.length() > MAX_OUTPUT_LINE_LENGTH) {
    System.out.println("-- " + message.substring(0, MAX_OUTPUT_LINE_LENGTH) + " ...");
  }
  else {
    System.out.println("-- " + message);
  }
}
//Main method

public static void main(String[] args) {
  TestSesTestClient1 client = new TestSesTestClient1();
  // Use the client object to call one of the Home interface wrappers
  // above, to create a Remote interface reference to the bean.
  // If the return value is of the Remote interface type, you can use it
  // to access the remote interface methods.  You can also just use the
  // client object to call the Remote interface wrappers.
  try{
    client.create();
    String message = client.getMessage();
    System.out.println("Hi Kashif! = " + message);
  }
  catch (Exception ex){}

  }
}
```

Deployment and Testing Session Bean on IBM WebSphere:

After generating the EJB.jar file from JBuilder, it's not difficult to import the Jar file into WebSphere. I needed to open the J2EE perspective and import the .jar file under EJB

Module by right clicks à Select Import. à Select EJB jar file à Next à Browse to the jar file à Type the new EJB Project Name à Type the Enterprise application project name à Click Finish. This imported the EJB with package and classes. In order to run the EJB on WebSphere server, we don't need to write a separate client. IBM WebSphere offers a feature to create a Test Environment and test your EJBs. To do this, I created a Test Environment for our Session Bean that provided me One WebSphere v.50 Test Environment for testing. It did have a sharp learning curve for setting up the test environment and configuring the server.

Once the test environment is created, go to the EJB Module à Right Click à Run on Server à Select the WebSphere Test Environment from the new popup window à Next à Remember to check the box Deploy EJB Bean (Generate EJB Deploy and RMIC code) à Finish.
This will start the Test Server Environment; deploy the EJB and provide a GUI Wizard based interface to test EJB Components. In the JNDI Explorer you can browse your EJB and then click on it à Click on Invoke button to create an object à Click on Work with Object button to work with the created object.
Once the object is available, you can browse all the available methods and invoke them and work with them to see the results and test your session bean.

Conclusion:

- Above experiment clearly shows the J2EE deployment in platform independent way. As I just exported one j2ee.jar file from EJB project and imported in two different environments without changing any content of that file. Few steps were taken differently to import the jar file. Test was successful on both servers using the same J2EE component.
- Testing was easy and user friendly on IBM WebSphere as compared to Sybase EAServer.
- Configuration and development was typical and hard on IBM Web Sphere as compared to Sybase EAServer.
- Learning curve was steep on WebSphere as compared to EAServer learning EJB development.
- IBM WebSphere provides excellent documentation and help to learn EJB development as compared to EAServer.
- In the case testing EJBs on EAServer, I had to do more coding and write a TestClient while WebSphere does this job for you and provides a wizard to deploy and test your EJB.
- Server execution of WebSphere was slow and required huge amount of free memory when compared to EAServer.
- WebSphere provides a rich development environment for EJBs and is based on Eclipse technology. EAServer, on the other hand, doesn't have as rich a development environment but it does allow you to create the components and method through wizards. To code the file you must user notepad interface which is not that rich.

Following Table summarize the amount of work done to Test Session Bean in terms of mouse click and extra line of code on each server.

No. of steps (mouse click and keyboard hits)	EAServer	WebSphere
To setup runtime configuration in JBuilder	8	N / A
To deploy ejb.jar file.	2	N/A
To import and deploy EJB jar file using Servers' administration software.	16	10
To Test EJB using Test Client in JBuilder	4	4
Total lines of code written for Test Client	130	0
To Test EJB through server.	N/A	20

Table 1_1

* While setting up the test environment and testing EJB on EAServer, my fingers have to do more work as compared to my brain. On WebSphere, my brain has to do more work as compared to fingers.

(Note: WebSphere 5 was release after JBuilder 8, so it is not supported, that's why currently you can't deploy your EJB directly to WS from JBuilder 8. The JBuilder 9 feature matrix indicates that it supports WS 5.)

Experiment # 2

Experiment # 2 on Sybase EAServer

Tasks list of Lab Experiment # 2:

