Enterprise JavaBeans (I)

K.P. Chow
University of Hong Kong
JavaBeans

- Components are self-contained, reusable software units that can be visually composed into composite components using visual builder tools. Builder tools include web page builder, visual application builder, GUI layout builder, server application builder, …

- A component model defines an environment supporting reusable application components, e.g. servlets and JSP pages

- Sun Microsystems defines a generic Java component model: JavaBean (JavaBean API 1.01 specification) for linking pieces of code to form an applet or application

- A component conforms to this application is called a JavaBean (or a Bean).

- A JavaBean must be developed in Java but can be used by any application on any platform
Enterprise JavaBeans

- The Enterprise JavaBeans (EJB) architecture is a component architecture for the development and deployment of object-oriented distributed enterprise-level applications.
- EJB is not JavaBean at the enterprise level, EJB is used to implement transactional middleware.
- Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification. (EJB 2.0 Specification)
EJB was designed to satisfy the two industry themes: components and multi-tier paradigm.

IIOP (Internet InterORB Protocol) is a stream protocol used to encode requests from a client to a server object and results returned from the server object.
Component Architecture

● Each component has a defined and published interface that provides application functionality
● Components are combined to produce a new application
● Component environment
  − Supports the following services: multi-threading, concurrency control, resource management, security, and transaction management
  − Provides runtime services by *component containers*: the container insulates the component from the runtime platform and manages the use of resources, such as execution threads, memory, and CPU
  − When a client invokes a server component, the container allocates the resources and initiates the component, and manages all interactions between the component and the external systems
To achieve Plug-and-Play assembly:
- Purchase a component, install it, and integrates it into their system with minimal or no effort
- Standardizing the contracts (interfaces)

Market availability:
- Fairly rich supply of off-the-shelf client products
- Server side components are limited
Multi-Tier Paradigm (1)

Web-based Application Partitioning

Client Workstation

Internet

Web Server

Business Logic

Data Access

DBMS
Multi-tier Paradigm (2)

- Applications can be partitioned into 4 layers:
  - presentation, business logic, data access, data management
- These layers can be distributed between the client and the server in various ways, e.g.
  - Two-tier model: Fat Client
  - 3-tier model
- Integrating different types of client systems and server systems is often quite complex
EJB Model

- EJB defines an architecture for the development and deployment of transactional, applications-based, server-side software components.
- The server-side components are called enterprise beans: they are *distributed* objects, which implement the business logic and provide remote services for clients distributed throughout the network.
- Java 2 Platform, Enterprise Edition (J2EE) supports web-based enterprise applications: a platform for the design, development, assembly and deployment of enterprise applications.
- J2EE defines the following components:
  - Client components: client applications and applets
  - Web components: Servlet and JSP
  - Business components: EJB components (enterprise beans)
EJB Model Key Elements (1)

- EJB Container
  - transaction control
  - persistence management
  - security services

- Server
- Client
- EJB Home Interface
- Transaction System
- DBMS

- JNDI
- JTS
- Security
- Messaging
EJB Model Key Elements (2)

- **Server:**
  - Provides a standard set of services (distributed transaction management and security) for the container
  - Provides at least one container for enterprise beans (the EJB container)

- **EJB container:** provides the environment for the deployment and runtime environment for the enterprise beans
  - Security: protect system resources against unauthorized access
  - Transaction: specify relationship among methods within a single transaction so that all methods in a transactions are treated as a single unit
  - Access naming and directory service: JNDI
  - Deployment of enterprise beans
  - Persistence of enterprise bean state information
  - Concurrency control over enterprise bean access
  - Remote connectivity between clients and enterprise beans to achieve remote invocation
  - Manage the life cycle of an enterprise bean instance

- **Client interface**
- **Interface to backend systems and DB**
The container isolates the enterprise beans from direct access by client applications.

