The Need for Component-Based Technologies

The following distributed computing development paradigms have benefits but also have many problems:

- Remote Procedure Calls (RPC)
- 2-Tier Client/Server (Java RMI, DCOM)
- Object Request Brokers (CORBA)

Deficiencies:
- Objects written in this environment are completely stateless, storage of the state of these objects is left to the developer.
- Developers must also write non-trivial code to deal with transactions, concurrency and other the integration of other services.

Problems with Previous Paradigms

Coding was far too complex because developers could not focus exclusively on writing Business Logic; rather there was too much focus on code for the infrastructure...

- Writing code to register Servers with Directory Services.
- Writing code to deal with Security and Transaction Services.
- Lifecycle issues
- Scalability issues
- Data Access

Gartner Group estimates that 65% of programmer time is spent building infrastructure, instead of distributed application functionality [Financial Times, 04/07/92]

Components

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.”

[Clemente Suyyapa, Component Software: Beyond Object-Oriented Programming, 1997]

Enterprise JavaBeans, COM+, .Net and the CORBA Component Model are examples of distributed server-side component systems.

Enterprise JavaBeans Components

Built on top of RMI
- RMI over EJF allows integration with other existing CORBA products and middleware

Makes Distributed programming (RMI/CORBA) easier
- Declarative Programming.

EJB has a Container Model

Let the EJB Server do all of the hard work!

What EJB Accomplishes

You can take a Java class and with little effort make it a distributed, secure, transactional class
- These are hard to bespoke implement

You can take any a source and make the data source appear to be a collection of Java objects

- Eliminates distinction between data from a database and any other source.
- All information is accessed through Java objects
- All SQL is cleanly encapsulated in Java objects
- Over object-oriented programming
- Database objects work with the full Java class library
Programming Implications

Developers can focus on writing business logic rather than writing low-level infrastructure like data access, concurrency, transactions, threading, etc.
- Reduces development time
- Reduces complexity
- Increases quality and reliability

However...
- The programming model contains many restrictions as we will see later

Enterprise JavaBeans Defined

Enterprise JavaBeans defines a server component model for the development and deployment of Java applications based on a multi-tier, distributed object architecture.

The Enterprise JavaBeans specification defines:
- A container model
- A definition of the services the container needs to provide to an Enterprise JavaBean, and vice versa
- How a container should manage Enterprise JavaBeans

Enterprise JavaBeans Architecture

The EJB architecture specifies the responsibilities and interactions among EJB entities:

EJB Server
EJB Container
EJB Bean

- The EJB Bean is nothing more than a Java class that implements one of the required EJB interfaces. This provides what the Business Logic or Data Access Logic.

EJB Clients
- An EJB Client is any program, Java or not, which makes invocations on the EJB Server

1. EJB Server

The EJB Server provides system services and manages resources
- Process and thread management
- System resources management
- Database connection pooling and caching
- Management API
- Provides a Runtime Environment

The EJB Server (or Application Server as it is commonly known) is on its way to becoming an operating system

2. EJB Container

Hosts the Enterprise JavaBeans

Provides services to Enterprise JavaBeans
- Naming
- Life cycle management
- Persistence (state management)
- Transaction Management
- Security

Likely provided by server vendor
- Provides a Run-time Environment for an Enterprise Bean
3. Enterprise Beans

A specialized Java class where the real business logic lives

Enterprise Java Beans may be:
- developer-written or tool-generated
- Distributed over a network
- Transactional
- Persistent
- Secure

EJB vendors provide tools that automatically generate distribution, transaction, persistence and security behavior.

---

4. EJB Clients

Client access is controlled by the container in which the enterprise Bean is deployed
- via a Home Interface

Clients locate an Enterprise JavaBean through a Java Naming and Directory Interface (JNDI) Registry

RMI is the standard method for accessing a bean over a network
- CORBA clients can access a bean via an IIOP/RMI Bridge

---

Roles in EJB

The subdivision of EJB development into different parts allows for the division of labour [A. Smith]

Roles in EJB
- Client
- EJB Developer
- EJB Deployer
- Container Developer (Normally a Vendor)

---

EJB Portability

Portable Data
- XML

Systems Portability
- Java is OS independent

Application Server Portability
- Container API
- Servlet API
- Other J2EE APIs

---

EJB Supporting Technologies

EJB is part of the Java 2 Enterprise Edition Specification (J2EE)

J2EE also includes:
- Java Naming and Directory Interface (JNDI)
- Java Transaction API (JTA)
- Java Message Service API (JMS)
- The Object Bus (RMI & EJB in CORBA mappings)
- EJB Security Model
- See http://java.sun.com/j2ee/ for full details of the spec.

