Components and the J2EE Platform

Mariano Cilia
cilia@informatik.tu-darmstadt.de

The Evolution to Components

**Component-based Dev.**
CORBA 3.0, EJB, .NET, Business comp. approach

**Distributed Objects**
CORBA 2.0, DCOM, UML

**Object-Orientation**
C++, Eiffel, OOA/D

Structured Programming
Pascal, Ada, COBOL, RPG
structured methodologies


The goal of Component-based Development

- achieve the same levels of **plug-and-play** that are available in other industries

- integrated circuits --> software components
- the socket --> application frameworks, containers (or CTMs)
- the bus --> the object bus (ORB)

What is a Component?

- has a well-defined **plug**
  - plugs into a specific **socket**

- a piece of software that is:
  - accessed only via interfaces
  - built for customization, composition and collaboration with other components
  - small enough to reuse, replace and maintain
  - big enough to deliver, deploy, and support
  - delivered in a self-contained package
  - can be independently developed, delivered and installed

Properties of a Minimalist Component

- Self-contained
- It is not a complete application
- It can be used in unpredictable combinations
- It has a well-specified interface
- Toolability

- Metadata and introspection
- Configuration and property management
- Event notification
- Scripting
- Interoperability
- Accurately documented
- Ease of use

Properties of a Server-side Component

- Security
- Licensing
- Versioning
- Lifecycle management
- Transaction control and locking

- Persistence
- Relationships
- Self-testing
- Semantic messaging
- Self-installing
J2EE Platform

- Defines standard for developing multi-tier enterprise apps
- Based on standardized, modular components
- Provides a complete set of services to those components
- Handles many details of application behavior automatically (without complex programming)
- Write once, run anywhere
- CORBA for interaction with existing enterprise resources

Web Container and Components

- Web Container provides request dispatching, security, concurrency and life-cycle mgmt
- Web app: HTML/XML pages, JSPs, Servlets, Java classes, applets, images
- Web client packages into a Web App archive (WAR) and deployed into a web container for execution
- Servlets
  - Java classes to dynamically process requests and construct responses
  - HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services
- JSPs
  - Text-based documents that execute as servlets

J2EE - Application Architecture

Multi-tier J2EE-based Architecture.

- A J2EE app with all its modules delivered in an Enterprise Archive (EAR) file = WARs + JARs
Java 2 Enterprise Edition 1.4
- Java Servlets & Java Server Pages (JSP) 2.0
- Enterprise Java Beans (EJBs) 2.1
- Java Transaction API (JTA) 1.0
- Java API for XML Parsing (JAXP) 1.2
- Java Transaction Service (JTS)
- Java Messaging Service (JMS) 1.1
- Message Driven Beans (MDB)
- Remote Method Invocation (RMI)
- Java Database Connection 2 (JDBC) 3.0
- Java Connector Architecture (JCA) 1.5
- Java Naming and Directory Interface (JNDI) 1.2
http://java.sun.com/j2ee/1.4/

Java Enterprise Edition 5
- JEE 5.0 under review (JSR 244)
  - EJBs 3.0 (JSR 220)
  - JSF 1.2 (JSR 252)
  - JSP 2.1 (JSR 245)
  - JAX-WS 2.0 (JSR 224)
  - ...
http://java.sun.com/j2ee/5.0/

J2EE Certification
- J2EE Compatibility Test Suite (CTS)
  - Helps to ensure that products (Application Servers) support portability of apps
- J2EE Application Verification Kit (AVK)
  - Tests apps for correct use of J2EE APIs and portability across different J2EE app servers

J2EE Compatibility
- Main value proposition for J2EE Platform: Portability of Applications
- This value is achieved through J2EE Compatible Brand
- Sun, in partnership with Java Community Process, delivers:
  - J2EE Platform Specification
  - J2EE Platform Reference Implementation
  - Compatibility Test Suite (CTS)
  - J2EE Compatible Brand
  - J2EE Blueprints

Enterprise JavaBeans (EJBs)
- encapsulate the business logic and data of a business concept
- server-side components which handle
  - scalability
  - transactions
  - persistence
  - security
- distributed objects hosted in EJB containers
- portable components
  - allow multi-vendor interoperability

Enterprise JavaBeans - A Server-side Component Model
Roadmap

- EJB Specification
- Container
- Classes and Interfaces
- EJB Classification
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- Four-Tier Architecture
- Development Process
- Server Primary Services
- Deployment
- Making a Purchase Decision
- Summary

EJB Specification

- defines an architecture for a transactional, distributed object system based on components
- programming model:
  - conventions or protocols
  - set of classes and interfaces (which make up the EJB API)
- defines the bean-container contract

