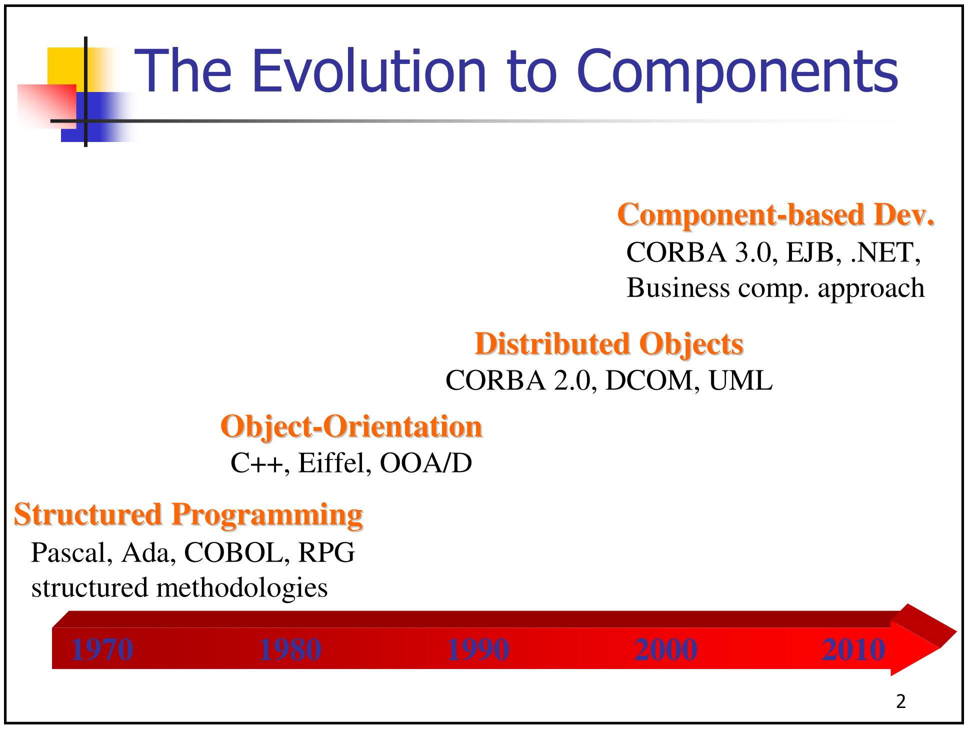
Components and the J2EE Platform

Mariano Cilia
mcilia@gmail.com

1



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

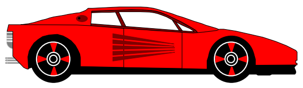
Structured Programming
Pascal, Ada, COBOL, RPG
structured methodologies

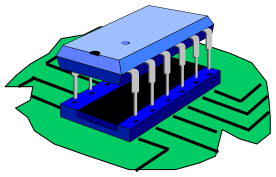
1970 1980 1990 2000 2010

2

The goal of Component-based Development

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

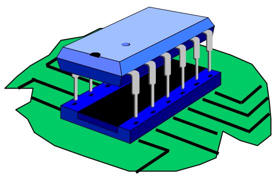




- integrated circuits --> software components
- the socket --> application frameworks, containers (or CTMs)
- the bus --> the object bus (ORB)

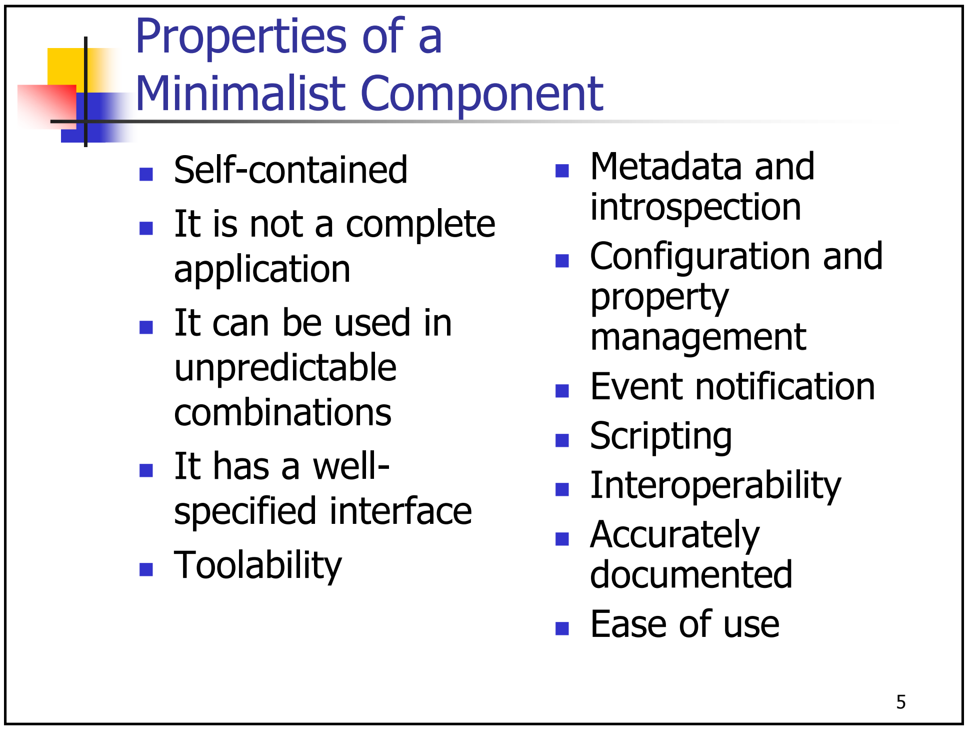
3

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

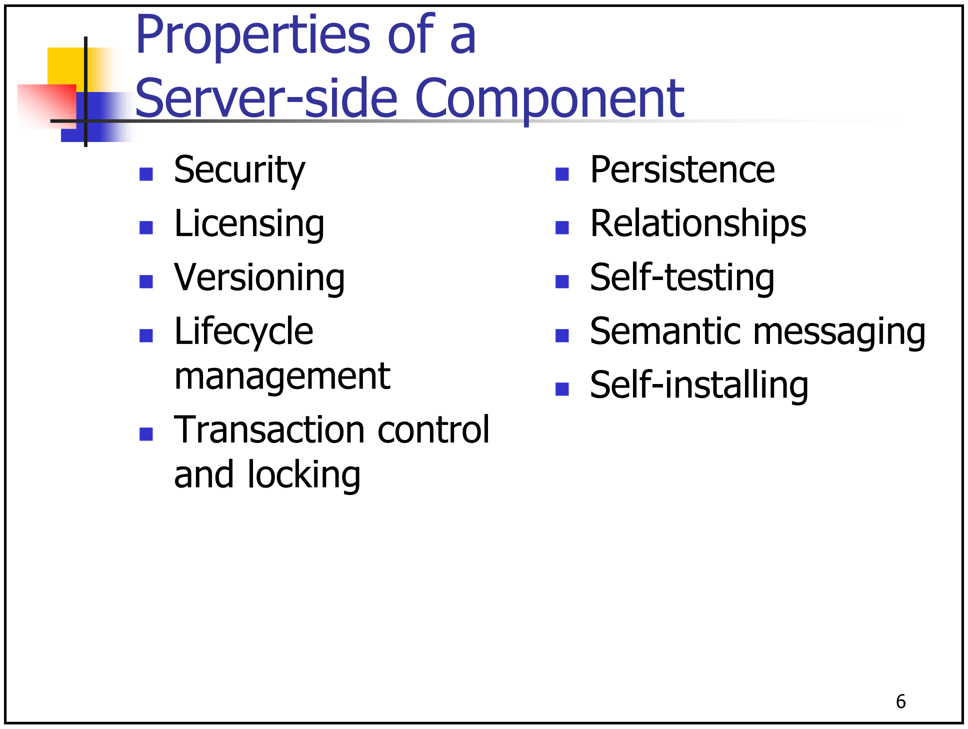
4



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

5



Properties of a Server-side Component

- Security
- Licensing
- Versioning
- Lifecycle management
- Transaction control and locking
- Persistence
- Relationships
- Self-testing
- Semantic messaging
- Self-installing

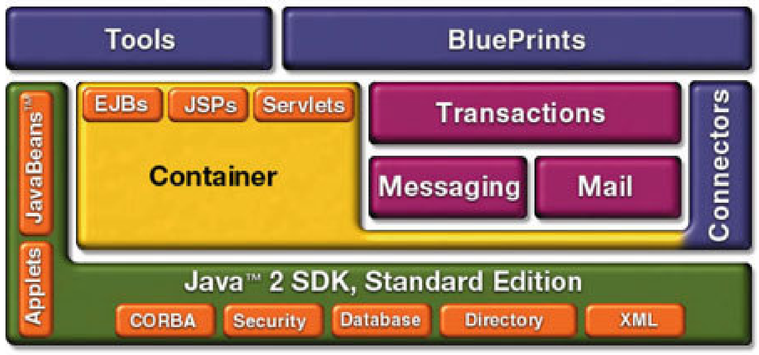
6

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

7

J2EE Platform – Big Picture



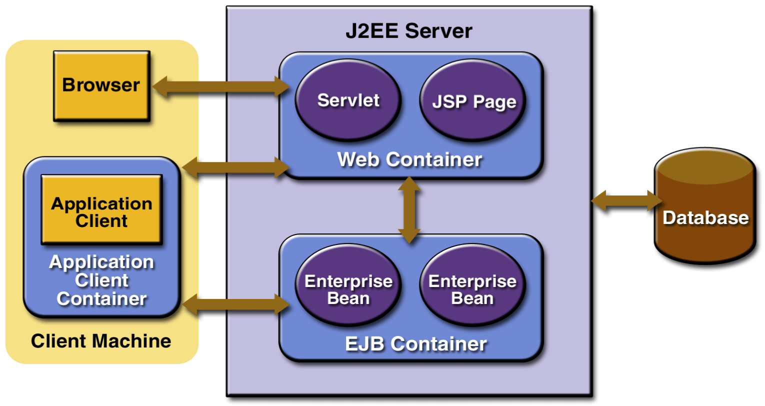
8

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

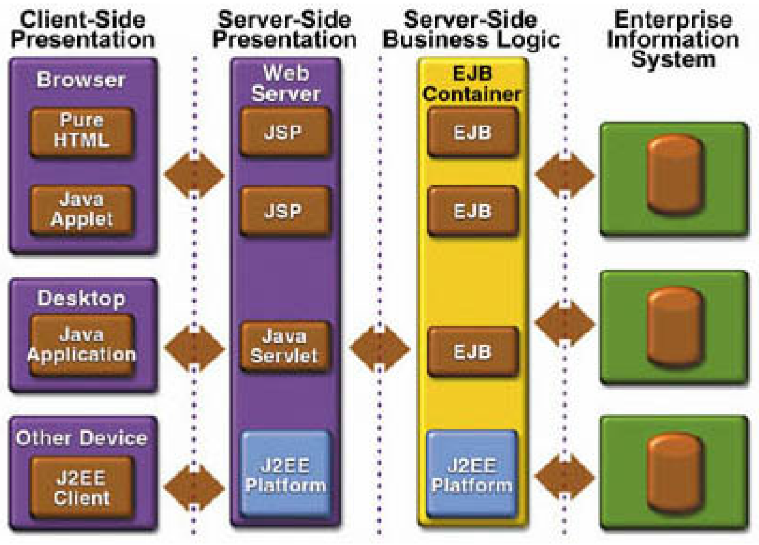
9

J2EE Server and Containers



10

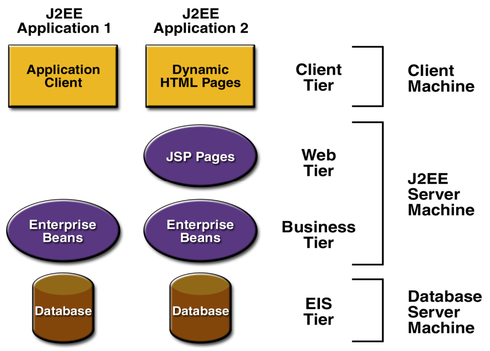
J2EE - Application Architecture



11

Multi-tier J2EE-based Architec.

- A J2EE app with all its modules delivered in an Enterprise Archive (EAR) file = WARs + JARs



12

Set of 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.)

13

Set of 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

14

Set of APIs (cont.)

- JMS - Java Messaging Service supports asynchronous communication through reliable queueing 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

Java 2 Enterprise Edition 1.3

- Enterprise JavaBeans Specification (EJB) 2.0
- J2EE Connector Specification (JCA) 1.0
- JavaServer Pages Specification (JSP) 1.2
- Java Transaction API Specification (JTA) 1.0.1B
- Java Transaction Service Specification (JTS) 1.0
- Java Naming and Directory Interface Specification (JNDI) 1.2.1
- Java Message Service Specification (JMS) 1.0.2b
- JDBC Specifications (JDBC) 3.0
- RMI over IIOP
- Java Servlet Specification 2.3
- Enterprise JavaBeans to CORBA Mapping 1.1
- Java IDL API
- JavaMail API Specification 1.2
- JavaBeans Activation Framework Specification 1.0.1

- http://java.sun.com/j2ee/1.3/

Java 2 Enterprise Edition 1.4

- Enterprise Java Beans (EJBs) 2.1
- Java Connector Architecture (JCA) 1.5
- Java Servlets & Java Server Pages (JSP) 2.0
- Java Transaction API (JTA) 1.0
- Java Transaction Service (JTS)
- Java Naming and Directory Interface (JNDI) 1.2
- Java Messaging Service (JMS) 1.1
- Java Database Connection 2 (JDBC) 3.0
- Java API for XML Parsing (JAXP) 1.2
- Message Driven Beans (MDB)
- Remote Method Invocation (RMI)

- http://java.sun.com/j2ee/1.4/

17

Java Enterprise Edition 5 (JSR 244)

- Web Services Technologies
  - Implementing Enterprise Web Services (JSR 109)
  - Java API for XML-Based Web Services (JAX-WS) 2.0 (JSR 224)
  - Java API for XML-Based RPC (JAX-RPC) 1.1 (JSR 101)
  - Java Architecture for XML Binding (JAXB) 2.0 (JSR 222)
  - SOAP with Attachments API for Java (SAAJ) (JSR 67)
  - Streaming API for XML (JSR 173)
  - Web Service Metadata for the Java Platform (JSR 181)
- Web Application Technologies
  - Java Servlet 2.5 (JSR 154)
  - JavaServer Faces 1.2 (JSR 252)
  - JavaServer Pages 2.1 (JSR 245)
  - JavaServer Pages Standard Tag Library (JSR 52)
- Enterprise Application Technologies
  - Enterprise JavaBeans 3.0 (JSR 220)
  - J2EE Connector Architecture 1.5 (JSR 112)
  - Common Annotations for the Java Platform (JSR 250)
  - Java Message Service API (JSR 914)
  - Java Persistence API (JSR 220)
  - Java Transaction API (JTA) (JSR 907)
  - JavaBeans Activation Framework (JAF) 1.1 (JSR 925)
  - JavaMail (JSR 919)
- Management and Security Technologies
  - J2EE Application Deployment (JSR 88)
  - J2EE Management (JSR 77)
  - Java Authorization Contract for Containers (JSR 115)
- http://java.sun.com/javaee/

