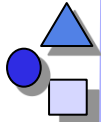


12) Enterprise Java Beans

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und
Multimediatechnik
<http://st.inf.tu-dresden.de>
12-1.0, April 25, 2012

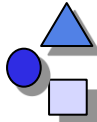


1. Basics
2. Different Kinds of Beans
 1. EJB in 2.0
3. EJB in 3.0
4. Evaluation



Obligatory Reading

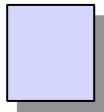
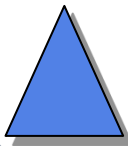
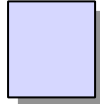
- ▶ Sun's enterprise bean tutorial
<http://java.sun.com/javaee/reference/tutorials/index.jsp>
- ▶ Szyperski, Chap 14
- ▶ <http://xdoclet.sourceforge.net>
- ▶ EJB 3.0 Features
http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/
- ▶ JBoss has a EJB 3.0 tutorial
<http://docs.jboss.org/ejb3/docs/tutorial/1.0.7/html/index.html>

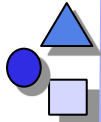


Literature

- ▶ JBoss EJB 3.0 Documentation
<http://docs.jboss.org/ejb3/app-server/>
- ▶ <http://developers.sun.com/docs/web/swdp/r1/tutorial/doc/toc.html>
- ▶ http://java.sun.com/developer/technicalArticles/Interviews/community/bien_qa.html
- ▶ The Java EE 5 Tutorial. For Sun Java System Application Server 9.1. Sun Microsystems, Sept. 2007.
<http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf>
- ▶ Ed Roman: Mastering EJB. Wiley & Sons.
<http://www.theserverside.com/books/wiley/masteringEJB/index.jsp>
- ▶ B. Tate, M. Clark, B. Lee, P. Linskey: Bitter EJB. Manning Publications Co.

12.1 Basics





Basics of Enterprise Java Beans (EJB)

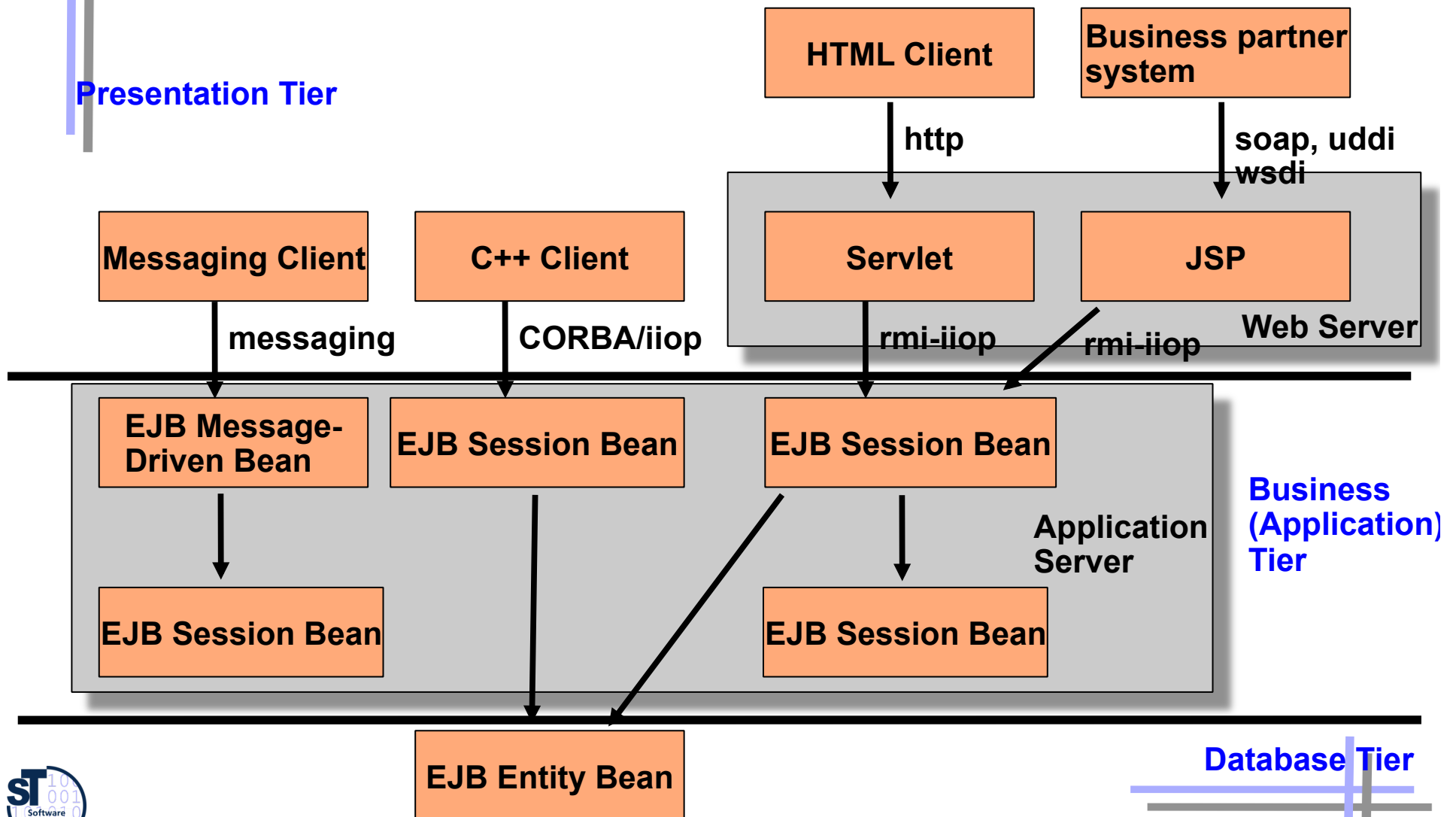
- ▶ Developed by SUN, now Oracle
 - Server-side component architecture for building distributed OO business applications in Java
 - Separation of business logic and lower-level concerns (e.g., networking, transactions, persistence, ...) into *implicit middleware*
- ▶ EJB 1.0 1998, EJB 2.0 2001, current version is 3.0
- ▶ EJB integrates several principles:
 - Adapters
 - Interceptors
 - Containers as application servers for transparency of transaction and persistency
 - Annotation-based code generation
 - A simple XML-based composition language