EJB Contracts

EJBs Specification - Model

- Components (Beans) - reusable building block, pre-built piece of encapsulated application
- Containers - execution environment for components, provides management and control services for components (i.e. an OS process or thread)
  - clients: visual containers (e.g. form, compound document, Web page)
  - servers: non-visual containers provided by application server (e.g. TPM, DBMS, Web server)

Enterprise Java Beans (EJBs)

- CORBA provides infrastructure for EJBs
- EJBs are the component model for CORBA and J2EE
- Support multi-tier apps by defining support for
  - Client-Server distribution
  - Transactions
  - Scalable state mgmt
  - Deployment
  - Security
- Bean (or component): implements business logic
- Support transient and persistent objects
- Bean provider indicates several choices via deployment descriptor
EJB v2.0
- Declarative specification of relationships between EJBs
- Declarative query language based on abstract schema (DBMS-/vendor-independent way to find entity beans at run time, based on various search criteria)
- Local interfaces for efficient invocation of EJBs in same container (call-by-reference parameter passing)
- Inter-server app interoperability RMI/IIOP
- Caching is fundamental to improve performance
- EJB v2.1
  - Focus on web services, EJB QL enhancements

Enterprise Java APIs
- Java offers a component model --> Java Beans
- Enterprise Java Platform defines a set of standard Java APIs that provide access to existing infrastructure services (ODBC metaphor)
- EJB specification defines standard model for a Java application server that supports complete portability and implements standard services
- JNDI - Java Naming and Directory Interface (access to DNS, NIS+, NDS, LDAP, etc.)

Enterprise Java APIs (cont.)
- RMI - Remote Method Invocation API creates remote interfaces for distributed computing on the Java platform
- Java IDL - creates remote interface to support CORBA communication.
  - Java IDL includes an IDL compiler and a lightweight replaceable ORB that supports IIOP
- Servlets and JSP - Servlets and Java Server Pages support dynamic HTML generation and session management

Enterprise Java APIs (cont.)
- JMS - Java Messaging Service supports asynchronous communication through reliable queuing or publish/subscribe
- JTA - Java Transaction API provides a transaction demarcation API
- JTS - distributed transaction service based on CORBA's OTS
- JDBC - database access API provides uniform DB access to relational databases

Roadmap
- EJB Specification
- Container
  - Classes and Interfaces
  - EJB Classification
    - Session Beans
    - Entity Beans
    - Message Driven Beans
  - Four-Tier Architecture
  - Development Process
  - Server Primary Services
  - Deployment
  - Making a Purchase Decision
  - Summary

The EJB Container
- Enterprise Beans run in a special environment (Container)
  - hosts and manages enterprise beans
  - manages every aspect of an enterprise bean at run time:
    - remote access to the bean
    - security
    - persistence
    - transactions
    - concurrency
    - access to and pooling of resources
The EJB Container (2)

- isolates the bean from direct access by client applications
- manage many beans simultaneously (reduce memory consumption and processing)
- pool resources
- manage lifecycles of all beans
- the client application is totally unaware of the containers resource management activities

Enterprise Beans

- do not need to write code into the bean about:
  - transactions
  - security
  - Persistence
  - (threads)
- can focus on encapsulating business logic and rules
- depend on the container for everything it needs
- cannot function outside of an EJB Container

Anatomy of an EJB Container

Enterprise Beans - Classes and Interfaces

- **home interface**: defines life-cycle methods for creating, finding, destroying beans
  - local: exposes home as local
  - remote interface: defines the public business methods of the bean
    - local: exposes remote as local
    - endpoint: exposes remote as WS
- **bean class**: where the state and behavior of the bean are implemented

Roadmap

- EJB Specification
- Container
- Classes and Interfaces
  - EJB Classification
    - Session Beans
    - Entity Beans
    - Message Drive Beans
  - Four Tier Architecture
  - Development Process
  - Server Primary Services
  - Deployment
  - Making a Purchase Decision
- Summary

Enterprise Beans - Externals

**Home Interface**

- provides life-cycle operations
- create, locate and remove objects
- provides metadata for the bean
- beans have one home interface

**Remote Interface**

- the same signature of all create methods must be used in the bean class (ejbCreate, ejbPostCreate)
- FindByPrimaryKey(key) is a standard method which locates beans based on the attributes of the primary key
Enterprise Beans - Externals
Remote Interface
- defines the public business methods of the bean
- EJB clients interact with remote interfaces that are implemented by EJB Objects
- (must throw at least java.rmi.RemoteException)

Enterprise Beans - Classes and Interfaces

Enterprise Beans - Bean Class
- implements the state and behavior of the bean
- business methods defined in the remote interface must be duplicated in the bean class (exactly the same signature)

Bean Instantiation (1)

Bean Instantiation (2)

Local Interface
- Life-cycle (home) and Business Methods (remote)
  - Avoids overhead of a distributed object protocol
    - better performance if same JVM
  - BUT
    - eliminates location transparency
      - cannot move beans to other containers
      - location of beans defined at development-time
Enterprise Beans - Internals

- interact with their container through:
  - **Callback Methods**
  - **EJBContext**
  - **Java Naming and Directory Interface (JNDI)**

Enterprise Bean - Callback Methods

- callback methods alert/inform/notify the bean about different events in its life cycle
- the container will invoke these methods to notify the bean, for example,
  - create, e.g. ejbCreate(), ejbPostCreate()
  - activate the bean, e.g. ejbActivate(), ejbPassivate()
  - retrieve or save its state from/to the database, e.g. ejbLoad(), ejbStore()
  - destroy the bean, e.g. ejbRemove()
- these methods must be implemented in the bean
- allow the bean to do some work immediately before or after some event

Enterprise Bean - EJBContext

- every bean obtains an EJBContext object (a reference to the container)
- the EJBContext interface provides methods for interacting with the container
- a bean can request information about its environment, like
  - identity of its client,
  - status of a transaction,
  - obtain a remote reference to itself, etc.

Enterprise Bean - JNDI

- for accessing naming systems
- every bean automatically has access to a special naming system called Environment Naming Context (ENC)
- the ENC is managed by the container and accessed by beans using JNDI
- allows beans to access resources like
  - JDBC connections
  - other enterprise beans
  - properties specific to that bean
  - home interfaces

Anatomy of an EJB Component

Roadmap

- **EJB Specification**
- **Container**
- **Classes and Interfaces**
- **EJB Classification**
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- **Four-tier Architecture**
- **Development Process**
- **Server Primary Services**
- **Deployment**
- **Making a Purchase Decision**
- **Summary**
**EJB Component Types**

- **Entity Beans**: model real-world objects
  - for example, Customer, Item, Supplier, ...
  - seen as persistent records in a database
- **Session Beans**: responsible for managing processes or tasks
  - coordinate the use of other (entity) beans
  - for example, Making a reservation, Sale, ...
  - transient, does not represent something in the database
- **Message Driven Beans**: react to incoming messages
  - for example, credit card verification, invoice proc.

---

**EJB Component Types v2.0**

- **Enterprise Bean**
  - **Stateful**: BMP
  - **Stateless**: CMP
  - **Bean-Managed Persistence**
  - **Container-Managed Persistence**

---

**EJB Component Types - Interfaces**

- **Entity Bean**
  - javax.ejb.EntityBean
- **Session Bean**
  - javax.ejb.SessionBean
- **Message Driven Bean**
  - javax.ejb.MessageDrivenBean

---

**Entity Beans**

- object representation of persistent data
  - describing business concepts (nouns)
  - maintained in a persistent storage (e.g. DBMS)
  - encapsulate operations of the data they represent
  - reusable and consistent interface to data in the database
  - relationships with other entities can be modeled
  - are shared by many clients
  - designed to service multiple clients, providing fast, reliable access to data and behavior while protecting the integrity of data changes
  - transactional
  - recoverable after system crash

---

**Entity Beans - Primary Key Class**

- simple class that provides a pointer into the database
- a primary key instance uniquely identifies an entity bean
- defines attributes that can be used to located a specific bean in the database
- may have several attributes (compound)
- all of them declared public
- can be undefined until deployment
  - allows to choose a system-specific key at deployment-time

---

**Entity Beans - Primary Key Class**

```java
public class CustomerPK implements java.io.Serializable {
    public int id;
    public CustomerPK() {} // id = value;
    public CustomerPK(int value) {
        id = value;
    }
    public int hashCode() {
        return id;
    }
    public String toString() {
        return String.valueOf(id);
    }
    public boolean equals(Object obj) {...}
}
```
Entity Beans Types

- **Container-Managed Persistence (CMP)**
  - manage the persistence of the entity bean
  - no database access code is written in the bean class
  - vendor tools map the entity fields to the database
- **Bean-Managed Persistence (BMP)**
  - responsible for reading and writing its own state to the database
  - the container will alert the bean as to when it is necessary to make an update or read its state from the database
  - (the container can also handle any locking or transaction)

Entity Beans - CMP

- the container is responsible for managing the persistence of the entity bean
- no database access is coded in the bean class
- callback methods must be implemented, even with no code { }
  - ejbCreate parameters are used to initialize fields
  - ejbLoad, ejbStore can be used to calculate derived values
  - at deployment time
    - vendor tools map the entity fields to the database
  - finder methods are generated automatically
  - reduces code and bugs
  - easy to port to different databases
  - if the database is not populated

Entity Beans – CMP / Seq. of Events

Entity Beans – CMP Relationships

- Relationships among entity beans specified in DD
  - cardinality: 1:1, 1:N, N:M
  - uni- and bi-directional
  - foreign keys
  - cascade delete

Entity Beans - BMP

- responsible for reading and writing its own state from/to the database
  - the container will alert the bean when necessary
  - the container can also handle any locking or transaction
- persistence logic is explicitly coded in bean class
  - depends on the DB paradigm (OO, Rel, ..)
  - how to map the persistent fields to the DB
- flexibility on how state is managed (between the bean and the DB)
  - a combination of different DBs, legacy systems, complex joins
  - is not as DB-independent as a CMP-entity

BMP - Example
**Entity Beans – BMP / Seq. of Events**

- **Client Application**
- **Container**
- **CustomerBean instance**
- **Database**

- custHome.create(2,"joe")
- ejbCreate(2,"joe")
- create entity representation in the DB
- ejbPostCreate()
- EJBObject ref
- remove entity representation from the DB
- ejbRemove()
- ejbStore()
- ejbPassivate()

**Entity Beans - Life Cycle**

- **create**
- create entity representation in the DB
- ejbPostCreate()
- **pooled**
- remove entity representation from the DB
- ejbRemove()
- ejbStore()
- ejbPassivate()

**Entity Beans - Some Considerations**

- **CMP**
  - saves time and effort if you are building from scratch (+)
  - very quickly development of a simple application (+)
  - (the new app does not rely on a complex legacy system)
  - code completely independent of the underlying DB schema (+)
  - mapping is influenciable by defining a mapping file (+)
  - difficult to migrate EJB between containers (-)

- **BMP**
  - to control how data is persisted (+)
  - for example to ERPs, legacy systems
  - very complex relationships (+)
  - JDBC makes your bean portable to any EJB Container (+)
  - many man-hours to build and maintain (-)
  - bugs (-)
  - cannot validate at compile time (-)

**EJB QL**

- query definitions are portable across DBMSs and EJB vendors
- statically compiled (from the deployment descriptor) at deployment-time
- can return a unique object or a collection
- two kinds:
  - find methods: invoked by EJB clients to obtain EJB obj references for specific beans
  - select methods: more versatile than find

**EJB QL – Some Restrictions**

- select statement is restricted to a single object or attribute
  - no support for multiple columns involving different tables
  - nested queries
  - dynamic queries
  - lack of support for Date type

**Session Beans**

- Session bean is created by client and (in most cases) exists only for one C/S session
  - performs operations on behalf of client (DB access, calculations, etc.)
  - may be transactional but not recoverable in case of system crash
  - may be stateless or maintain conversational state across methods and transactions
  - container manages context if session bean is swapped from memory
  - must maintain its own persistent data
Session Beans
- manage business process or tasks, acting as agents for the client
- work with entity beans, data or other resources to control workflow
  - workflow expresses how entities interact to model the business
- do not represent persistent data
- there are no find methods

Session Beans (2)
- hiding the fine-grained details of workflow is important because:
  - it provides flexibility in how the system evolves
  - how clients are allowed to interact with the EJB system
  - helps to thin down the client application
  - reduce traffic network and connections
- can be either:
  - Stateless
  - Stateful

Stateless Session Beans
- each method is completely independent
- everything it needs to known has to be passed via the method's parameters
- executes from beginning to the end and returns the result
- after a method execution, nothing about the details of the request are remembered
- e.g. report generation, batch processing, validation of a credit card

Stateless Session Beans - Example
- Payment Bean
  byCreditCard(Customer, CreditCard, Amount)
  {
    // verify card expiration
    // contact card company service
    // debit the amount from the Credit Card
    // all OK? --> return transaction number
  }

Stateless Session Beans - Life Cycle
- each method is completely independent
- everything it needs to known has to be passed via the method's parameters
- executes from beginning to the end and returns the result
- after a method execution, nothing about the details of the request are remembered
- e.g. report generation, batch processing, validation of a credit card

Stateful Session Beans
- often thought of as extensions of the client
  - fill in the fields on a GUI client (conversational state)
  - pressing a button executes an operation based on info entered previously
  - conversational state is kept in memory while a client uses a session
  - it is dedicated to one client for its entire life cycle
Stateful Session Beans (2)
- encapsulate the business logic and the conversational state of a client
- moving it to the server (Workspace-tier to the server)
- model workflow, managing the interaction with several other beans while maintaining conversational state
- time out: the SSB instance is destroyed and the remote reference is invalidated
- do not use instance pooling

Stateful Session Beans - Example

```java
public class reservationBean implements javax.ejb.SessionBean {
    public Customer customer;
    public Date exDate;
    public Int exCode;

    ejbCreate(cust) { customer = cust; }
    setDatesetDate(( date )) { exDate = date; }
    setExcursionsetExcursion(( code )) { exCode = code; }
    bookExcursion(( cc, price )) { ... }
    resHome.create(customer, exDate, exCode);
    payment.byCreditCard(customer, cc, price);
    printTicket(customer, exCode, exDate, price);
    ...
}
```

Stateful Session Beans - Life Cycle

Message-Driven Beans
- A Message Driven Bean (MDB) is an EJB that consumes messages
- MDBs:
  - consume messages from Queues or Topics
  - do not have home or remote interfaces
  - execute as stateless services
  - do not have return values or propagate exceptions back to the clients
  - do not have client-visible identities
  - are controlled by a container

Message Driven Beans (MDBs)
- For async consumption of messages
  - Acts as a JMS message listener
  - Resembles a stateless session bean
  - On receiving message satisfying message selector (SQL WHERE)
    - Container invokes onMessage method of MDB
- MDBs mandatory since EJB 2.0 (J2EE 1.3)
  - J2EE 1.3: Restricted to JMS
  - J2EE 1.4: Messages of provider by means of plug-ins (J2EE Connector adapter)
    - Support any messaging system (e.g. SMTP, SNMP, ...)
MDBs – Life Cycle

Transactionable Behavior

- EJB model supports notion of implicit transactions
  - EJBs don’t have to specify the transactional demarcation point in order to participate in distributed transactions
  - EJB execution environment automatically manages transaction on behalf of the EJBs
  - transaction policies can be defined during deployment
  - transactions may be controlled by client-side applications

Roadmap

- EJB Specification
  - Container
  - Classes and Interfaces
  - EJB Classification
    - Session Beans
    - Entity Beans
    - Message Driven Beans
- Four-Tier Architecture
  - Development Process
  - Server Primary Services
  - Deployment
  - Making a Purchase Decision
  - Summary

Four-tier Architecture

The EJB Development Process
Packaging - JAR files

- platform-independent file for compressing, packaging and delivering several files together
- based on a ZIP file format
- classes and interfaces associated with Beans are packaged in a JAR file
- has an index (DD) describing all beans in the file

Roadmap

- EJB Specification
- Container
- Classes and Interfaces
- EJB Classification
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- Four-Tier Architecture
- Development Process
- Server Primary Services
- Deployment
- Making a Purchase Decision
- Summary

EJB Server - Primary Services

- CORBA services
  - add-on subsystems explicitly utilized by the application code
  - complicated when they are used in combination
- CTMs automatically manage all primary services
- Primary Services:
  - Persistence
  - Transactions
  - Concurrency
  - Distributed Objects
  - Naming
  - Security

Primary Services - Persistence

- entity beans are persistent
- CMP bean's state is automatically managed by a persistence service
- the container is responsible for synchronizing an entity bean's instance fields with the data in the database
- at deployment time, (vendor) tools map the entity fields to the database automatically or according to a mapping file
  - Objects-to-Relations mappings

Primary Services - Transactions

- a transaction is a set of tasks (unit-of-work) that is executed together
- Atomic
  - all tasks in a transaction must be completed successfully
- an EJB Server monitors the transaction to ensure that all the tasks are completed successfully
- JTS is realized on top of CORBA OTS
- How to manage the bean at run time:
  - declaring transactional attributes at deployment time

Primary Services - EJB Transaction Attributes

- not supported
  - requires new
- supports
  - mandatory
- required
  - never

Diagram: EJB Server - Primary Services

Diagram: Roadmap

Diagram: Primary Services - Persistence

Diagram: Primary Services - Transactions

Diagram: Primary Services - EJB Transaction Attributes
Primary Services - Concurrency
- entity beans are shared components
- EJB needs to protect the data represented by the shared bean
- EJB (by default) prohibits concurrent access to bean instances
  - if one client invokes a method on the EJB Object, no other client can access that bean instance until the method invocation is complete
- beans can not be multi-threaded (own threads)
  - by default, non-reentrant

Primary Services – Distrib. Obs.
- as long as the EJB server supports the EJB client view (remote interface and home interface), any distributed object protocol can be used
- Clients (implemented using different programming languages) can access beans using different protocols
- Remote Method Invocation (RMI): language abstraction (or programming model) for any kind of distributed object protocol
  - RMI over JRMP (Java Remote Method Protocol)
  - RMI over IIOP (Inter-ORB Protocol) CORBA-compliant

Primary Services - Naming
- provide clients with a mechanism for locating distributed objects
- two purposes:
  - object naming: association of a distributed object with a natural language name or identifier
    - a name is really a pointer or an index to a specific distributed object
  - lookup API: provides the client with an interface to the naming system
    - allow clients to connect/bind to a distributed service and request a remote reference to a specific object
- EJB spec mandates the use of the JNDI as a lookup API on Java clients