1. Setup database Configuration in JBuilder for the server
2. Create EJB Entity Bean using JBuilder.
3. Deploy EJB Entity Bean to Sybase EAServer through JBuilder.
4. Write Java Client to access EJB Entity Bean, deployed on EAServer.
5. Run the Test Client and test the Entity Bean

Lab Description:

For this lab I am going to create an Entity Bean that will represent a business entity object that exists in persistent storage (database). Following are the steps to perform this lab for EAServer.

- Create a new Project in JBuilder.
- Go to Project properties and set the target server to EAServer.
  - Right Click à Properties à Server Tab à Select EAServer.
- Add the required EAServer Libraries in the project Path.
  - Right Click à Properties à Path Tab à Required Libraries à Add à Select EAServer Client à OK.
- Create a new EJB Module in JBuilder as EJB 2.0 compliant.
- Create Database Configuration in JBuilder.
  - Tools à Configure Libraries à Add à Add Jconn2.jar.

- Create a Database URL.
  - Tools à Database Pilot à In Database Pilot window click File à New à in new window enter the following information.
  - Driver = com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
  - RUL = jdbc:Sybase:Tds:localhost:2638
- Create a CMP 2.0 Entity Bean.
  - Type the Bean Name
  - Interface: Remote
  - Click on Classes and Packages and type the package name.
  - Add Primary Id field
  - Add another field like : name

By Entity Bean has been successfully created. Now I need to deploy this bean to EAServer.

- Deploy and Run Entity Bean
  - Create Run time configuration in JBuilder for EAServer.
  - Build your project that will create a .jar file and Sybase specific deployment descriptor.
  - Deploy EJB JAR file to Sybase EAServer.
- Before building the jar file, make sure to override the EAServer Connection settings in EJBModule properties.
  - Right Click on Module à Properties à Deployment à Check the box override connection settings à Enter EAServer connection information .e. user id, host etc.
- Make the Jar File and deploy it to EAServer.
- Test Entity Bean
  - Create an EJB Test client using EJB Test Client Wizard in JBuilder.
  - Code your main method to create your EJB Component and call findMethod and get () method.
    - client.create(empId,empName);
    - client.findByPrimaryKey(empId);
    - client.getEmpName();

- Run the Test Client.
- Debug the Session Bean
  - Assign Breakpoints in your bean.
  - Start EAServer in Debug mode.
  - Run Test Client.

Experiment # 2 on IBM WebSphere

Tasks list of Lab Experiment # 2:

1. Setup database Configuration in WebSphere
2. Create New EJB Project
3. Create CMP Entity Bean.
4. Create a Database in DB2 and Run database.
5. Generate EJB to RDB Mapping for Entity Bean.
6. Configure the Test Server to run Entity Bean.
7. Create a Data Source to DB2 Database.
8. Add the data source to Server Configuration.
9. Deploy and RMIC Code to the Test Server.
10. Run the Test Client and test the Entity Bean

Lab Description:

For this lab I am going to develop an Entity Bean in WebSphere and will use DB2 as a database, because IBM WebSphere doesn't support Sybase ASA database. Therefore I am going to do the same experiment on WebSphere and then monitor the ease of development on both servers. Following are the steps to perform this lab for EAServer.

- Open the J2EE Perspective.
- Create a new EJB Project under EJB Module in WebSphere.
- Create TOP-Down Entity Bean.
  - select New => Enterprise Bean
- Create TOP-Down Entity Bean.
- On the Create an Enterprise Bean dialog:
  - Select the project
  - Select Entity bean with container managed persistence (CMP) fields.
  - Enter Bean namre.
  - Enter ejbModule as the Source folder.
  - Enter Default package.
  - Click Next.
- Check Remote Client View Box.
- Add CMP Attribute
  - Add Id and name attribute and make id as a primary key.
  - Make sure to select Key field while adding Id attribute
  - Also make sure to check the box for promoting Getter and Setter to Remote interface.
- Click Finish

Now CMP Entity Bean has been created but here I need to do few steps to map it to the database.

- Generate EJB to RDB Mapping.
  - Right Click on Package à Generate EJB to RDB Mapping à Create new back end Folder à Next à Select Top-Down à Next à Select DB2 Universal Database Driver v.6.1 à Enter Database Name à Enter Schema Name à Finish.
- After Mapping it creates a table for you and maps it to the DB2 database too.

- Open Deployment descriptor of Entity Bean and make sure you have JNDI name correct.
- Define Data Source Binding.
- Create a new Server Project.
  - From the main menu, select Perspective à Open à Other.
  - Select Server from the dialog and click OK.
