ORACLE-CERN CHAPTER

CERN's EDH APPLICATION

Oracle Customer Experience

Primary Authors:

Derek Mathieson, CERN

Jurgen De Jonghe, CERN

Contributors from Oracle Corporation:

Mike De Groot

Anand Ramakrishnan

Arun Srinivasan

Ellen Barnes

Kuassi Mensah

Moe Fardoost

Roel Stalman

Sudhakar Ramakrishnan

TABLE OF CONTENTS

CERN 4

EDH Application 4

Why move to J2EE technologies? 5

Why Use EJB? 5

Why Choose Oracle J2EE products? 6

The EDH Component Model 6

EDH DOCUMENT SERVLET 7
EDH SERVLET EXECUTOR 7
COMMON BUSINESS OBJECTS 7
COMMON INPUT OBJECTS 8
RUNTIME SCENARIO 8

Migration TO EJB: First Steps 10

THE CURRENCY BEAN 11
THE CONVERTER BEAN 14
TAKING STOCK 14

The CERN Material Request 15

THE CERN STORES CATALOG 16
THE MATERIAL REQUEST BEAN 16
Persistence Methods 17
OBJECT RELATIONSHIPS 17
Business Methods 18
The Material Request Line Items 18

Deployment Descriptors 19

THE MATERIAL REQUEST BEAN 19
Transaction Control 19
EJB Relationships 19
Deployment Environment 20
THE MATERIAL REQUEST EXECUTOR (SESSION) BEAN 21
Transaction Control 21
THE CATALOGITEM BEAN 21
Obtaining the Delivery Lead Time 21

Putting It All Together 22

THE ORACLE ENTERPRISE JAVA ENGINE (ORACLE EJE) 23
- *Accelerator* 1
- *Generational Scavenging Garbage Collector* 1
- *Scalability* 1
- *Database Integration* 1

JVM OVERLOAD 24

LOAD BALANCING 25

CERN's experience 26

EXPENSIVE ENTITIES 26

ORACLE EJE ACCELERATOR 26

Future Work 27

References 27

CERN

CERN is the leading particle physics research laboratory. Some 6,500 scientists, half the particle physicists in the world, use the facilities at CERN. The business of CERN is pure science, exploring the most fundamental questions of nature. The tools of the laboratory, particle accelerators and detectors, are among the largest and most complex scientific instruments in the world. CERN is currently working on the construction of a new 27-kilometer accelerator, the Large Hadron Collider (LHC), due for completion in 2005. When CERN was established in the 1950s, it set the standard for European Collaboration in science—with the LHC, it is set to become the first truly global laboratory. However, as with many businesses in the current economic climate, CERN is expected to continue growing at the same time that staff levels are planned to shrink in the coming years—in essence, achieving higher productivity with less resources. One way of accomplishing this is with the use of fast, efficient, and streamlined organization-wide electronic workflow.

EDH Application

Among computer scientists, CERN is more often remembered as being the birthplace of the World Wide Web (Tim Berners-Lee and Robert Cailliau) than an organization that has been awarded several Nobel Prizes in Physics. In this context, it is important to remember that the World Wide Web was developed to meet the needs of CERN physicists, collaborating and exchanging information in a global physics community. Similarly, an organization-wide e-business system at CERN must be available and must meet the requirements of the physicists and engineers working or collaborating with CERN, whether they are on the CERN site in Geneva or working from their home institutes in California, Moscow, or Delhi. Currently, CERN collaborates with more than 500 such institutes around the world.

CERN's internal e-business application is known as EDH (Electronic Document Handling). EDH currently has more than 5000 active users, with more than 1000 different users a day. An electronic document is processed every 20 seconds. The system is multilingual, Web-based using a Java servlet architecture, and runs Oracle Workflow as the routing engine. Oracle Workflow enables information to be routed in accord with the business rules.

CERN uses EDH to:

- Purchase any of 16,000 standardized items from the CERN Stores Catalogue
- Create purchase requisitions that are processed and transmitted to any of CERN's 20,000 suppliers or any new supplier in the world
- Create a request to import or export goods
- Create a request to attend a course from our on-site training catalogue
- Create a request to attend external training, conferences, or other events
- Create a request for vacations time
- Create a request for overtime compensation
- Create a request for additional human resources for a project or activity

The preceding tasks, using EDH, generate about 100,000 electronic forms a year. Paper forms are no longer used for any of the above tasks. Other smaller, low-volume procedures are also under consideration for integration into EDH.

EDH understands the structure, roles, and responsibilities of the organization and so can employ this knowledge to accelerate procedures even further. For example, EDH never sends a document to someone who is absent. If a document is not signed within a given timeframe, then it is automatically routed to a deputy or someone else with equivalent responsibilities in that domain. Oracle Workflow also offers tools to show and edit the workflow graphically, thereby fostering streamlining and standardization of procedures across the organization. In addition, Oracle Workflow supports both synchronous and asynchronous processes and enables post-notification, such as functions to be executed. The net result is that the average processing time of an e-document takes less than a few hours, compared with days, if not weeks, for the previous paper version.

Why move to J2EE technologies?

Using servlets with (JDBC), access to a database can easily serve dynamic content. But this approach lacks scalability if the Web applications become more complex, or the client base grows. A key performance measure for the Web is the speed with which content is served to users. Caching is one of the key strategies to improve performance. Oracle addresses this problem with Oracle Web cache to improve the application server performance, and a database cache to improve the performance in the database tier. In addition, the Internet calls for rapidly changing user interfaces, making it important to clearly separate the business logic layer from the presentation layer.

More particularly, our goal is to decrease the complexity of the application development by using the system-level services of the EJB container, including security, resource pooling, persistence, concurrency, and transactional integrity. By acquiring off-the-shelf components, the sheer amount of code developed in-house is reduced, at the same time increasing internal reusability within the confines of a standardized component model. Use of frameworks enables developers to productively build application components out of smart, reusable parts. Oracle offers a comprehensive component framework and tool set for writing server-side business applications. Called Business Components for Java, the framework significantly simplifies building Java and XML-based application components.

The J2EE platform thus emerges as an industry standard, and many vendors provide solutions in the area of UML tools, Integrated Development Environments, EJB application servers, monitoring tools, and so on. By using the best products that come out of this competitive environment, and by adhering to standard, well-documented J2EE technologies, the goal of CERN is to reduce the learning curve for its new team members, allowing them to quickly integrate into their projects. Because CERN is partly a training facility, it has a particularly large turnover of temporary student programmers who work on an application for only a short time.

Why Use EJB?

Although the EDH application has existed since 1990, a complete rewrite in Java was undertaken in 1998, with the relatively new EJB specification in mind. Unfortunately, at the time few commercial application servers implemented the EJB specification, and those that did were thought to be insufficiently scaleable for the relatively large user population at CERN.

As a compromise, we implemented a component architecture closely following the EJB specification, with the intention of migrating to real EJBs once the commercial implementations of the technology matured. The EDH development team believes that this is indeed the case now.

Why Choose Oracle J2EE products?

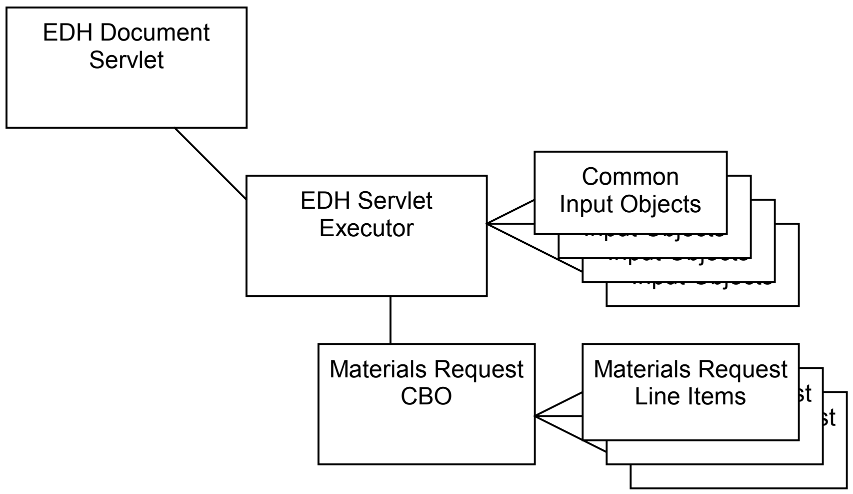
Given the existing business relationship between CERN and Oracle, it seemed logical to start the J2EE journey at Oracle. Although the multi-vendor support is a major argument in favor of J2EE, Oracle's J2EE offering is a compelling choice, because Oracle offers products for all J2EE tiers.