18

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

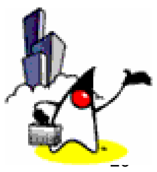
19

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

20

Enterprise JavaBeans - A Server-side Component Model



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

27

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

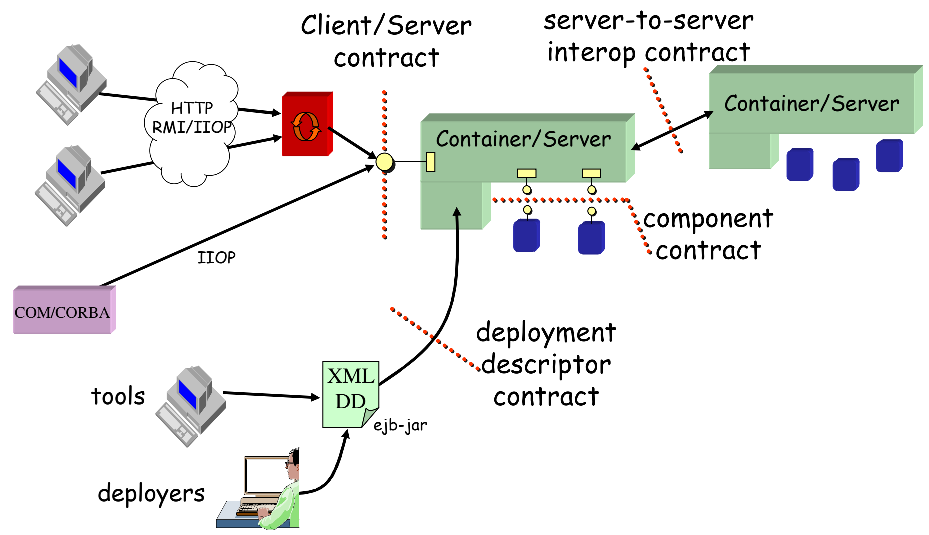
28

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

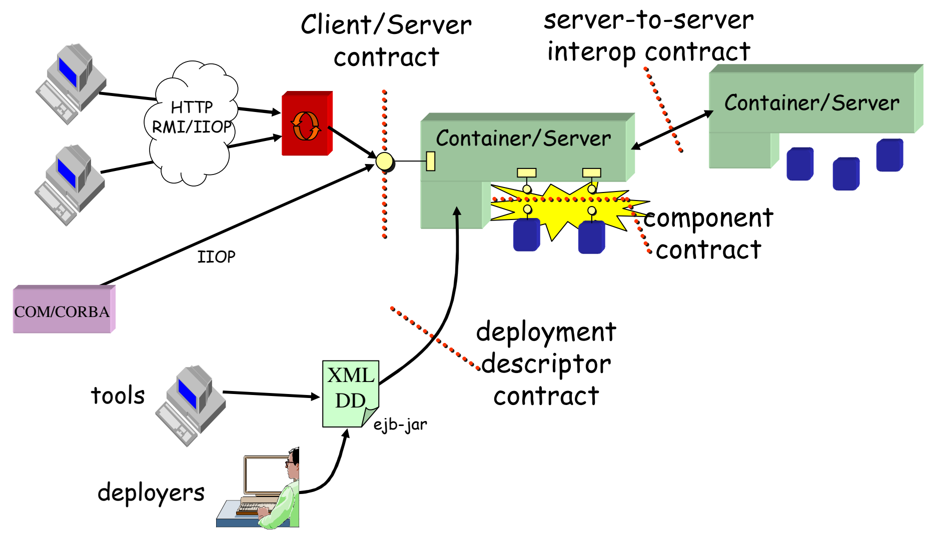
29

EJB Contracts



30

EJB Contracts



31

EJBs Specification - Model

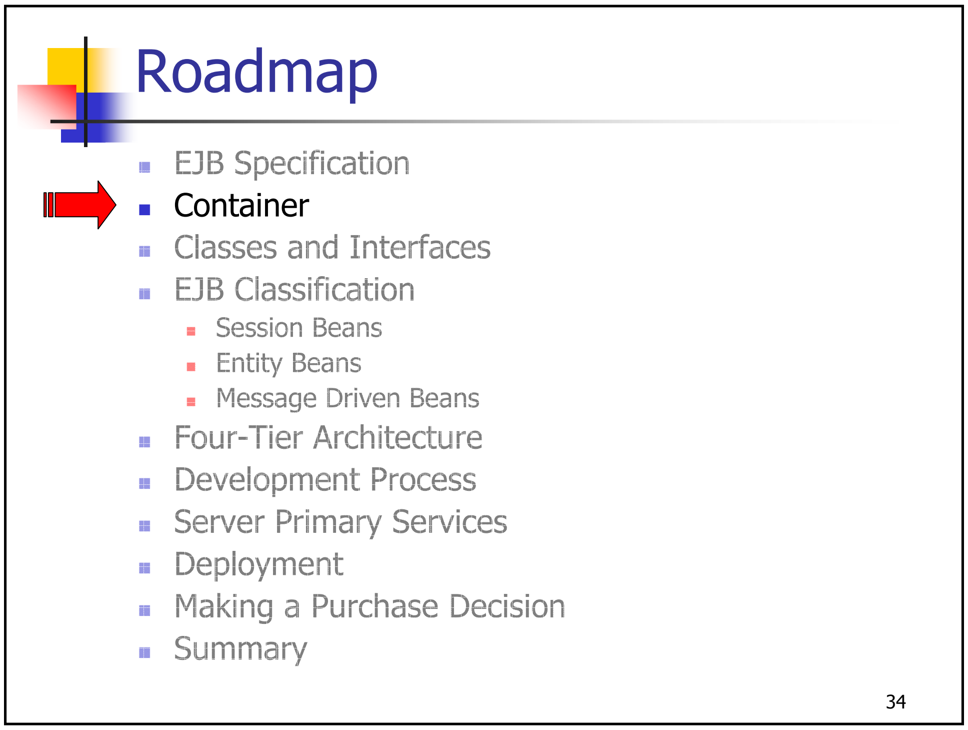
- Components (Beans) - reusable building block, pre-built piece of encapsulated application
- Containers - execution environment for components, provides management and control services for components (i.e. an OS process or thread)
  - clients: visual containers (e.g. form, compound document, Web page)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

32

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

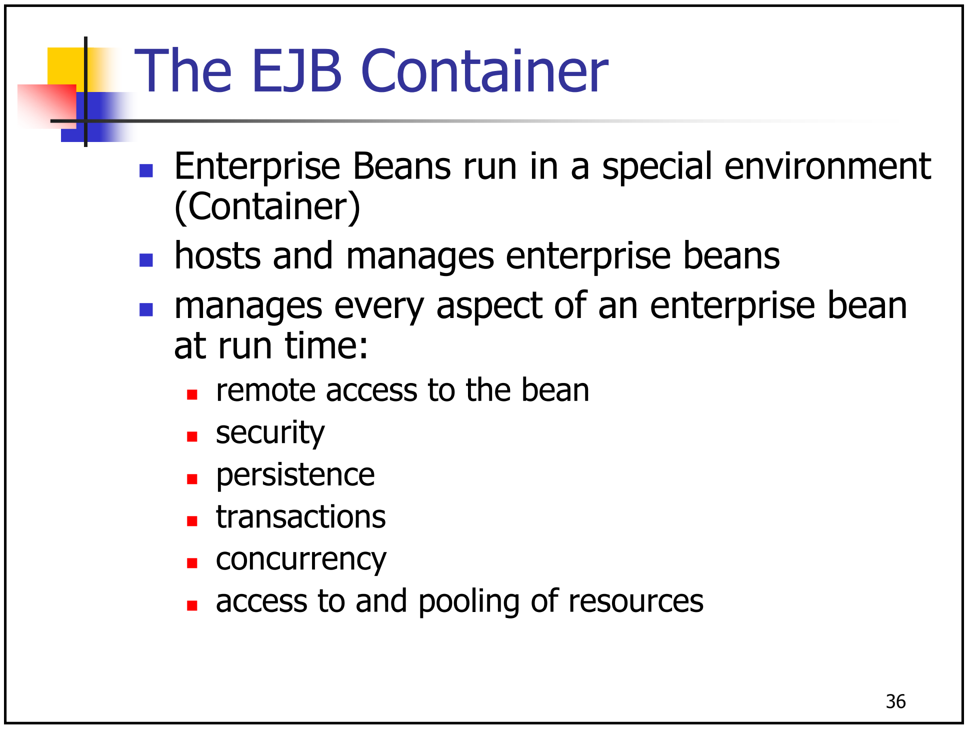
33



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

34



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

36

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

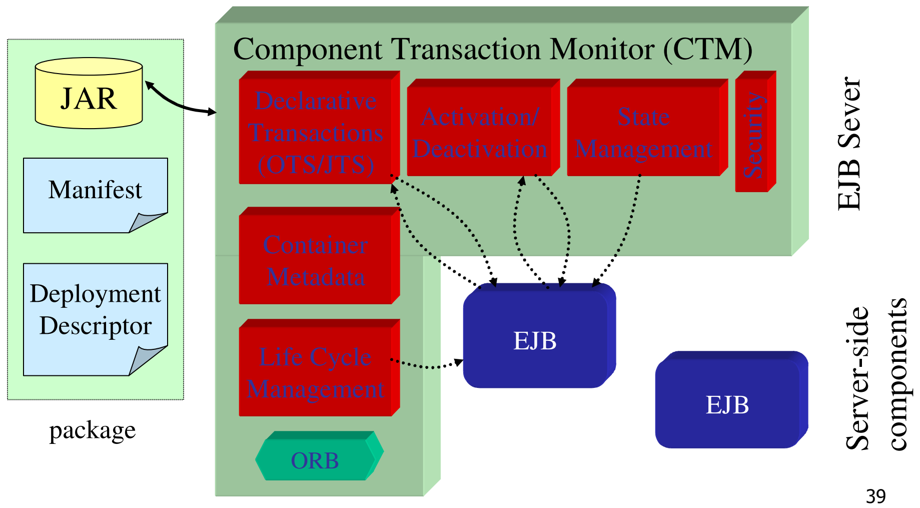
37

Enterprise Beans

- developers
  - 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

38

Anatomy of an EJB Container



39

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

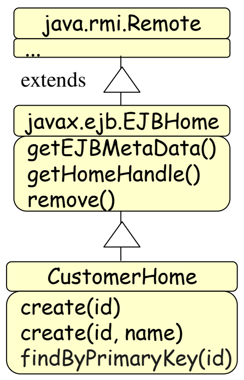
40

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

41

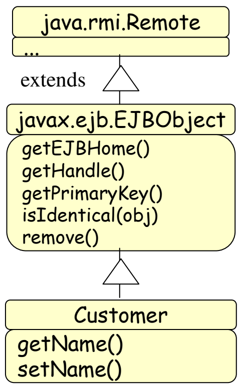
Enterprise Beans - Externals Home Interface



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

42

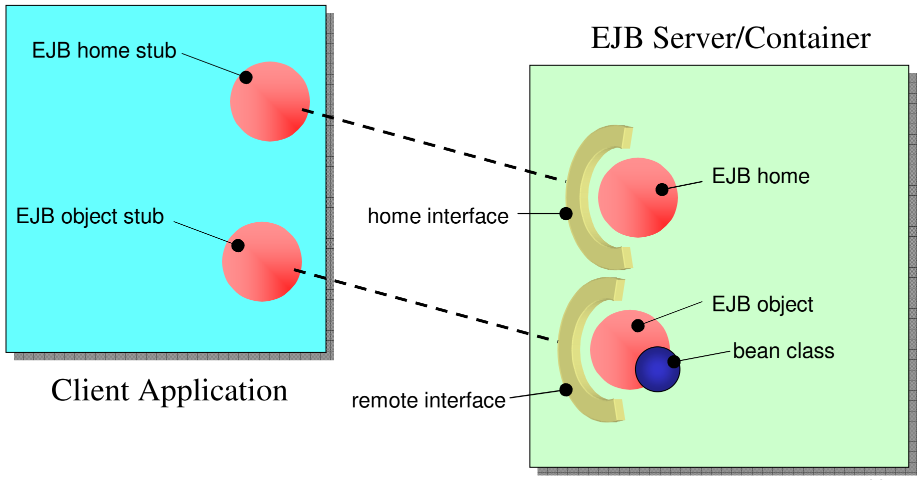
Enterprise Beans - Externals Remote Interface



- 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)

43

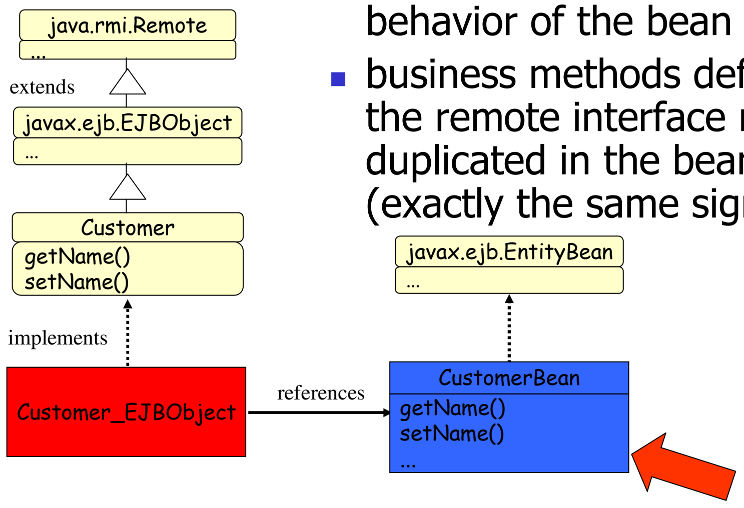
Enterprise Beans - Classes and Interfaces



44

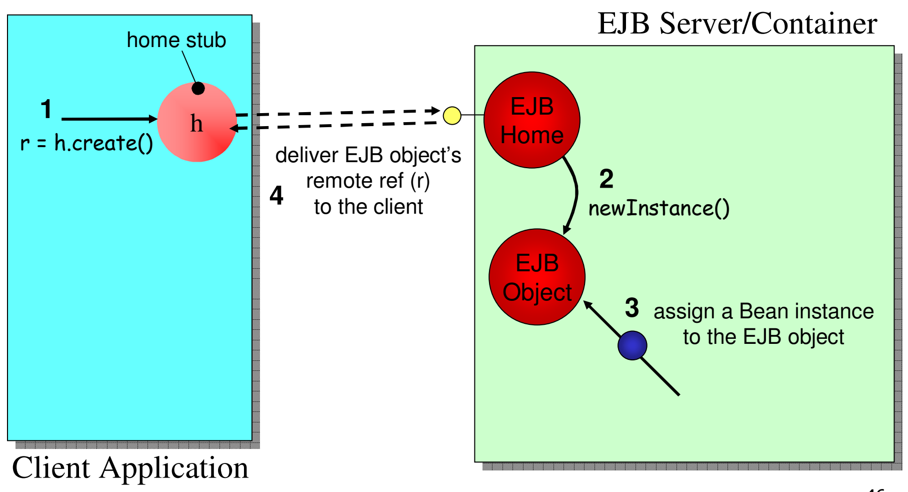
Enterprise Beans - Bean Class

- 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)



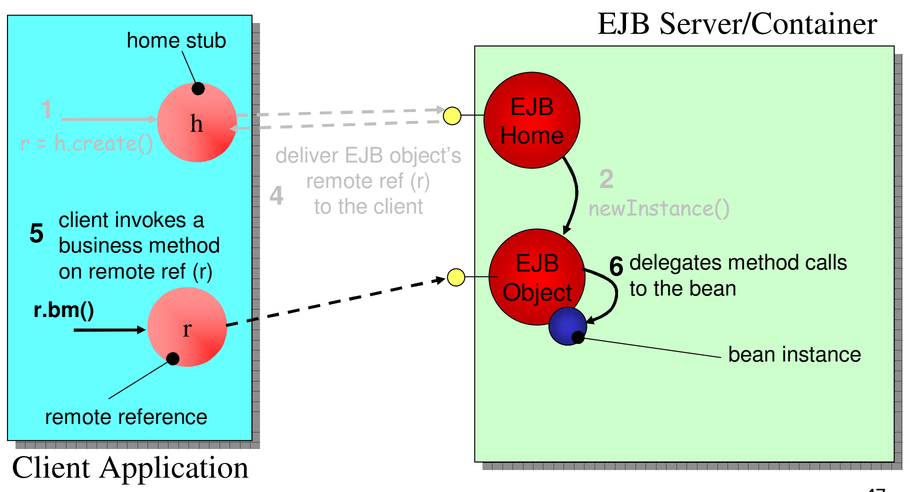
45

Bean Instantiation (1)



46

Bean Instantiation (2)


EJB Server/Container
home stub
1
r = h.create()
h
deliver EJB object's remote ref (r) to the client
4
EJB Home
2
newInstance()
5 client invokes a business method on remote ref (r)
r.bm()
r
EJB Object
6 delegates method calls to the bean
bean instance
remote reference
Client Application

47

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

48

Enterprise Beans - Internals

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

49

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

50

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.