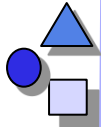


Ingredients of EJB

- ▶ Java-based Component Model (language specific)
 - Static components contain classes
 - Dynamic components contain objects
 - **Session Beans:** for business logic and application algorithms
 - **Message-Driven Beans:** Same function as session beans
 - Called by sending messages instead of calling methods
 - Have a message queue, react to an asynchronous message connector
 - **Entity Beans:** for business objects (data)
 - Persistent object that caches database information (an account, an employee, an order, etc)
 - Component factory (Home bean)
 - Customization possible by deployment descriptors
- ▶ Composition Technique
 - Adaptation/Glue:
 - Transparent distribution (Almost, see local/remote interfaces)
 - Transparent network protocols
 - Transparent transaction and persistency via Containers
 - No connectors

Interactions in an EJB Component System (Where are the Beans?)

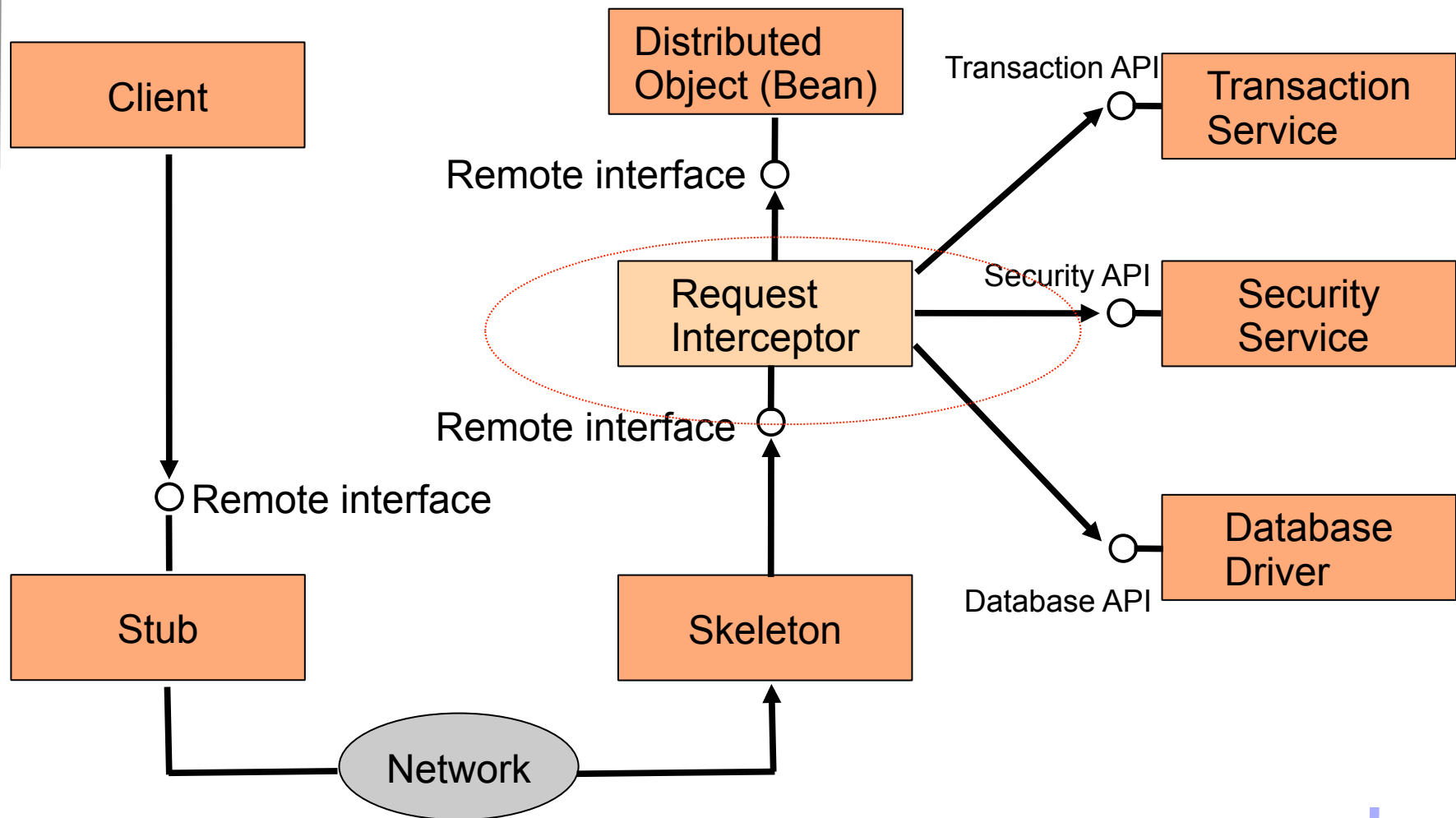




The Container/Application Server

- ▶ The Bean Container (application server)
 - The container is a façade of all beans, and a wrapper (Decorator, Interceptor) of an individual bean
 - In a container, some business logic may run on the server, hiding the direct data access
 - The container manages the beans
 - Factory: create; Repository: find, remove, Facade
 - The container provides middleware services the beans can use (*implicit middleware*)
 - Write *only* business logic
 - Declare the middleware services that you need (transactions, persistence, security, resource management, ...etc)
 - The middleware services are provided automatically
 - In explicit middleware (see CORBA), middleware services have to be addressed by the programmer
- ▶ Some common application servers