  - Right-click anywhere in Navigator view and select New à Server and Server Configuration à Enter the Server Name à Select Test Environment from the list à Finish.
- Add a Project to the new Server.
  - Right Click on server à Add à Select the EJBEAR à Finish.
- Now Define the Data Source under Server Configuration.
  - Right Click on server à Open à Data Source Tab à Add the Data Source for JDBC provider à Select Default DB2Server JDBC à Next à Enter Data source name à JNDI Name à Next à Enter Database Name à Finish.
- Generate Deploy Code
  - Right Click on EJB Project à Generate à Deploy and RMIC Code à Finish.
- Launch the Test Client
  - Right Click on the EJB Project à Select Run on Server.
  - EJB Test Client will open Up à Click JNDI Explorer.
  - Expand com package up-to the bean.
  - Click on Bean and Click Create à work with Objects.
  - Use the Universal Test Client to perform the following tasks:
    - Finding an enterprise bean
    - Calling an EJB method
    - Removing an enterprise bean
    - Viewing fields
    - Viewing methods
    - Verifying the results.
- Verify the Database
  - Go to Database and make sure you have a table and data in there.

Interpretation of Experiment # 2 Results

This experiment includes many steps and configuration settings. I used ASA Database with EAServer for the Entity Bean and DB2 Database with IBM WebSphere. I developed CMP Entity Bean, which represent a business entity object in each database, on each server separately but in the same way because IBM WebSphere doesn't support ASA database and JBuilder 7 doesn't have support WebSphere ver. 5.0 (JBuilder 9 support it). I could just import the same jar file and do rest of the settings related to database and WebSphere; but in this experiment I tried to show how much ease of development WebSphere provides to the developer. Therefore, I created same simple Entity Bean through WebSphere EJB creation wizard.
(Note: I didn't have Oracle database or ASE database that they both support therefore I used same structure in both database to do this experiment)

I have noticed that development of Entity Bean requires more steps and configuration settings on both servers and it is clearly described in lab description above.
Following is more detail of deployment and testing on each server

Deployment and Testing Entity Bean on EAServer:

I was able to deploy my Entity Bean to EAServer directly from JBuilder using the plug-in for EAServer. It deploys the EJB to EAServer and installs it on the server it also creates the stubs, skeleton and Java Connection Cache too. Manual deployment of EJB is the same as in Lab experiment # 1 above.
To deploy Entity Bean from JBuilder to EAServer you need to create a runtime configuration for applications same like I did in experiment # 1.
Before I test my EJB on EAServer, there are couples of changes those are required to run the test successfully. I need to Edit the CMP field mapping and Table name in the component properties. Following are the steps:

Log on the Jaguar Manager à Servers à Installed Packages à FirstEJBMod à Right Click FirstEJB à Component Properties à Persistence tab à under persistence Tab General Tab à Edit the Table field (it has Module name as a prefix of the table, remove that) à Go to Field Mapping Sub tab à **Modify database Column [Type]** values of the CMP fields as follows:
(Note: match the column names, as they should appear in database table)

Field	Database Column [Type]
empId	empid
empName	empname

Testing EJB on EAServer
EAServer doesn't provide any testing tool to test the deployed EJB. In order to test the EJB I had to use JBuilder to create a Test Client again.
The test client runs within JBuilder by creating another tab, TestSesTestClient, in the message pane. You can see the successful execution of the Test Client from the last line

- **Execution time: 80 ms.**
  -- Return value from findByPrimaryKey(1):
  IOR:000000000000003e524d493a636f6d2e746573742e656a622e6669727 ...
  -- Calling getEmpName()
  -- Succeeded: getEmpName()
  -- Execution time: 40 ms.
  -- Return value from getEmpName(): Kashif Ahmed.

After Successful testing I can go in my database and can see the new table is created with the name of **FIRST EJB** and there is one row there with id =1 and name = Kashif Ahmed. Thus during this experiment we mapped and Entity Bean to the database table.

(Note: JBuilder creates a Test Client for you but you need to make sure in the Context Lookup you have **yourmodule/yourBean**, if you don't have lookup like this it will fail to lookup Entity

Bean on the Server. JNDI name of the component can be send through component properties on EAServer.)

Here is the code for the Test Client that was written to Test Entity Bean on EAServer:

