# J2EE, now called JEE: introduction

Luca Cavallaro
cavallaro@elet.polimi.it

# Focus of this lecture

- Objectives
- The approach
  - Multi-tiers architectures and JEE
  - Components and containers in JEE
  - Resource management and primary services for EJB 3.0

# JEE objectives (1)

- To define an architectural model to build enterprise applications that are **distributed**, **component-based**, and **transaction-oriented**
- To offer a wide set of APIs to
  - Reduce development time
  - Reduce application complexity
  - Improve performance
  - Allow the application to  access to various data sources
  - Offer the application functionality to various kinds of clients

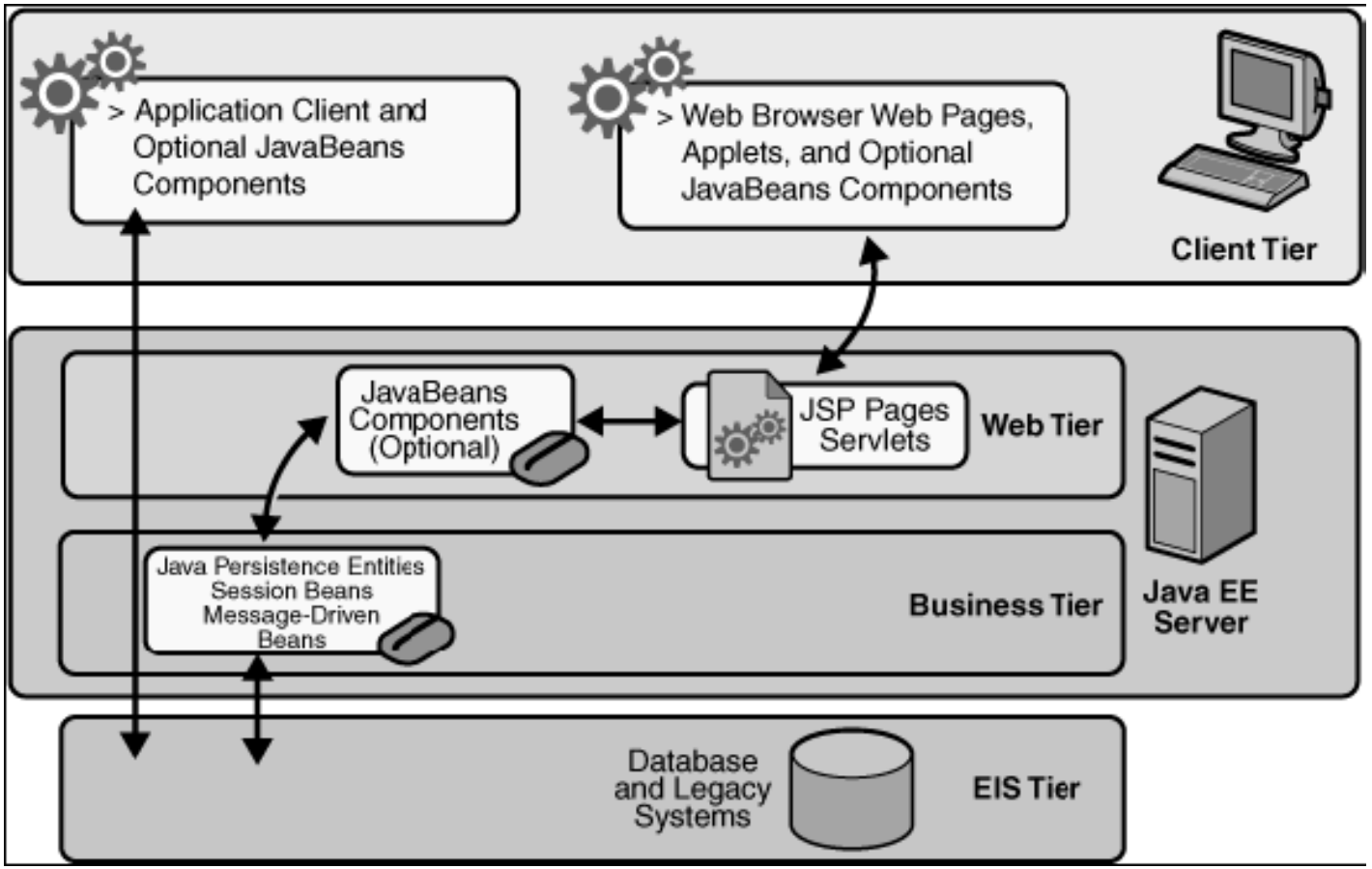
# JEE objectives (2)