 - JBoss – free software www.jboss.org, Apache Geronimo
 - BEA's WebLogic, IBM's WebSphere, Oracle's Oracle 11g

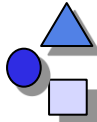
Implicit Middleware by Interceptors (Decorators)





Parts - Overview

- ▶ Bean class
- ▶ Home – a factory
- ▶ Local interface [3.0: annotation]
- ▶ Remote interface [3.0: annotation]
- ▶ Deployment descriptor



The Parts of an EJB

- The Enterprise Bean Class

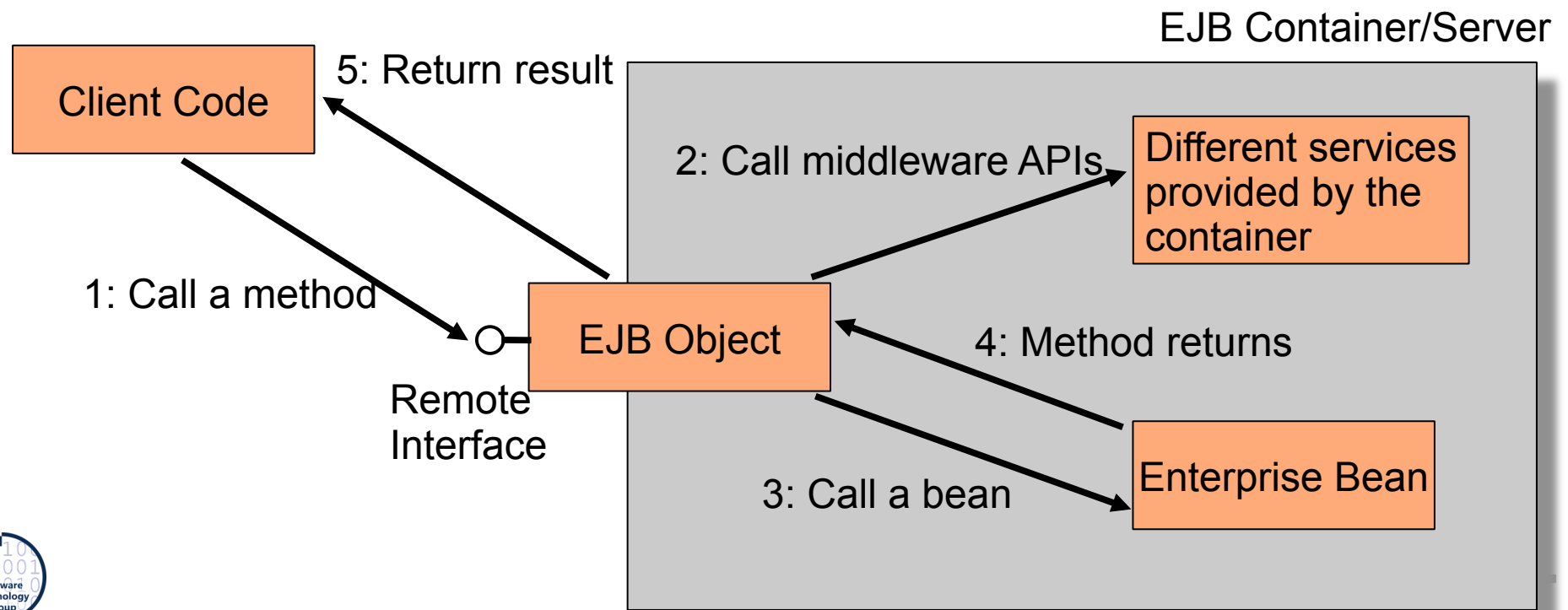
- ▶ The implementation of the bean looks different depending on which kind of bean
- ▶ Session beans
 - Business-process-related logic
 - e.g., compute prices, transfer money between accounts
- ▶ Entity beans
 - Data-related logic
 - e.g., change name of a customer, withdraw money from an account
- ▶ Message-driven beans
 - Message-oriented logic
 - e.g., receive a message and call a session bean

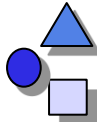


The Parts of an EJB

- The EJB Object

- ▶ The enterprise bean is not called directly
 - Instead an EJB object is generated by the container (facade object, proxy)
 - The EJB object filters the input and intercepts calls and delegates them to the bean
 - The EJB object is responsible for providing middleware services





The Parts of an EJB

- The Remote Object Interface

- ▶ The interface to the bean that the client sees
 - Must contain all methods the bean should expose
- ▶ As the EJB object lies between the client and the bean, it has to implement this interface
 - Must extend `javax.ejb.EJBObject`

```
public interface Bank extends javax.ejb.EJBObject {  
  
    public Account getAccount(String name)  
        throws java.rmi.RemoteException;  
  
    public void openAccount(String name)  
        throws java.rmi.RemoteException;  
  
}
```