```
package com.test.ejb.firstejbproj;

import java.util.*;

import javax.naming.*;
import javax.rmi.*;

public class FirstEJBTestClient1 {
 static final private String ERROR_NULL_REMOTE = "Remote interface reference is null.  It must be created by calling
one of the Home interface methods first.";
 static final private int MAX_OUTPUT_LINE_LENGTH = 100;
 private boolean logging = true;
 private FirstEJBRemoteHome firstEJBRemoteHome = null;
 private FirstEJBRemote firstEJBRemote = null;

 //Construct the EJB test client
 public FirstEJBTestClient1() {
  long startTime = 0;
  if (logging) {
   log("Initializing bean access.");
   startTime = System.currentTimeMillis();
  }

  try {
   //get naming context
   Context ctx = getInitialContext();

   //look up jndi name
//     Object ref = ctx.lookup("FirstEJBRemote");
   Object ref = ctx.lookup("FirstEJBMod/FirstEJB");

   //cast to Home interface
   firstEJBRemoteHome = (FirstEJBRemoteHome) PortableRemoteObject.narrow(ref, FirstEJBRemoteHome.class);
   if (logging) {
    long endTime = System.currentTimeMillis();
    log("Succeeded initializing bean access.");
    log("Execution time: " + (endTime - startTime) + " ms.");
   }
  }
  catch(Exception e) {
   if (logging) {
    log("Failed initializing bean access.");
   }
   e.printStackTrace();
  }
 }

 private Context getInitialContext() throws Exception {
  String hostname="kashi";
  String url = "iiop://kashi:9000";
  String user ="jagadmin";
  String password ="kashi";
  Properties properties = null;
  try {
   properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY, "com.sybase.ejb.InitialContextFactory");
   properties.put(Context.PROVIDER_URL, url);
   properties.put(Context.SECURITY_PRINCIPAL, user);
   properties.put(Context.SECURITY_CREDENTIALS, password);
   Thread curr=Thread.currentThread();
   ClassLoader l=curr.getContextClassLoader();
   ClassLoader c=l;
```

```
    java.net.URL classPath = new java.net.URL("http",hostname,8080,"/classes/");
    haveUrlSomewhere:
{//Check
do
{
            if(c instanceof java.net.URLClassLoader)
            {
                        java.net.URLClassLoader ul = (java.net.URLClassLoader)c;
                        java.net.URL[] urls = ul.getURLs();
                        if(java.util.Arrays.asList(urls).contains(classPath))
                                    break haveUrlSomewhere;
            }
}
while((c=c.getParent()) != null);
curr.setContextClassLoader(java.net.URLClassLoader.newInstance(new java.net.URL[]{classPath},l));
}

    return new InitialContext(properties);
   }
   catch(Exception e) {
    log("Unable to connect to EAServer at " + url);
    log("Please make sure that the server is running.");
    throw e;
   }
  }

  //---------------------------------------------------------------------------
  // Methods that use Home interface methods to generate a Remote interface reference
  //---------------------------------------------------------------------------

  public FirstEJBRemote create(Integer empId, String empName) {
   long startTime = 0;
   if (logging) {
    log("Calling create(" + empId + ", " + empName + ")");
    startTime = System.currentTimeMillis();
   }
   try {
    firstEJBRemote = firstEJBRemoteHome.create(empId, empName);
    if (logging) {
     long endTime = System.currentTimeMillis();
     log("Succeeded: create(" + empId + ", " + empName + ")");
     log("Execution time: " + (endTime - startTime) + " ms.");
    }
   }
   catch(Exception e) {
    if (logging) {
     log("Failed: create(" + empId + ", " + empName + ")");
    }
    e.printStackTrace();
   }

   if (logging) {
    log("Return value from create(" + empId + ", " + empName + "): " + firstEJBRemote + ".");
   }
   return firstEJBRemote;
  }

  public FirstEJBRemote findByPrimaryKey(Integer empId) {
   long startTime = 0;
   if (logging) {
    log("Calling findByPrimaryKey(" + empId + ")");
    startTime = System.currentTimeMillis();
   }
   try {
    firstEJBRemote = firstEJBRemoteHome.findByPrimaryKey(empId);
    if (logging) {
     long endTime = System.currentTimeMillis();
     log("Succeeded: findByPrimaryKey(" + empId + ")");
     log("Execution time: " + (endTime - startTime) + " ms.");
    }
```

```
    }
    catch(Exception e) {
      if (logging) {
        log("Failed: findByPrimaryKey(" + empId + ")");
      }
      e.printStackTrace();
    }

    if (logging) {
      log("Return value from findByPrimaryKey(" + empId + "): " + firstEJBRemote + ".");
    }
    return firstEJBRemote;
  }

  //----------------------------------------------------------------------------
  // Methods that use Remote interface methods to access data through the bean
  //----------------------------------------------------------------------------

  public Integer getEmpId() {
    Integer returnValue = null;
    if (firstEJBRemote == null) {
      System.out.println("Error in getEmpId(): " + ERROR_NULL_REMOTE);
      return returnValue;
    }
    long startTime = 0;
    if (logging) {
      log("Calling getEmpId()");
      startTime = System.currentTimeMillis();
    }

    try {
      returnValue = firstEJBRemote.getEmpId();
      if (logging) {
        long endTime = System.currentTimeMillis();
        log("Succeeded: getEmpId()");
        log("Execution time: " + (endTime - startTime) + " ms.");
      }
    }
    catch(Exception e) {
      if (logging) {
        log("Failed: getEmpId()");
      }
      e.printStackTrace();
    }