Primary Services - Security
- Authentication: validates the identity of the user
  - login, secure ID cards, security certificates, etc.
- Access Control: applies security policies that regulate what a specific user can and cannot do
- Secure Communication: between a client and a server
  - encrypting the communication: message encoding
    - normally cryptographic keys
  - physical isolation: a dedicated network connection
    - expensive, limiting and pretty much impossible on the Internet

Roadmap
- EJB Specification
- Container
- Classes and Interfaces
- EJB Classification
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- Four-Tier Architecture
- Development Process
- Server Primary Services
- Deployment
  - Making a Purchase Decision
- Summary

Deployment
- the bean code contains only business logic
- primary services are handled automatically by the EJB Server
- how to apply primary services to each bean class at run time?
  - deployment descriptors
    - describe the runtime attributes of server-side components
    - allow to customize behavior of software at runtime without having to change the software itself (assembly)
Deployment Descriptors (DD)

- tells the deployment tools:
  - what kind of beans are in the JAR file (Session or EntityBeans)
- when deploying
  - the container uses the deployment descriptor to learn about the beans contained in the JAR file
- once bean class and interfaces have been defined, a deployment descriptor is created:
  - ejb-jar.xml
- deployment descriptor is packaged together with the JAR file

DD - Deployment Example

```xml
<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>
        This Customer enterprise bean entity represents a customer.
      </description>
      <ejb-name>CustomerBean</ejb-name>
      <home>com.example.customer.CustomerHome</home>
      <remote>com.example.customer.Customer</remote>
      <ejb-class>com.example.customer.CustomerBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <primary-key-class>com.example.customer.CustomerPK</primary-key-class>
      <reentrant>False</reentrant>
      <cmp-field><field-name>id</field-name></cmp-field>
      <cmp-field><field-name>name</field-name></cmp-field>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

DD - Assembly Descriptor

- how the beans are assembled into an application
- the assembler customizes/configures the bean (sets attributes) for a particular application
- these attributes tell the container:
  - how they should be managed in transactions (container transaction)
  - who has access to the beans at runtime (security roles)
  - who can execute methods (method permission)

DD - Assembly Example

```xml
...<assembly-descriptor>
  <security-role>
    <description>
      This role represents everyone who is allowed full access to the customer bean.
    </description>
  </security-role>
  ...<method-permission>
    <role-name>everyone</role-name>
    ...<method-name>*</method-name>
    <trans-attribute>Required</trans-attribute>
  </method-permission>
</assembly-descriptor>
```

The EJB Deployment Process

Remember! EJBs - Classes and Interfaces
Clients

- interact with a set of interfaces that provide access to beans and their business logic
- JNDI API: to find and access beans (regardless of their location on the network)
- EJB client-side API:
  - set of interfaces and classes that a developer uses on the client to interact with beans
  - client-jar
    - includes the interfaces and classes needed by a client app
      - e.g. remote, home, primary key, app exceptions
    - Also access from Servlets, JSPs, CORBA, ...

Clients - Example