---

Java Naming and Directory Interface (JNDI)

JNDI is an API specification that is used to provide naming/directory services
- JNDI provides a common API on top of any directory service product
  - Directory service products (LDAP, DNS, NDS) all implement most of the specification
- JNDI is used by clients in conjunction with RMI and EJB to locate Enterprise JavaBeans on a server.
Java Naming and Directory Interface (JNDI)

Clients accessing a Bean with JNDI

EJB Internals

EJB Class is written by the developer.

EJBHome and EJBOBJECT interfaces and classes control access to the Bean class.

Deployment Descriptor and MANIFEST describe security and transactional characteristics of the Bean.

EJBHome Interface and Class

Used to get a reference to a bean's remote interface by creating or finding it.

- Must extend the class server.ejb.EJBEjb.
- Provides bean creation services for clients.
- myFbn = ejbHome.create() instead of myFbn = new Fbn().
- Supports multiple signatures to create EJB instances.
- Provides a bean-removal interface also.
- The specialised EJBHome interface is written by the developer.
- The EJBHome Class implementation is generated by a tool.
- Also manages the bean:
  - starting (Entity Bean)
  - deleting (Entity Bean).
EJBObject Interface and Class

A.k.a. the remote interface. The remote interface is written by the developer.
• Must extend the interface java.rmi.Remote
• The EJBObject Class implementation is generated by a tool.
EJBObject class has the same methods as the bean and delegates to the bean for actual behavior.
• Intercepts calls to the EJB Class to add support for transactions, security & threading.
• EJBObject class checks security and sets up transaction before delegating method call to the bean.
Clients can never get a reference to a bean’s EJB Class, only the EJBObject interface.

Clients accessing a Bean

After the EJBHome Class instantiates the Bean...
• The EJBHome Class will instantiate the EJBObject Class, initializing it with the remote object reference to the Bean Class.
• The Client will now communicate to the EJBObject Class. The EJBObject Class will delegate the call to the Bean.
• Client now has reference to an instance of the EJBObject Class, add a reference to the bean class.

Clients accessing a Bean

EJB Class

A bean has a single Java class at its core
• This class is written by a developer if it’s a session bean
• This class is sometimes generated by a tool if it’s an entity bean
Implements one of the following contracts:
• java.ejb.EntityBean
• java.ejb.SessionBean
These contracts provide for consistent behavior when activating beans, passivating beans, reading data, writing data.
• Every container can expect these methods in every bean.

EJB’s Declarative Programming Model

The EJB specification mandates a container model where common services are declared, not programmed
• At development and/or deployment time, attributes defining the bean’s transaction and security characteristics are specified
• At deployment time, the container intercepts the Enterprise JavaBean attributes for the runtime services it requires and wraps the bean with the required functionality.
• At runtime, the container intercepts all calls to the object
  • The container provides transactional, threading and security delegation required before the method invocation.
  • Invokes the method on the object
  • Cleans up after the call

Deployment Descriptor

Allows you to declare transaction and security attributes. NO PROGRAMMING REQUIRED!!!
An EJB Deployment Descriptor describes the classes, interfaces and declarative behavior of an EJB.
The deployment descriptor is generated by server tools.
The EJB Deploidy fills in the XML deployment descriptor at deployment time to define properties of the EJB such as:
• JNDI name
• Define certain properties of bean methods
• Principal identity, type of transaction management, etc.
• Specify an Access Control List to be associated with the runtime environment of the EJB
Generating Code with the Deployment Tool

Write your Bean implementation
- Compile this Java source into Java bytecode

Tool from the vendor is responsible for creating a serialized deployment descriptor for the bean

EJB tools use the Reflection API on the compiled EJB Bean to determine:
- name of Bean class, methods, parameters, return values

The EJB Tool uses the above information to generate a Deployment Descriptor file and an editor is used with which to declare attributes on different classes and methods

EJB Tool generates an Empty Deployment Descriptor...

Now the DEPLOYER can edit the transaction & security attributes for each individual method of the EJB Bean. The deployer DECLARES these attributes, there is no coding!

Once the Deployment Descriptor is edited, the EJB Tools parse the file and generate the appropriate container code based on what is defined.

All of the code is compiled and packed in the ejb-jar JAR file for distribution.