    if (logging) {
      log("Return value from getEmpId(): " + returnValue + ".");
    }
    return returnValue;
  }

  public void setEmpName(String empName) {
    if (firstEJBRemote == null) {
      System.out.println("Error in setEmpName(): " + ERROR_NULL_REMOTE);
      return ;
    }
    long startTime = 0;
    if (logging) {
      log("Calling setEmpName(" + empName + ")");
      startTime = System.currentTimeMillis();
    }

    try {
      firstEJBRemote.setEmpName(empName);
      if (logging) {
        long endTime = System.currentTimeMillis();
        log("Succeeded: setEmpName(" + empName + ")");
        log("Execution time: " + (endTime - startTime) + " ms.");
      }
    }
```

```
    catch(Exception e) {
      if (logging) {
        log("Failed: setEmpName(" + empName + ")");
      }
      e.printStackTrace();
    }
  }

  public String getEmpName() {
    String returnValue = "";
    if (firstEJBRemote == null) {
      System.out.println("Error in getEmpName(): " + ERROR_NULL_REMOTE);
      return returnValue;
    }
    long startTime = 0;
    if (logging) {
      log("Calling getEmpName()");
      startTime = System.currentTimeMillis();
    }

    try {
      returnValue = firstEJBRemote.getEmpName();
      if (logging) {
        long endTime = System.currentTimeMillis();
        log("Succeeded: getEmpName()");
        log("Execution time: " + (endTime - startTime) + " ms.");
      }
    }
    catch(Exception e) {
      if (logging) {
        log("Failed: getEmpName()");
      }
      e.printStackTrace();
    }

    if (logging) {
      log("Return value from getEmpName(): " + returnValue + ".");
    }
    return returnValue;
  }

  public void testRemoteCallsWithDefaultArguments() {
    if (firstEJBRemote == null) {
      System.out.println("Error in testRemoteCallsWithDefaultArguments(): " + ERROR_NULL_REMOTE);
      return ;
    }
    getEmpId();
    setEmpName("");
    getEmpName();
  }

  //--------------------------------------------------------------------------
  // Utility Methods
  //--------------------------------------------------------------------------

  private void log(String message) {
    if (message == null) {
      System.out.println("-- null");
      return ;
    }
    if (message.length() > MAX_OUTPUT_LINE_LENGTH) {
      System.out.println("-- " + message.substring(0, MAX_OUTPUT_LINE_LENGTH) + " ...");
    }
    else {
      System.out.println("-- " + message);
    }
  }
  //Main method

  public static void main(String[] args) {
```

```
    FirstEJBTestClient1 client = new FirstEJBTestClient1();
    // Use the client object to call one of the Home interface wrappers
    // above, to create a Remote interface reference to the bean.
    // If the return value is of the Remote interface type, you can use it
    // to access the remote interface methods.  You can also just use the
    // client object to call the Remote interface wrappers.
    String empName = new String ("Kashif Ahmed");
    Integer empId = new Integer (001);

    try {
      client.create(empId,empName);
      client.findByPrimaryKey(empId);
      client.getEmpName();