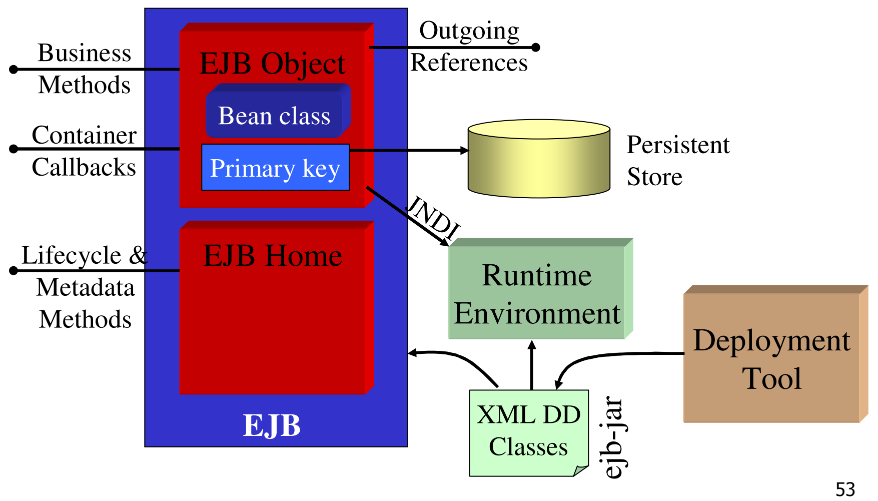
51

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

52

Anatomy of an EJB Component



53

Roadmap

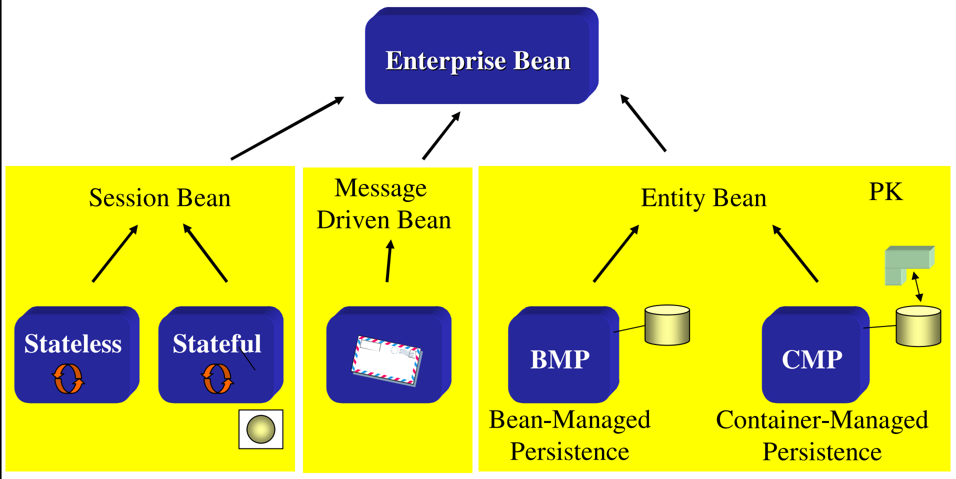
- EJB Specification
- Container
- Classes and Interfaces
- EJB Classification
  - Session Beans
  - Entity Beans
  - Message Driven Beans
- Four-Tier Architecture
- Development Process
- Server Primary Services
- Deployment
- Making a Purchase Decision
- Summary

54

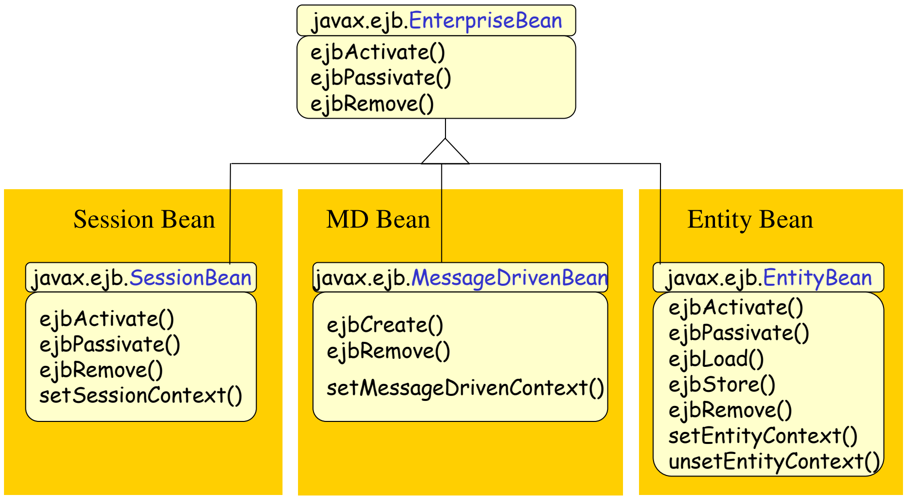
EJB Component Types

- **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



EJB Component Types - Interfaces



58

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

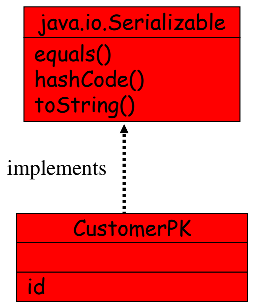
59

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

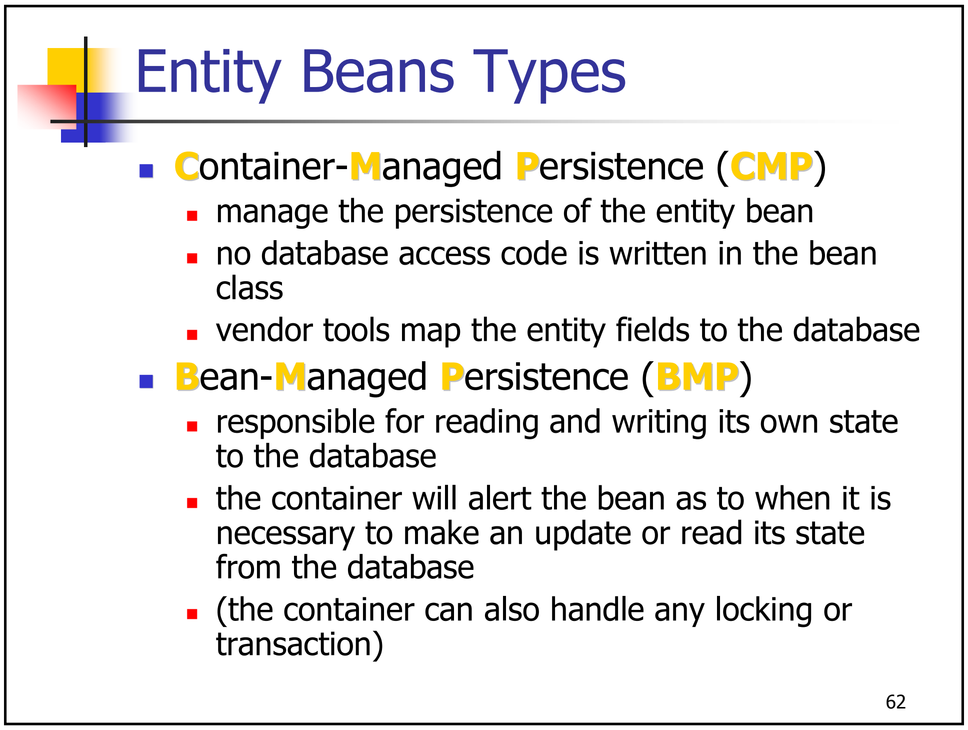
60

Entity Beans - Primary Key Class



```
public class CustomerPK implements java.io.Serializable {

        public int id;

        public CustomerPK() {}
        public CustomerPK(int value) {
                id = value;
        }

        public int hashCode() {
                return id;
        }
        public String toString() {
                return String.valueOf(id);
        }
        public boolean equals(Object obj) {...}
}
```

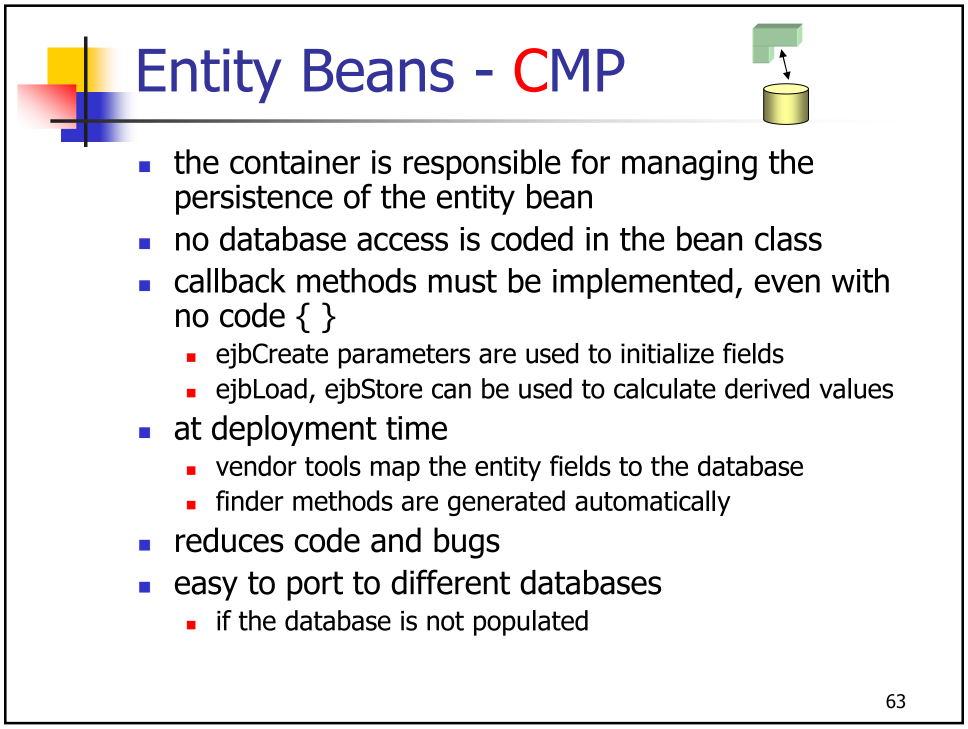
61



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)

62

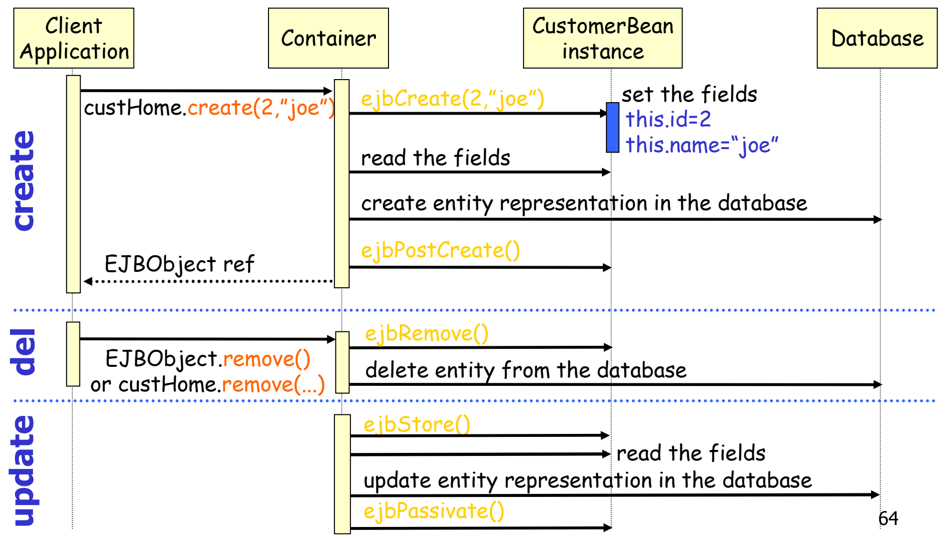


Entity Beans - CMP

- 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

63

Entity Beans – CMP / Seq. of Events



64

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

65

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

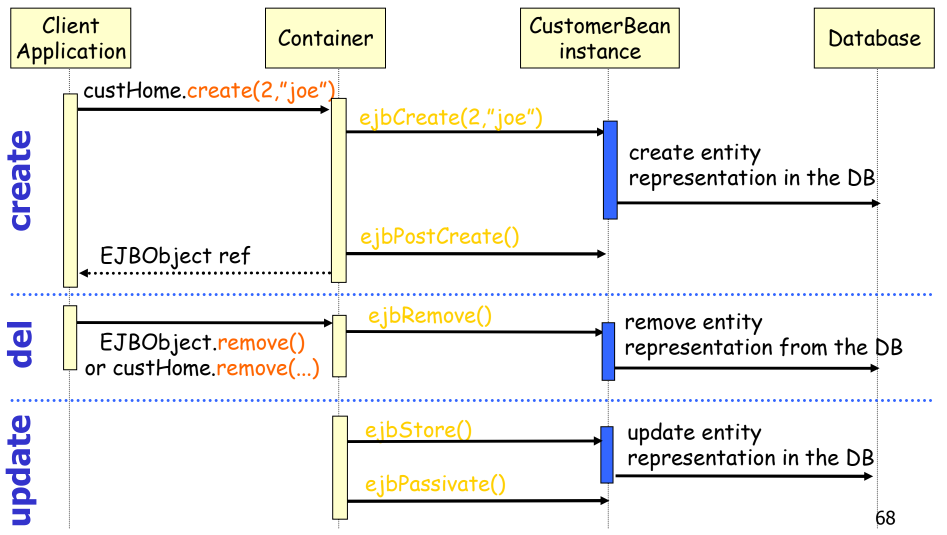
66

BMP - Example

```
public CustomerPK ejbCreate(id, name){ ...
    con = this.getConnnection();
    s = con.prepareStatement("insert into Customer (id, name) values (?,?)");
    // set values, execute statement, return primary key ...
}
public void ejbLoad(){...
    con = this.getConnnection();
    s = con.prepareStatement("select id, name from Customer where id=? ");
    ps.setInt(1,pk.id)
    // execute query, if successful set variables ...
}
public void ejbStore(){ ...
    con = this.getConnnection();
    s = con.prepareStatement("update Customer set id = ?, name=? where
    id=? ");
    // set values and execute statement...
}
```