- Allow the developer to abstract from low level problems such as
  - Transaction managament
  - State management
  - Multi-threading
  - Connection pool management
  - …
- Follow the philosophy Write Once, Run Anywhere
  - Define a contract that makes it possible to use platforms from various vendors
- Provide mechanisms to support interoperability with non Java systems
- Compatible with CORBA protocols

# The approach

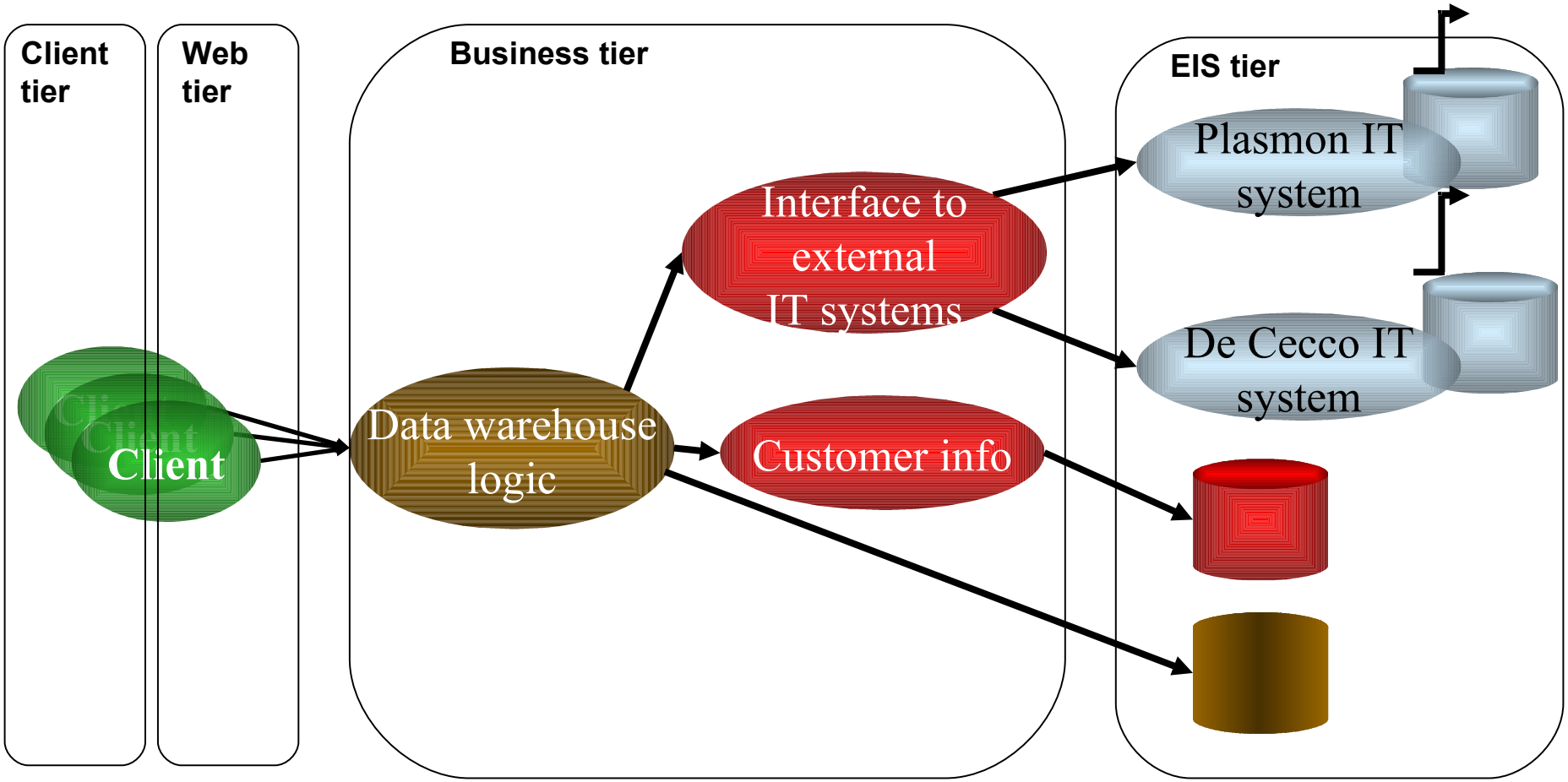
- A **three tier architectural model**
  - The business logic is encapsulated in the middle tier
    - directly controlled by the IT department
    - has direct and complete access to all enterprise services
    - possibly connected with external applications
- A predefined set of **components** that offer various functions
- A set of **containers** that control the components execution and manage their life cycle, communication, security, …