Oracle9*i* offers a unified Java-XML-SQL environment for both the Oracle9*i* Database and Oracle9*i* Application Server. From native data storage (such as XML documents) to standards-based data access (such as JDBC) to common APIs for complex application processing (such as messaging, queuing, transactions, multimedia and file management, online analytical processing, and data warehousing), the three languages can be used to write applications. This approach ensures that applications rich in these languages will never suffer from inter-language incompatibilities, and allows Oracle to ensure optimal performance and scalability.

Furthermore, Oracle is uniquely positioned to distribute these logical tiers over different physical tiers, depending on the operational requirements. Oracle9*i* Application Server and Oracle9*i* database have a complete implementation of the J2EE enterprise APIs, including Servlet, EJB, JSP, XML, JMS, and JNDI. Oracle9*i* Application Server (Oracle9*i*AS) builds on Apache's extensible architecture by adding modules to extend the core functionality. Oracle9*i*AS Web Cache provides cacheability rules for both static and dynamic content created by J2EE components. Oracle9*i*AS intelligently distributes loads across multiple servers and provides no single point of failure using its sophisticated listener-dispatcher architecture, Web cache load balancing, and architecture of Oracle HTTP Server. In particular, all logical tiers can be collapsed into the Oracle9*i* database featuring EJB 1.1 support and an integrated servlet engine. Because the EJBs reside inside the database, access to the database can be highly optimized. Servlets running in the Oracle Enterprise Java Engine have direct in-memory access to EJBs and benefit from a shorter-path JDBC driver. Another boost in performance can be expected from the ability to compile the final production code into native code that gets linked dynamically with the database.

The EDH Component Model

As mentioned previously, the existing EDH architecture is based on the EJB specification, but does not use any commercial EJB container. The following diagram summarizes the main components of the existing EDH architecture.



EDH Component Model Architecture Diagram

EDH Document Servlet

Each electronic document in EDH has its own specific Document Servlet. The role of the document servlet is that of a dispatcher. It uses a cookie stored on the user's browser to identify the user's Servlet Executor instance and then forwards the call directly to it.

EDH Servlet Executor

The Servlet Executor is the component that holds the *conversational state* information about the current interaction. An instance of a Servlet Executor exists for each open document on the user's screen. Its role is to hold references to the Common Business Object representing the document being edited and its user interface components, and to coordinate their activity. (See "Runtime Scenario" on page 9). Oracle EJE addresses the scalability of such stateful components with a VM architecture that reduces the response time by making sure that each session perceives its own *virtual* Java VM, Java global variables, threads, and garbage collector. The underlying runtime supplies the needed scalability. This enables the VM to efficiently perform memory management, provide a low session footprint, and advance garbage collection.

Common Business Objects

Within the EDH application, the equivalent of an Entity Bean is the Common Business Object (CBO). This object effectively provides an object representation of one of our business objects—for example, a purchase order, a person, or a currency. CBOs provide `getXXX()` and `setXXX()` methods for their properties, implement access control, have the ability to be persisted to the EDH database, and implement any necessary business logic (such as calculation of totals) and business rules (such as requiring mandatory fields, permitting only allowed types of cost center, and so on).

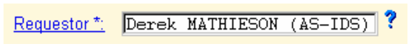
In accord with the EJB specification, we get our CBOs from a *Home* interface through its `findByXXX()` and/or `create()` methods. An EJB client first obtains a reference to the home interface with the Java Naming and Directory Interface (JNDI) mechanism. Oracle EJE behaves like a namespace server by implementing a JNDI interface resulting in location transparency, which is particularly useful when changes must be made to the server environment that do not seriously disrupt the client. In addition, the namespace has fast access to indexed entries in the cache, resulting in performance improvement.

Common Input Objects

The Common Input Objects (CIOs) are a set of classes responsible for building the user interface for our Web applications. Each CIO represents a single data type within the document. CIOs can be Numbers, Dates, and Text, as well as more complex objects such as People, Currencies, and Suppliers.

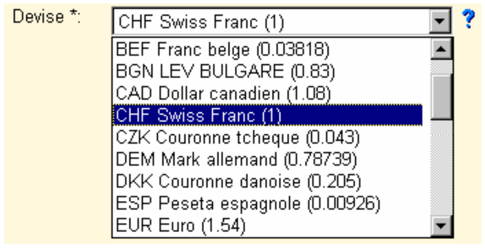
Within an EDH document, a CIO instance is created for each field. When the HTML form is submitted, the CIO is responsible for parsing the data from the user and performing some basic validation, such as maximum string length for Text input objects, or verifying if the person exists for Person input objects. One of the advantages in moving to frameworks like Oracle Business Components for Java is that it provides mechanisms for defining, implementing, and executing validation logic in the business logic tier. The validation framework provides a consistent programming model that hides internal implementation details.

When a response is sent back to the user, the CIO is capable of creating an HTML representation of its value that can be substituted into an HTML template.



Requestor *: Derek MATHIESON (AS-IDS) ?

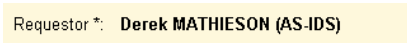
Person CIO



Devise *: CHF Swiss Franc (1) ?

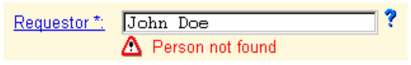
BEF Franc belge (0.03818)
BGN LEV BULGARE (0.83)
CAD Dollar canadien (1.08)
CHF Swiss Franc (1)
CZK Couronne tcheque (0.043)
DEM Mark allemand (0.78739)
DKK Couronne danoise (0.205)
ESP Peseta espagnole (0.00926)
EUR Euro (1.54)

Currency CIO



Requestor *: **Derek MATHIESON (AS-IDS)**

Person CIO (Read Only)



Requestor *: John Doe ?
Person not found

Person CIO (Error State)

User Interface Generation Employing CIOs

The preceding examples illustrate two CIOs in operation. As you can see, the CIO can represent itself in a manner that is appropriate to the data type and can also take into account the user's preferred language (the Currency CIO is rendered in French). Oracle supports one of the richest sets of multi-language character support in the market, incorporating many open and vendor-specific character sets within the database. Among the standards supported by Oracle is UTF8, or Unicode 2.1. Thus, database utilities and error messages, sort order, date, time, monetary, numeric, and calendar conventions automatically adapt to the native language and locale.