Enterprise bean developers are insulated from platform dependent APIs and therefore can focus on encapsulating business rules while the EJB container takes care of the rest, e.g. security.
Container and Bean Interactions (1)

- **Callback methods:**
  - Each bean implements a subtype of EnterpriseBean interface, which defines several methods called *callback methods*.
  - Each callback method alerts the enterprise bean to a different event and the container will invoke these methods to notify the enterprise bean when it encounters the events, such as:
    - activates the enterprise bean
    - persists the enterprise bean state to DB
    - ends a transaction
    - removes the enterprise bean from the memory

- **EJBContext:**
  - Every enterprise bean obtains an EJBContext object, which is a reference directly to the container.
  - The EJBContext object provides methods for interacting with the container to request information about its environment, e.g. identity of the client, status of a transaction, its own remote references.
Container and Bean Interactions (2)

- Java Naming and Directory Interface (JNDI)
  - Every enterprise bean can use JNDI to access a special naming system called the Environment Naming Context (ENC), managed by the container
  - The default context is in the name space (directory) called “java:comp/env” and its subdirectories
  - When an enterprise bean is deployed, any beans it uses are mapped into the subdirectory “java:comp/env.ejb”, and bean references can be obtained at runtime through the default context
  - JNDI ENC also allows an enterprise bean to access resources like JDBC connections, other enterprise beans, and properties specific to that bean

- EJB specification defines a bean-container contract
  - Include the above interaction mechanisms and a set of strict rules describes the various behavior of the enterprise beans and their containers
  - Design to make an enterprise bean portable among containers so that an enterprise bean once developed can be deployed to any EJB container
EJB Runtime Environment

- EJB models are implemented on standards-based scalable platform:
  - Based on industry standard protocols: TCP/IP, IIOP
  - Appropriate for small-scale applications to large-scale business transaction systems
  - Supports Internet-based application
  - Interoperability with heterogeneous clients: use EJB-to-CORBA mapping, IIOP
  - EJB containers provide transparent attachment to vendor-specific backend systems
  - Supports higher level of integration and interoperability:
    - EJB applications can be *developed* in any EJB-compliant environment
    - EJB applications can be *deployed* in any EJB-compliant environment
  - Allows customization without access to source code
  - Application behaviors and runtime settings can be changed at development and deployment time (by modifying the bean properties)
EJB Products

- Products that support EJB architecture:
  - TP monitors, e.g. BEA Tuxedo or IBM TXSeries
  - Component Transaction Servers, e.g. Sybase Jaguar CTS, Microsoft Transaction Server
  - CORBA, e.g. Borland VisiBroker/ITS
  - Web platforms, e.g. IBM WebSphere Application Server
  - DBMS, e.g. IBM DB2
  - JBoss (open source)
EJB Lifecycle

EJB Server Provider

EJB Container Provider

EJB Developer

EJB User / GUI programmer

EJB Application

Enterprise JavaBean

EJB Deployer
EJB Core Elements

EJB Server

EJB Container

Home Interface

Home Object

Remote Interface

EJB Object

Enterprise JavaBeans

Databases

EJB Client

Locate, Create and Remove instance of EJB

Invoke Business methods of EJB

Enterprise Services and API

JNDI

JTS

Security

Messaging
EJB Server-side Component

- 2 interfaces defining the EJB server-side component’s business methods:
  - *home interface* and *remote interface*: use to access the enterprise bean
  - Home interface: for methods with life cycle behaviors of an enterprise bean that are not specific to a single bean instance, e.g. create and locate the bean
  - Remote interface: for methods with behaviors that are specific to a single bean instance, e.g. exposed business methods for the bean

- Actual implemented class of the bean: instantiated to become a distributed object during run time
EJB Client

- **Home interface**: CustomerHome
- **Remote interface**: Customer

```java
// get the JNDI naming context
Context initialCtx = new InitialContext();
// use the context to lookup the EJB home interface CustomerHome
CustomerHome home=(CustomerHome)initialCtx.lookup("/Customer");
// Use the home interface to create a new instance of customer bean
Customer customer = home.create(customerID);
// Use business method in the customer remote interface Customer
customer.changePassword(somePassword);
```
EJB Home Interface and Home Objects

- The home interface of an enterprise bean is used to control the life cycle of its bean objects by providing the methods: create, find and remove.
- **Home object** is the implementation of the home interface.
- A home interface extends the interface `javax.ejb.EJBHome`, which defines a standard set of utility methods and provide a common base type for all home interfaces.
- A home interface is defined by the enterprise bean provider and implemented by the enterprise bean container, i.e.
  - EJB programmer defines the home interface.
  - EJB container vendor provides tools that **automatically generates** the `EJBHome` object that implements home interface.

```
<Interface>
java.rmi.Remote
</Interface>
extends

<Interface>
javax.ejb.EJBHome
</Interface>
extends

<Interface>
CustomerHome
</Interface>
```
Example of Home Interface

Home interface provides life cycle methods for creating, destroying and locating beans

```java
package com.club.customer;

import java.ejb.*;
import java.rmi.RemoteException;

public interface CustomerHome extends EJBHome {
    public Customer create (String lastName, String firstName, String middleInitial)
        throws RemoteException, CreateException, NameException;
    public Customer create (String preApprovalCode)
        throws RemoteException, CreateException, CodeException;
    public Customer findByName (String name)
        throws RemoteException, FinderException;
}
```
Methods in Home Interface

- In the example, CustomerHome has 2 create methods:
  - Which create method to invoke depends on the type and number of arguments
  - The create methods return the enterprise bean remote interface, Customer
  - The create methods throw EJB exceptions and application specific exceptions: NameException and CodeException, and the exceptions are defined in an exception class as part of the package com.club.customer

- The name of a find methods always begins with find, e.g. findByName to obtain an enterprise bean remote interface

- The remove method is defined in the superinterface EJBHome

- Not all EJB have all the types of the methods

- Home interface itself is a valid remote interface for the Java RMI-IIOP in the sense that its method throws java.rmi.RemoteException and the types of arguments and return values are legal for RMI-IIOP
EJB Remote Interface and EJBOObjects

- The enterprise bean remote interface defines the business methods that a client can invoke on an individual enterprise bean object.
- Clients use the bean’s home interface to obtain references to the bean’s remote interface.
- A remote interface extends the interface javax.ejb.EJBOBJECT, which defines a standard set of utility methods and provides a common base class for all remote interfaces.
  - EJB programmer defines the remote interface.
  - EJB container vendor provides tools that generates the code for the EJBOBJECT that implements the remote interface.
- The client makes a call to a method in the remote interface, which is then forwarded to the corresponding method of the enterprise bean.
- EJBOBJECTs are managed by the EJB container.
Example of Remote Interface

package com.club.customer;

import java.ejb.**;
import java.rmi.RemoteException;

public interface Customer extends EJBOBJECT {
    public Float getBalance ( ) throws RemoteException;
    public void changePassword (String newPassword) throws RemoteException, PasswordException;
    ...
}

- 2 methods are shown in the above remove interface Customer: these are the exposed methods
- Note that the arguments and the return values must be legal types for RMI-IIOP
- Applications access enterprise beans using the remote and home interfaces at run time
EJB Client

- Locate the specific EJB container through JNDI
- Use home object to locate, create or destroy instances of Enterprise Beans
- Make use of the EJBOBJECT to invoke bean methods
- Only get reference to EJBOBJECT
- When client invokes a method, the EJBOBJECT receives the request, delegates to the corresponding bean instance
Example of EJB Client

- **Home interface** is `CustomerHome`
- **Remote interface** `Customer`
  
  ```java
  // get the JNDI naming context
  Context initialCtx = new InitialContext();
  // use the context to lookup the EJB home interface CustomerHome
  CustomerHome home=(CustomerHome)initialCtx.lookup("/Customer");
  // Use the home interface to create a new instance of customer bean
  Customer customer = home.create(customerID);
  // Use business method in the Customer remote interface
  customer.changePassword(somePassword);
  ```
EJB Runtime Behavior

1. Lookup("/Customer")
2. create() find…()
3. ejbFind…()
4. new()
5. ejbCreate()
6. bean methods
7. bean methods
8. bean methods
9. bean methods