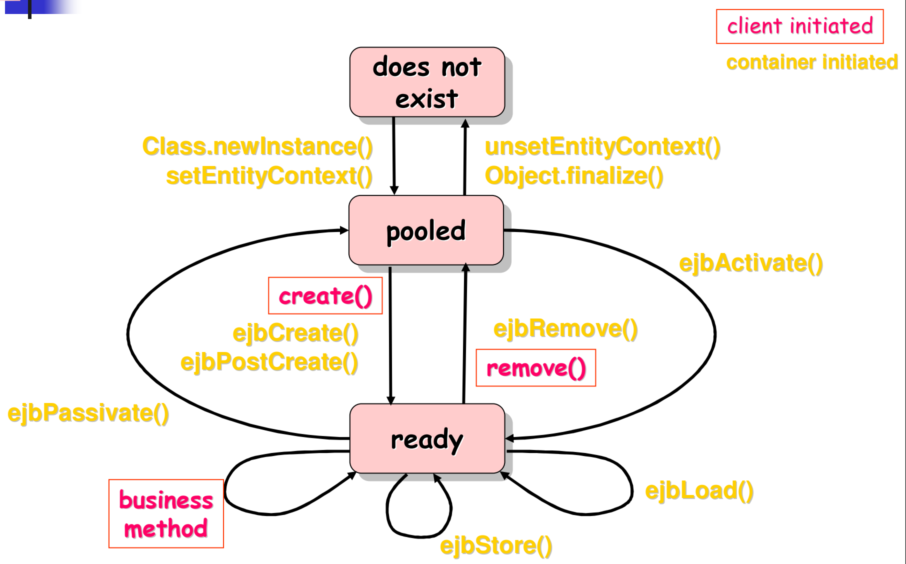
67

Entity Beans – BMP / Seq. of Events

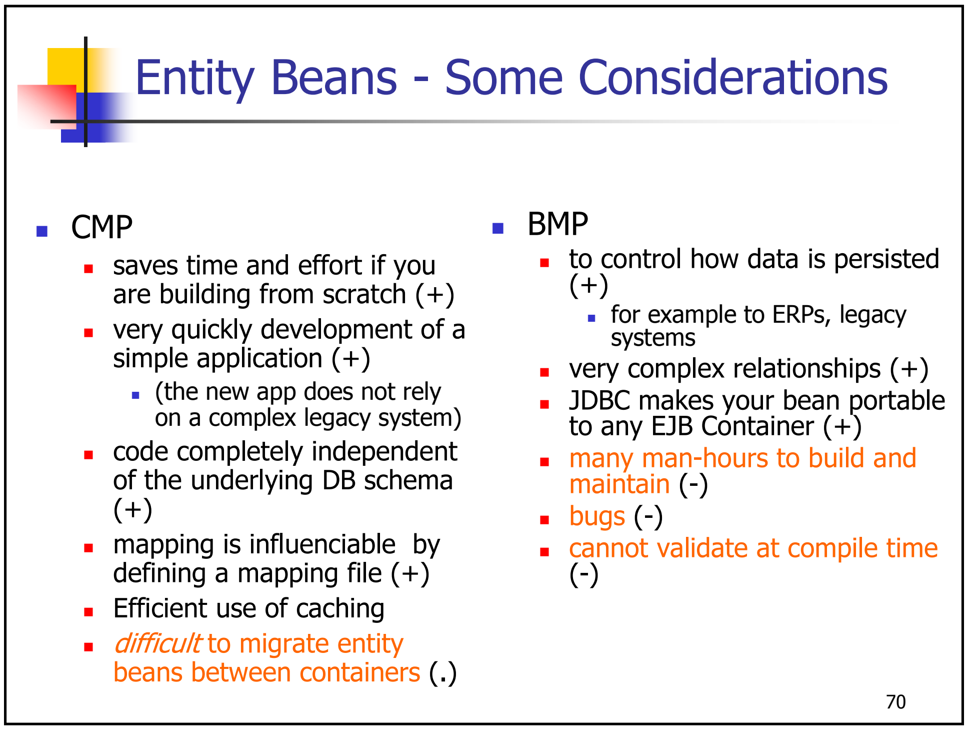


68

Entity Beans - Life Cycle



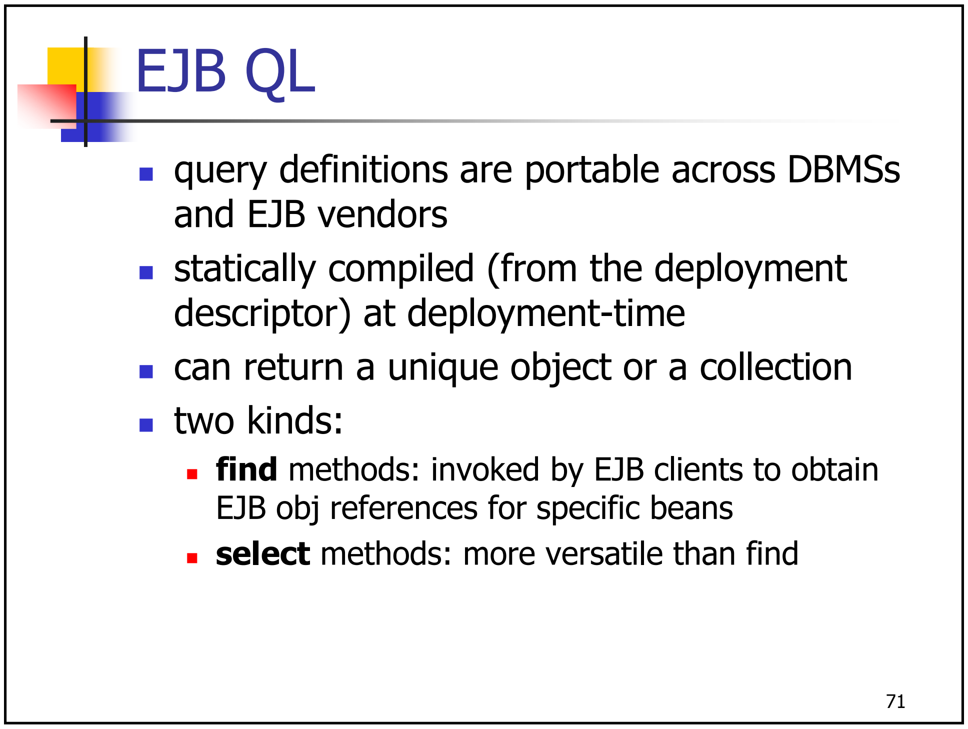
69



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 (+)
  - Efficient use of caching
  - *difficult* to migrate entity beans 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 (-)

70



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

71

EJB QL – Some Restrictions

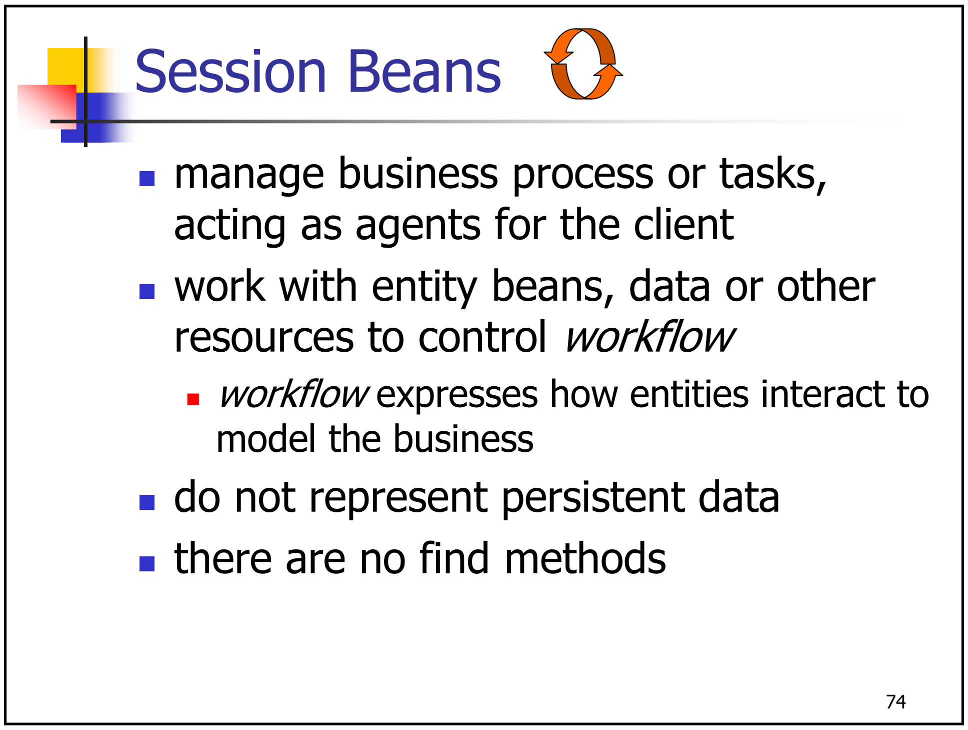
- select statement is restricted to a single object or attribute
  - no support for multiple columns involving different tables
- nested queries
- dynamic queries
- lack of support for Date type

72

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

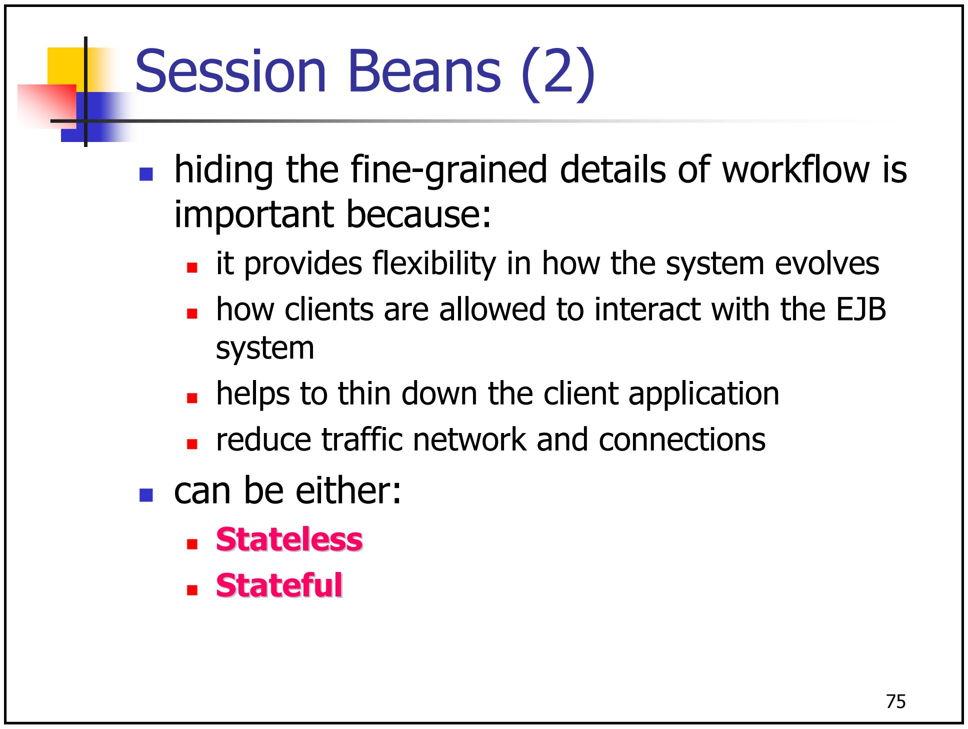
73



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

74



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**

75

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

76

Stateless Session Beans - Example

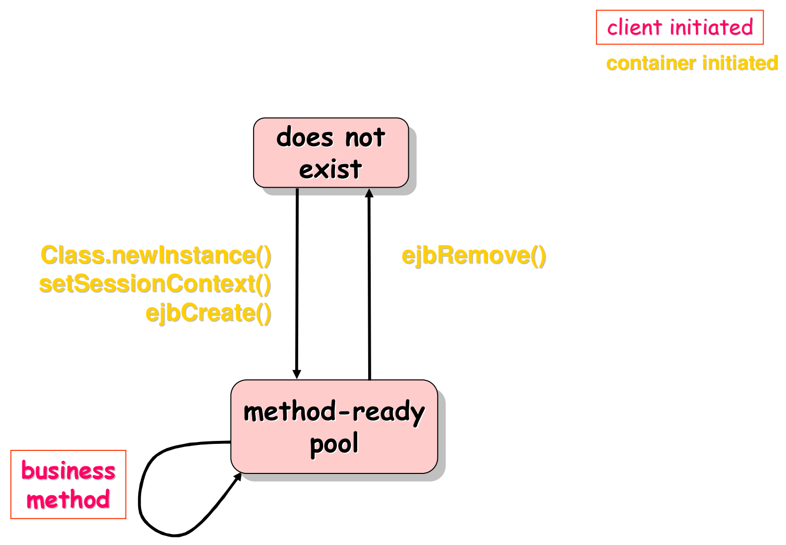
- Payment Bean

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

workflow

77

Stateless Session Beans - Life Cycle



78

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

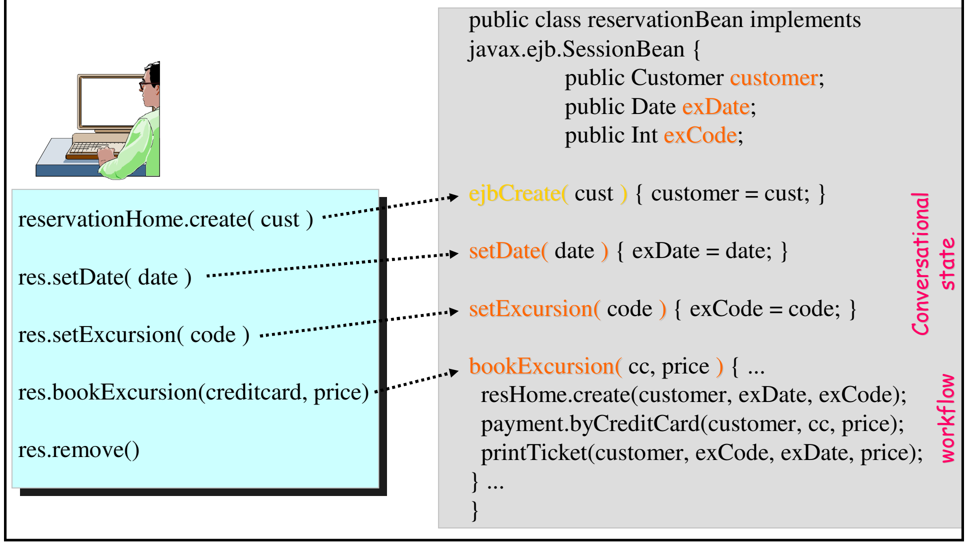
79

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

80

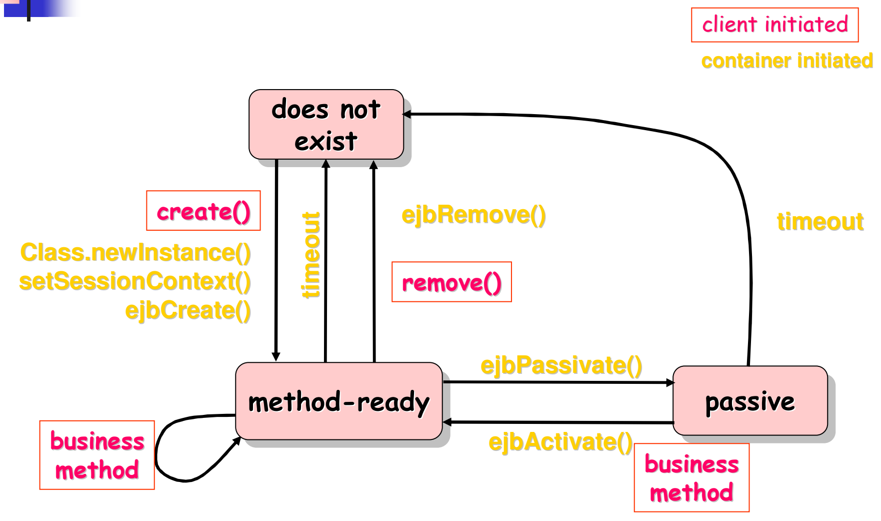
Stateful Session Beans - Example



```
reservationHome.create( cust )
res.setDate( date )
res.setExcursion( code )
res.bookExcursion(creditcard, price)
res.remove()
```

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

ejbCreate( cust ) { customer = cust; }

setDate( date ) { exDate = date; }

setExcursion( 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



82

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

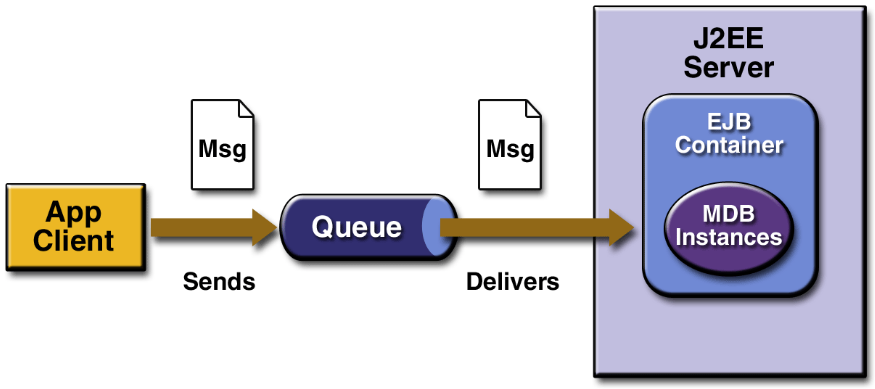
83

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, …)