The Parts of an EJB

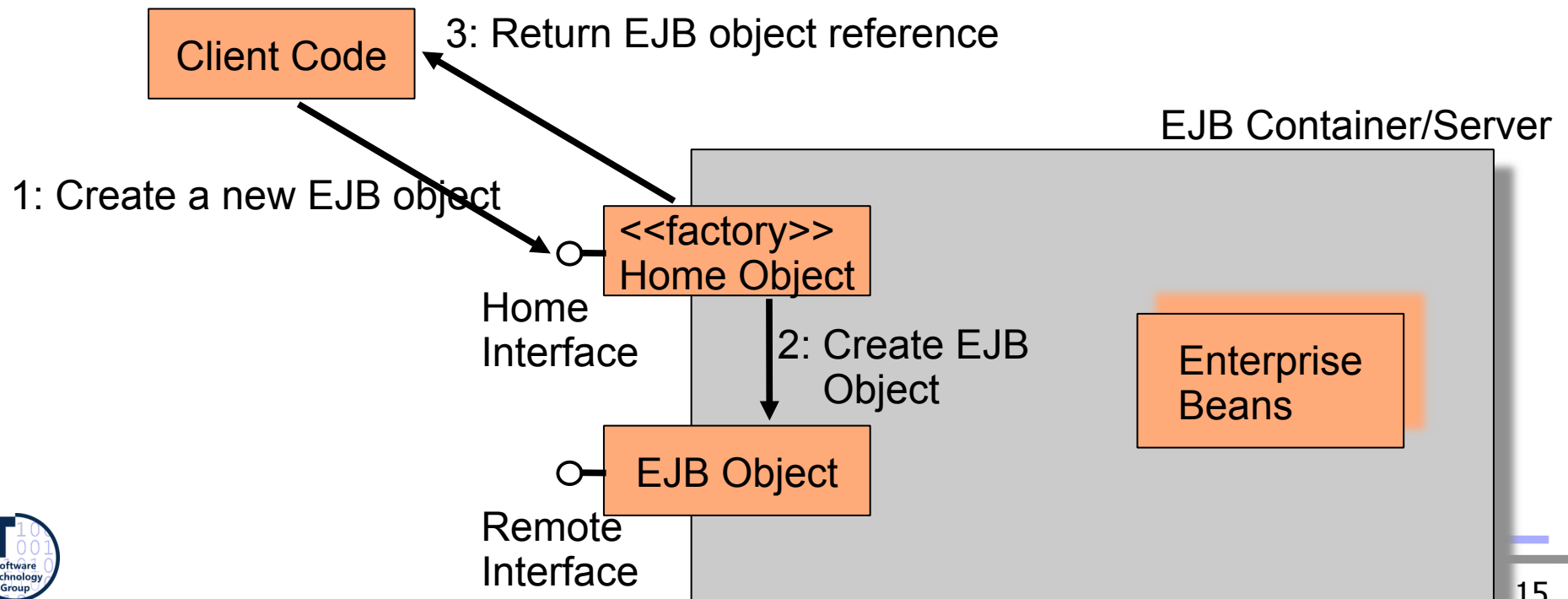
- The Home Object (the Factory)

- ▶ How does the client get hold of an EJB object?
 - The EJB object can exist on a different machine
 - EJB promotes location transparency, so the client shouldn't have to care where the EJB object is located
- ▶ An EJB object *factory* and *repository* is needed: The home object
 - Create EJB objects
 - Find existing EJB objects
 - Remove EJB objects

The Parts of an EJB

- The Home Object and Interfaces

- ▶ The home object needs a *home interface* (factory)
 - Defines methods for creating, finding and removing EJB objects
- ▶ The communication uses Java RMI over IIOP
 - If an argument is serializable, it is sent as pass-by-value
 - RMI can also simulate pass-by-reference
 - A serialized stub for the remote object is sent instead





The Parts of an EJB

- Local Interfaces

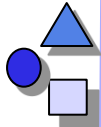
- ▶ Optionally, you can provide local interfaces
 - local interface corresponding to remote interface
 - local home interface corresponding to home interface
- ▶ When beans are located locally it is possible to use local calls

Remote:

- ▶ Client calls a local stub
- ▶ Marshalling
- ▶ Stub calls skeleton over a network connection
- ▶ Unmarshalling
- ▶ EJB object is called, performs middleware services
- ▶ Bean is called
- ▶ Repeat to return result

Local:

- ▶ Client calls a local object
- ▶ Local object performs middleware services
- ▶ Bean is called
- ▶ Control is returned to the client



Drawbacks of Using Local Interfaces

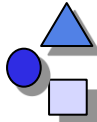
- ▶ They only work when calling beans in the same process
 - Code for local interfaces differs from code for remote interfaces
 - To switch between local and remote calls it is necessary to change the code
 - Location transparency is not preserved
- ▶ The marshalling of parameters is by reference
 - This is different from remote calls which are by value
 - There is a definite speed gain...
 - ...but it can be error-prone because the semantics are different from remote calls
- ▶ Horrible: this should be encapsulated in a connector!