# The architectural model of JEE



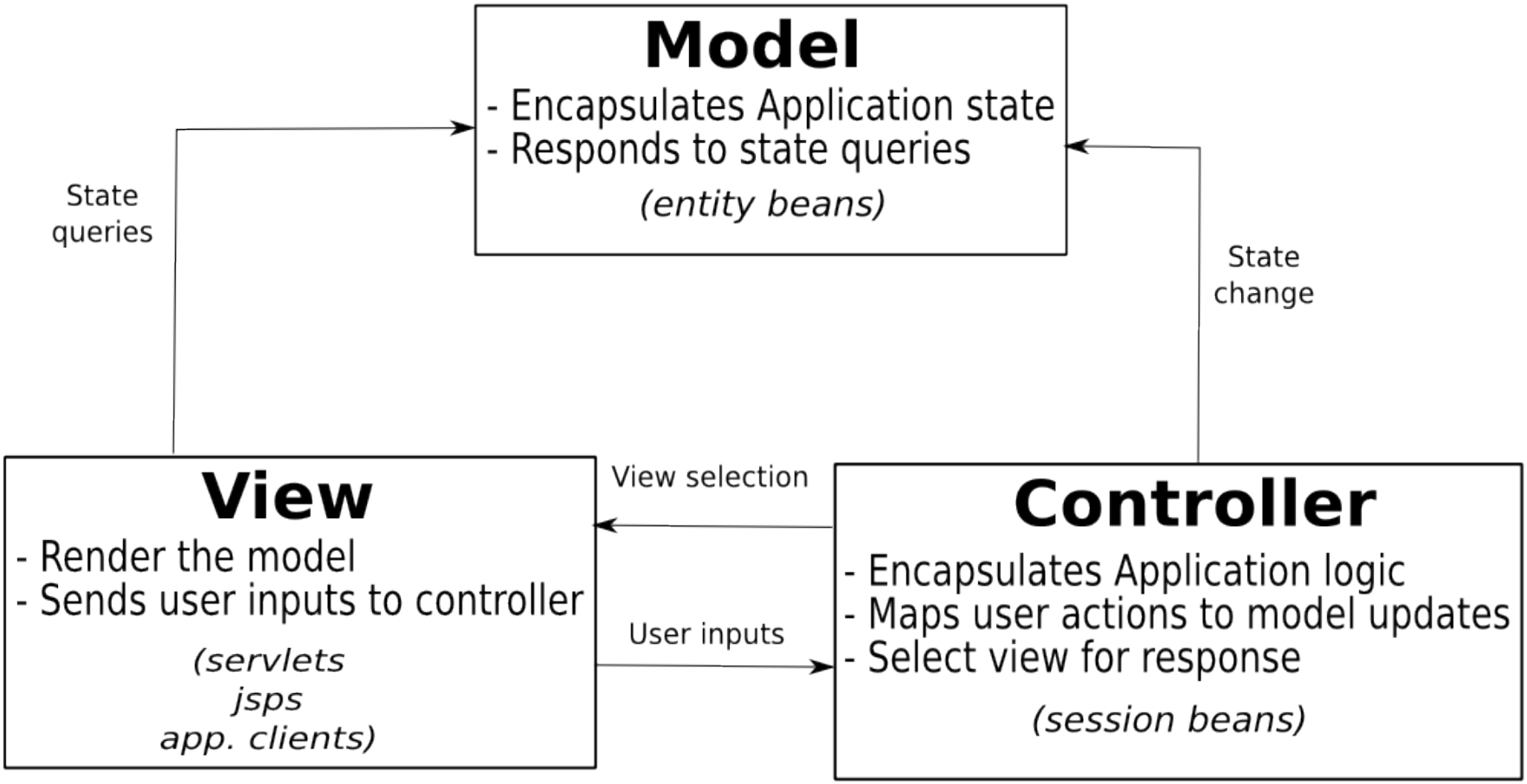

- (from Ball et. al. *The Java EE 5 tutorial)*

# The supermarket data warehouse example in the JEE model



# JEE: a logical point of view

- JEE implements the MVC pattern



# Components and component models: general definitions

- ***Component:*** software element that
  - Encapsulates the reusable implementation of some functionality
  - Can be composed without modifications
  - Respects the constraints imposed by *a component model*
- ***Component model***: is a combination of
  - Standard governing how to build individual components
  - How to organize (compose) components to build an application
  - How component can communicate and interact among each other
- Chris Luer, André van der Hoek – “Composition Environments for Deployable Software Components” – August 2002