Container Services

The EJB deployment tool generates code to handle:
- Security
- Naming
- Transactions
- Connection Pooling
- Thread Management
- Lifecycle Management

Sample Deployment Descriptor

PS\Tomcat\Java\src\deployment\deployment_descriptor\entity:
VersionNumber (1.0)
BeaBeanName (Controller)
ControllerDescriptor:
  TransactionType:
    TRANSACTION_Bean_CoMMitted
Method:
  setGetId(): throws java.rmi.RemoteException
  UseTransaction:
    @ EJBTransaction
  TransactionAttribute:
    @ J2EE_Purpose

EJBean Packaging

Enterprise JavaBeans are comprised of many Java files

These files are packaged in a JAR file
- A JAR file is a ZIP file with a MANIFEST file that describes the contents of the file
- A MANIFEST is a simple file
- “Name bean AccessDeployment.scripts EJBean: True”
- A JAR file can contain more than one Enterprise JavaBean.

I.C.E. Paradigm

Install
Configure
Execute
Two Types of EJBean

Session Beans
- Stateful/Stateless
- The Bean Lifecycle
- Passing a Session Bean’s Object Reference
- Accessing Environment Entries

Entity Beans
- Persistence
- Shared Access
- Primary Key
- Primary Key Value
- Bean-Managed Persistence
- Container-Managed Persistence

Session and Entity EJBeans

<table>
<thead>
<tr>
<th></th>
<th>Session Bean</th>
<th>Entity Bean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Performs a task for a client.</td>
<td>Represents a business entity object that exists in persistent storage.</td>
</tr>
<tr>
<td>Shared Access</td>
<td>May have one client.</td>
<td>May be shared by multiple clients.</td>
</tr>
<tr>
<td>Persistence</td>
<td>Not persistent. When the client terminates its session bean is no longer available.</td>
<td>Persistent. Even when the EJB container terminates, the entity state remains in a database.</td>
</tr>
</tbody>
</table>

Database Connections in EJB (Persistence)

Coded Connections
- Logical name for DBC
- JNDI lookup when obtaining the database connection
- When To Connect
- Long- versus short-term connections
- Specifying Database Users and Passwords

Container-Managed Connections
- Code Generated by Deployment Tool

Transactions in EJB

Container-Managed Transactions
- Transaction Attributes
- Rolling-Back Transactions
- Synchronizing a Session Bean’s Instance Variables

Bean-Managed Transactions
- JDBI Transactions
  - JTA Transactions
    - Join Transaction STM
      - Inside the JTA methods, which then call the lower-level JTS (Java Transaction Service) methods
      - The JDBI STM implements the transaction manager with the JTS.
Transactions in EJB

Security

Authentication
• In J2EE, Users, Realms, and Groups are managed using realmtool.
  `realmtool -r default -a dunk -i import certificate file`

Authorization
• Declare roles using EJB JAR file in the Deployment Tool or XAR descriptor.
  In J2EE, using the Application Deployment Tool the administrator maps roles to J2EE users and groups.

Common EJB Architectural Pattern

Typical EJB E-Commerce Application

Bean Development Process

Programming Restrictions for Enterprise Beans

Implement the EJB Class
Specify the remote interface
Specify the home interface
Specify security and transactional characteristics using vendor tools or XML descriptors
Use vendor tools to generate supporting code and package components in EJB jar files
Iterate...

See JBoss/Tomcat “Introduction to EJBs” Tutorial
(http://www.esign.co.tdl.cn/~scheffing/teaching/lruds/103/)

Enterprise beans make use of services provided by both the J2EE container and the EJB (Application) Server.
To avoid conflicts with these services, enterprise beans are restricted from performing certain operations:
• Managing or synchronizing threads
• Accessing files or directories with the java.io package
• Using JIT functionality to display information or to accept information from a keyboard
• Listening on a socket, accepting connections on a socket, or using a socket for multicast
• Setting a socket factory used by Server/Socket, Socket, or the stream handler factory used by the URL class
• Loading a native library
When do you need EJB's?

If you really need any of:
- 2 Phase commit across multiple data sources
- method-level object security
- object distribution (especially shared logic for multiple client types)
- standard object-relational mapping capabilities

You should look at EJB's
- but they're not right for every project

---

Critique of EJB

Makes distributed systems programming easier and faster.
- Declarative programming
- Separation of roles between developer and deployer of bean.
  - Developer writes Java classes
  - Deployer is a system administrator setting ACLs, etc

Programming model is different to standard Java.
- Learning curve and restrictions.
- Not language independent.
- Performance hit.
- Not firewall friendly.

---

References

- JBoss Tutorial: