PORTING JAVA PET STORE APPLICATION USING JDX™ OR-MAPPER FOR PERSISTENCE

by

Rajini Raju
Chaya Sudindrakumar
(rajini_chaya@yahoo.com)

May 25, 2004

ABSTRACT

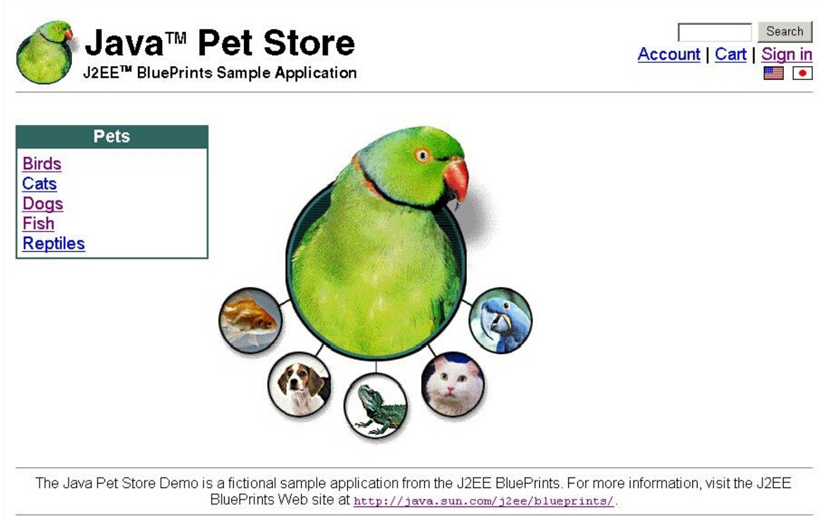
The Java Pet Store represents a typical J2EE e-commerce application, presenting views of products and services for sale. Pet Store calls upon several distributed components (EJB session beans), whose role is to interact with the data layer implemented using EJB components (entity beans) and CMP (Container Managed Persistence). Entity EJBs are pretty heavyweight and hard to use. This report describes how Entity EJBs can be replaced with POJOs (Plain Old Java Objects) and a lightweight Object-Relational Mapping (OR-Mapping) technology provided by JDX product from Software Tree to act as the persistence layer. In addition to greatly simplifying the architecture, this approach also seems to provide substantial performance improvements over the original implementation.

CONTENTS

1. INTRODUCTION

The Java Pet Store application is a decoupled enterprise architecture that can interoperate with existing data sources and business partners' systems, all built on top of the J2EE platform. Pet Store represents a typical e-commerce application, presenting views of products and services for sale. The key design pattern used in the Java Pet Store is the Model-View-Controller (MVC) architecture, which separates three distinct forms of functionality within the application.

The sample application comprises four separate sub-applications that cooperate to fulfill the enterprise's business needs, each of which is a J2EE application. Pet Store calls upon several distributed components (EJB session beans), whose role is to interact with the data layer implemented using EJB components (entity beans) and CMP (Container Managed Persistence).



The Java Pet Store Demo is a fictional sample application from the J2EE BluePrints. For more information, visit the J2EE BluePrints Web site at http://java.sun.com/j2ee/blueprints/.

Java Pet Store Welcome Page

Entity EJBs are pretty heavyweight and hard to use. In this project, we decided to use POJOs (plain old Java objects) and a lightweight Object-Relational Mapping (OR-Mapping) technology provided by JDX product from Software Tree to act as the persistence layer. The primary objective was to assess the design and coding simplification achieved using such architecture. A secondary objective was to measure performance improvements, if any.

This report describes the design, source code, build, and deployment changes to use POJOs with JDX OR-Mapper for the Java Pet Store application. It also provides some performance numbers under different load conditions. We used BEA WebLogic Application Server version 8.1 (eval), Pointbase database version 4.4, and JDX version 4.1 to build the application and used eValid testing tool (eval) for performance measurements.

2. DESIGN CHANGES

The objective of the project was to replace the complexity of using Entity EJBs with the simplicity of using POJOs (plain old Java objects) backed by JDX OR-Mapper for persistence. So the essential design changes involved the following:

- POJOs replace all entity beans of Pet Store application.
- Session beans use POJOs instead of entity beans.
- Session beans use JDX for data access (persistence).