# JEE components

- A JEE component is a self-contained software functional unit
  - Can be assembled in a JEE application
  - Is able to communicate with other components
- JEE components available to
  - Develop clients
    - Client applications and applets (executed at the client site)
    - Servlet, JavaServer Faces, JSP (executed at the JEE server site)
  - Develop the application logic and the data interaction mechanisms
    - Enterprise JavaBean (EJB, executed at the JEE server)

# JEE Clients (1)

- Web client: composed by
  - Web pages dynamically generated by the web tier components
  - Web browser
  - Usually, thin client
    - They do not access directly to the data
    - They do not execute some complex business logic
- Applet
  - Small clients written in Java
  - Downloaded from the web tier as part of a web page
  - Executed on the client JVM
  - Require plugins are enabled on the web browser and a proper security policy file
- Note: usually the web client solution is preferable to the applet
  - Less requirements on the client configuration
  - Possibility to delegate the development of web pages to a web design group not expert of Java

# JEE clients (2)

- Application client
  - Executed directly at the client site
  - Useful when the user interface is particularly rich and difficult to develop with a markup language
  - Interact directly with EJBs
  - … but is can interact also with the web tier opening an http communication channel
  - It can be written in any programming language
    - It can act as a bridge between the JEE application and other development environment (e.g., for embedded systems)
- The rule to choose between a web client and an application
  - The web client simplifies the distribution, deployment, and update of the system
  - The application client allows to build a more complex client
    - It can improve the reactivity of the system from the user point of view

# Web tier components

- Servlets: Java classes that
  - run on the web server,
  - serve the client HTTP requests, and
  - dynamically generate pages in reply;
  - are executed within a web container
- JSPs (JavaServer Pages): separate content generation from presentation. Developers exploit
  - HTML or XML tags to define the structure of a page, and
  - JSP tags or scriptlets to generate dynamic contents;
  - JSP tags or scriptlets are executed server side
- JavaServer Faces: provide
  - a set of standard graphic widgets,
  - mechanisms to connect
    - those widgets to some data source, and …
    - client-side events to server-side event handlers

# Business level components

- EJB: implement the logic that addresses the needs of the specific business domain
- EJB types
  - Session Beans
  - Entity Beans
  - Message Driven Beans

# Session bean

- Component that*:*
  - *Serve a single client*
  - *Has a relatively short lifecicle*
  - *Can be transaction aware*
  - *Can access persistent data stored in a database, but…*
  - *Does not directly represent persistent data (see the entity bean)*

# Entity beans

- Component that*:*
  - *Offers an object-oriented view on data stored in a database*
  - *Can have a long life (as long as the one of data in the database)*

# Message driven beans (MDB)

- Similar to a session bean. Differences:
  - *It subscribes to receive events (messages)*
  - *It can be activated when the subscribed events are received*
  - *Events are asynchronously communicated to the MDB*
  - *The event generator does not wait for a reply*

# The role of (non enterprise) JavaBean

- Components with properties and set and get methods
- Sometimes used to transfer data between clients and the JEE server or between the server and the database
- They are not considered JEE components

# JEE container

- The JEE containers offer services to manage
  - the component lifecycle
    - Deployment
    - Activation/instantiation
    - Configuration
    - Execution scope
    - Termination
  - transactions
  - security
  - lookup of other components
  - communication between components
  - ….
- They are the interface between components and the low level functionalities (primary services) offered by the platform
- Allow the developer to focus on the application problem, not on the “details”
- They always mediate the interaction between components

# JEE container types

- EJB container
- Web container
- Application client container
- Applet container

# EJB container and resource management

- Goal: allow application resource access to a high number of clients
- Solution: EJB available in more than one instance
  - one per active client
- Advantage: every client sees a dedicated service
- Problems:
  - Needs to instantiate and destroy a large number of components
  - Often the interaction with the client is very short
  - Thus, the instantiation and destroy time generates a significant overhead