The Parts of an EJB

- The Deployment Descriptor

- ▶ An XML file in which the middleware service requirements are declared (There is a DD-DTD)
 - Bean management and lifecycle requirements
 - Transaction, persistence, and security requirements
- ▶ Composition of beans (references to other beans)
 - Names: Name, class, home interface name, remote-interface name, class of the primary key
 - States: type (session, entity, message), state, transaction state, persistency management - how?
- ▶ The application assembler may allocate or modify additional different information
 - Name, environments values, description forms
 - Binding of open references to other EJB
 - Transaction attributes



Example of a Deployment Descriptor

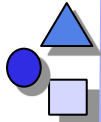
```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise  
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
```

```
<ejb-jar>  
  <enterprise-beans>  
    <session>  
      <ejb-name>Bank</ejb-name>  
      <home>com.somedomain.BankHome</home>  
      <remote>com.somedomain.Bank</remote>  
      <local-home>com.somedomain.BankLocalHome</local-home>  
      <local>com.somedomain.BankLocal</local>  
      <ejb-class>com.somedomain.BankBean</ejb-class>  
      <session-type>Stateless</session-type>  
      <transaction-type>Container</transaction-type>  
    </session>  
  </enterprise-beans>  
</ejb-jar>
```



The Parts of an EJB (2.0) ***- Putting It All Together***

- ▶ Finally all the above mentioned files are put into an EJB-jar file
 - bean class
 - home (and local home) interface
 - remote (and local) interface
 - deployment descriptor, i.e., the composition specification
 - (possibly vendor-specific files)



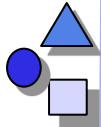
Deployment of an EJB

- ▶ The *deployment* of a bean is a new step in component systems we have not yet seen
- ▶ The application server is notified of the new bean by
 - using a command-line tool,
 - dropping the EJB in a specific directory,
 - or in some other way
- ▶ The EJB-jar file is verified by the container
- ▶ The container generates an EJB object and home object
- ▶ The container generates any necessary RMI-IIOP stubs and skeletons



Roles in the EJB Software Process

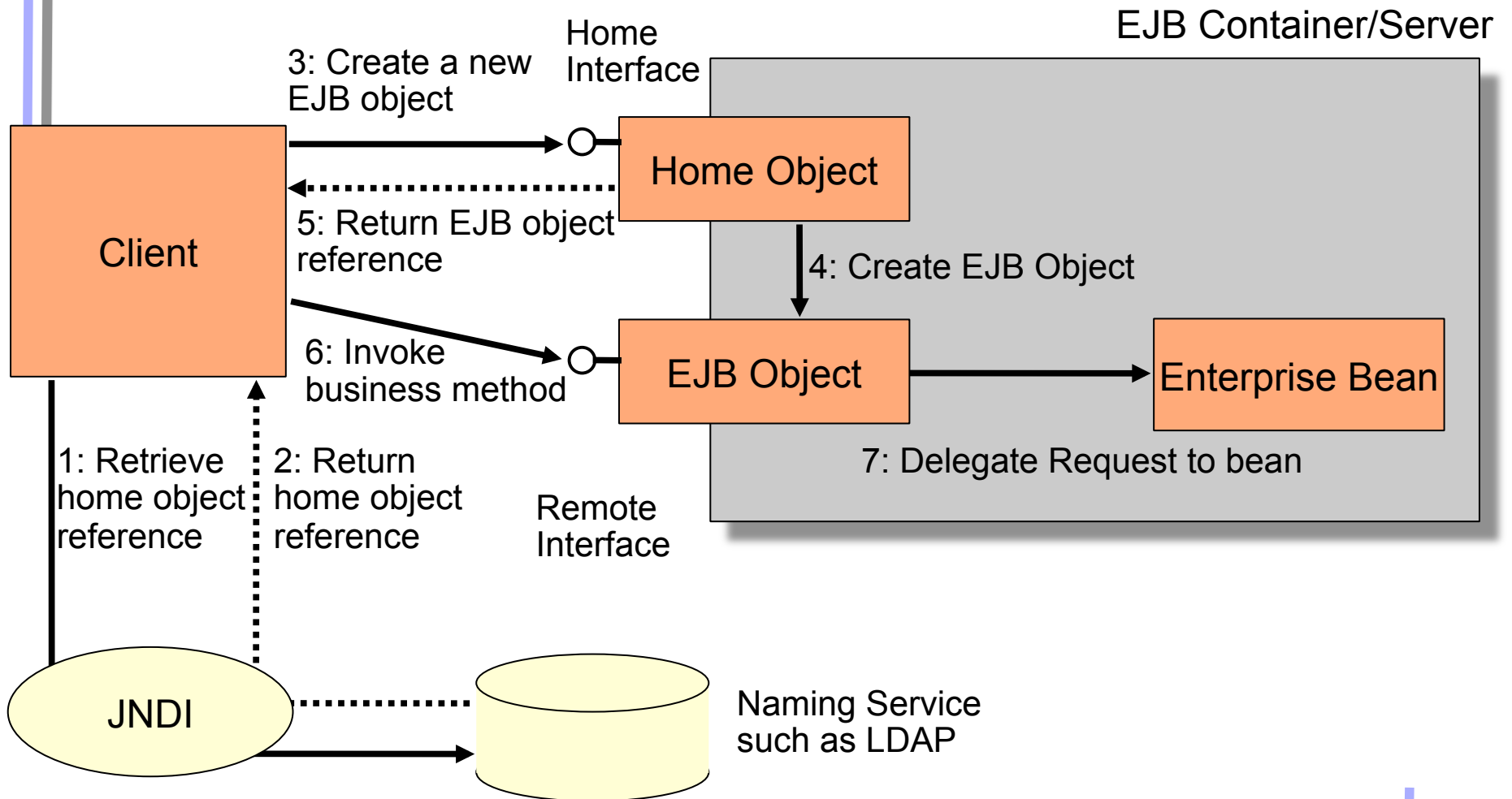
- ▶ *Bean provider* (bean producer) is an application expert
 - Builds a EJB-jar with application specific methods, deployment-descriptor, remote, home interface
- ▶ *Application assembler* composes EJB to larger EJB, i.e., applications units.
 - She extends the deployment-descriptors
- ▶ *Employer* (deployer) puts the EJB into a environment, consisting of a EJB Server and Container (Adapter).
 - Is the EJB connected to a EJB-Container, it is configured and usable
- ▶ *Server-provider* is a specialist in transaction management and distributed systems.
 - Provides basic functionality for distribution
- ▶ *Container-provider* (container provider) delivers the container tools for configuration and for run time inspection of EJB
 - The Container manages persistency of Entity Beans, generation of communication code (glue code) to underlying data bases