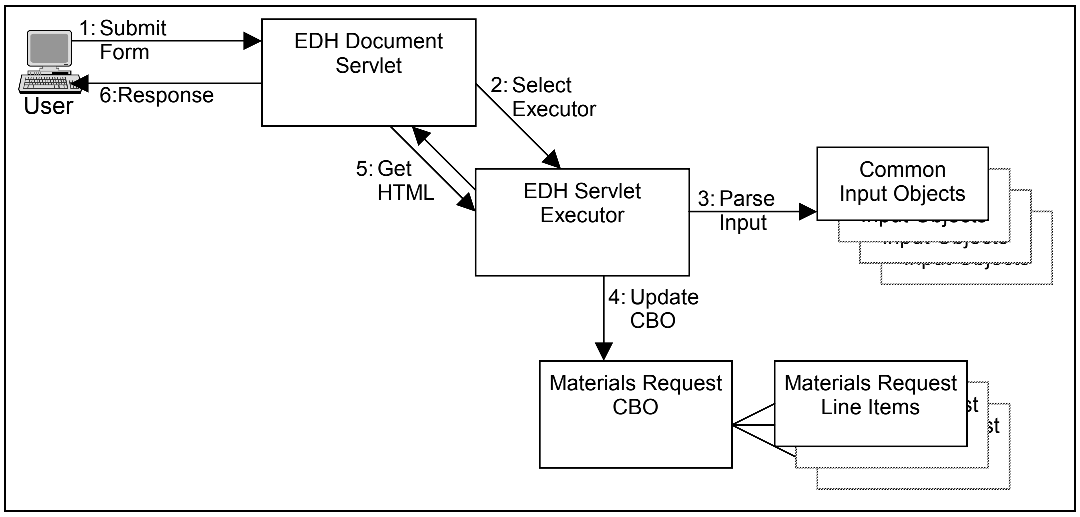
Runtime Scenario

Two classes, a DocumentServlet and a ServletExecutor, provide the user interface to the common business objects.

A single instance of the DocumentServlet exists for each type of document and is shared by all users. Its job is to find the appropriate ServletExecutor for the user and forward the requests to it.

A ServletExecutor instance is created for each open document window during the session. The job of the ServletExecutor is to interpret the information in the user's request and make the appropriate calls to the CBO. Once the ServletExecutor has processed the entire request, it then creates the response HTML by substituting placeholders in an HTML template with the values from the now updated CBO.

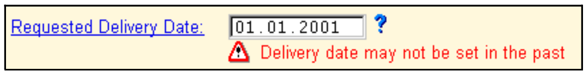
The following diagram and steps below it describe the sequence of events and how each object interacts during a typical user form submission.



Runtime Scenario

1. When the user submits the HTML form, the Document Servlet uses the session cookie to identify the correct Servlet Executor Instance. If there is no Servlet Executor for the document in the current session, then the Servlet creates a new instance.
2. The Document Servlet then calls the Servlet Executor so that it can process the user's input.
3. The Servlet Executor first makes a call to each of the CIOs so that they have an opportunity to interpret the user's input.
4. If the CIO successfully interprets the form data, the Servlet Executor then calls the corresponding `setXXX()` method on the CBO. The CBO either accepts the new value or throws a `ConstrainedPropertyException` ,indicating that the value is not valid.

   If either the parsing of the user input or the `setXXX()` method call of the CBO failed, then the CIO is set to error state, which appears to the user as an error message.


Requested Delivery Date: 01.01.2001 ?
Delivery date may not be set in the past

Date CIO showing Error Message

   After all the CIOs have been processed, the Servlet Executor returns control to the Document Servlet.

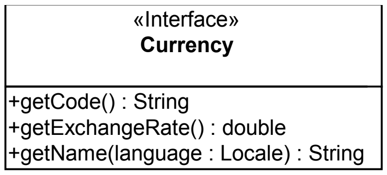
5. The Document Servlet then makes a call to the `getHTML()` method of the Servlet Executor that generates the HTML representation of the Document by using an HTML template.
6. Finally, the Document Servlet sends the resulting HTML back to the user.

MIGRATION TO EJB: FIRST STEPS

As a first step in the migration to EJB, one of the simpler CBOs was re-implemented as an EJB.

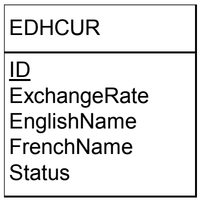
The Currency CBO was chosen because it is a self-contained bean with a simple interface, without dependencies on other parts of EDH, and is based on a single table.

The Currency CBO is defined as follows:


«Interface»
Currency
+getCode() : String
+getExchangeRate() : double
+getName(language : Locale) : String

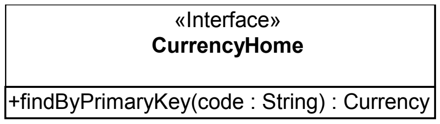
The Currency CBO

All the CBOs are defined as interfaces with an underlying implementation class. It was designed this way so that the implementation could be replaced at a later date without having to change any client code.



The Currency Table

The currency CBO uses only a single database table. Each instance of a Currency CBO maps to a single row in the EDHCUR table.

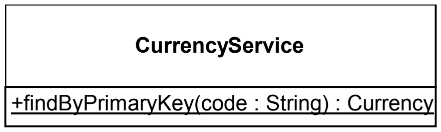


The CurrencyHome

The CurrencyHome provides the definition of the finder methods. Because the currency CBO is read-only, there is neither a `create()` method in the Home interface, nor `setXXX()` methods in the actual bean.

Within a servlet, the program obtains a reference to a class that implements the CurrencyHome interface by calling the `findByName("X")` method of the HomeHome interface.

The HomeHome interface performs the same role as the `lookup()` method in a JNDI Context. The implementation does not make use of JNDI, but instead employs `Class.forName()` to obtain a reference to an implementation of the Home interface.



The CurrencyService

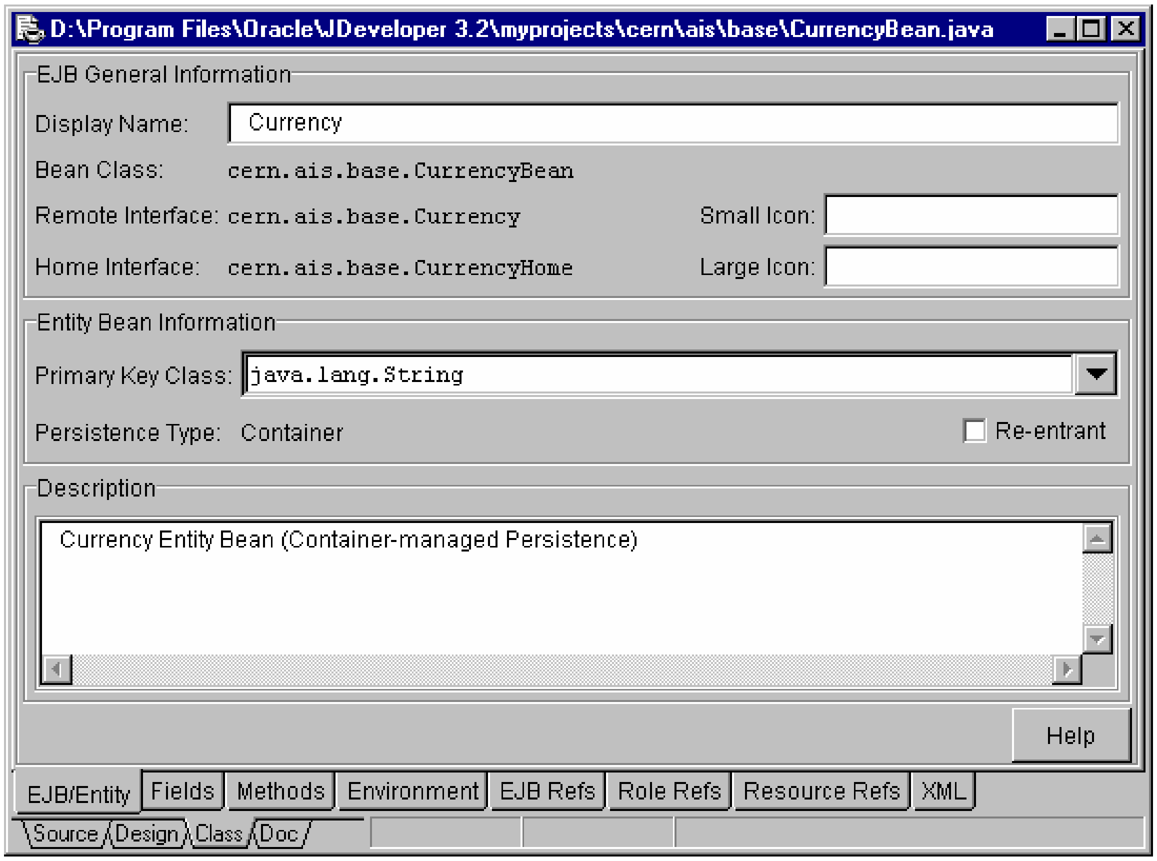
The CurrencyService supplies the interface to the relational database. As a measure to improve the maintainability of EDH, all the SQL for a particular CBO was located in a single class, rather than being spread over the code. By grouping together all the SQL into one place, it is easier to identify what statements need to be changed if the underlying tables change their definition (as they have done in the past for tables provided by third-party products). JDBC is preferable for dynamic SQL, and SQLJ for static SQL. Developers often use SQLJ because it gives the ability to embed static SQL in Java code, thereby raising the level of abstraction, resulting in increased productivity. The Oracle JDBC Drivers support the JDBC 2.0 standard. They also provide support for Oracle-defined extensions for Oracle datatypes, Connection Pooling, and Distributed Transactions.