# EJB container and resource management (2)

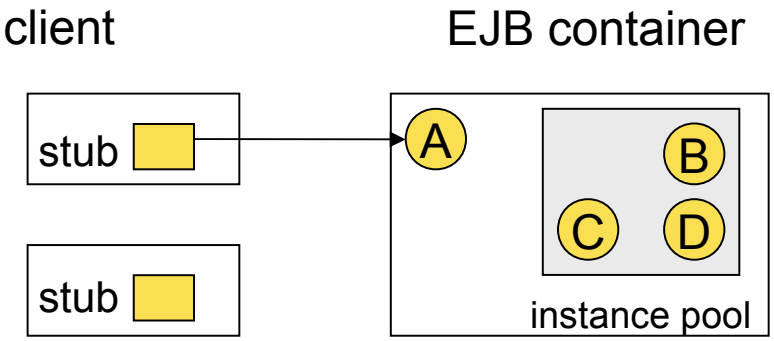
- Solution: instance pooling
  - The client sent a request to the EJB container EJB (never directly to the EJB)
  - The container selects an EJB instance from a pool and sends the request to it
  - After satisfying the request, the instance goes back to the pool

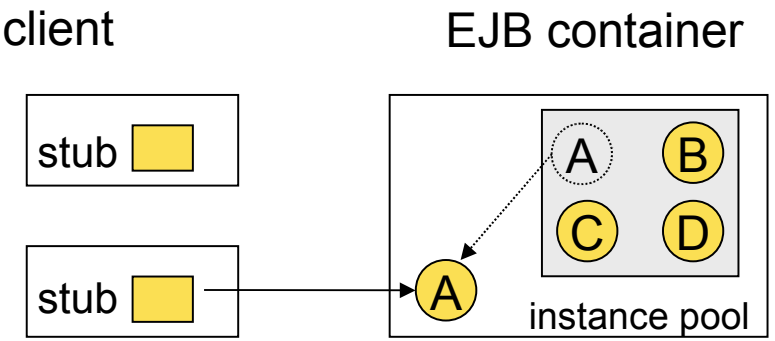
# Instance pooling and session beans

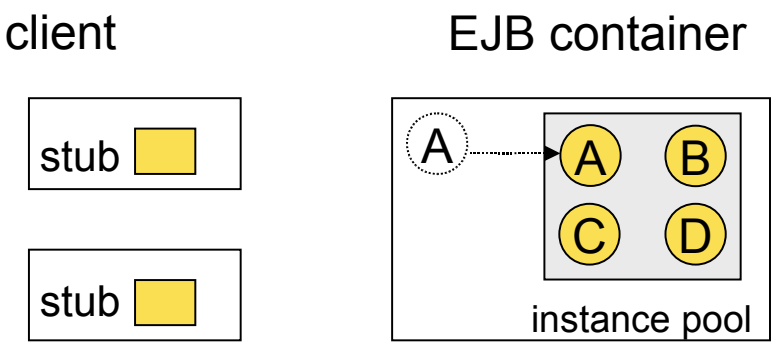
- **Stateless** session beans do not maintain any notion of state between different method invocations
  - We will see stateful session beans later on

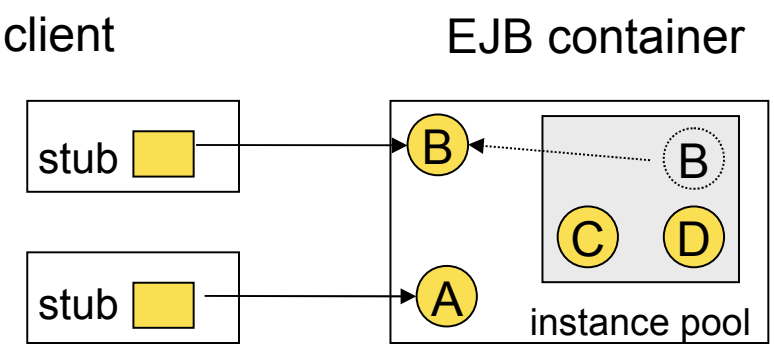
→A pool of stateless session beans are equivalent