How to Find a Home Object

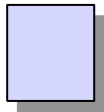
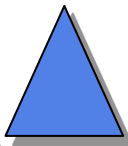
- ▶ To achieve location transparency the machine address of the home object should not be hard-coded
- ▶ Instead the Java Naming and Directory Interface (JNDI) is used to lookup home objects
 - JNDI is a standard interface for locating resources, similar to the Corba name service
 - Only the address to the JNDI server is needed
 - JNDI provides a mapping between the name of a resource and its physical location

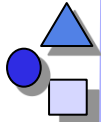
The Entire Process



12.2 A Closer Look at the Different Kinds of Enterprise JavaBeans

(only 2.0)
(for your own study)



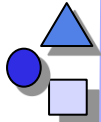


Session Beans Overview

- ▶ Reusable components that contain logic for business processes
 - The lifetime of a session bean is roughly equivalent to the lifetime of the client code calling it
 - A session bean is nonpersistent
- ▶ Two kinds of session beans
 - Stateful
 - Stateless

java.ejb.SessionBean

- **setSessionContext(SessionContext context)**
The bean can query the SessionContext for information concerning the container
- **ejbCreate()**
Used to perform initialization when the bean is created
- **ejbPassivate()**
Used by stateful session beans, explained later
- **ejbActivate()**
Used by stateful session beans, explained later
- **ejbRemove()**
Used to release any resources the bean has been holding before it is removed



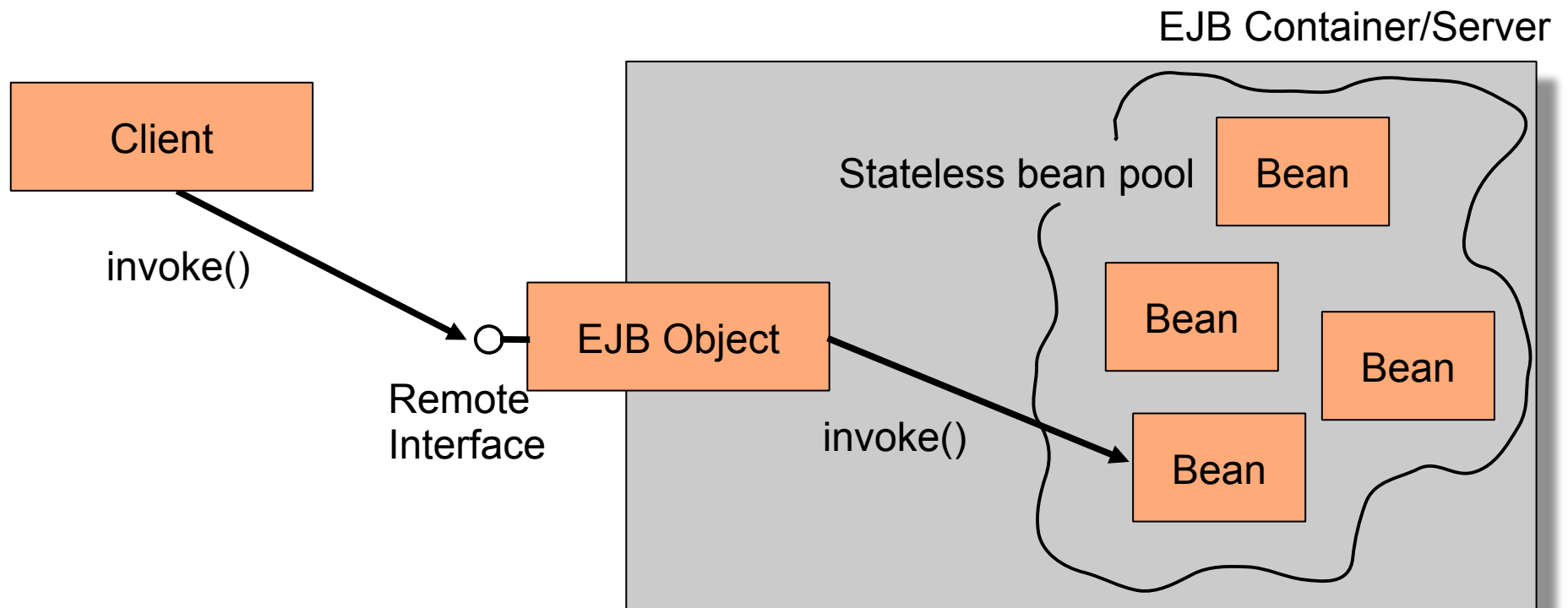
Stateless Session Beans

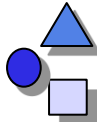
- ▶ Handle single requests
 - Conversations that span a single method call
 - Does not hold a conversational state
- ▶ The bean may be destroyed by the container after a call or it has to be cleared of old information
- ▶ Examples of stateless session beans
 - A user verification service
 - An encoding engine
 - Any service that given some input always produces the same result