THE CURRENCY BEAN

In the new EJB implementation, the Currency and CurrencyHome interfaces were kept, but implemented using EJBs. The remainder of the EDH application could then conveniently be used to test the new EJB implementation. Eventually, the interface could be extended to also allow updates of the exchange rate (restricted to the Accounting Department).

A wizard in Oracle JDeveloper allows for the creation of a Container Managed Persistence (CMP) Entity Bean by pointing to the existing currency table (EDHCUR) and indicating the primary key and which columns need persisting. By default, the field names of the EJB correspond to the column names in the table, but you can override this. With this information, the wizard creates the remote interface (Currency.java), the home interface (CurrencyHome.java), its implementation (which was called CurrencyCMPBean.java), and the standard deployment descriptor (Currency.xml), as well as the vendor-specific descriptor (Currency_oracle.xml).

Oracle EJE provides a hook to plug in custom Object Relational (O/R) mapping through the Persistence Storage Interface (PSI). This is a Java API that allows container managed persistence (CMP) providers, such as Oracle Business Components for Java, to manage O/R mappings from entity beans to database entities. You can use JDeveloper's wizards to design an O/R mapping that Oracle Business Components for Java manages with the PSI. An Entity bean class also needs a primary key, corresponding to one or more columns of the table, that allows instances to be retrieved using the `findByPrimaryKey()` method. CMP beans can also make use of other business component classes: domains that allow EJB fields to be based on Oracle object types, and secondary view objects corresponding to EJB finder methods. Using the EJB/Oracle9*i* Deployment Object Wizard in Oracle JDeveloper, CMP persistence can be added to the entity bean.


D:\Program Files\Oracle\JDeveloper 3.2\myprojects\cern\ais\base\CurrencyBean.java

EJB General Information

Display Name: Currency
Bean Class: cern.ais.base.CurrencyBean
Remote Interface: cern.ais.base.Currency
Home Interface: cern.ais.base.CurrencyHome
Small Icon:
Large Icon:

Entity Bean Information

Primary Key Class: java.lang.String
Persistence Type: Container
Re-entrant: ☐

Description

Currency Entity Bean (Container-managed Persistence)

Oracle JDeveloper EJB Designer Screen

```
public interface Currency  extends javax.ejb.EJBObject
{
     public String getName(Locale lang) throws java.rmi.RemoteException;
     public double getExchangeRate()    throws java.rmi.RemoteException;
     public void setExchangeRate(double newExchangeRate)
                                        throws java.rmi.RemoteException;
...
```

Excerpt from Currency.java

In the following example, the Oracle-specific deployment descriptor mainly defines the mapping of **the** Currency bean (in the package `cern.base`) to its JNDI name `ais/base/Currency`. In addition, it specifies the CMP provider (here ~~we use~~ the simple Reference Implementation PSI-RI **was used**) and the mapping of the columns in the database table to their field names:

```
<oracle-descriptor>
   <mappings>
      <ejb-mapping>
```

```
         <ejb-name>Currency</ejb-name>
         <jndi-name>ais/base/Currency</jndi-name>
      </ejb-mapping>
   </mappings>
   <persistence-provider>
      <description> demo persistence manager </description>
      <persistence-name>psi-ri</persistence-name>
      <persistence-deployer>
          oracle.aurora.ejb.persistence.ocmp.OcmpEntityDeployer
      </persistence-deployer>
   </persistence-provider>
   <persistence-descriptor>
      <description> simple persistence-mapping  </description>
      <ejb-name>Currency</ejb-name>
      <persistence-name>psi-ri</persistence-name>
      <psi-ri>
         <schema>demo</schema>
         <table>edhcur</table>
         <attr-mapping>
            <field-name>CurrencySymbol</field-name>
            <column-name>CUR</column-name>
         </attr-mapping>
         <attr-mapping>
            <field-name>EnglishDesc</field-name>
            <column-name>ELG</column-name>
         </attr-mapping>
         <attr-mapping>
            <field-name>ExchangeRate</field-name>
            <column-name>EXG</column-name>
         </attr-mapping>
      …
```

Excerpt from Currency_oracle.xml

The EJB Designer in JDeveloper can be used to further define the (remote and home) interfaces. After this, business logic is added to the bean implementation. For example, in the `setExchangeRate()` you could check whether the new value is within the same range of the existing value, to reduce data entry errors. The `getName()` method was also added to support more languages in the future without changing the interface. After this, the bean is ready to be deployed to the Oracle container.

Because Container Managed Persistence was chosen, the container automatically invokes a persistence manager on behalf of the bean, without any extra programming to load or store the data in the database. In addition, you automatically get many CMP-only Finder methods to perform a SQL query against the persistent data table (*findAllCurrencys* takes a String that denotes the "where" clause of a SQL query). Assuming the CurrencyHome, curHome, was obtained, all Currencies can be listed (Collections are not currently supported):

```
Currency cur;
Enumeration e = curHome.findAllCurrencys("");
while(e.hasMoreElements())
{
     cur = (Currency) e.nextElement();
     System.out.println (" name = " +  cur.getName(Locale.ENGLISH)+
                         " has rate = " + cur.getExchangeRate() );
}
```

The Converter Bean

With the Currency Bean defined, a facility was added to convert any amount from one currency to another through a *Converter* Stateless Session Bean in the same *cern.base* package. After creating the bean through the appropriate Wizard in JDeveloper, the interface files, the bean implementation and the deployment descriptors are automatically generated. The remote interface in Converter.java is:

```
public interface Converter extends javax.ejb.EJBObject
{
      double convert (Currency from, Currency to, double amount)
                        throws  java.rmi.RemoteException;
}
```

With its implementation in the bean:

```
public double convert (Currency from, Currency to, double amount)
             throws java.rmi.RemoteException
{
     return ( to.getExchangeRate() / from.getExchangeRate() ) * amount;
}
```

After deploying to the container, you can now write the following client code:

```
// Create the currencies we want to convert from their primary key
// (ISO Code)
Currency swissFrancs = curHome.findByPrimaryKey("CHF");
Currency euro = curHome.findByPrimaryKey("EUR");

// Obtain a new Exchange Calculator from the
// ConverterHome interface convHome
Converter myXChangeCalc = convHome.create();

System.out.println("150 " + swissFrancs.getEnglishDesc() +
                   " corresponds to " +
                   myXChangeCalc.convert(swissFrancs, euro, 150f)
                   + " " + euro.getEnglishDesc());

// Clean up neatly
myXChangeCalc.remove();
```