84

Message-Driven Beans

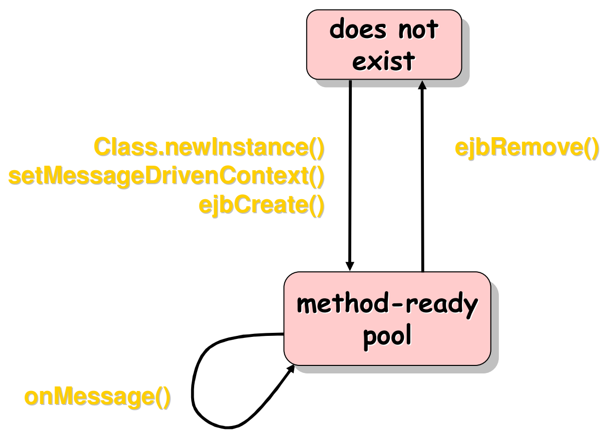


85

MDBs – Life Cycle

client initiated

container initiated



86

Transactional 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

87

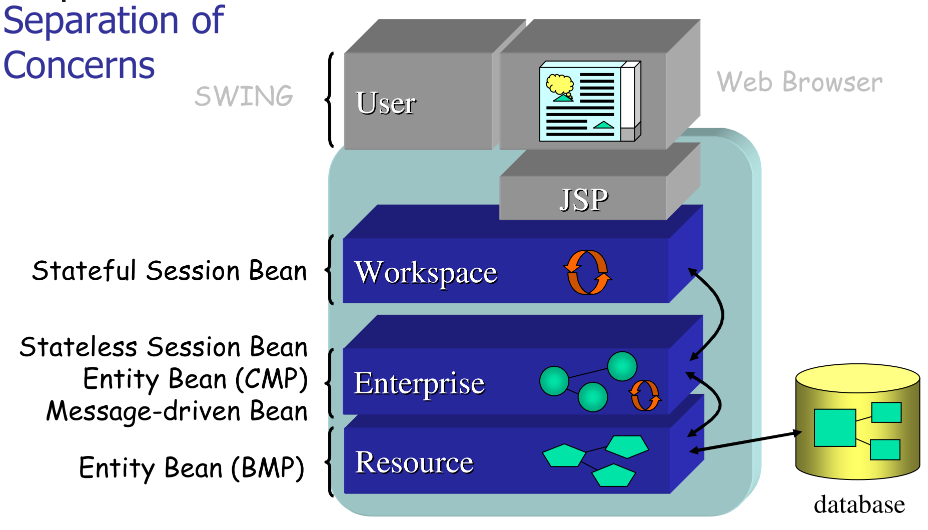
Roadmap

- 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

88

Four-tier Architecture

Separation of Concerns



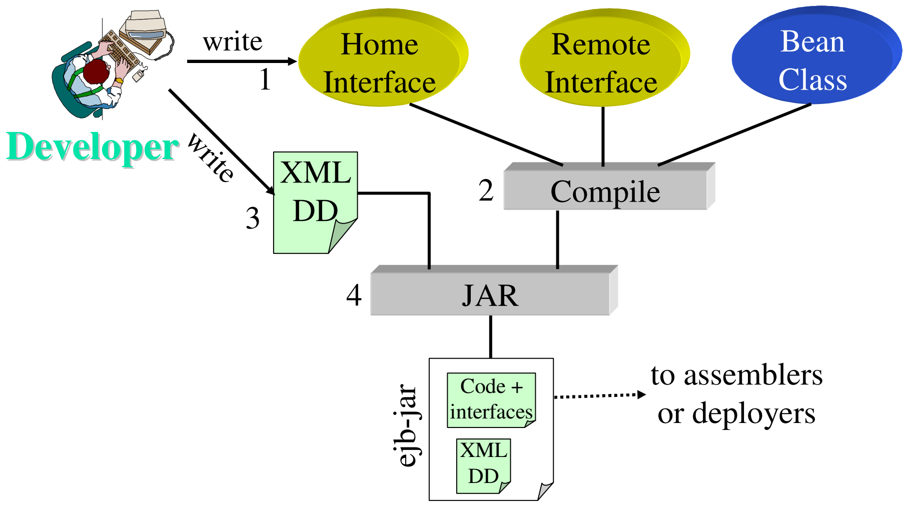
89

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

90

The EJB Development Process



91

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

92

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

93

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

94

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

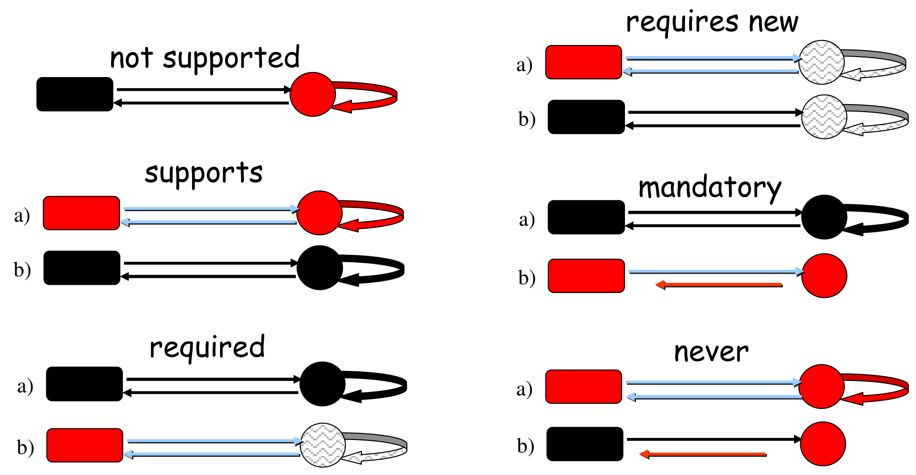
95

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

96

Primary Services - EJB Transaction Attributes



■ client's TX context 🟥 Non-TX context ▨ bean's TX context → exception

97

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

98

Primary Services – Distrib. Objs.

- 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

99

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

100

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

101

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

102

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)

103

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

104

DD - Deployment Example

```
<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>
     <prim-key-class>com.example.customer.CustomerPK</prim-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>
…
```

105

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)

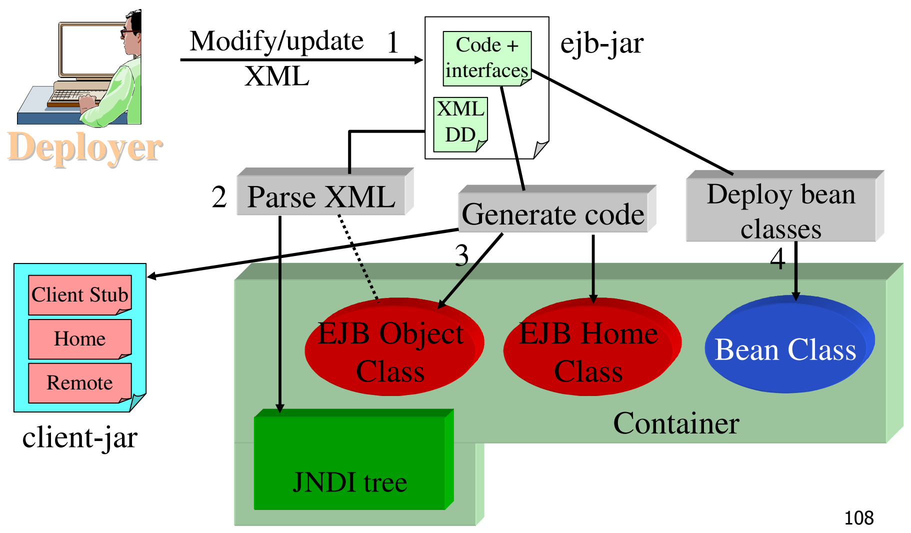
106

DD - Assembly Example

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

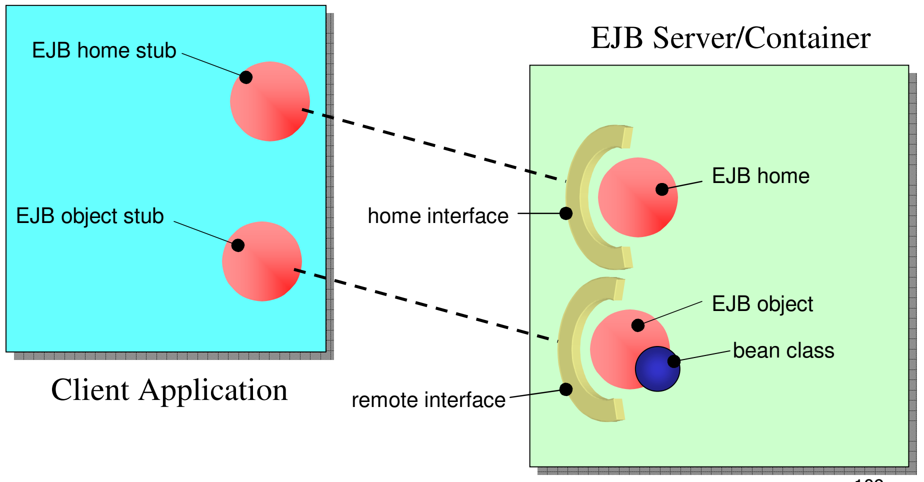
107

The EJB Deployment Process



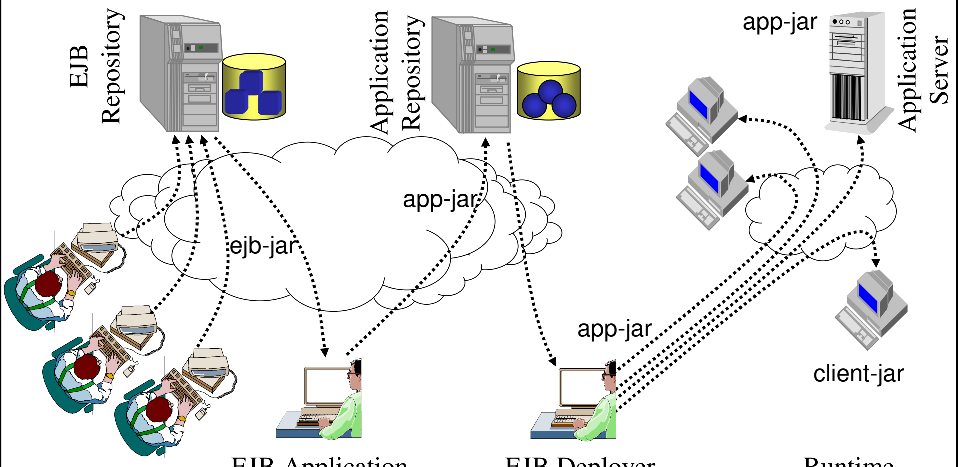
108

Remember! EJBs - Classes and Interfaces



109

EJB Lifecycle - Roles



EJB Developers

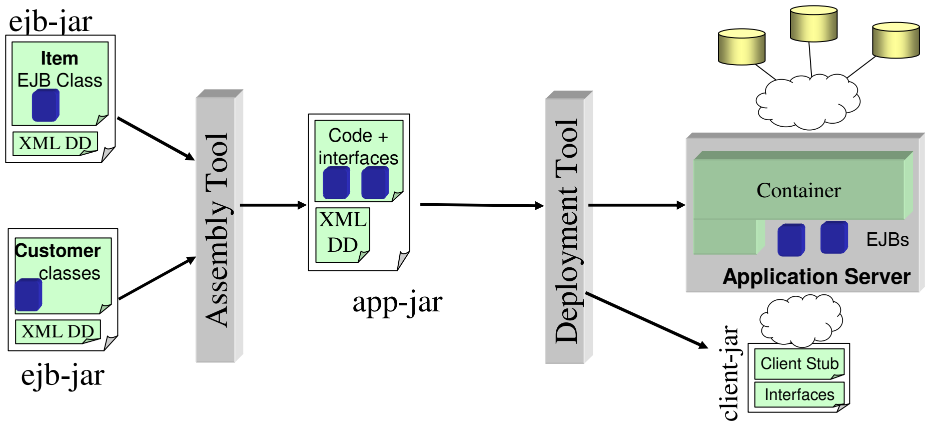
EJB Application Assembler

EJB Deployer (System Manager)

Runtime Environment

110

The EJB Assembly Line



EJB Developers

EJB Application Assembler

EJB Deployer (System Manager)

Runtime Environment

111

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, ...

112

Clients - Example

```
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);
```

113

EJB spec evolution → v3.0

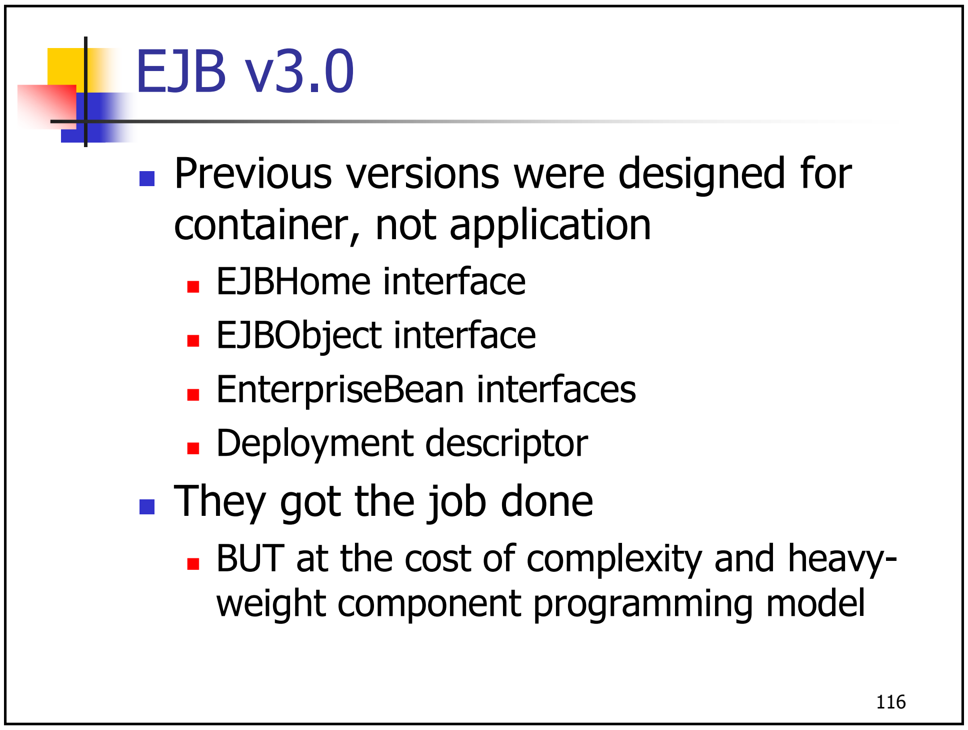
[JavaOne'06]