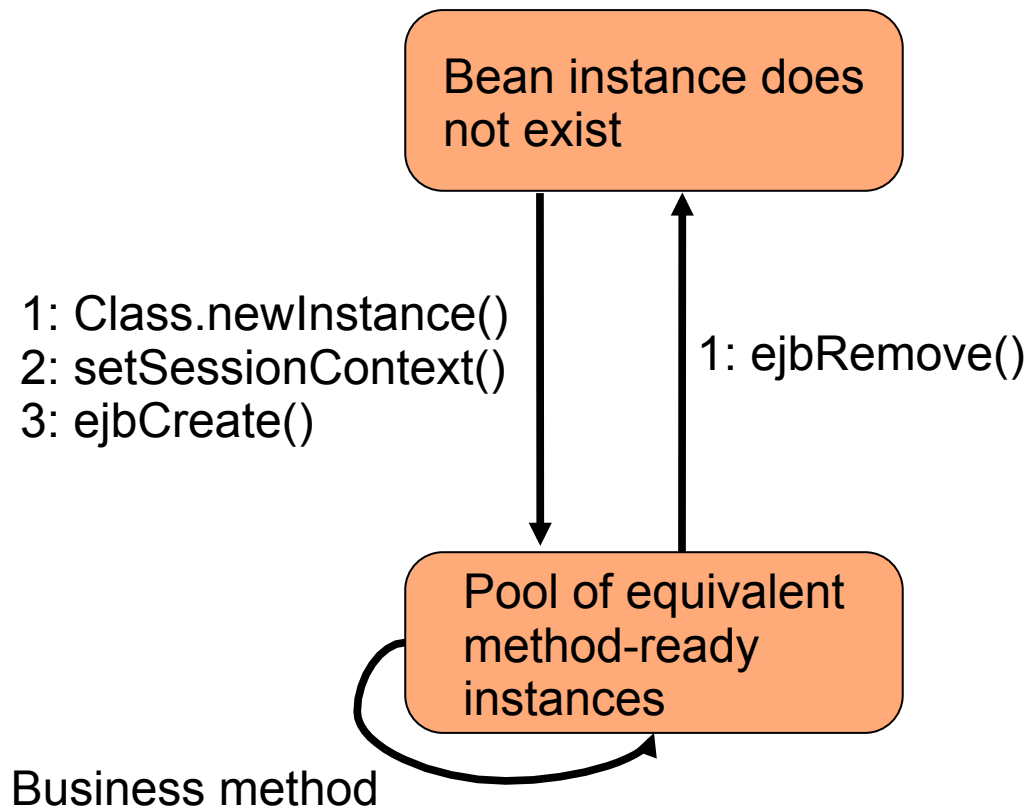
Pooling Stateless Session Beans

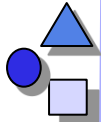
- ▶ Stateless session beans can easily be pooled (reused) to allow better scaling
 - They contain no state