Which produces:

```
150 Swiss Franc corresponds to 97.4 Euro.
```

Taking Stock

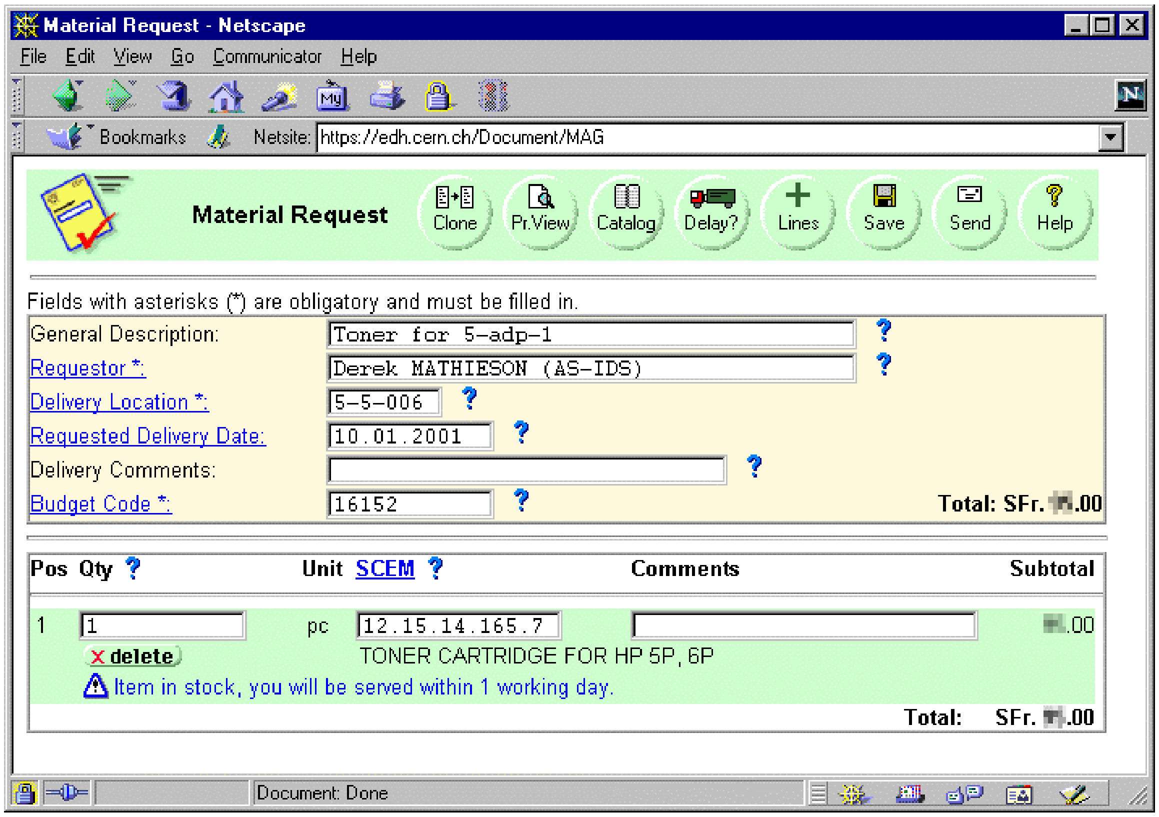
The Currency Entity Bean can now be accessed by anyone through its JNDI name. During deployment of the EJB, access rights were set up so that exchange rates can be updated only centrally from the Accounting department with the setExchangeRate() method. Because the beans and the database table reside in the same database, the loading and storing of data is efficient. Any changes to the underlying database table will affect only our bean.

Any application that wants to use the *Converter* retrieves these new rates immediately. The remote `convert` method call will access the exchange rates of the two *Currency* beans that reside in the same container, and there is no further overhead.

After the first simple EJB was completed, we were ready to tackle a more complex example.

THE CERN MATERIAL REQUEST

The CERN Material Request is a Web-based electronic document that allows any of the 10,000 people working on CERN activities to purchase any of 16,000 standardized items from the CERN Stores Catalogue.



Material Request - Netscape
File Edit View Go Communicator Help
Bookmarks
Netsite: https://edh.cern.ch/Document/MAG
Material Request
Clone
Pr.View
Catalog
Delay?
Lines
Save
Send
Help
Document: Done

Fields with asterisks (*) are obligatory and must be filled in.

General Description:	Toner for 5-adp-1
Requestor *:	Derek MATHIESON (AS-IDS)
Delivery Location *:	5-5-006
Requested Delivery Date:	10.01.2001
Delivery Comments:	
Budget Code *:	16152

Total: SFr. [redacted].00

Pos	Qty	Unit	SCEM	Comments	Subtotal
1	1	pc	12.15.14.165.7		[redacted].00
	delete		TONER CARTRIDGE FOR HP 5P, 6P		
	Item in stock, you will be served within 1 working day.				
				Total:	SFr. [redacted].00

The Material Request Document

The Material Request document resembles almost any on-line order form. It consists of a simple *header* that holds information about the customer, and how the goods are to be paid for, followed by one or more line items that contain quantity, description, and price fields.

One of the first steps in converting the document to EJB is to identify the existing Objects:

```
BudgetCode
CatalogItem
Currency
Location
MaterialRequest
MaterialRequest Line Item
Person
```

List of Material Request Objects

The list was divided into "simple" objects (Person, Location, Budget Code, and CatalogItem) that were converted to EJBs in the same way as the Currency Object, and the more complicated updateable objects.

THE CERN STORES CATALOG

Every item in the CERN catalog has a unique identification number (the SCEM† code) that allows for identification in the Materials Management System (MMS) database. CERN uses a commercial MMS (Baan) running on an Oracle database. The CERN Stores Catalog itself is a separate custom application that provides two primary interfaces:

- A set of tables in an Oracle database that contain all the information available for every item in the catalog
- A Web application (written in Oracle's PL/SQL programming language) that has the ability to "paste" a chosen product ID to another Web form, using JavaScript

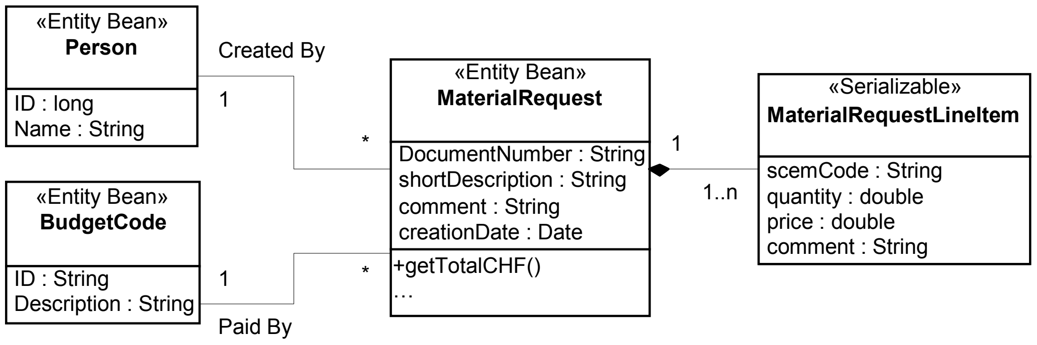
The exact details of this application are outside the scope of this book, and although we plan to re-implement the Web application in Java, no changes will be required to its interface.

The Material Request form accepts the product ID pasted into it, and then revalidates it on the server.

THE MATERIAL REQUEST BEAN

The Material Request Bean is the primary business component in **the** application. It contains all the business logic for implementing our Material Request Document.

† **S**tandard de **C**lassification pour **E**quipement et **M**atériels



UML Diagram of MaterialRequest Bean and its Associations

At this stage Bean-Managed persistence was chosen, rather than deferring it to the container because our Material Request maps to more than one table and has relationships to other EJBs that must be persisted. As mentioned previously, Oracle does offer support for Container Managed persistence when the mapping is relatively simple.

Persistence Methods

With Bean Managed Persistence, you must provide implementations for the `ejbCreate()`, `ejbLoad()`, `ejbStore()`, and all the finder methods.