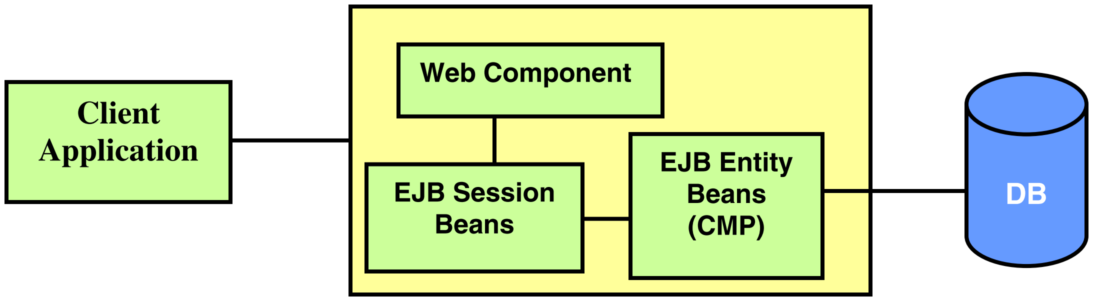


Fig 1.Original Pet Store application high-level architecture using CMP

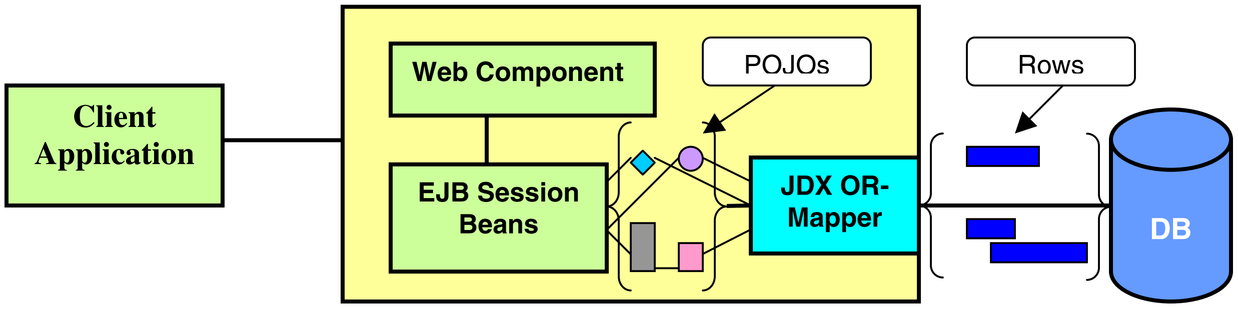


Fig 2. Pet Store application architecture using JDX OR-Mapper for persistence

In the following sections, we describe the code, build, and deployment changes to accomplish the architectural transformation.

3. SOURCE CODE CHANGES

This section explains the source code changes for using JDX OR-Mapper instead of EJB CMP for persistence. The old code is shown with a light gray background color and the new code is shown with an aqua background color. This section is divided into the following subsections:

3.1 Replacing an Entity Bean with a POJO class

For each entity bean class (e.g. UserEJB.java), we created a POJO class (e.g., User_JDX.java) with the following simple changes.

- Replaced the abstract EntityBean declaration with POJO class declaration
- Replaced the ejbCreate () method with a constructor taking the same arguments
- Removed all EJB life-cycle management methods such as ejbPostCreate() and ejbActivate()
- Added an empty constructor
- Added a member variable for each persistent attribute
- Implemented the accessor methods (thus making them concrete methods)

```
//UserEJB.java -- Old
public abstract class UserEJB implements EntityBean {
    private EntityContext context = null;
    // CMP fields
    public abstract String getUserName();
    public abstract void setUserName(String userName);
    public abstract String getPassword();
    public abstract void setPassword(String password);

    // EJB create methods
    public String ejbCreate(String userName, String password)
                                        throws CreateException {
        // code for checking the input data goes here
        setUserName(userName);
        setPassword(password);
        return null;
    }
    public void ejbPostCreate(String userName, String password)
                                        throws CreateException {
    }
    // Business methods
    public boolean matchPassword(String password) {
        return password.equals(getPassword());
    }
    // Misc Method
    public void setEntityContext(EntityContext c) {
        context = c;
    }
    public void unsetEntityContext() {
    }
    public void ejbRemove() throws RemoveException {
    }
    public void ejbActivate() {
    }
    public void ejbPassivate() {
    }
    public void ejbStore() {
    }
    public void ejbLoad() {
    }
}
```

```
 // User_JDX.java   -- New
 public class User_JDX {
    private String userName;
    private String password;

    public User_JDX(){
    }
    public User_JDX(String userName, String password) {
            // code for checking the input data goes here
       this.userName= userName;
       this.password=password;
    }
    public String getUserName() {
       return userName;
    }
    public void setUserName(String userName) {
       this.userName=userName;
    }
    public String getPassword() {
       return password;
    }
    public void setPassword(String password) {
       this.password=password;
    }

    public boolean matchPassword(String password) {
       return password.equals(getPassword());
    }
```