    }catch (Exception ex ){
      ex.printStackTrace();
    }
  }
}
```

Deployment and Testing Entity Bean on IBM WebSphere:

I created same table in DB2 database under DB2ADMIN schema. Then I created CMP Entity Bean using WebSphere EJB creation Wizard that was not very difficult, just had couples of windows with advance options. As I already mentioned in the lab#1 experiment that in order to run the EJB on WebSphere server, we don't need to write a separate client. IBM WebSphere offers a feature to create a Test Environment and test your EJBs.

But before creating the Test Server environment I had to do several steps to set the Data Source in both EJB deployment descriptor and Server Configuration, verify the JNDI name and have JNDI name setup in server configuration. All steps have already been mentioned in the lab# 2 description above.

Finally, I got a Test Environment for Entity Bean. The learning curve was high as compared to lab # 1 experiment because for Entity Bean, Server requires additional configuration for Data source and mapping to database and it was complex. Also there are not many examples and articles available on web for WSAD 5 because it is a new release.

Once the test environment is created, go to the EJB Module à Right Click à Run on Server à Select the WebSphere Test Environment from the new popup window à Next à Remember to check the box Deploy EJB Bean (Generate EJB Deploy and RMIC code) à Finish.

This will start the Test Server Environment; deploy the EJB and provide a GUI Wizard based interface to test EJB Components. In the JNDI Explorer you can browse your EJB and then click on it à Click on Create à Click on Work with Object button to work with the created object à Invoke Method
Once the object is available, you can browse all the available methods and invoke them and work with them to see the results and test your Entity bean.
To verify the result, I went to my database my table was there and it was the row with the data I created my entity bean.

Conclusion:

- Above experiments shows that development and deployment of Entity Bean on both servers is very different as compared to session bean and involve several more steps, most of them are related to run time configuration settings.
- Once every thing completely and correctly setup for IBM WebSphere Test Environment then Testing of Entity Bean is more interactive and easy as compared to EAServer but time consuming. On the other hand after writing Test Client for EAServer, testing was simple and fast.
- Configuration and development of Entity Bean was typical and tough on IBM Web Sphere as compared to Sybase EAServer.
- Database configuration, mapping and settings involve more steps on IBM WebSphere as compared to EAServer.
- Learning curve was steep on WebSphere as compared to EAServer in Entity Bean deployment and mapping to database too.
- Coding for developer was less on IBM WebSphere as compared to EAServer.
- Server execution remains memory hungry on WebSphere as compared to EAServer.
- In EAServer Database mapping is taking care by server, you just need to make sure the table names and field names are right, on the other hand WebSphere maps the table correctly and you don't need to modify them but you need to do setup database configuration correctly and generate EJB to RDB mapping yourself, it doesn't takes place on run time.

Following Table summarize the amount of work done to Test Entity Bean in terms of mouse click and extra line of code on each server.

No. of steps (mouse click and keyboard hits)	**EAServer**	**WebSphere**
Mouse Clicks to modify the component properties for database configuration	18	25
Keyboard Hits to modify the component properties for database configuration	6	9
Total lines of code written for Test Client	242	0
To Test EJB through server.	N/A	20
To Test EJB using Test Client in JBuilder	4	4
To setup runtime configuration in JBuilder	8	N / A
To deploy ejb.jar file.	2	N/A

Table 1_2

- While setting up the test environment and testing Entity Bean on EAServer, my fingers and brains have to do less work as compared to on WebSphere. Testing Entity Bean on WebSphere involved high brainstorming.

Summary / Conclusion

This section concludes our result, which is based on the analysis and interpretation of both experiments performed on both application servers.

I developed two labs and those were performed on both servers one by one and I monitored the ease of use of servers, time factor, learning curve, how much time it takes to deploy an EJB application on any server and how much server configuration and other settings / steps are required. I examined how user friendly the environment of each server was with respect to rapid development and testing. I also analyzed openness of the server w.r.t. EJBs development and if there is any need to do extra configuration to deploy and install a generated ejb.jar file that contains our ejb application.

- Both application servers provide full J2EE platform support for EJB application development.
- The main role of JBuilder is to have one common tool where I develop my EJBs and then deploy them to each server and then monitor. Also monitor, when I have a J2EE war file, how much configuration and extra steps I need to do in each application server. Although WebSphere itself provide rich development environment for J2EE development but the learning curve is steep and it required significant amount of training.