114

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

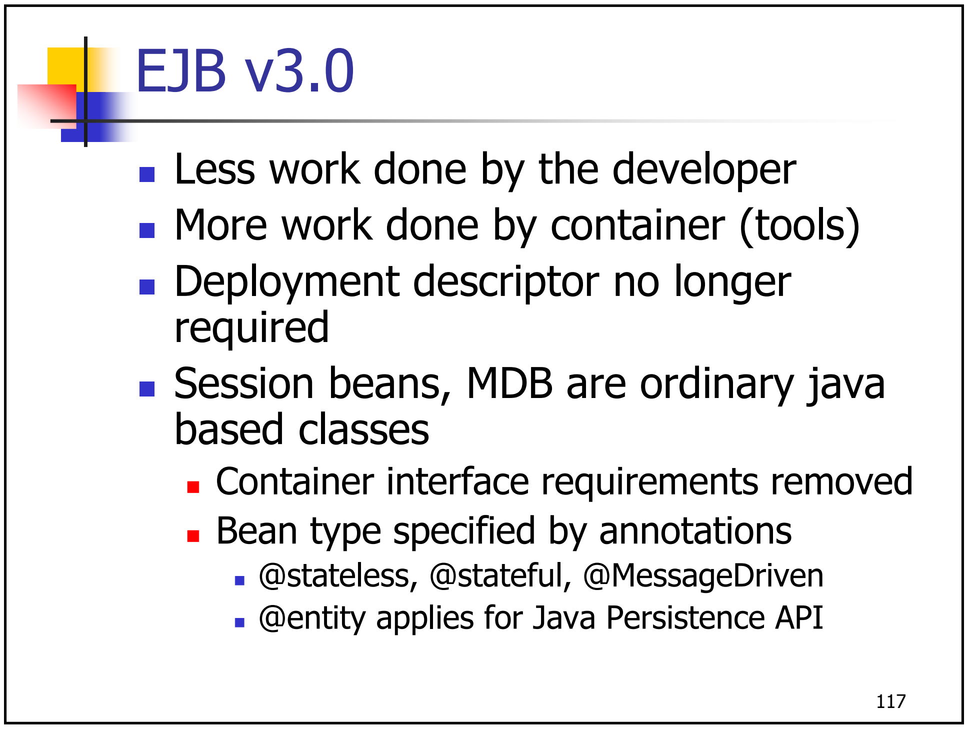
115



EJB v3.0

- Previous versions were designed for container, not application
  - EJBHome interface
  - EJBObject interface
  - EnterpriseBean interfaces
  - Deployment descriptor
- They got the job done
  - BUT at the cost of complexity and heavy-weight component programming model

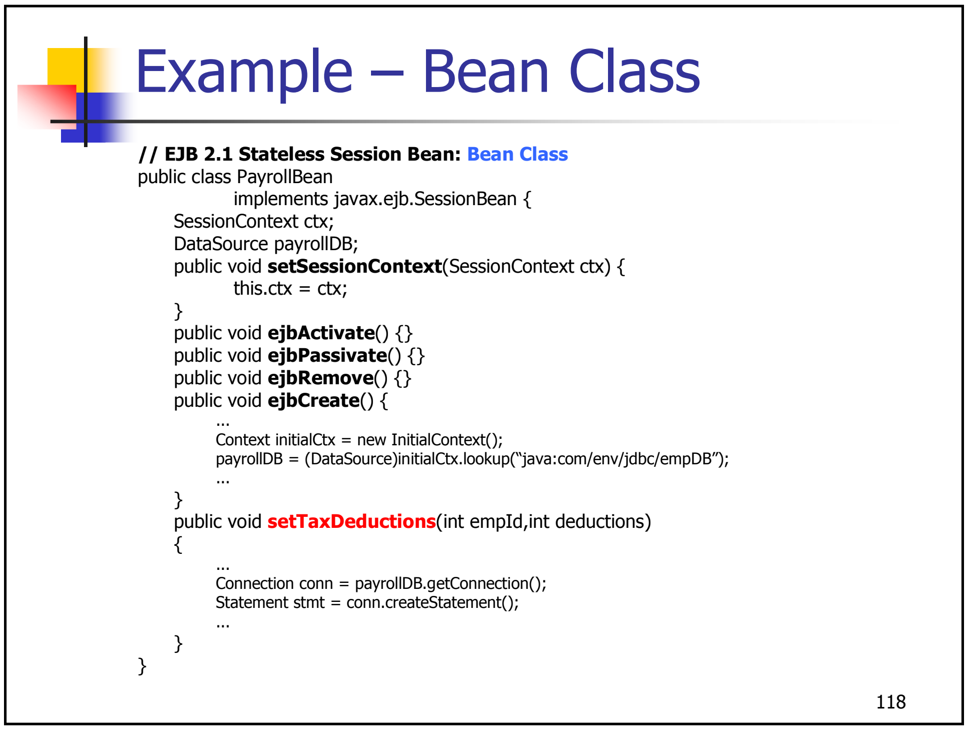
116



EJB v3.0

- Less work done by the developer
- More work done by container (tools)
- Deployment descriptor no longer required
- Session beans, MDB are ordinary java based classes
  - Container interface requirements removed
  - Bean type specified by annotations
    - @stateless, @stateful, @MessageDriven
    - @entity applies for Java Persistence API

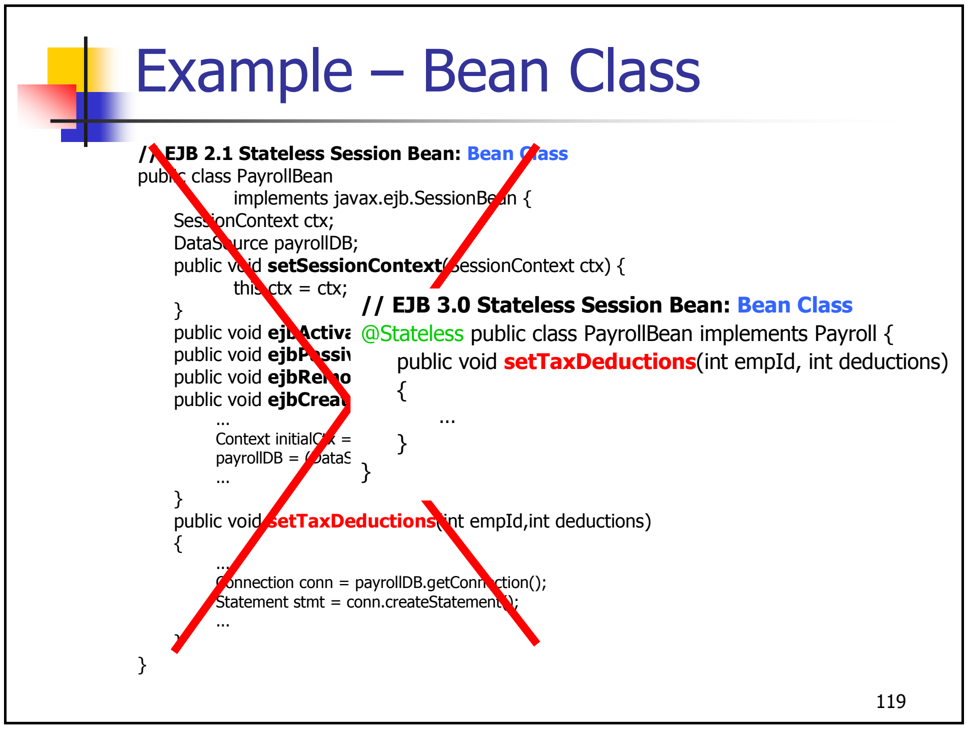
117



Example – Bean Class

```
// EJB 2.1 Stateless Session Bean: Bean Class
public class PayrollBean
            implements javax.ejb.SessionBean {
    SessionContext ctx;
    DataSource payrollDB;
    public void setSessionContext(SessionContext ctx) {
            this.ctx = ctx;
    }
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
    public void ejbCreate() {
        ...
        Context initialCtx = new InitialContext();
        payrollDB = (DataSource)initialCtx.lookup("java:com/env/jdbc/empDB");
        ...
    }
    public void setTaxDeductions(int empId,int deductions)
    {
        ...
        Connection conn = payrollDB.getConnection();
        Statement stmt = conn.createStatement();
        ...
    }
}
```

118

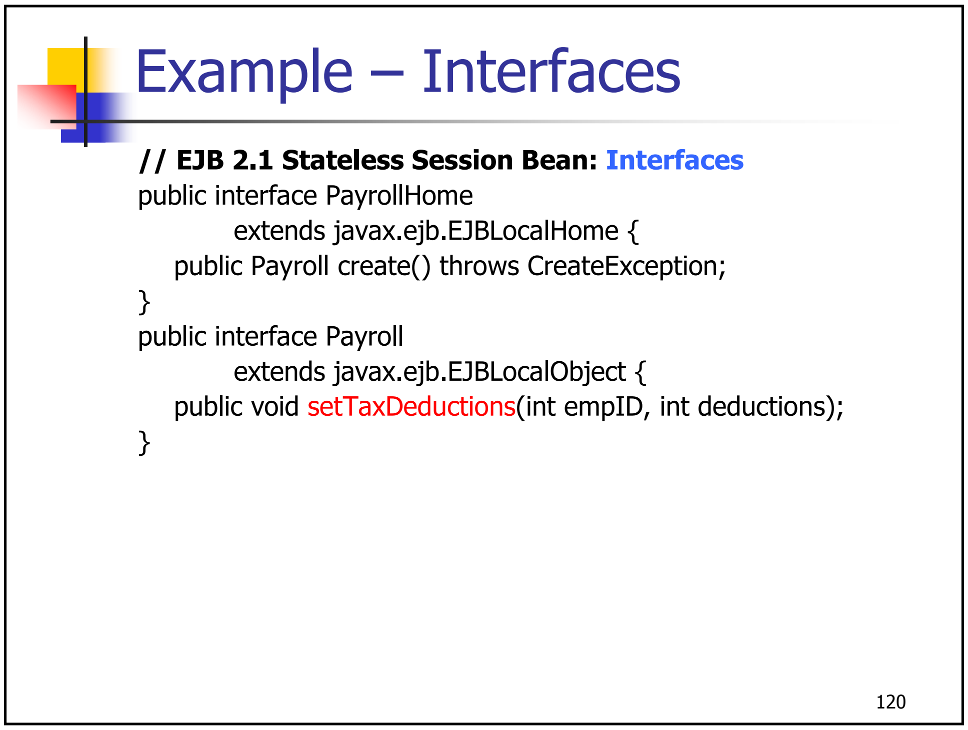


Example – Bean Class

```
// EJB 2.1 Stateless Session Bean: Bean Class
public class PayrollBean
            implements javax.ejb.SessionBean {
    SessionContext ctx;
    DataSource payrollDB;
    public void setSessionContext(SessionContext ctx) {
            this.ctx = ctx;
    }
    public void ejbActiva
    public void ejbPassiv
    public void ejbRemo
    public void ejbCrea
        ...
        Context initialCtx =
        payrollDB = (DataS
        ...
    }
    public void setTaxDeductions(int empId,int deductions)
    {
        ...
        Connection conn = payrollDB.getConnection();
        Statement stmt = conn.createStatement();
        ...
    }
}
```

```
// EJB 3.0 Stateless Session Bean: Bean Class
@Stateless public class PayrollBean implements Payroll {
    public void setTaxDeductions(int empId, int deductions)
    {
        ...
    }
}
```

119

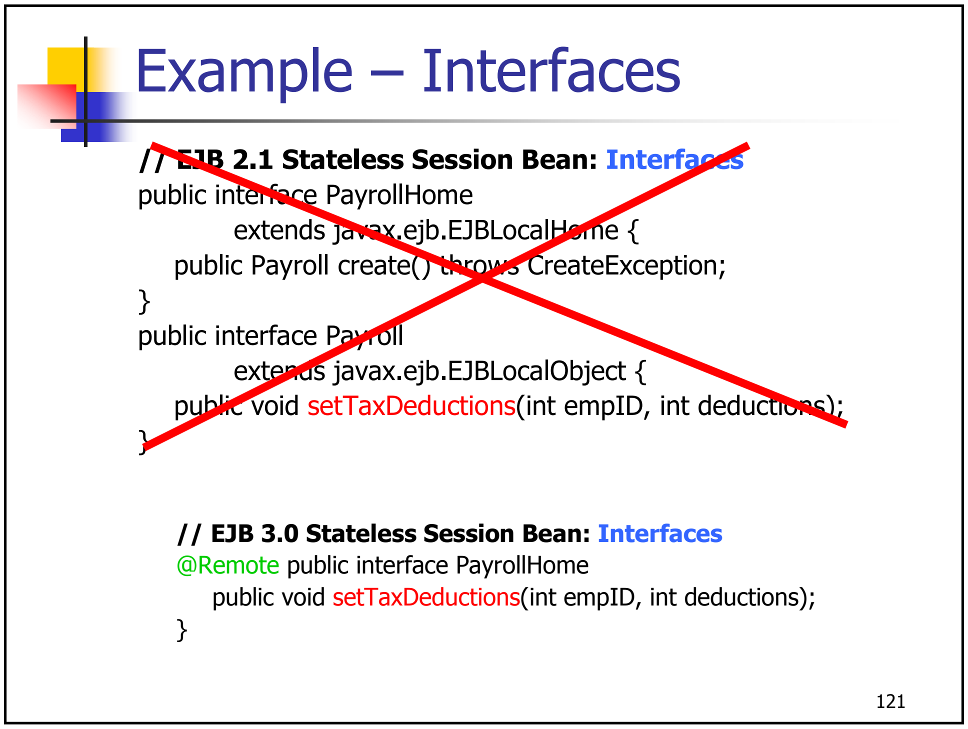


Example – Interfaces

// EJB 2.1 Stateless Session Bean: Interfaces

```
public interface PayrollHome
            extends javax.ejb.EJBLocalHome {
    public Payroll create() throws CreateException;
}
public interface Payroll
            extends javax.ejb.EJBLocalObject {
    public void setTaxDeductions(int empID, int deductions);
}
```

120



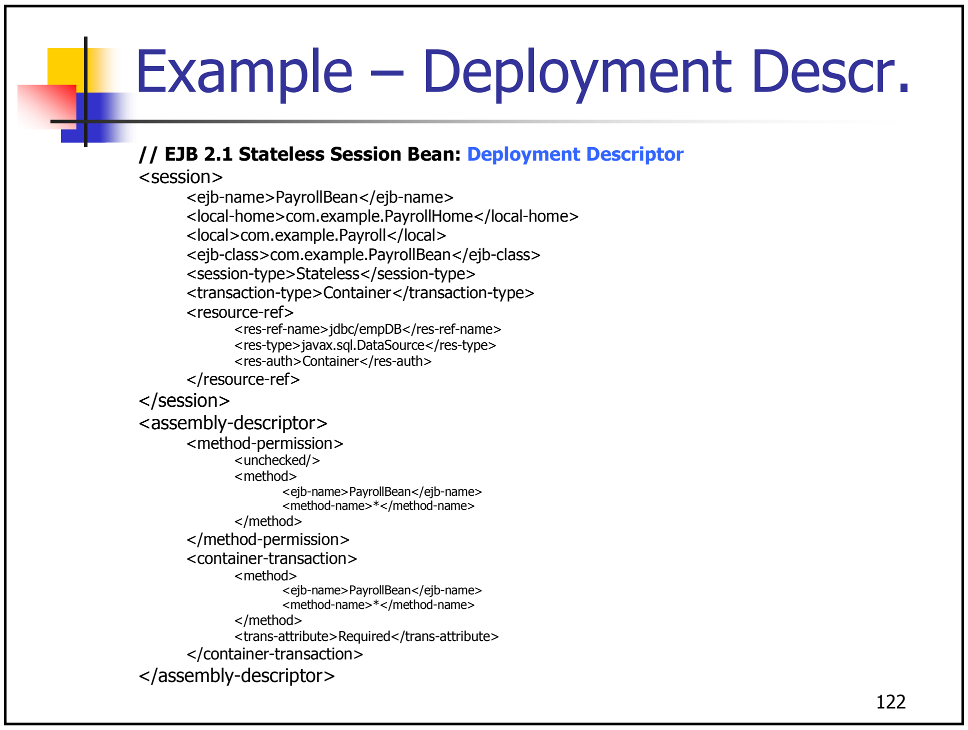
Example – Interfaces

~~**// EJB 2.1 Stateless Session Bean: Interfaces**~~
~~public interface PayrollHome~~
~~extends javax.ejb.EJBLocalHome {~~
~~public Payroll create() throws CreateException;~~
~~}~~
~~public interface Payroll~~
~~extends javax.ejb.EJBLocalObject {~~
~~public void setTaxDeductions(int empID, int deductions);~~
~~}~~

// EJB 3.0 Stateless Session Bean: Interfaces

```
@Remote public interface PayrollHome
    public void setTaxDeductions(int empID, int deductions);
}
```

121

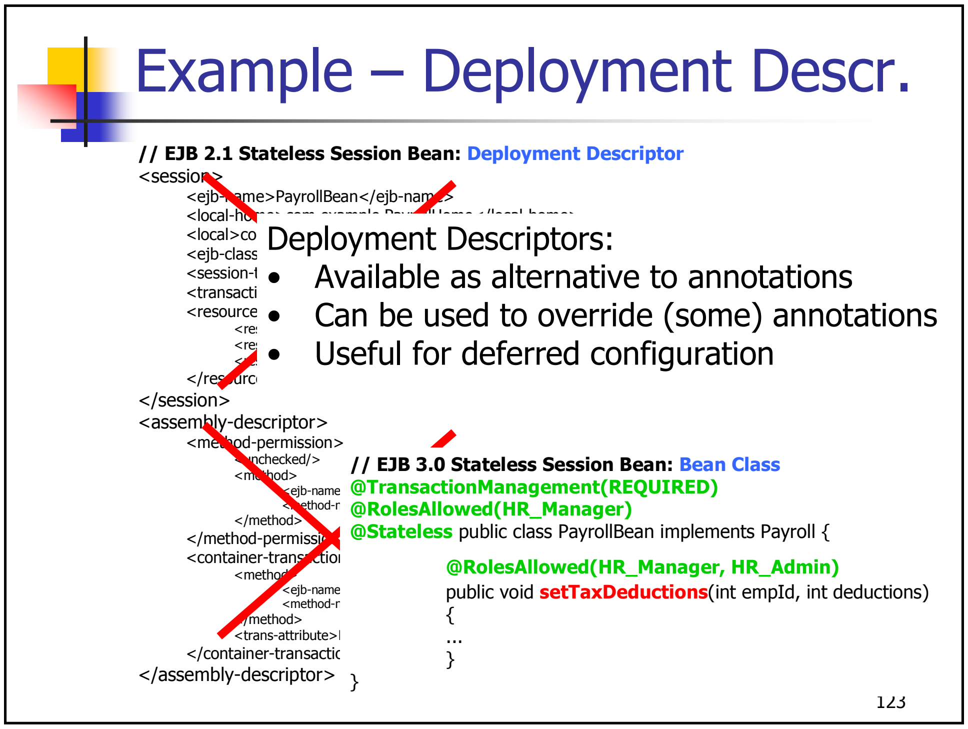


Example – Deployment Descr.

// EJB 2.1 Stateless Session Bean: Deployment Descriptor

```
<session>
        <ejb-name>PayrollBean</ejb-name>
        <local-home>com.example.PayrollHome</local-home>
        <local>com.example.Payroll</local>
        <ejb-class>com.example.PayrollBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
        <resource-ref>
                <res-ref-name>jdbc/empDB</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>
</session>
<assembly-descriptor>
        <method-permission>
                <unchecked/>
                <method>
                        <ejb-name>PayrollBean</ejb-name>
                        <method-name>*</method-name>
                </method>
        </method-permission>
        <container-transaction>
                <method>
                        <ejb-name>PayrollBean</ejb-name>
                        <method-name>*</method-name>
                </method>
                <trans-attribute>Required</trans-attribute>
        </container-transaction>
</assembly-descriptor>
```

122



Example – Deployment Descr.

// EJB 2.1 Stateless Session Bean: Deployment Descriptor

```
<session>
        <ejb-name>PayrollBean</ejb-name>
        <local-h
        <local>co
        <ejb-class
        <session-t
        <transacti
        <resource
                <re
                <re
                <
        </resourc
</session>
<assembly-descriptor>
        <method-permission>
                <unchecked/>
                <method>
                        <ejb-name
                        <method-r
                </method>
        </method-permissi
        <container-transactio
                <metho
                        <ejb-name
                        <method-r
                </method>
                <trans-attribute>
        </container-transactic
</assembly-descriptor>
```

Deployment Descriptors:

- Available as alternative to annotations
- Can be used to override (some) annotations
- Useful for deferred configuration

// EJB 3.0 Stateless Session Bean: Bean Class

```
@TransactionManagement(REQUIRED)
@RolesAllowed(HR_Manager)
@Stateless public class PayrollBean implements Payroll {

        @RolesAllowed(HR_Manager, HR_Admin)
        public void setTaxDeductions(int empId, int deductions)
        {
        ...
        }
}
```

123

(System) Exceptions

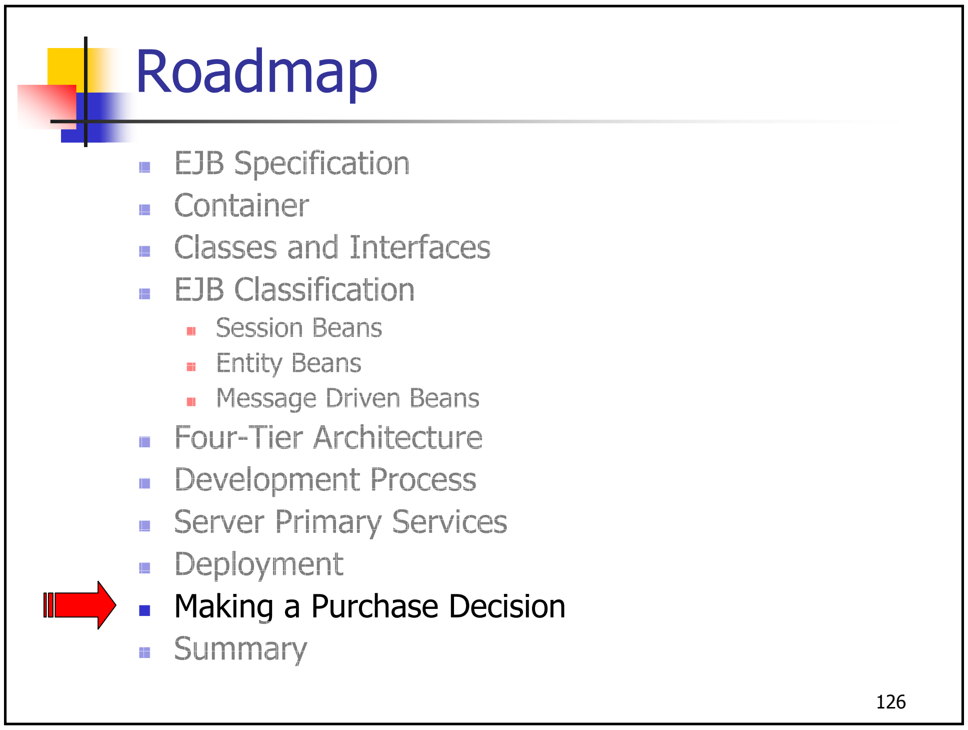
- In EJB 2.1 specification
  - Remote system exceptions subtypes of java.rmi.RemoteException
  - Local system exceptions subtypes of EJBException
- In EJB 3.0 simply extend EJBException
  - Independent whether is local or remote
  - Business logic exceptions
    - @ApplicationException()

124

EJB 3.0 - Summary

- Major simplification of EJB technology
  - Beans are plain java-based classes
  - APIs (re)focussed on ease-of-use for developers
  - Easy access to container services and environment
  - Deployment descriptors available, but generally unneeded
  - All-in-one (code + annotations)
    - Is this good for reusability?
- EJB 3.0 based components interoperate with existing components/apps
- Easy-to-use of powerful functionality

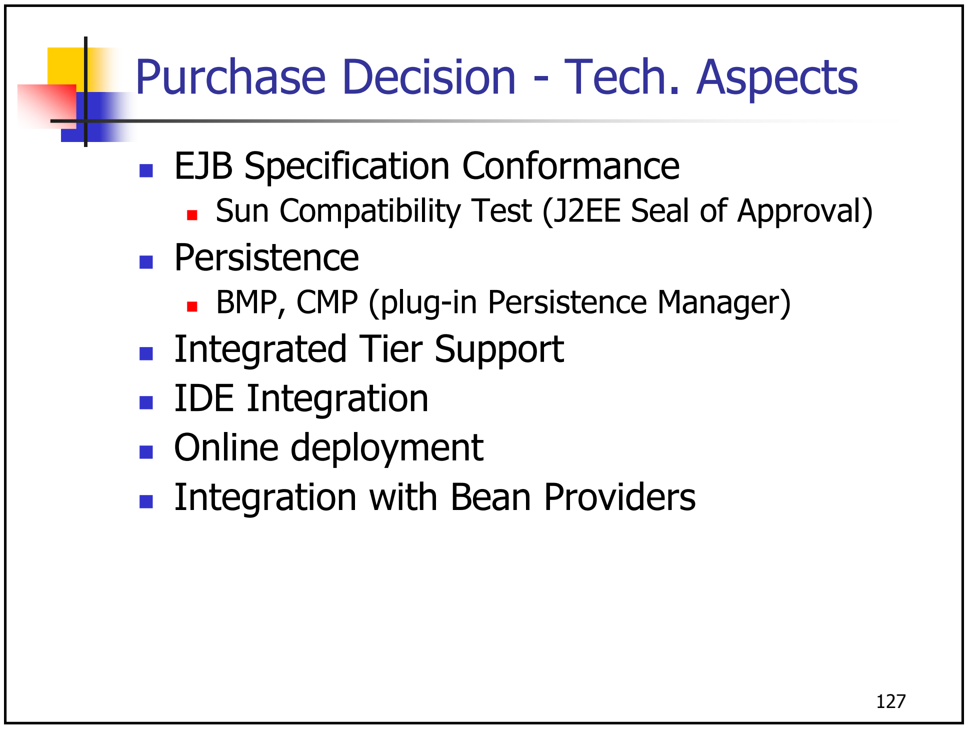
125



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

126



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

127

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

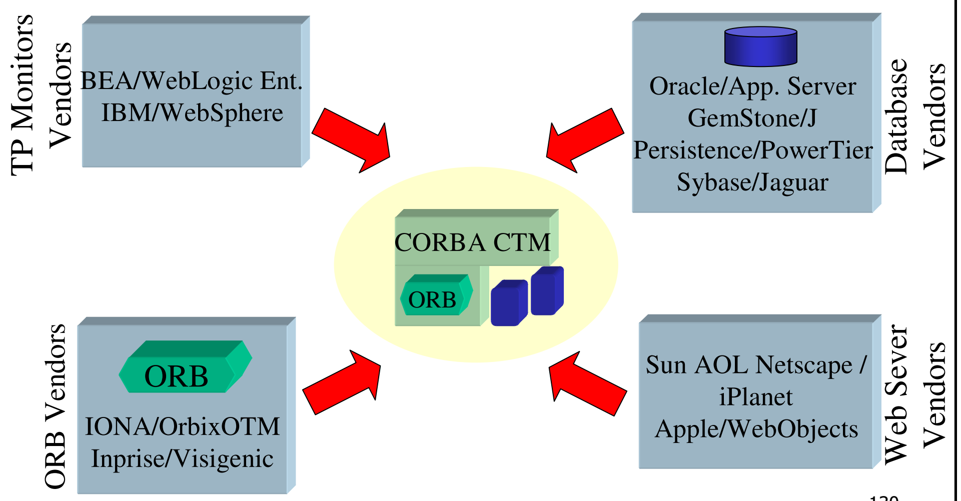
128

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/ITS)
  - 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 ...

129

App Server Vendors



130

Making a Purchase Decision - Prods

- Borland:
  - JBuilder/Together – BAS
- IBM:
  - Java Visual Age/Rational/Eclipse - WebSphere
- BEA:
  - Visual Café - Web Logic Server
- SUN/Netscape:
  - NetBeans - iPlanet App Server
- Oracle:
  - JDeveloper (+Oracle 10g) - Oracle App. Server
- ...

131

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

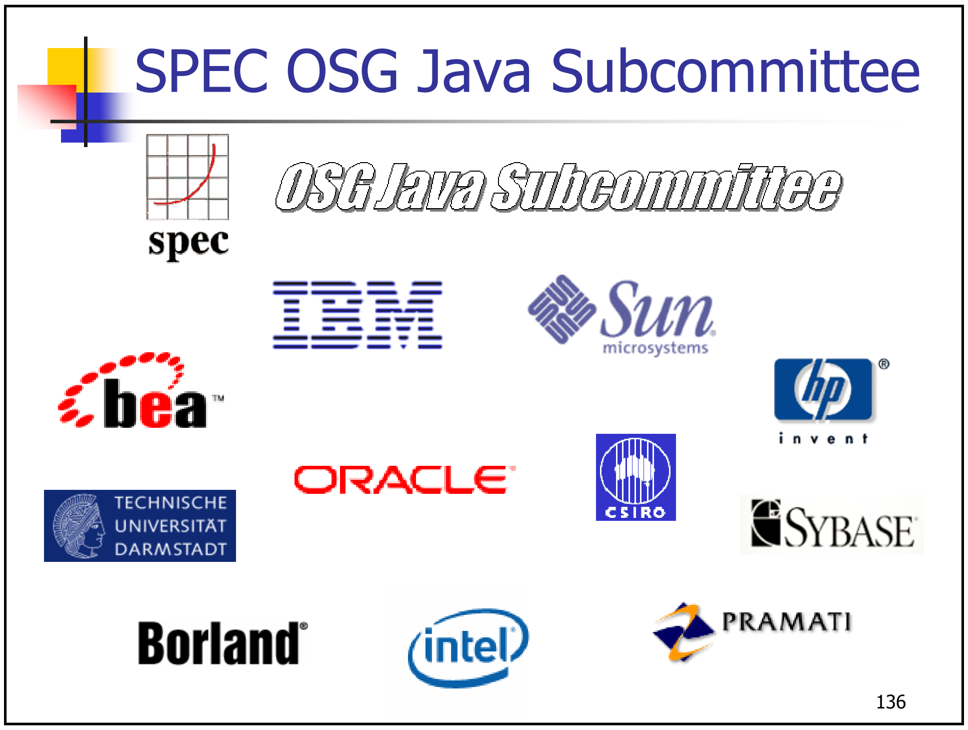
J2EE App Server Benchmarks

- *"The J2EE application server software stack has solidified, with products becoming much more similar and migration of code between platforms becoming steadily easier."* (META Group'04)
- **Once functionality is standardized, performance becomes the distinguishing factor!**
- Needed are industry standard benchmarks to measure the performance and scalability of J2EE platforms