A list of all the entity beans and their replacements is shown in the Appendix A.

3.2 Changes to a Session Bean

For each session bean class (e.g. UserEJB.java), we made the following modifications to use POJOs and JDX:

- Added code to access JDX OR-Mapper handles. Please note that this code (see Appendix E) can conveniently be placed into a common super class for these session beans.
- Modified methods to create entity beans with POJO constructors and JDX insert calls.
- Modified findByPrimaryKey() methods to use JDX named query mechanism (explained in Appendix F).
- Modified other finder methods to use JDX query calls.

The following code blocks shows mainly those methods and variables that are modified by the new implementation.

```
public class SignOnEJB implements SessionBean {

   /**
     * business method used to check if a user is allowed to sign on
     */
   public boolean authenticate(String userName, String password) {
        try {
            UserLocal user = ulh.findByPrimaryKey(userName);
            return user.matchPassword(password);
        } catch (FinderException fe) {
            return false; // User not found, so authentication failed.
        }
   }
   /** business method to create new users **/
   public void createUser(String userName, String password)
                         throws CreateException {
      UserLocal user = ulh.create(userName, password);
   }

}
```

```
public class SignOnEJB implements SessionBean {

   private User_JDX user = null;

   /** Insert JDX OR Mapper Handlers code mentioned in APPENDIX E **/

   public boolean authenticate(String userName, String password) {
      try {
         jxResource_ = checkoutJXResource();
         jdxHandle_ = jxResource_.getJDXHandle();
         Vector qParams = new Vector();  // param values for named quer
         qParams.add(userName);

         Vector users = jdxHandle_.executeNamedQuery("PKQuery",
                 "com.sun.j2ee.blueprints.signon.user.ejb.User_JDX",
                 qParams, -1,0, null);
         if (users.size()==0) {
            return false;
         }
         User_JDX user = (User_JDX) users.firstElement();
         return user.matchPassword(password);
       } catch (Exception ex) {
          return false; // User not found, so authentication failed.
       } finally {
          checkinJXResource();
       }
    }

    /** business method to create new users **/
    public void createUser(String userName, String password)
                   throws  EJBException{
       try {
          jxResource_ = checkoutJXResource();
          jdxHandle_ = jxResource_.getJDXHandle();
          user = new User_JDX(userName, password);
          jdxHandle_.insert(user, 0, null);
       } catch (Exception ex)  {
           throw new EJBException("Could not create user " + userName +
                                  ". Error:" + ex.getMessage());
       } finally {
           checkinJXResource();
       }
    }
}
```

A list of all the modified session beans is shown in Appendix B.

3.3 Changes to a Populator Module

Populator modules are used to load the sample instance data for the given object model in the database. For each populator class (e.g. UserPopulator.java), we made the following modifications to use POJOs and JDX:

- Added code to access JDX OR-Mapper handles.
- Modified methods to populate the database using JDX insert calls.
- Modified findByPrimaryKey() and other finder methods to use JDX query calls.

The following pages show changes to an example module (UserPopulator.java).