- The learning curve is steep on WebSphere due to its huge development environment; you need to know in which perspective you are supposed to write your code. What configuration do you need to setup the test environment? You don't just import the jar in WebSphere like EAServer, You need to do extra step by creating new EJB Project, add data source in server configuration, and do EJB to RDB Mapping etc. Therefore to know all these steps it has high learning curve. While in EAServer you just import the file and it create the package and every thing for you. At the most you just need to go to component properties and verify the database information and connection cache information.
- Due to GUI based deployment provided be JBuilder to most J2EE platform application servers' deployment of EJB components is no-brainer and developer doesn't need to do any coding for it except simple run time configuration settings. When we build the EJB in JBuilder, it writes the deployment descriptor for you, It writes it based on the configuration we do, like database, server configuration etc.It writes the xml deployment descriptor file i.e. ejb-jar.xml, which contains all the deployment settings for EJB and both servers, can read those when you import the jar file.
- Developer faces higher complexities on WebSphere as compared to EAServer. Like in lab # 2 experiment developer need to do all the database configuration and settings information to map it and this involve more steps as compared to steps involve for the same task on EAServer.

It can't be said that which server is the best and which server a company should used for J2EE Application Development because both offers full J2EE platform support and it depends on different other factors too like:

- What technical skills developers have?
- How much budget company has to spend on software purchase?
- How much hardware resources are available to the developer?
- What is the existing development platform?
- What is the time frame for project completion?
- and many more …

A prediction how the real students will do on the labs:

Based on the experiments the predictions about the students performing same experiments are:

- Student working on WebSphere will need enough training and training material on EJB development using IBM WebSphere 5.0 as compared to EAServer.
- Students working on WebSphere will have steep learning curve as compared to students working on EAServer, due to very rich development environment provided by WebSphere. WebSphere Studio provides one platform for complete J2EE Application development, therefore students needs to learn more, like where to write the code, what perspective is being used for what kind of component. Like if you are modifying the server configuration, a separate server perspective is provided, similarly for Database etc. On the other hand EAServer doesn't provide any rich development environment and you need to use any third party tool or Sybase PowerJ to do J2EE application development.
- Students working on EAServer will finish the lab early as compared to students working on WebSphere without enough training.
- After completion of significant amount of training students working on WebSphere will do rapid J2EE Application Development as compared to EAServer because WebSphere studio provides complete functionality for application development and for EAServer they would need another tool.
- Students working on EAServer will need to write more code as compared to the students working on WebSphere.
- Students working on WebSphere will be able to test their application first in the test environment provided by WebSphere without publishing it to the production but students working on EAServer won't be.
- Students working on WebSphere will feel more comfortable in testing their EJBs due to provided GUI based test environment by WebSphere as compared to EAServer.

(Note: These predictions have been done only based on my experiments and experience during the research. Performance of students could differ based on individuals.)

Acknowledgement

The author would first like to thank the course advisor, Dr. Ralph Bunker (Professor at Maharishi University of Management) for his full support, help and cooperation throughout the research. Mr. Dean Jones (CEO of PowerObjects) for his supportive ideas to select the research topic and direction on the right path. Mr. Doug Malmgren (Account Executive at PowerObjects) for his help providing various information and help in getting various references and material related to this research. Finally, a special thanks to Loren Corbridge (Senior Product Manager at Sybase e-Business Division) for providing the comparison matrix that was researched by Sybase and allowing me use it.

Appendices

The following is an appendix that lists important relevant information that would be helpful to students to better understand application servers and to perform the described experiments efficiently.

Comparison Matrix

Standards Support	**EAServer 4.2**	**WebSphere 5.0 Server**
Rich Client	Supports Application clients, C/C++ clients, ActiveX, and PowerBuilder clients.	Supports Application clients, C/C++ clients, ActiveX clients (requires ActiveX to EJB bridge).
Java (CORBA)	Yes.	No.
EJB Features		
Dynamic Finder Query	Not supported. Possible extension by providing declarative expressions for dynamic finder queries.	Yes thru custom execute Query dynamic API.
Bulk CMP SQL Operations	Supports inserts/updates/deletes thru CMP Driver Wrapper.	None.
Bean Inheritance	No.	Supported.
Tag Libraries Included	None.	Provides various custom tags for database activities.
Authentication	Supports OS authentication; Custom authentication; JAAS Support.	JAAS supported; Custom authentication; Supports Kerboros V5 for resource adapter authentication; Supports OS, RDBMS, LDAP;

Naming Optimization	None.	JNDI caching to reduce number of remote calls to name server for lookup calls.
EJB Utilities	None. Relies on IDE class and DTD generation capabilities.	Application Assembly Tool (AAT)