# Instance pooling and session bean

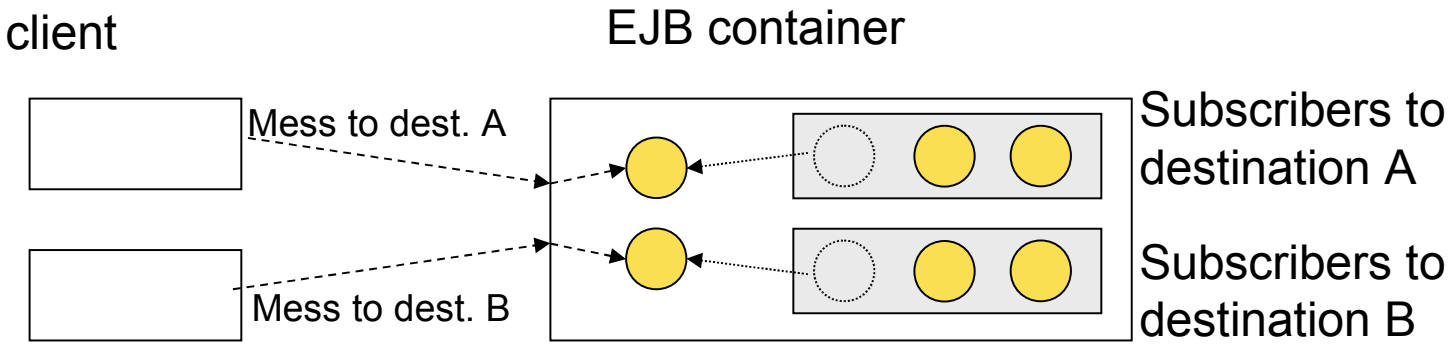



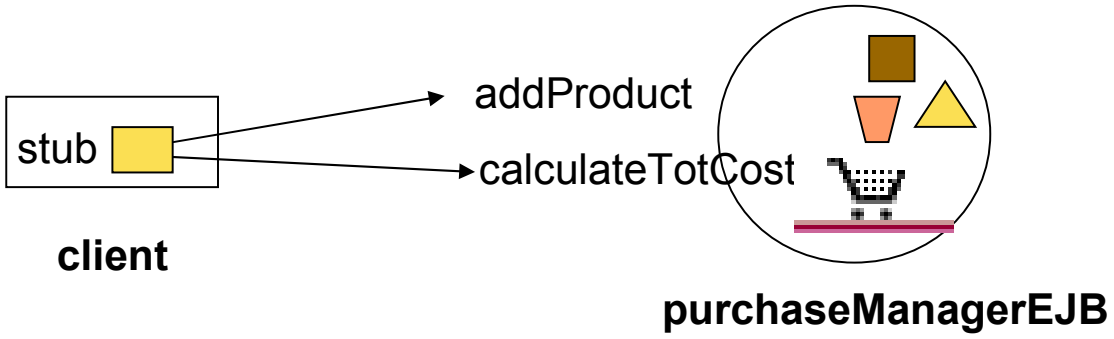



# Instance pooling and message driven bean



# Stateful session bean and passivation/activation mechanisms

- Stateful session bean: owns a *conversational state*
- Example:

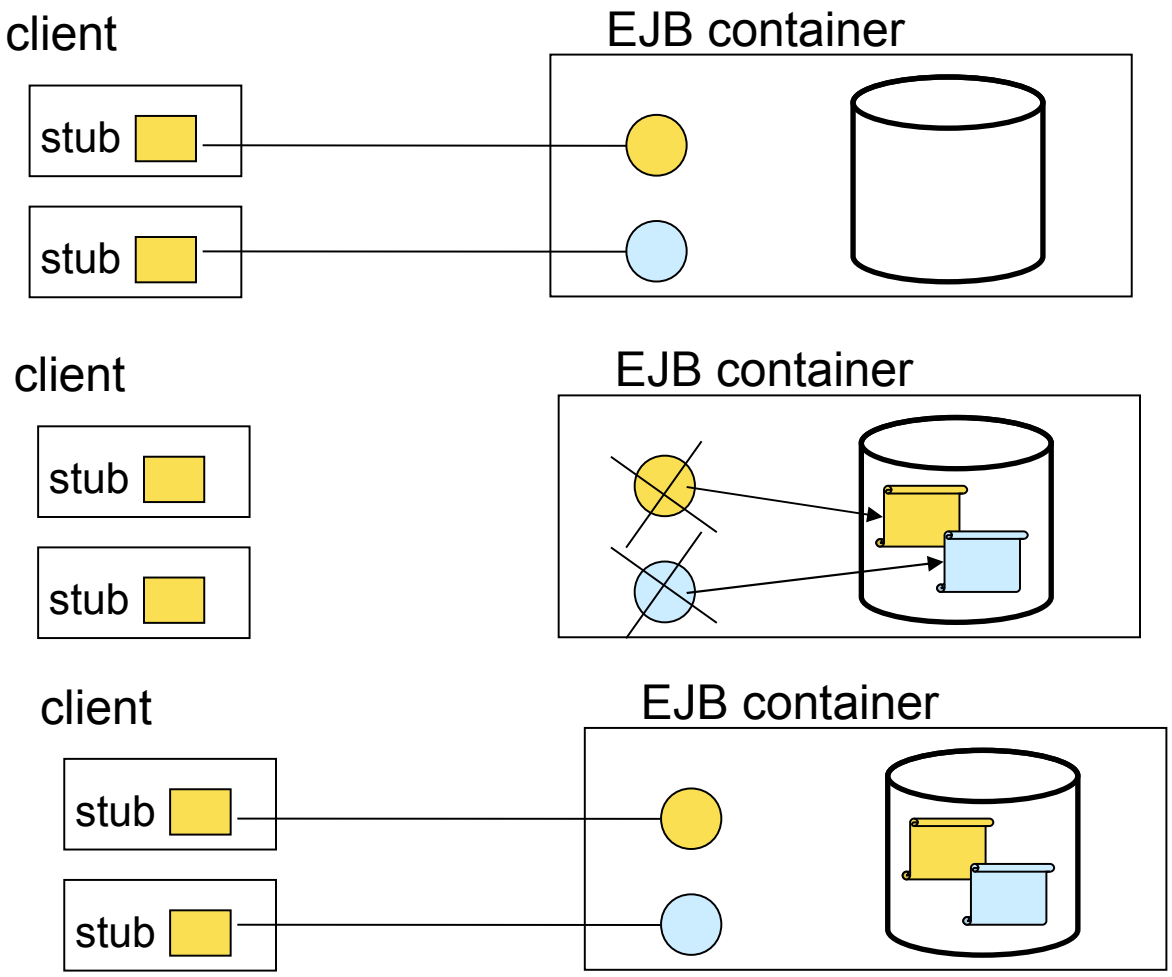



- The same EJB has to serve a client till the end of a conversation
- Stateful session bean do not participate to the istance pooling
- They are passivated when “idle” and activated when needed

# Passivation and activation (1)

- Passivation
  - The container serializes and stores the converational state of the EJB
  - The EJB is removed from the memory
- Activation
  - It is executed when a method is invoked on the EJB
  - The container creates a new instance of the EJB
  - Assign to this instance the state of the passivated instance
- Passivation and activation is transparent to clients

# Passivation and activation (2)



# Passivation and activation (3)

- Developers can define, as part of stateful EJBs, methods that are called
  - Right before passivation, e.g., to close a file
  - Right after activation, e.g., to assign values to transient variables
- The container rebuilds at the activation the remote references to the other beans

# Primary services

- They are added value services that an EJB platform is expected to offer
  - Support to concurrency
  - Transactions
  - Persistency
  - Distributed objects
  - Asynchronous messages
  - Timer Service
  - Naming
  - Security

# Concurrency

- Session bean and MDB: **NO** concurrent access
  - Stateful session bean stateful can be used only by the client that has created them
  - Stateless session bean do not need concurrency
    - They do not maintain a state
    - The container directs the call toward different instances
  - Similar to stateless sessions. Each MDB receives a different message
- Entity bean: concurrent access is **needed**
  - The persistence container protect shared data by copying them
  - Each client accesses to a different copy
  - Copy synchronization can be managed in various ways
    - Optimistic concurrency (through versioning)
    - Lock mechanisms

# Concurrency (2)

- … Thus, JEE manages concurrency, but this is a task of the container!
- It is outside the control of developers
  - It is forbidden to define synchronized methods and to create threads within an EJB

# Transactions

- Transaction: a set of instructions to be executed all together
  - Either all or nothing
- Transactions are automatically executed by the container
- The developer can exploit some APIs to gain direct control on them
  - … will see later on

# Persistency

- Persistency allows the entity beans to be durable
  - Their methods and public attributes can be accessed at any time
  - Information is not lost in case of a system failure
- The EntityManager service allows to
  - Create, find, query, remove, update entity beans
- EJB containers exploit these mechanisms to manage the lifecycle of entity beans
- Entity beans can be *detached* from their container when they are used to transfer data
- They can then be reattached to the container that, at this point, manages the synchronization with the main copy

# From objects to relations (1)

- Relational DBMSs are very common
- JEE defines a mechanism to map objects (entity beans) to relations in a relational DBMS (O/R mapping)

```
@Entity
@Table(name="Cabin")
public class Cabin {
 private int id;
 private string name;

 @Column(name="NAME")
 public string getName(){return name;}
 public void setName(String n){name=n;}

 @Column(name="ID")
 public int getId(){return id;}
 public void setId(int n){id=n;}
}
```

# From objects to relations (2)

- Whenever the mapping is identified, the EJB container is in charge to keep entities and relations in the DB synchronized
- One entity can map on more than one table (multiple joins and updates)
- … and it can maintain relations with other entities
  - One to one
  - One to many
  - Many to many