```java
CustomerHome custHome;
Object ref;
// obtain a reference to the CustomerHome
ref = jndiContext.lookup("java:comp/env/ejb/Customer");
// cast returned object to the appropriate datatype
custHome = PortableRemoteObject.narrow(ref, CustomerHome.class);
// use the home interface to create a new customer bean instance
Customer customer = custHome.create(customerID);
// use a business method (remote interface) on the customer
customer.setName(name);
```

Roadmap

- EJB Specification
- Container
- Classes and Interfaces
- EJB Classification
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- Four-Tier Architecture
- Development Process
- Server Primary Services
- Deployment
- Making a Purchase Decision
- Summary

Purchase Decision - Tech. Aspects

- EJB Specification Conformance
  - Sun Compatibility Test (J2EE Seal of Approval)
- Persistence
  - BMP, CMP (plug-in Persistence Manager)
- Integrated Tier Support
- IDE Integration
- Online deployment
- Integration with Bean Providers
Purchase Decision - Tech. Aspects (2)
- In-Memory Data Cache
- Distributed Transactions
- Scalability
- High Availability
- Security
- Intelligent Load Balancing
- Mirroring
- Clean Shutdown
- Existing Enterprise System Integration
- Asynchronous Messaging Support
- Specialized Services

EJB App Servers - Roots
- EJB Application Servers could be developed from scratch or through interfaces to existing products
  - TP monitors (IBM TXSeries, CICS/390, Tuxedo)
  - Component TX servers (Sybase Jaguar CTS)
  - CORBA systems (BEA M3, Inprise VisiBroker/ITTS)
  - Relational DBMSs (DB2, Oracle8i onwards)
  - OODBMSs (GemStone/J)
  - Object-relational caching systems (Persistence)
  - Web application servers (BEA WebLogic, IBM WebSphere, Sun iPlanet (Sun One), Borland...)

App Server Vendors
- BEA/WebLogic Ent.
- IBM/WebSphere
- TP monitors
- CORBA CTM
- ORB
- ORB Vendors
- IONA/OrbixOTM
- Inprise/Visigenic
- Database Vendors
- Oracle/App. Server
- GemStone/J
- Sybase/Jaguar
- Sun AOL Netscape
- iPlanet
- Web Server Vendors
- NetBeans - WebLogic Server
- ORB
- Application Servers (BEA WebLogic, IBM WebSphere, Sun iPlanet, Borland...)

Making a Purchase Decision - Prods
- Borland:
  - JBuilder/Together – BAS
- IBM:
  - Java Visual Age/Rational/Eclipse - WebSphere
- BEA:
  - Visual Café - WebLogic Server
- SUN/Netscape:
  - NetBeans - iPlanet App Server
- Oracle:
  - JDeveloper (+Oracle10g) - Oracle App. Server
- Specialized Services

Roadmap
- EJB Specification
- Container
- Classes and Interfaces
- EJB Classification
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- Four-Tier Architecture
- Development Process
- Server Primary Services
- Deployment
- Making a Purchase Decision
- Benchmarks, Patterns, Summary

Benchmarking for J2EE Application Servers
SPECjAppServer2002 Benchmark
- Industry Standard Application Server Benchmark
- J2EE1.3/EJB2.0 port of SPECjAppServer2001 (SJAS2001)
- Both SJAS2001 and SJAS2002 based on Sun's ECperf 1.1
- Heavy-duty synthetic B2B E-Commerce Workload
- Measures performance and scalability of J2EE AppServers
- For more info visit: http://www.spec.org/osg/jAppServer/

Business Problem Modeled
- Order / Inventory Management
  - B2C Customer Interactions, Online-Ordering
- Just-in-Time Manufacturing
- Production / Assembly Lines
- Supply-Chain Management
  - B2B Interactions with External Suppliers

Application Design
- Benchmark Components:
  1. EJBs – J2EE application deployed on System Under Test (SUT)
  2. Supplier Emulator – web app. simulating external suppliers
  3. Driver – Java app. simulating clients interacting with the system
- RDBMS is used for persistence
- Benchmark’s Throughput is function of chosen Transaction Injection Rate
- Performance metric provided is TOPS/sec = total number of business transactions completed in the customer domain + total number of workorders completed in the manufacturing domain, normalized per second

Components
- Driver
  - Client JVM
  - RMI
  - SUT
  - EJB X
    - Receivers
  - EJB Z
    - Buyers
  - J2EE AppServer
- Supplier Emulator
  - HTTP
  - Emulator Servlet
  - Web Container
Some Results – Dual-node

- Oracle = 431.26 total operations per second (TOPS), price/performance $160.62/TOPS
  - Configuration:
    - Application Server: Oracle AS 10g
    - on 2 Intel Xeon at 3000 Mhz, 2GB of main memory
    - Hardware: ProLiant ML370G3
    - JVM: BEA JRockit 8.1
    - JDBC Driver: Oracle JDBCDriver 10.1.0.1.1 (Thin)
    - OS: RedHat Enterprise Server 2.1
    - Database: Oracle 10g
      - on 2 Intel Xeon at 3000 Mhz, 2GB of main memory
    - Total System Cost: US$ 69,267

Some Results – Multi-node

- IBM = 2,575.34 total operations per second (TOPS), price/performance $330.07/TOPS
  - Configuration:
    - Application Server: WebSphere 5.1 Application Server,
      - 9 servers, with 2 Intel Xeon at 2400 Mhz, 1.5GB of RAM
    - Hardware: eServer xSeries 335 Cluster
    - JVM: IBM's 1.4.1 JRE
    - JDBC Driver: IBM DB2 JDBC Universal Driver Provider
    - OS: SuSE Linux Enterprise Server 8 SP2A with ReiserFS
    - Database: DB2 Universal Database v8.1.1.32 FP4, Enterprise Server Edition running on AIX
      - 1 DB Server, with 8 CPUs and 16GB of main memory
    - Total System Cost: US$ 850,025

Some Results – Distributed

- IBM = 435.57 total operations per second (TOPS), price/performance $856.79/TOPS
  - Configuration:
    - Application Server: WebSphere 5.0.1 Application Server,
      - 7 servers, with 2 Intel Xeon at 2400 Mhz, 1.5GB of RAM