A list of all the Populator files modified is shown in Appendix C.

```
// UserPopulator.java – Old
public class UserPopulator {

   /* Initialize attributes here*/
   public UserPopulator() {
      this(XML_USERS);
      return;
   }
   public UserPopulator(String rootTag) {
      this.rootTag = rootTag;
      return;
   }
   public XMLFilter setup(XMLReader reader) throws PopulateException {
      /*……………………………………*/
   }
   public boolean check() throws PopulateException {
      try {
         InitialContext context = new InitialContext();
         UserLocalHome userHome =
                  (UserLocalHome) context.lookup(JNDI_USER_HOME);
         Collection users = userHome.findAllUsers();
         if ((users == null) || (users.size() == 0)) {
            return false;
         }
     } catch (Exception exception) {
        return false;
     }
     return true;
   }
   private UserLocal createUser(String id, String password) throws
                                                    PopulateException {
      try {
         if (userHome == null) {
            InitialContext context = new InitialContext();
            userHome = (UserLocalHome) context.lookup(JNDI_USER_HOME);
         }
         UserLocal user;
         try {
            user = userHome.findByPrimaryKey(id);
            user.remove();
         } catch (Exception exception) {
         }
         user = userHome.create(id, password);
         return user;
      } catch (Exception exception) {
          throw new PopulateException (“Could not create: “ +
          exception.getMessage(), exception);
      }
   }
}
```

```
// UserPopulator.java - New
public class UserPopulator {

   /* Initialize attributes here*/
   public UserPopulator() {
      this(XML_USERS);
      return;
   }
   public UserPopulator(String rootTag) {
      this.rootTag = rootTag;
      return;
   }
   /** Insert JDX OR Mapper Handlers code mentioned in APPENDIX E **/
   public XMLFilter setup(XMLReader reader) throws PopulateException {
    /*…………………………………………*/
   }
   public boolean check() throws PopulateException {
      try {
         jxResource_ = checkoutJXResource();
         jdxHandle_ = jxResource_.getJDXHandle();
         Vector users=jdxHandle_.query
                  ("com.sun.j2ee.blueprints.signon.user.ejb.User_JDX",
                    null,1,0,null);
         if ((users == null) || (users.size() == 0)) {
            return false;
         }
      }  catch (Exception exception) {
           return false;
      } finally {
          checkinJXResource();
      }
      return true;
   }
   private User_JDX createUser(String id, String password)
                                  throws   PopulateException {
      try {
          try {
             jxResource_ = checkoutJXResource();
             jdxHandle_ = jxResource_.getJDXHandle();
             Vector users = new Vector();
             users=jdxHandle_.query
                   ("com.sun.j2ee.blueprints.signon.user.ejb.User_JDX",
                    "userName='"+id+"'",1,0,null);
             if (users .size() > 0) {
                jdxHandle_.delete(users.elementAt(0), 0, null);
             }
          } catch (Exception exception) {
          }
          user = new User_JDX(id, password);
          jdxHandle_.insert(user, 0, null);
          return user;
      } catch (Exception exception) {
           throw new PopulateException ("Could not   create: " +
                                  exception.getMessage(), exception);
      }
   }
}
```

3.4 Sequence Generators

In original Java Pet Store, the container takes care of automatic primary key generation for CMP entity beans with the help of application server specific deployment descriptor files.

In the new implementation, we have used JDX OR-Mapper's sequence generation mechanism to generate primary keys for POJO objects. In JDX, a named sequence can be defined declaratively in the mapping file. For example, the following specification defines a sequence "SEQUENCE_GEN" with an initial value of 1001.

```
SEQUENCE SEQUENCE_GEN START_WITH 1001
```

For runtime, we used JDXSeqUtil class to conveniently get persistently unique sequence numbers for primary key initialization, as illustrated in the following code block.
.

```
// Sequence generator initialization code
JDXSeqUtil jdxSeqUtil = new JDXSeqUtil("SEQUENCE_GEN", 20);

// Usage of sequence generator
address.setPrimaryKey((int)jdxSeqUtil.getNextSeq(jdxHandle_));
```

In order to accomplish sequence generation, a utility module, *`jdxsequtil.java`*, was created and placed in the directory *`C:\petstore1.3.1_02\src\components\util\jdx\src\com\sun\j2ee\blueprints\util\jdx`*

Public getter and setter methods for primary key were added in the POJO files, which use JDX sequence generator.

A list of all the files using Sequence generators is shown in Appendix D.