The database schema for the Material Request consists of two tables, one that holds the main contents of the documents (EDHMAG), and the other that holds the individual line items (EDHMAGLI).

```
public class MaterialRequestBean implements EntityBean {
  transient  EntityContext ctx;

  String     shortDescription;
  String     comment;
  Date       creationDate;
  ArrayList  line_items;
  Person     creator;
  BudgetCode budCode;

  ...
```

Start of the MaterialRequestBean Implementation

Object Relationships

The preceding UML diagram shows that the MaterialRequest also references two other objects, a BudgetCode that indicates the account being charged for the purchase, and a Person object corresponding to the document creator. These objects will be stored as attributes of the EJB, and the relationship will be persisted by storing the primary key of the referenced objects in the `ejbStore()` method and then re-finding the Person and BudgetCode objects again in the `ejbLoad()` method from the appropriate HomeInterface through its `findByPrimaryKey()`. Alternatively, you can also do this in the getter method if the related object is not always needed (lazy initialization).

```
InitialContext ic  = new InitialContext();
BudgetCodeHome bch = (BudgetCodeHome)ic.lookup
                           ("java:comp/env/ejb/BudgetCode");

           budCode =  bch.findByPrimaryKey(budCodePK);
...
```

Retrieving the Budget Code in the `ejbLoad()` method

Notice that the Oracle EJB container does not require you to use `PortableRemoteObject.narrow()` when obtaining object references through JNDI. Instead, a simple Java cast is all that is needed.

BUSINESS METHODS

The business methods of the Material Request are where the business-specific logic is implemented.

The business methods fall into two categories:

1. The business logic specific to the Material Request—for example, the methods responsible for validating the consistency of the materials request, and for calculating its total value in a specific currency
2. The methods responsible for manipulating the line items

THE MATERIAL REQUEST LINE ITEMS

The Material Request Line Items are represented as simple Java objects. They implement the *serializable* interface so that they can be transported over the network by value. The infrastructure support needed to enable this communication is performed using RMI/IIOP, which is supported in the Oracle EJE environment. The line items are not implemented as EJBs for two reasons. First, they exist only within the context of a Material Request document, and would never be referenced outside that context. Second, EJBs are relatively heavyweight objects that consume resources on the server and, therefore, should be used only where their added capabilities (such as transactions) are required.

As the UML Diagram illustrates, Line Items are contained within the Material Request. This is achieve by extending the `ejbLoad()` and `ejbStore()` methods so that the line items are retrieved along with the header in the `ejbLoad()`method, and stored at the same time as the header in the `ejbStore()`method.

Oracle JDeveloper will provide support for the Unified Modeling Language (UML) (not currently available). Initially it will consist of two UML modelers: a class modeler and an activity modeler. Developers can use the class modeler to generate Java classes or Oracle Business Components for Java applications. Additionally, a reverse engineering facility, allows developers to build UML models from existing code. The code is automatically synchronized with the UML model so that changes you make in the class modeler are immediately reflected in your code, and the converse.

Deployment Descriptors

The Material Request Bean

Every Enterprise JavaBean has a deployment descriptor. The deployment descriptor file is formatted in XML. It allows a developer to specify the bean's transactional and security attributes declaratively. The container reads the deployment descriptor and enforces transaction and security constraints, state management, lifecycle, and persistence. One of the benefits of using a tool like JDeveloper is the way that it helps you create the XML deployment descriptor through a series of wizards.

Transaction Control

All the business methods in the Material Request Bean have the transaction attribute set to `Requires`. This means that a new transaction context will not be created if the method is called within an existing one. This will usually be the case, because the Session Bean controlling the Web Interface will create a transaction context before calling the Material Request Bean methods. This is achieved by using the `RequiresNew` transaction attribute on the methods of the Session Bean.

Oracle9*i* database and the Oracle9*i* Application Server support declarative and programmatic transactions using JTA APIs. Oracle J2EE container supports both JTA client and server-side demarcation, and propagation of transaction contexts. Propagation of the transaction context is necessary for including the invoked object into the global transaction. The JDBC drivers supplied by both Oracle9*i* database and the Oracle9*i* Application Server are also JTA enabled, giving them the capability to incorporate client-side transaction demarcation.

EJB Relationships

In common with many EJB applications, the Material Request Bean must interact with other EJBs in the system. To do this in a portable way, you can define an EJB relationship within the deployment descriptor.

```
<ejb-ref>
  <description>BudgetCode Entity</description>
  <ejb-ref-name>ejb/BudgetCode</ejb-ref-name>
  <ejb-ref-type>Entity</ejb-ref-type>
  <home>cern.base.BudgetCodeHome</home>
  <remote>cern.base.BudgetCode</remote>
  <ejb-link>BudgetCodeBean</ejb-link>
</ejb-ref>
```

Doing this allows the Material Request Bean to always refer to the BudgetCode Bean using the logical JNDI name "`java:comp/env/ejb/BudgetCode`", giving a degree of independence. If the implementing class or the environment changes in some way, only the Deployment Descriptor would need updating without having to modify any Java code.

The `<ejb-link>` tag in the deployment descriptor gives the name of the entity bean. Currently, Oracle implements this differently from the EJB1.1 standard. In this case, the name relates to a mapping entry in the vendor-specific deployment descriptor that maps the name of an EJB to a specific JNDI path.

```
<ejb-mapping>
  <ejb-name>BudgetCodeBean</ejb-name>
  <jndi-name>ais/base/BudgetCode</jndi-name>
</ejb-mapping>
```

Part of an Oracle Specific Deployment Descriptor

Deployment Environment

Another important aspect of the deployment descriptor is the section that defines the EJB environment. You can use this feature to define resources that the EJB can retrieve in a platform-independent way at runtime. The following sample shows the part of the descriptor that is needed to connect to the database.

```
<resource-ref>
  <res-ref-name>jdbc/localDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
```

The database connection can then be obtained at runtime, using JNDI as follows.

```
if (ic == null)
  ic = new InitialContext ();

DataSource                    localDS              =
       (DataSource)ic.lookup("java:comp/env/jdbc/localDB");

return localDS.getConnection();
```

How you bind the DataSource object to the JNDI namespace is dependent on the application server that you are using. With Oracle9*i*, you bind a DataSource using the `bindds` command in the session shell.

```
bindds  /edh/DataSource/localDB -rebind -url
      jdbc:oracle:thin:@edhdb:1521:ORCL
      -user scott -password tiger
```

Example of Using the `bindds` to Bind a DataSource to the JNDI Namespace

Oracle provides support for TAF—Transaction Application Failover. In case of connection failure, the application is automatically reconnected. Developers should also be aware that Oracle supports strong connection pooling to improve performance when interacting with the database.

The Material Request Executor (Session) Bean

The code for the Material Request Executor Bean remains largely unchanged with the introduction of EJB. As stated previously, the Executor Bean is responsible for maintaining the conversational state information. The Executor Bean also plays the role of coordinator as it passes the HTTP request from the client to the User Interface components and then obtains the resulting HTML stream for passing back to the browser.

This Executor Bean is implemented as a stateful session bean, it has relatively few methods, and most of them simply delegate their implementation to other objects.

Transaction Control

As with many EJB applications, the Session Bean has the responsibility for demarcating transactions. This is done, in our case, mainly for efficiency. The entity beans have been defined such that all of their business functions will run in an existing transaction context if one exists (the Requires attribute in the application assembly section of the deployment descriptor). When users submit the HTML forms, it is possible that they have updated many fields in the forms. This means that the Material Request Executor Bean will make several calls to the Material Request Entity Bean. By making all the calls part of the same transaction, the container will commit the changes to the database only after all the updates are complete.