SPECjAppServer 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/

SPECjAppServer



SPEC OSG Java Subcommittee

spec

OSG Java Subcommittee

IBM

Sun microsystems

bea™

hp® invent

TECHNISCHE UNIVERSITÄT DARMSTADT

ORACLE®

CSIRO

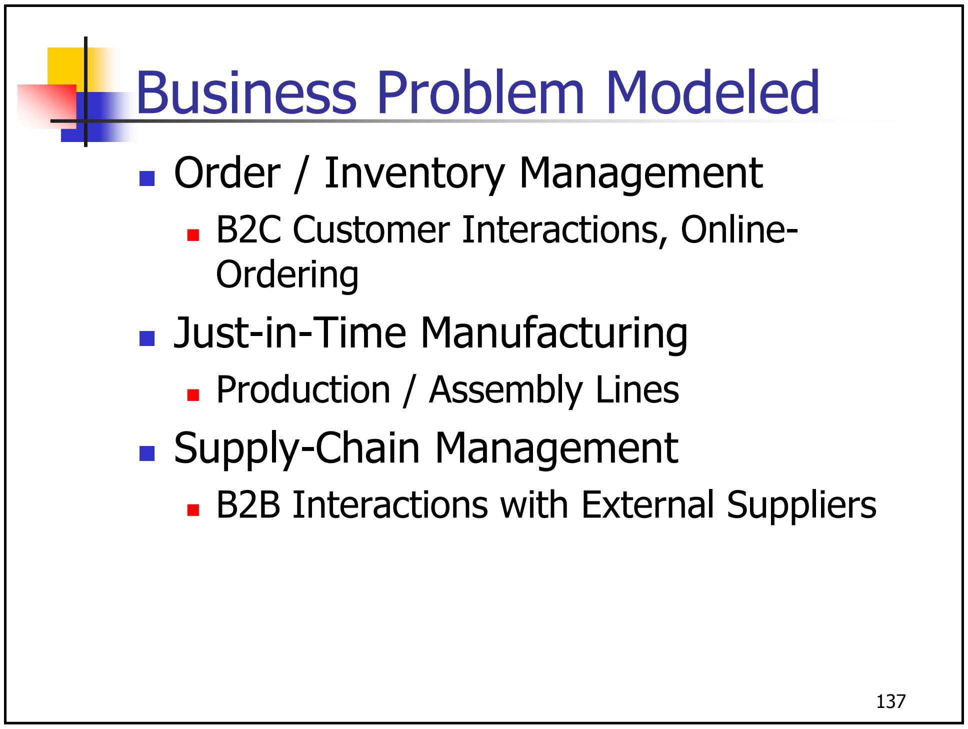
SYBASE®

Borland®

intel®

PRAMATI

136

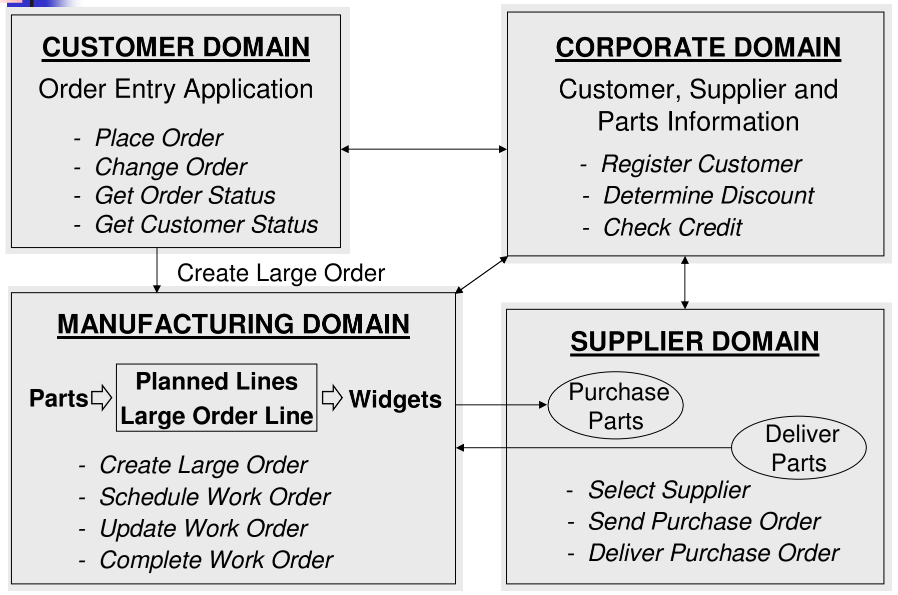


Business Problem Modeled

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

137

Business Domains

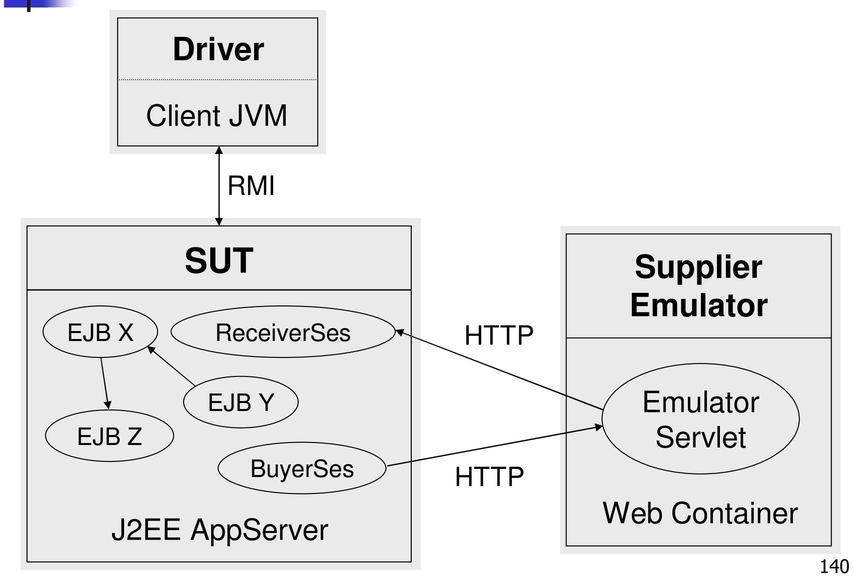


Application Design

- 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

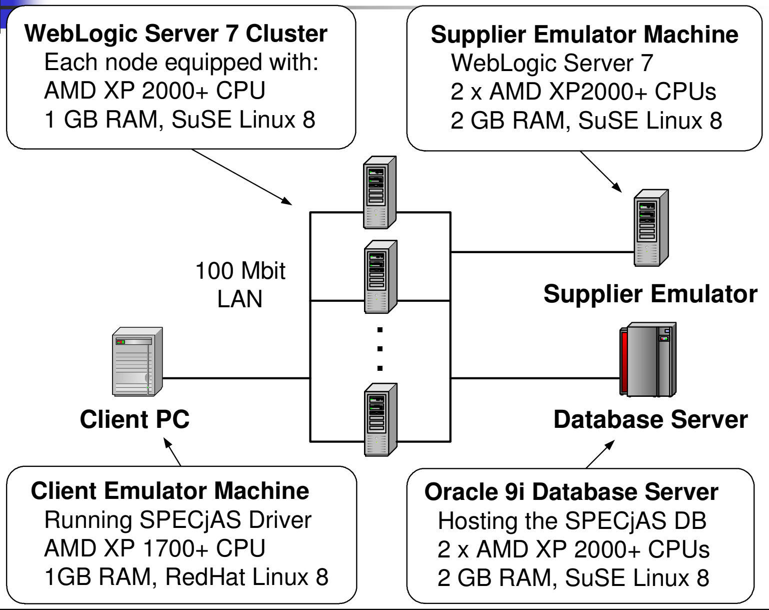
139

Components



140

Sample of IT Infrastructure


WebLogic Server 7 Cluster
Each node equipped with:
AMD XP 2000+ CPU
1 GB RAM, SuSE Linux 8

Supplier Emulator Machine
WebLogic Server 7
2 x AMD XP2000+ CPUs
2 GB RAM, SuSE Linux 8

Client Emulator Machine
Running SPECjAS Driver
AMD XP 1700+ CPU
1GB RAM, RedHat Linux 8

Oracle 9i Database Server
Hosting the SPECjAS DB
2 x AMD XP 2000+ CPUs
2 GB RAM, SuSE Linux 8

141

Some Results – Dual-node

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

142

Some Results – Multi-node

- IBM = 2,575.34 total operations per second (TOPS), price/performance $330.07/TOPS
- Configuration:
  - Application Server: WebSphere 5.1 Application Server,
    - 9 servers, with 2 Intel Xeon at 3200 Mhz, 3GB of main memory
  - Hardware: eServer xSeries 335 Cluster
  - JVM: IBM's 1.4.1 JRE
  - JDBC Driver: IBM DB2 JDBC Universal Driver Provider
  - OS: SuSE Linux Enterprise Server 8 SP2A with ReiserFS
  - Database: DB2 Universal Database v8.1.1.32 FP4, Enterprise Server Edition running on AIX
    - 1 DB Server, with 8 CPUs and 16GB of main memory
- Total System Cost: US$ 850,025

143

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:
- http://www.spec.org/jAppServer2004/

144

J2EE Patterns

From JavaOne Tutorials
and Books

145

What are patterns?

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

146

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 Facade
- Application Service
- Business Object
- Composite Entity
- Transfer Object
- Transfer Object Assembler
- Value List Handler

Integration Tier
- Data Access Object
- Service Activator
- Domain Store
- Web Service Broker

147

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

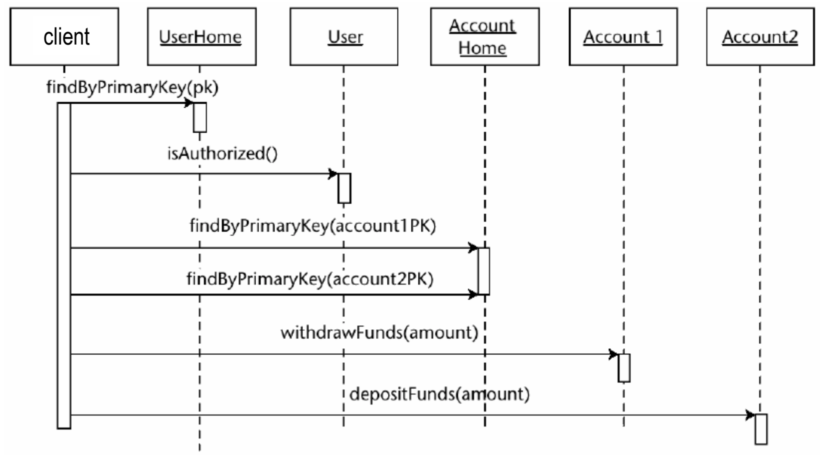
148

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

149

Situation I - Motivation



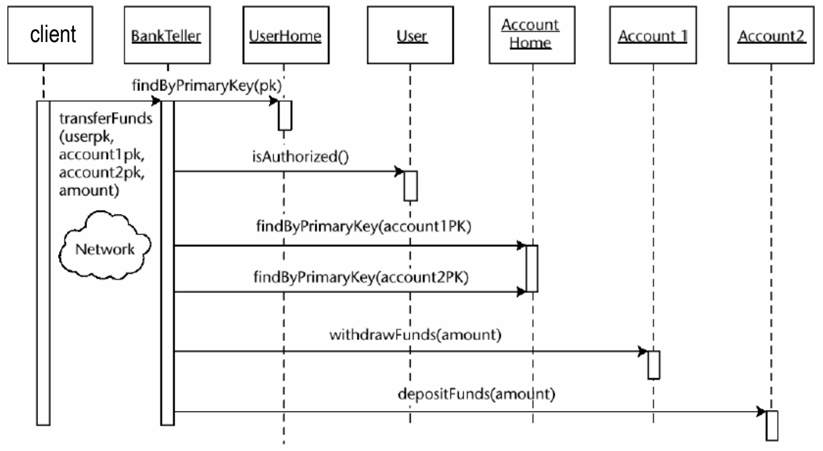
150

Situation I – Motivation (cont.)

- 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?

151

Solution: Session Façade



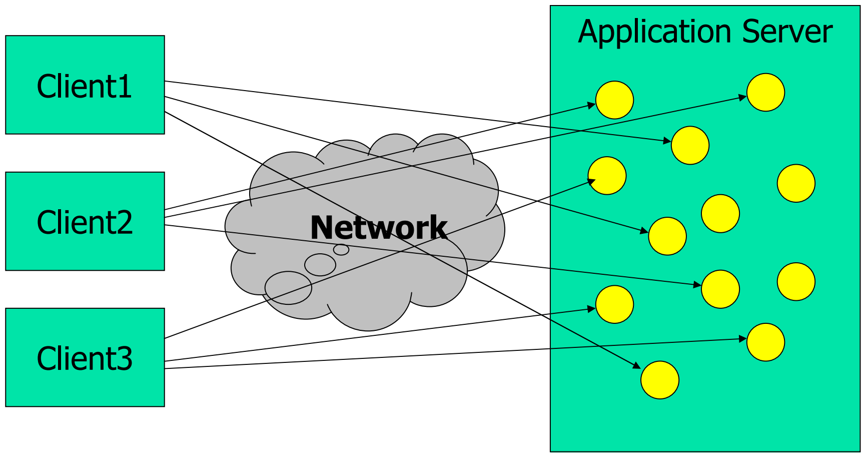
152

Session Façade Benefits

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

153

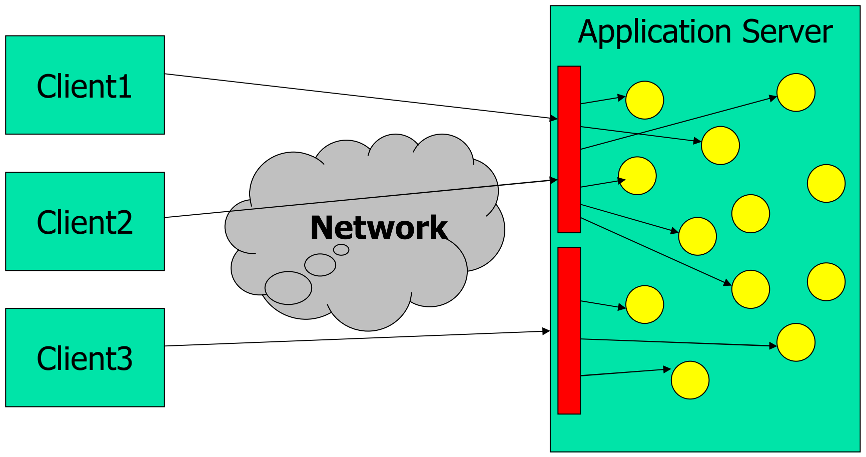
Before Session Façade



Direct Entity Bean Access

154

With Session Façade



155

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?

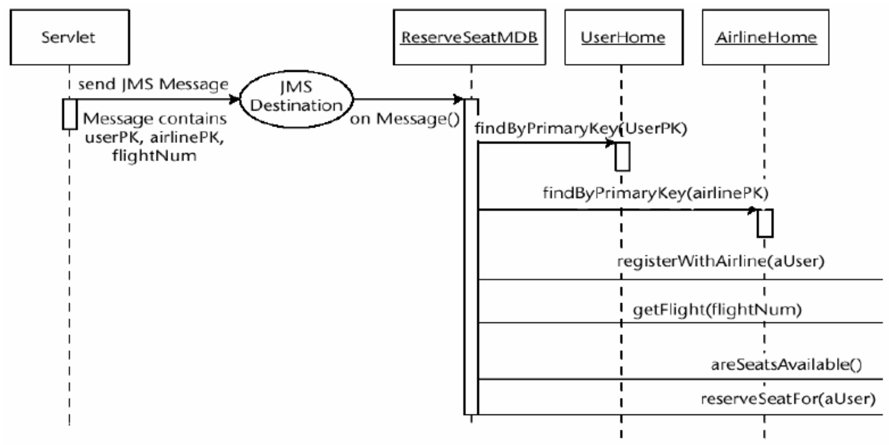
156

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

157

Solution: Message Façade



158

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

159

Summary and Conclusions

160

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

161

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

162

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)

163

Compatibility (cont.)

- 4 Versions released in 4 years (1.0,1.1,2.0, 2.1)
  - 3.0 since June 2006
- 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)
  - ease-of-use for the developer (EJB 3.0)

164

Summary and Outlook

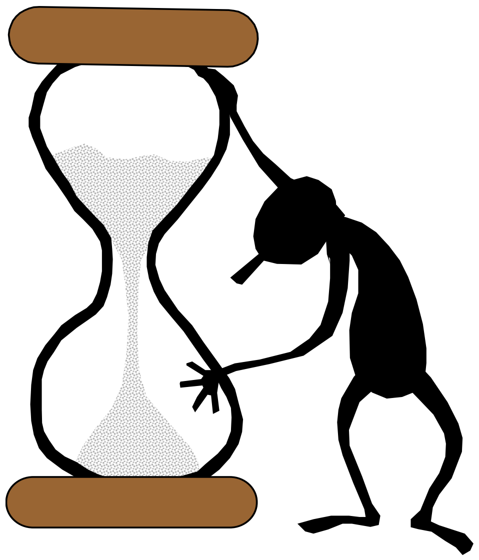
- J2EE has become widely successful
- M$ attempting to fight with .NET
- Revival of object to relational DB mapping
- XML parsing overhead is high
- Standardized benchmark
  - SPECjAppServer
- Web services becoming popular, even for intranet usage (being extended with TX, business process,…)

165

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

166



168