3.5 OR-Mapping File (petstore.jdx)

JDX provides an innovative declarative way of specifying human readable OR-Mapping information based on a simple grammar. The mapping specification contains mapping information for all the persistent classes belonging to an application domain. It includes, among other things, table names, primary key attributes and object relationships. This mapping specification can conveniently be put in a text file (ORMFile) that can be created in many different ways including using a text editor, a modeling tool, JDXStudio or even programmatically. An ORMFile serves to create a JDX mapping unit that provides a runtime exchange for Java business objects and relational data.

The following snippets from *`petstore.jdx`* file describe the mappings for 2 different classes. Example 1 shows mapping for User_JDX class to UserEJB table in the database. The attributes userName (primary key) and password are mapped to columns USERNAME and PASSWORD column respectively. PKQuery defines a JDX named query.

Example 2 shows the mapping for Account_JDX class. Among other things, it also shows mappings for complex attributes (creditCard and contactInfo) involving cascade-delete (BYVALUE) relationships.

Example 1: JDX Mapping for class USER_JDX

```
CLASS com.sun.j2ee.blueprints.signon.user.ejb.User_JDX TABLE UserEJB
   PRIMARY_KEY userName
   SQLMAP FOR userName COLUMN_NAME USERNAME
   SQLMAP FOR password COLUMN_NAME PASSWORD
   QUERY_NAME PKQuery PREDICATE 'userName=?'
;
```

Example 2: JDX Mapping for class Account_JDX

```
CLASS com.sun.j2ee.blueprints.customer.account.ejb.Account_JDX TABLE AccountEJB
   PRIMARY_KEY primaryKey
   SQLMAP FOR primaryKey COLUMN_NAME PRIMARYKEY
   SQLMAP FOR status COLUMN_NAME STATUS
   IGNORE jxResourcePool_ jxResource_ jxSession_ jxResourcePoolInitialized_ jdxHandle_
   IMPLICIT_ATTRIB CREDITCARD_PRIMARYKEY ATTRIB_TYPE int
   IMPLICIT_ATTRIB CONTACTINFO_PRIMARYKEY ATTRIB_TYPE int
   RELATIONSHIP creditCard REFERENCES
       !com.sun.j2ee.blueprints.creditcard.ejb.CreditCard_JDX
          BYVALUE WITH CREDITCARD_PRIMARYKEY
   RELATIONSHIP contactInfo REFERENCES
       !com.sun.j2ee.blueprints.contactinfo.ejb.ContactInfo_JDX
           BYVALUE WITH CONTACTINFO_PRIMARYKEY
;
```

To learn more about the OR-Mapping configuration files, please refer to JDX User Manual.

The file *`PetStoreORMapping.config`* contains settings to initialize JDX subsystem in the class InitPetStoreJDXServlet.

The files *`petstore.jdx`* and *`PetStoreORMapping.config`* are located in the *`\JDXConfig`* directory of the distribution.

4. DIRECTORY STRUCTURE CHANGES

Here is a summary of the directory structure changes for using JDX backed POJO objects in Pet Store. Assuming:

PETSTORE_HOME: Root of the Pet Store folder (e.g., *c:\petstore1.3.1_02*)
BEA_HOME: Root of WebLogic 8.1 server installation (e.g., *c:\bea*), and
JX_HOME: Root of JDX software installation (e.g., *c:\jdx4.1*)

- Created new folder "jdx" under *PETSTORE_HOME\src\components\util\* and a directory structure in parallel to the "tracer" folder in *PETSTORE_HOME\src\components\util*

  Under the "jdx" folder:

  - Source files *InitPetStoreJDXServlet.java* and *JDXSeqUtil.java* are created in the directory *PETSTORE_HOME\src\components\util\jdx\src\com\sun\j2ee\blueprints\util\jdx*

  - The build process automatically places the corresponding class files in the directory *PETSTORE_HOME\src\components\util\jdx\build\classes\com\sun\j2ee\blueprints\util\jdx*

- Created new folder *\JDXConfig* to hold OR-Mapping configuration files *petstore.jdx* and *PetStoreORMapping.config* .

5. BUILD CHANGES

Changed `build.xml` in the following directories:

A `build.xml` file is created in

- *`PETSTORE_HOME\src\components\util\jdx\src`* similar to the one present in *`PETSTORE_HOME\src\components\util\tracer\src`*

- In *`PETSTORE_HOME\src\components\build.xml`* file, the following lines have been added.