# Distributed objects

- Clients do not see directly the EJBs nor their containers
- They access the remote interface implemented by session beans
- Various communication protocols are possible
  - Java RMI-IIOP (mandatory)
  - CORBA IIOP
  - SOAP
  - …
- Every protocol has to map to Java RMI-IIOP so that clients can exploit the java EJB APIs for communication
- Clients can be written in any programming language

# Asynchronous enterprise messages

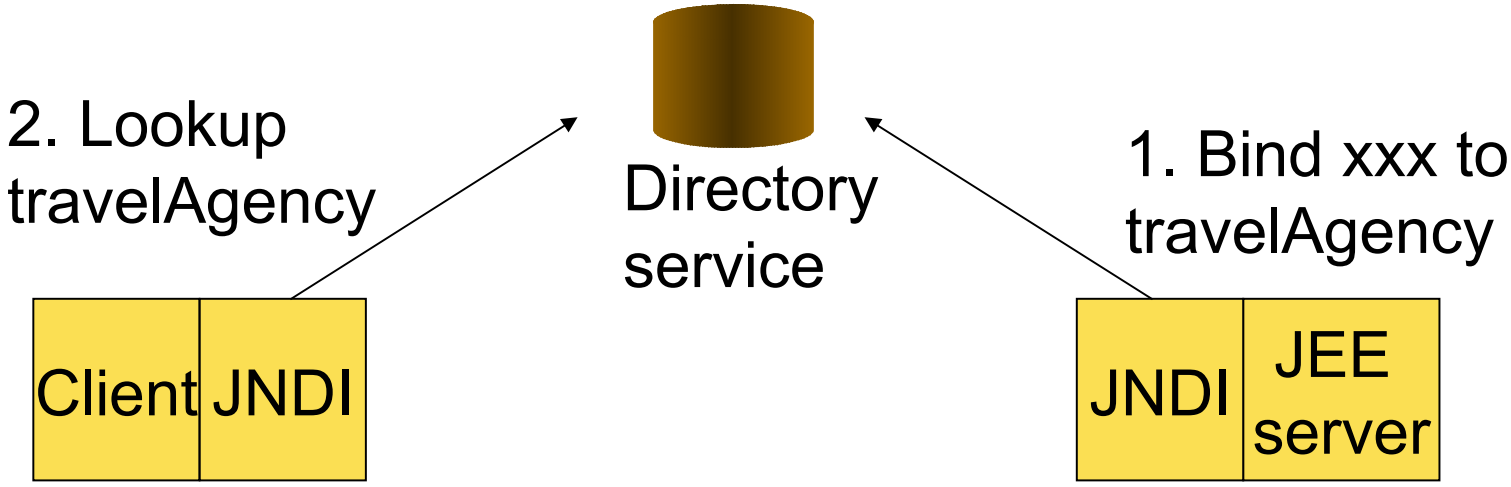
- They activate MDBs
- The can be generated by any kind of EJB as well as by clients
- Usually they exploit protocols and APIs defined by JMS (Java Messaging Service)
- … with one more characteristic: routing is reliable
  - In case of failure, the message is retransmitted (some platforms fix a limited number of attempts)
  - The messages can be persistent till their delivery
  - Messages are transactional
    - If the MDB processing a message fails, the corresponding transaction is aborted…
    - the container delivers the message to another MDB

# The timer service

- Timers are used to plan notifications to be sent in specified time instants
  - Example: in a bank application, every time a loan is started, we can define some timers corresponding to the deadline for payments
- Timers can be associated to
  - entity bean
  - stateless session bean
  - message-driven bean

# Naming

- Java Naming Directory Interface (JNDI) providers mechanisms to publish and find objects
- It allows the usage of various directory services
  - LDAP, DNS, …



# Security

- Offered services
  - Authentication: validates the users identity
  - Authorization: authorizes the user to access some information/operations
  - Secure communication: offers mechanisms to encript a communication channel

# JEE Connector Architecture

- JEE Connector Architecture (JCA) defines the way Enterprise Information Systems (EISs) can be integrated with JEE containers.
- A *resource adapter* allows JEE components to interact with the resource manager of an EIS
- Providers of JEE platforms and application integrators can use JCA to develop new resource adapters

# References


- Burke & Monson-Haefel. *Enterprise JavaBeans 3.0.* O Reilly, fifth edition 2006.
- Ball et. al. *The Java EE 5 tutorial*. Sun Microsystems 2006. http://java.sun.com/javaee/5/docs/tutorial/doc/