The CatalogItem Bean

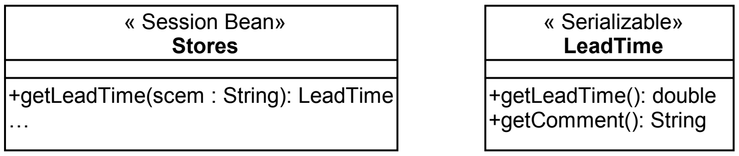
The CatalogItem bean is the representation of a product in the Stores Catalog. As with many of the objects in the Material Request, it is effectively a read-only object (maintenance of the catalog is performed using a previously existing administration tool build, using Oracle Forms). One extra piece of functionality that has been added to the CatalogItem bean is an interface to the delivery lead-time information in our MMS through a stateless session bean.

Obtaining the Delivery Lead Time

For efficiency, a local copy (snapshot) of the catalog database table from the MMS database is taken nightly. This allows faster access to the catalogue than through a database link, and the content of the catalogue does not change often enough to require more frequent updates. One piece of information that does change often, however, is inventory information. If an order can be fulfilled from stock, delivery happens within 24 hours; otherwise, an order must be placed with an external supplier, which usually implies a longer delivery time. In order to give the EDH users some idea of the expected delivery time, live feedback from the MMS is required.

Minimizing the number of round-trips between the middle tier and data tier can influence the response time. The relative percentage of read-only tables in the middle tier application can have an effect in the number of round trips your application has to perform. Database caches are effective in multi-tier environments where databases are located on separate nodes. These caches reduce the load on the database tier by processing the most common read-only requests to data sets on the application server tier. Oracle9*i*AS Database Cache keeps track of queries and can intelligently route to the database cache or to the origin database, without any application code modification. Oracle9iAS Database cache is not just an in-memory cache—it is both an in-memory and disk backed cache. This combination does not limit the cache size of the memory footprints and also means that the cache is not "cold" immediately after a machine reboot.

This is a good example of where to use a stateless session bean. The bean has only a single method that calls a stored procedure on the MMS database, which returns a *value* object containing the delivery lead-time and an associated comment. This feature creates a nice separation of the two systems; the Ordering application (EDH) has no idea of how lead-times are calculated—it knows only what service to ask.

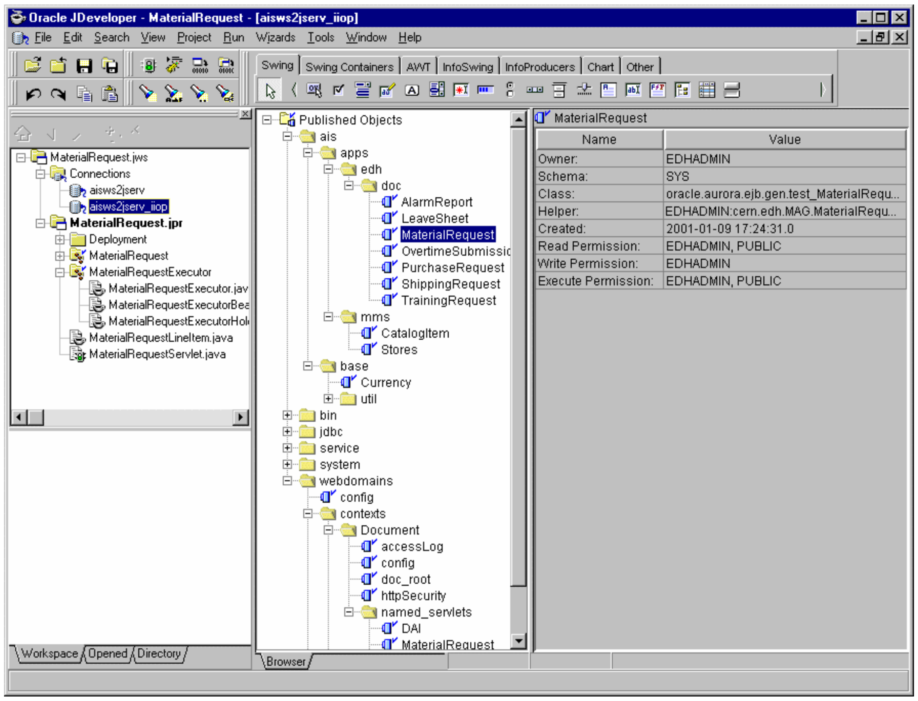


The Stores Stateless Session Bean and LeadTime *Value* Object

To simplify the use of this bean, a simple helper method, `getLeadTime()`, was then added to the CatalogItem bean. This method obtains an instance of the Stores bean and calls its `getLeadTime()` method.

PUTTING IT ALL TOGETHER

Oracle JDeveloper is a powerful Java IDE, which helped a great deal when creating this EJB application. The many wizards enable the rapid creation of the EJBs, and the integrated debugger supports debugging EJB implementations, even when running within the database. When the project-wide JNDI namespace was defined, the built-in JNDI browser proved to be very useful.



MaterialRequest

Name	Value
Owner:	EDHADMIN
Schema:	SYS
Class:	oracle.aurora.ejb.gen.test_MaterialRequ...
Helper:	EDHADMIN:cern.edh.MAG.MaterialRequ...
Created:	2001-01-09 17:24:31.0
Read Permission:	EDHADMIN, PUBLIC
Write Permission:	EDHADMIN
Execute Permission:	EDHADMIN, PUBLIC

Using JDeveloper to Browse the JNDI Namespace

THE ORACLE ENTERPRISE JAVA ENGINE (ORACLE EJE)

The choice of Java virtual machine can have a large influence on the overall performance and stability of a Java Web application. Oracle has addressed this problem by developing its own Java virtual machine implementation that runs within each server product—Oracle9*i* database and Oracle9*i* Application Server. By running *within* the database, Java applications can leverage all the scalability and reliability of the database server. In the application server, Oracle EJE runs within the mid-tier SQL cache, allowing the data proximity benefits to exist in the application server as well. It is the only JVM implementation that uses the scalability of an RDBMS to be able to support thousands of concurrent users.

The developers of Oracle EJE have been able to make significant improvements compared with traditional JVM designs. For example:

- *ACCELERATOR*[†]

  An innovative feature of Oracle EJE is its use of native compilation. In a server environment, the portability offered by interpreted Java bytecode is not necessary. Server applications are typically longer-lived than those of a client, which means it is worthwhile investing more CPU cycles to produce highly optimized native code than could be done by JIT (or HotSpot™) compilers. Oracle calls this "way-ahead-of-time-compilation", and it can significantly improve server performance.

† Accelerator has been used to natively compile all the standard Java classes provided with the JVM. It became available for application developers to compile their own classes in Oracle8*i* Release 3 (version 8.1.7).

- *Generational Scavenging Garbage Collector*

  Oracle EJE uses an advanced garbage collection strategy that greatly improves the memory utilization of Java Applications.

- *Scalability*

  Oracle has made significant advances in reducing the amount of memory required for a typical stateful Java session. On Oracle EJE, each session appears to be running on its own private JVM, yet the memory overhead is typically under 50 KB.

- *Database Integration*

  Because most e-business Web sites typically involve extensive use of relational databases, it is vital that Java applications efficiently access the database. By integrating the JVM within the database, the communications overhead imposed by running in an external JVM is eliminated. Oracle has also supplied a special "Server" JDBC driver that can be used to directly access the database's SQL engine through the same high-performance APIs that were previously available only to PL/SQL stored procedures.

Oracle EJE is a relatively new feature, yet we have been using it successfully in production with EDH for the past two years and have found it to be a very stable implementation.

JVM Overload

One area that was identified very quickly when developing EDH was the need to spread the load of the application on several Java virtual machines at the same time. This was because a single virtual machine rapidly became overloaded with only a relatively small number of users. The problem usually manifested itself with the JVM running out of available memory, even though the JVM had been allocated 64 MB of RAM. This is because as the JVM gets loaded and the garbage collector fails to keep up with the number of objects being freed, resulting in new memory allocation calls failing, even when there is plenty of free memory (except all of it is currently waiting to be marked free by the garbage collector).