  `<ant dir=”util/jdx/src” target=”core”/>` (under target name=”core”)
  `<ant dir=”util/jdx/src” target=”clean”/>`(under <target name=”clean”)

6. WEBLOGIC CONFIGURATION CHANGES

To use JDX OR-Mapper with WebLogic application server, following lines were added to the startup file
`BEA_HOME\user_projects\domains\petstore\startPetStoreWebLogic.cmd`

- `set JX_HOME=c:\jdx4.1`
- `set JDX_OPTS=”-DJX_HOME=%JX_HOME%”`
- `set CLASSPATH=%CLASSPATH%;%JX_HOME%\classes\jxclasses.jar;c:\petstore1.3.1_02\src\components\util\jdx\build\classes;c:\petstore1.3.1_02\src\components\signon\build\classes;………`

Note: In the above `CLASSPATH`, like the Signon component, we have also included the classes folders under all other components present in `c:\petstore1.3.1_02\src\components` and `c:\petstore1.3.1_02\src\apps` directories.

7. SUMMARY OF THE PORTING STEPS TO USE JDX OR-MAPPER IN JAVA PET STORE

We downloaded all the required software (petstore1.3.1_02, JDX 4.1, Weblogic 8.1) and followed the steps of sections 2 to 6 . For instance, we created POJO files (with suffix _JDX) to replace the entity beans of the original Pet Store application (refer to section 3.1). We changed the session bean files, populator files and created sequence generators as outlined in sections 3.2, 3.3, and 3.4.

We also made the following changes to the original Pet Store source files:

- To initialize the JDX subsystem, we added two new source files, *`InitPetStoreJDXServlet.java`* and *`JDXSeqUtil.java`*, as mentioned in section 4.

- To initialize database for JDX, we added two new tables, JDXSEQUENCE and JDXMETADATA, to the existing Pet Store database using the following commands that invoke JDXSchema utility program.

  ```
  C:\JDXConfig> forward –metacreate
  C:\JDXConfig > forward –metainit
  ```

Note: Other Instructions to run Pet Store with JDX on Weblogic application server are given in detail in the file *`README.html`* provided with the distribution of the software.

8. PERFORMANCE TESTING

Performance testing (*Load Testing)* was done using *e-Valid*, a browser based testing tool. Using this tool we created test scripts, which simulated 1-50 users accessing certain pages, starting from the home page to the order confirmation page of an item selected (Homepage-> Item Selection Page-> Sign on Page-> Check out Page-> Order Page), from the Pet Store site. The tool generated test results in the form of graphs and charts.

Server throughput was calculated using this equation:

Throughput = Number of requests / Elapsed time (min)

Each test script of the tool was run *with 'cache on'.* So results of the tests were considered after the first two runs of the application.

Hardware and Software Configuration

Hardware and OS	Software
• 2.40 GHz Intel Pentium4 • 256 MB RAM • 40 GB Hard Disk • Win XP Professional (v 2002, SP 1)	• JDK Version 1.4.1_03 • BEA WebLogic Server Version 8.1 • PointBase RDBMS Version 4.4 • JDX OR-Mapper Version 4.1

We used out-of-the-box configuration of WebLogic, PointBase, and JDX. No special tuning for the application or the environment was attempted.

Unfortunately we cannot publish the performance numbers without explicit permission from the vendors. We will augment this section with actual results after receiving appropriate permissions. In the meantime, you are welcome to run the performance tests and get the numbers for your own settings. The details on running the performance tests are included with the Pet Store implementation package available from Software Tree's web site. Here is a sample result:

Table 1. Sample load test results for n users

Applications	**Fastest Load Time (sec)**	**Slowest Load Time (sec)**	**Avg. Load Time (sec**	**Elapsed Time (sec)**	**Throughput (no. of reqs. / min)**
Original JPS 1.3 on Weblogic 8.1	X1	Y1	Z1	T1	N1
JPS 1.3 with JDX on Weblogic 8.1	X2	Y2	Z2	T2	N2

CONCLUSION

The project has shown that the complexity of using EJB Entity Beans can be removed by using POJO (Plain Old Java Object) classes with JDX, a lightweight OR-Mapper provided by Software Tree. We have outlined the easy steps needed to convert Pet Store, a representative J2EE application, from using EJB with CMP (Container Managed Persistence) to using POJOs with JDX (OR-Mapper for Persistence). The resulting application is not only simpler, but also performs much better as proven by load testing with out-of-the-box settings.