IDE Integration	PowerJ, Jbuilder, PowerDesigner for WebServices, Visual Café, Together Control Center.	Jbuilder, Websphere Studio, Visual Café.
Server Administration		
Repository Versioning	Yes.	No.
Web Console	No.	Yes. Multi-user support; Struts and Tiles Implementation.
Wizard based Deployment	No.	Yes.
Administration Customization	Not Possible.	Provides various administration roles (operator, monitor, configurator, administrator) with different sets of admin. permissions;
OEM features	**EAServer 4.2**	**WebSphere 5.0 Server**
Self-optimizing	No.	Yes.
Self-configuring	No.	Yes.
Self-protecting (security)	No.	Yes.
Troubleshooting Tools	None.	Application Server toolkit; RAS Collector Tool for gathering information for IBM service people; Log Analyzer.
Migration tool	None.	Earconvert (also part of AAT) to convert j2ee 1.2 application to j2ee 1.3.

Interpretation of Servers' Installation

- The installation of IBM WebSphere Studio took longer time as compared to installing Sybase EAServer.
- IBM WebSphere requires more physical hard drive space as compared to Sybase EAServer.
- IBM WebSphere takes longer time to start as compared to Sybase EAServer.
- It takes a long time to start on 512 RAM as compared to EAS.
- IBM WebSphere takes more time to shutdown the server as compared to EAServer. EAServer requires less memory as compared to WebSphere.

- Sybase EAServer administration tool, Jaguar Manager (A Java based GUI tool), is slower to start.
- BM WebSphere has richer interface as compared to Sybase EAServer.

(Note[5]: Following statistic are taken on IBM NetVista machine, Intel Pentium III 650 MHz processor with 512 RAM)

IBM WebSphere

WebSphere Studio startup time	Approx: 90 sec to 1 min.
Memory used during startup	Approx: 90,000 KB to 106,000 KB.
Memory used while idle	Approx: 15,000 KB
CPU Usage while starting up	Approx: 85% - 97%
Shutdown Time	Approx: 5 sec.

Sybase EAServer

Jaguar Server startup time	Approx: 15 sec to 25 sec.
Memory used during startup	Approx: 25,000 KB to 30,000 KB.
Memory used while idle	Approx: 13,000 KB
CPU Usage while starting up	Approx: 60% -80%
Shutdown Time	Approx: 1 sec.
Jaguar Manger startup time	Approx: 15 sec to 20 sec.
JagMgr. memory used during startup	Approx: 20,000 KB to 25,000 KB.
JagMgr. memory used while idle	Approx: 21,000 KB
JagMgr. CPU Usage while starting up	Approx: 85% - 90%
Shutdown Time	Approx: 1 sec.

Although Sybase says that minimum memory space requirement for EAServer is 32 MB and 64 MB is recommended, but based on my practical experience during and research one should have at-least 128 MB of RAM for better and efficient performance. Otherwise, Jag Manager (GUI Server Administration Tool) takes enough time to load. Normally Jaguar Server takes up-to 40 MB easily while running any application.

[5] These statistics are performed by me and has been taken on the Intel Pentium III machine with 512 MB RAM memory.

Reference Citations

1. Charles Welty (University of Southern Maine); and David W.Stemple (University of Massachusetts) - Human factors comparison of a procedural and a nonprocedural query language, 1981
2. Enterprise JavaBeans: (2nd Edition) by Richard Monson-Haefel- O'Reilly
3. Professional Java Server Programming J2EE Edition: published by Wrox. (ch: 1, 18-23).
4. Beginning Java 2: (JDK 1.3 Edition) by Ivor Horton- published by Wrox.
5. CS540: Distributed Computing Course: by Gregory Guthrie and Rene DeJong Spring 2001- M.U.M.
6. EJB Development Using Borland JBuilder 8 and Sybase EAServer 4.1.3.: by Sudhansu Pati, Systems Engineer, Borland Software Corporation - January 2003
7. [1] Sun J2EE Tutorial:
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Overview2.html
8. Enterprise Bean:
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts.html
9. Enterprise Java Bean Intro:
http://developer.apple.com/techpubs/webobjects/Enterprise_JavaBeans/Introduction/chapter_2_section_2.html
10. [2] Sybase Web Site:
http://www.sybase.com
11. Online EAServer Installation Guide:
http://sybooks.sybase.com/onlinebooks/group-eag/eap0420e/easvr_ig/@ebt-link;pt=49?target=%25N%13_495_START_RESTART_N%25
12. IBM Developers Network Web Site:
http://www-106.ibm.com/developerworks/
13. IBM WebSphere Technical information Site:
http://www7b.software.ibm.com/wsdd/
14. JBuilder Enterprise Information:
http://www.borland.com/jbuilder/enterprise/index.html
15. Sybase ASA Documentation:
http://www.sybase.com/products/anywhere
16. Dynamo vs. iPlanet Application Server Comparison Matrix – CORVIA Inc
http://www.ugcs.caltech.edu/~werdna/nnh/technology/ Dynamo-iPlanetMatrix.pdf
17. Association of Computing Machinery
http://www.acm.org
18. University of Minnesota Research Department
http://www.cs.umn.edu/research.html
19. University of Minnesota Digital Library
http://sciweb.lib.umn.edu/
20. IT World Site Network
http://www.itworld.com
21. Java Guru World
http://www.jguru.com
22. Java World
http://www.javaworld.com
23. Flashline's Open Source Frameworks Matrix
http://www.flashline.com/components/appservermatrix.jsp?sid=1046444527500-3637481644-51
24. Introduction to WSAD 5.0 – J2EE Introduction – EJB Introduction
http://www.intertech-inc.com

25. WSAD home page
http://www-3.ibm.com/software/ad/studioappdev/
26. IBM WSAD support
http://www-3.ibm.com/software/ad/studioappdev/support/
27. WebSphere related topics
http://www.websphere-world.com/
28. WebSphere Central
http://www.websphere central.com
29. IBM RedBooks
http://www.redbooks.ibm.com/
30. Integrating EJBs into J2EE apps in WebSphere Studio
http://www6.software.ibm.com/reg/devworks/dw-ejbj2ee-i?S_TACT=102B7W63&S_CMP=WSDD
31. Easy, Breezy EJB Tooling in WebSphere Studio
http://www6.software.ibm.com/reg/devworks/dw-ejbtls-i?S_CMP=WSDD&S_TACT=102B7W60

Thanks