    - Hardware: eServer xSeries x335
    - JVM: IBM's J2RE 1.3.1
    - JDBC Driver: IBM DB2 JDBC Driver
    - OS: M$ Windows 2000 Server
    - Database: DB2 Universal Database v8.1 Workgroup Server
      - on 4 Intel Xeon MP at 2000 Mhz and 4GB of RAM
    - Total System Cost: US$ 373,188
  - Check it yourself online under:

What are patterns?

- Patterns communicate:
  - "Solution to a recurring Problem in a Context"
  - A design which is used by others
  - An abstraction which can be realized
  - Discovered, proven expert solutions
  - Creates a higher level vocabulary
  - Combined to solve bigger problem

J2EE Patterns

From JavaOne Tutorials and Books
J2EE Pattern Catalog

**Presentation Tier**
- Intercepting Filter
- Front Controller
- Context Object
- Application Controller
- View Helper
- Composite View
- Service To Worker
- Dispatcher View

**Business Tier**
- Business Delegate
- Service Locator
- Session Façade
- Application Service
- Business Object
- Composite Entity
- Transfer Object
- Transfer Object Assembler
- Value List Handler

**Integration Tier**
- Data Access Object
- Service Activator

Patterns - Benefits
- Reduce re-inventing the wheel
- High-level language for discussing design issues
- Great way for organizing knowledge
- Combinations of patterns create reusable architectures
  - Promoting design re-use
- Increase developer productivity, communication
- Large community around patterns

Support/Adoption
- Developer Community
- 3rd Party
  - Logic Library
  - Object Venture
  - TogetherSoft
  - Rational
  - The Server Side
- Sun
  - Sun™ ONE Studio (5.x)
  - J2EE BluePrints/Java
  - Pet Store/Adv Builder
  - Sun Education Courses
- patterns.java.net

Situation I - Motivation

Issues with previous diagram:
- High network overhead
- Poor concurrency
- High coupling
- Poor reusability
- Poor maintainability
- Poor separation of development roles

Problem:
- How can an EJB client execute a use case business logic in one transaction and one bulk network call?

Solution: Session Façade
Session Façade Benefits

- Low network overhead
- Transactional benefits
- Low coupling
- Good reusability
- Good maintainability
- Clean verb-noun separation
- High concurrency

Before Session Façade

Direct Entity Bean Access

With Session Façade

Motivation

- Consider a client request uses methods of multiple EJB components in a transaction
- The use case is a long running
- Client need not or cannot wait for a response
- Problem:
  - How can a client execute a long running transaction without blocking?

Solution: Message Façade

- What’s needed:
  - A server-side intermediary to client (like session façade)
  - But one that doesn’t require client to block and wait
- Solution: Message Driven Beans
  - MDBs are designed just for this!
  - Use MDBs to create a fault tolerant, asynchronous façade
  - Clients should have access to MDBs only, not session/entity beans

Solution: Message Façade
Message Façade

Benefits
- Asynchronous execution
- Eliminates single point of failure
- Asynchronous execution is guaranteed

Drawbacks
- Messages are weakly-typed input parameters
- MDBs have no return values
- MDBs do not propagate exceptions back to clients

Summary and Conclusions

EJB Container System

- EJB model defines relationship between an EJB component and an EJB container system
- No specific container system is required
  - any application execution system, e.g. an application server, can be adapted to support EJBs by adding support for services defined in the EJB specification
  - application execution system provides portability
- ==> EJBs can run in any execution system (EJB container) that supports EJB standard

EJB Server

- An EJB execution system is called an EJB Server
- EJB server provides standard services to support EJB components
  - management and control services for a class of EJB objects, life-cycle management, persistence management, security services
  - provide access to standard distributed transaction management mechanism

Compatibility

- Specification does not provide details about concurrency control, resource management, thread pooling, etc.
  - ==> multiple implementation options
  - ==> product differentiation
  - ==> product incompatibility
- Sun is accelerating standardization and certification to guarantee portability (i.e. create an EJB in one compliant tool and run it within any compliant container)

Compatibility (cont.)

- 4 Versions released in 4 years (1.0, 1.1, 2.0, 2.1)
  - 3.0 coming soon
- major differences:
  - message driven beans for asynchronous interactions
  - CMP relationships for explicit representation of relationships between beans and their mapping
  - standardization of EJB-QL
  - access to persistent attributes via abstract accessor methods (needed to realize container managed relationship)
Summary and Outlook

- J2EE has become widely successful
- M$ attempting to fight with .NET
- Revival of object to relational DB mapping
- XML parsing overhead is high
- Standardized benchmarks beginning to take off
  - SPECjAppServer
- EJBs yet to expose full spectrum of DB functionality
- Web services becoming popular, even for intranet usage (being extended with TX, business process,...)

Summary and Outlook (cont.)

- Compensating TX support begins to appear
- Research needed on self-tuning/adapting systems
- Caching appears everywhere
- Identity mgmt is crucial
- There will be vendor consolidation
- DBMS researchers and practitioners paying more attention on app server area