Even though Entity Beans offer advantages like transaction and security management, resource pooling, JNDI (Java Naming and Directory Interface), and component lifecycle management, the following disadvantages make them hard to use in many situations.

Entity Bean Disadvantages:

- Forces the use of a heavy component mechanism for fine-grained business objects.
- More complex than JDX, limiting developer productivity.
- Inheritance not supported.
- Cannot be used for persistence in non-application server environments.
- There is no dynamic query mechanism to lookup entity beans (finders are specified at compile time).
- It is not easy to write unit tests for beans, as it is not possible to use them outside of an application server.

JDX Advantages:

- Works with POJOs providing all the associated object-model advantages like inheritance and optimized placement of business logic.
- Lightweight and efficient mapping engine.
- High performance and scalable implementation.
- Small set of simple and flexible APIs.
- Smart and elegant mapping specification.
- Supports many different application architectures; works with JSP, Servlets, EJBs and standalone programs.
- Works with most popular databases, existing schema, and application servers.
- Easy to learn and easy to use**.**

Although this report details the steps taken to transform the existing Pet Store application to use a simpler persistence model with JDX OR-Mapper, it should be easy to infer that using JDX will also benefit development of fresh Java/J2EE applications.

APPENDIX A: ENTITY BEANS

The following table lists the names of the entity beans and the corresponding POJO classes replacing them in the new implementation using JDX OR-Mapper for persistence. For example, UserEJB.java is replaced by User_JDX.java.

The components 1-12 are present in the directory
`c:\petstore1.3.1_02\src\components\*`

The component No. 13 is present in the directory
`c:\petstore1.3.1_02\src\apps\supplier\src\com\sun\j2ee\blueprints\supplier`

No.	COMPONENT	ENTITY BEAN	POJO OBJECT
1.	Signon	UserEJB.java	User_JDX.java
2.	Uidgen	CounterEJB.java	Counter_JDX.java
3.	processmanager	ManagerEJB.java	Manager_JDX.java
4.	Lineitem	LineItemEJB.java	LineItem_JDX.java
5.	Customer	CustomerEJB.java	Customer_JDX.java
6.	Address	AddressEJB.java	Address_JDX.java
7.	Creditcard	CreditCardEJB.java	CreditCard_JDX.java
8.	Contactinfo	ContactinfoEJB.java	ContactInfo_JDX.java
9.	purchaseorder	PurchaseOrderEJB.java	PurchaseOrder_JDX.java
10.	supplierpo	SupplierOrderEJB.java	SupplierOrder_JDX.java
11.	account	AccountEJB.java	Account_JDX.java
12.	profile	ProfileEJB.java	Profile_JDX.java
13.	inventory	InventoryEJB.java	Inventory_JDX.java

APPENDIX B: SESSION BEANS

The following table lists the names of the session beans along with the POJO classes they are using within their new implementation employing JDX OR-Mapper for persistence. For example, SignOnEJB.java uses User_JDX.java.

Session beans 1-3 are located in the path `c:\petstore1.3.1_02\src\components.`

Session bean No.4 is located in `c:\petstore1.3.1_02\src\apps\opc`,

Session bean No.5 is located in `c:\petstore1.3.1_02\src\apps\petstore`, and

Session bean No.6 is located in `c:\petstore1.3.1_02\src\apps\supplier`.

No.	SESSION BEAN	POJO OBJECT
1.	SignOnEJB.java	User_JDX.java
2.	ProcessManagerEJB.java	Manager_JDX.java
3.	UniqueIdGeneratorEJB	Counter_JDX.java
4.	OPCAdminFacadeEJB.java	PurchaseOrder_JDX.java
5.	ShoppingClientFacadeLocalEJB.java	Customer_JDX.java
6.	DisplayInventoryBean.java, OrderFulfillmentFacadeEJB.java	Inventory_JDX.java

APPENDIX C: POPULATOR FILES

The following table lists the populator classes, which populate the Pet Store demo database using JDX OR-Mapper.