This problem was solved in the past by spreading the user population across several virtual machines (up to 20), running on the same host. This effectively created 20 garbage collector threads, each managing a smaller pool of objects. This type of load balancing is necessary on any large-scale, Java-based Web application, and can significantly complicate its design.

With Oracle EJE memory architecture, this scheme is no longer necessary; in fact, each session (user) runs in its own *virtual* JVM. Typical servlet engines use threads to isolate concurrent users. This becomes a bottleneck when there are a large number of users and a stateful application. True, a good design will limit session state to only the essentials; however, even this can cause resource (thread) contention within the container. Oracle's approach is to use a JVM architecture that isolates concurrent users. The result is less contention and a container that is more resilient to application failures within an individual session.

In addition, when many users share the same JVM it is almost impossible to prevent the activity of one user from affecting the others. By using Oracle EJE, however, the resources consumed by a session can be strictly controlled using standard Oracle tools. Limits on memory, CPU, and so on can all be set.

LOAD BALANCING

Another area identified when developing EDH was the need to understand how load balancing is handled by the platform on which you deploy your J2EE components. This led us to understand how the server platform dealt with response times and how Oracle9*i* platform distributes connection loads across multiple servers to provide no single point of failure. Oracle's clustering technology is used to load-balance and provide fault tolerance.

Oracle supports many ways by which you can help improve the response times:

- Round-Robin DNS for distributing HTTP requests across the cluster
- Use of hardware load-balancing devices and IP sprayers
- Spawning a number of HTTP processes to handle load
- Using Oracle HTTP Servers in reverse proxy mode.

In addition, Oracle9*i*AS offers techniques to improve the performance of servlets, EJBs and J2EE components delivering dynamic content. J2EE components rely on HTTP for communication between Web clients and servers, and RMI/IIOP for communication between objects requesting service from each other in a distributed computing environment. The following paragraphs address how different components of Oracle9*i*AS work together in intelligently routing HTTP and IIOP requests to nodes, thereby providing smooth response times and no single point of fail over.

When an HTTP request arrives, the load-balancing device redistributes the load over Web caches that sit in front of the application server farm. Oracle9*i*AS Web Cache distributes the HTTP requests according to the relative capacity of each application server, which is configurable. If one of the application servers in the farm were to fail, Web cache automatically redistributes the load among the remaining servers. Oracle9*i*AS Web Cache reduces application server load not only on Oracle9*i*AS, but other application servers too.

OracleHTTP Server load balances servlet processes that use the Apache JServ servlet engine. Several JServ processes can serve a single or multiple instances of Oracle HTTP Server. This is done using a weighted load-balancing algorithm that can be configured, depending upon the load handling capability of the target machines.

Oracle9*i*AS listener/dispatcher architecture provides a fault-tolerant and resilient environment, without a single point of failure. With this architecture, each physical machine has a listener on a designated port and a number of dispatchers to service J2EE container requests. The bridge in this paradigm is that each dispatcher registers itself with any number of nodes in the application server farm. Thus if a particular node is no longer able to service requests, then the listener will send incoming requests to another dispatcher on another node. It is important that an application server redirects both HTTP and IIOP requests intelligently for the purpose of load balancing. Redirection is essential to load balance at a *protocol level*; however, there is no concept of "redirection" in HTTP—but there is one for IIOP. Oracle leverages its listener/dispatcher architecture with Apache modules to provide HTTP redirection. For example, mod_ose load balances servlet requests across nodes by redirecting HTTP requests.

Oracle9*i*AS provides mechanisms to load-balance incoming requests, be it IIOP or other network formats in multi-threaded server mode. This has great benefits to enterprise components residing on multiple nodes. The listener process has the ability to load-balance across these nodes using IIOP redirection. This is possible because of the listener's ability to inspect IIOP headers for session ids and redirect to the appropriate node. Incoming IIOP connections are redirected to dispatchers on the node with least load. The load is computed based on number of parameters, including CPU utilization, number of sessions in progress, and so on. With load balancing you can split the load across multiple servers. The Listener process within Oracle9*i*AS performs load balancing by distributing EJB lookup across multiple application servers. Services that are located in a number of places can be grouped together by specifying a service name in the URL. Listener handles HTTP connections by transferring such requests to configured dispatchers on a node.

CERN'S EXPERIENCE

The EDH application has been running using an EJB-style component model since November 1998 (already quite old in Java terms!). This design strategy has been extremely positive. The well-defined and relatively simple structure makes the architecture very effective for developing a variety of e-business applications.

CERN's experience with Oracle's EJB container has been reasonably successful. Although~~,~~ the dream of complete Application Server independence is tantalizingly close, most EJB containers still have some way to go before an application can be moved from one to another without modification.

Once these (relatively minor) incompatibilities have been resolved, application developers will be able to choose an application server based on factors such as performance, scalability, and maintenance tools.

EXPENSIVE ENTITIES

In the EDH application, several entities (Currency, CatalogItem) could have been implemented as simple value objects, obtained from a session bean. It was possible to do this because they were simple.

It is important not to forget, Entity Beans are relatively expensive in terms of resources, and they should be used only where the complicated machinery for authorization, transaction control, and persistence is necessary. Although the CatalogItemBean exists, it would *not* be a good choice for implementing the on-line catalog. When the user browses the catalog, the system should not have to create thousands of instances of the CatalogItemBean. Instead, a session bean should access the database directly. Only once the user had made a selection would a CatalogItemBean instance be created.

It would, of course, be preferable not to need this hybrid approach, and in fact the choice of when to use Entity Beans may eventually depend on the level of optimization of your EJB container. Hopefully, with time, the EJB specification will evolve to make this decision unnecessary (early indications show that the forthcoming EJB 2.0 specification will indeed address this issue). In so doing, it will simplify the design and reduce the amount of code that needs to be aware of the actual database design.

ORACLE EJE ACCELERATOR

Currently Oracle EJE Accelerator has been available to application developers for only a short time. In recent tests, it was found that Oracle EJE Accelerator was two to three times faster at running the EDH application as compared to the previous fastest JVM (Symantec JIT).

The response time of any interactive application is a critical success factor, and using Oracle EJE Accelerator gives a significant advantage compared to EJB application servers relying on classical JVM technology.

FUTURE WORK

Although this is early in the life of EJB containers, EJBs appear to be an excellent way to develop highly scalable Java Web applications.

EDH is part of a larger set of AIS (Advanced Information System) applications that manage all corporate information of CERN, including financial, accounting, supplier, purchasing, materials management, HR, and payroll data. CERN aimed at acquiring 80 percent of the functionality through best-of-breed ERP suites in the various areas, all running Oracle on top of Sun servers. The remaining 20 percent was implemented at CERN for specific needs.

The integration and communication between the developed and acquired software modules is currently achieved with a Foundation layer, which consists of a set of database tables with an extensive interface through stored procedures. The aim is to re-architecture this layer on top of the EJB component framework in which all business objects are truly unique and have a well defined interface to the outside world. Areas such as portals, wireless, business intelligence, dynamic Web services, collaboration, and process integration are already part of the Oracle9*i* Application Server infrastructure. This would greatly simplify integration of enterprise components and prepare the EDH application for use on any wireless or mobile device.

In making a move to J2EE technologies, the server platform chosen to deploy J2EE components can prove to be a critical success factor. By choosing Oracle to deliver the J2EE infrastructure services needed to power the EDH application, CERN believes it made the right decision.

REFERENCES

1. CERN Web site, **Error! Reference source not found.**
2. CERN EDH application, http://edh.cern.ch
3. Oracle Corporation web site, http://www.oracle.com
4. Oracle Technology Network for developers, http://otn.oracle.com
5. Oracle9*i* Database home page, http://www.oracle.com/ip/deploy/database/9i/
6. Oracle9*i* Application Server home page, http://www.oracle.com/ip/deploy/ias/index.html
7. Oracle Internet Developer Suite home page, http://www.oracle.com/ip/develop/ids/index.html

*****