These classes are located in the directory *`c:\petstore1.3.1_02\src\apps\petstore\src\com\sun\j2ee\blueprints\petstore\tools\populate`*.

No.	**POPULATOR FILES**
1	UserPopulator.java
2	CustomerPopulator.java
3	AccountPopulator.java
4	CreditCardPopulator.java
5	ContactInfoPopulator.java
6	AddressPopulator.java
7	ProfilePopulator.java

APPENDIX D: SEQUENCE GENERATORS

The following table lists the POJO classes, which use the JDX Sequence generator mechanism to initialize the primary key attributes of new instances.

No.	SEQUENCE GENERATORS
1	Address_JDX
2	Account_JDX
3	CreditCard_JDX
4	ContactInfo_JDX
5	Profile_JDX
6	LineItem_JDX

APPENDIX E: MANAGING JDX SUBSYSTEM HANDLES

The following common code is used to initialize and use JDX subsystem handles. Please note that this code can conveniently be placed into a common super class for those session beans, which need to interact with persistent POJOs.

```
private boolean jxResourcePoolInitialized_ = false;
private JXResourcePool jxResourcePool_;
private JXResource jxResource_ = null;
private JXSession jxSession_ = null;
private JDXS jdxHandle_ = null;

/* This method initializes the initializing parameters. */
public void init() {
   if (!jxResourcePoolInitialized_) {
      jxResourcePool_ = InitPetStoreJDXServlet.getJXResourcePool();
      jxResourcePoolInitialized_ = true;
   }
}

/* JDX checkin checkout methods */
public JXResource checkoutJXResource() {
   init();
   if (jxResource_ == null) {
      jxResource_ = (JXResource) jxResourcePool_.getResource();
   }
   return jxResource_;
}
/* This method releases the database resource handle into the pool
   of resource handles. */
public void checkinJXResource() {
   if (jxResource_ != null) {
      jxResourcePool_.releaseResource(jxResource_);
   }
   jxResource_ = null;
}
```

Notice that JDX provides the resource pooling utility components (JXResource and JXResourcePool) to help develop multi-client applications. These components may be used to simplify the creation of an extensible pool of JX/JDX handles and share those handles efficiently in a multi-threaded application. The above code uses the static method getJXResourcepool() in InitPetStoreJDXServlet class, which returns a pool of resource handles.

It is interesting to note that the code can be simplified by using aspect-oriented programming (AOP). In particular, the code to checkout/checkin JDX handles can easily be refactored with an advice.

APPENDIX F: NAMED QUERIES

JDX provides the facility to define a named query with optional parameter markers. The named query can be defined once and used (executed) multiple times. Named queries can provide greater convenience, more flexibility, and better performance in many situations. The query will be executed by JDX using a prepared statement for faster performance. The named query is automatically defined and executed for all the subclasses.

For example consider SignOnEJB.java(new) mentioned earlier. There we have used a named query (PKQuery) to retrieve a User object having a given username as the following code shows:

```
Vector qParams = new Vector();
qParams.add(userName);
Vectorusers=jdxHandle_.executeNamedQuery(“PKQuery”,
        “com.sun.j2ee.blueprints.signon.user.ejb.User_JDX”,
         qParams, -1,0, null);
```

In the mapping file (*petstore.jdx*), the following specification defines the named query PKQuery for the class User_JDX class.

```
 QUERY_NAME PKQuery PREDICATE ‘userName=?’
```

ACKNOWLEDGEMENTS

We want to thank our friends Vidya Hungud and Gayathri Prakasam for their help in carefully reviewing this document for readability and usability. Chaya wants to thank her husband Jayaprakash for his cooperation and support during the project work. Finally, we want to thank Damodar Periwal, the architect of JDX OR-Mapper, for his help throughout the project.

RESOURCES

- To download Sun's Java Pet Store Demo V.1.3.1_02, go to
  http://java.sun.com/developer/releases/petstore/petstore1_3_1_02.html
- To download Software Tree's JDX 4.1 OR-Mapper, go to
  http://www.softwaretree.com/
- To download BEA WebLogic 8.1 Application server, go to
  http://commerce.bea.com/index.jsp
- To download Software Research's e-Valid 4.0 Testing tool, go to
  http://www.soft.com/eValid/Products/Download.40/down.evalid.40.phtml