Life Cycle of a Stateless Session Bean



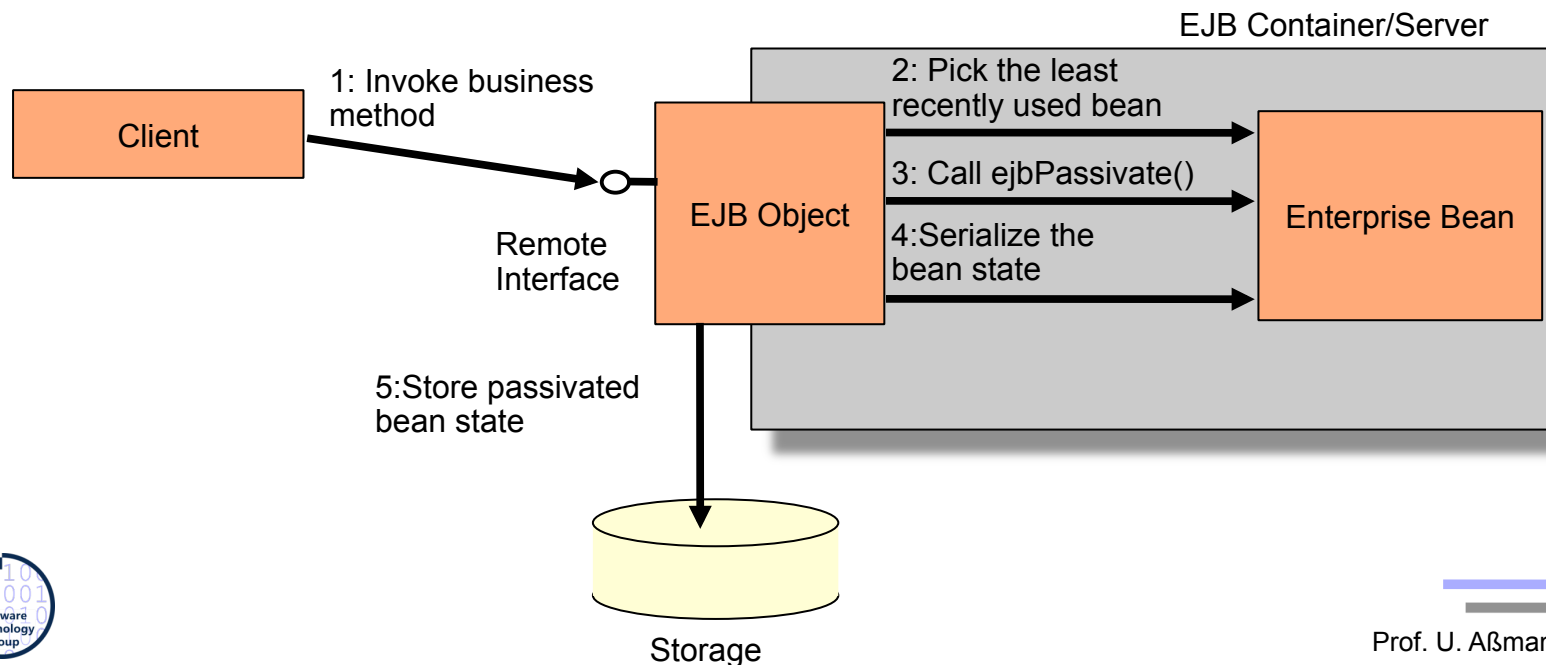


Stateful Session Beans

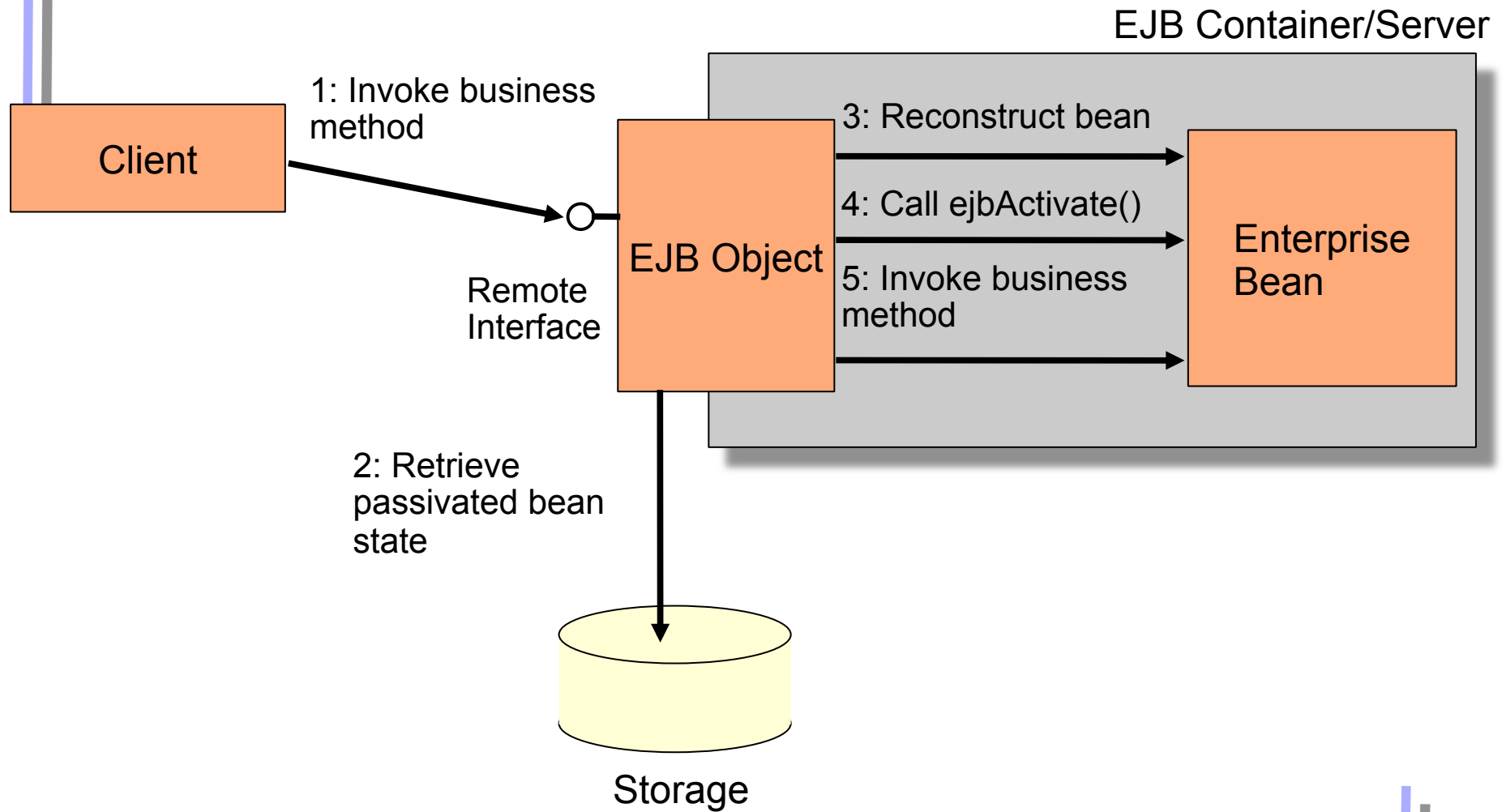
- ▶ Handles drawn-out conversations
 - E-commerce web store with a shopping cart
 - Online bank
 - Tax declaration
- ▶ Thus it has to retain its state between invocations

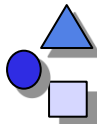
Pooling Stateful Session Beans

- ▶ Pooling becomes more complicated
 - Beans must be swapped from physical memory to disk
- ▶ A stateful session bean has to implement:
 - `ejbPassivate()`: Called to let the bean release any resources it holds before it gets swapped out
 - `ejbActivate()`: Called right after the bean has been swapped in to let it acquire the resources it needs

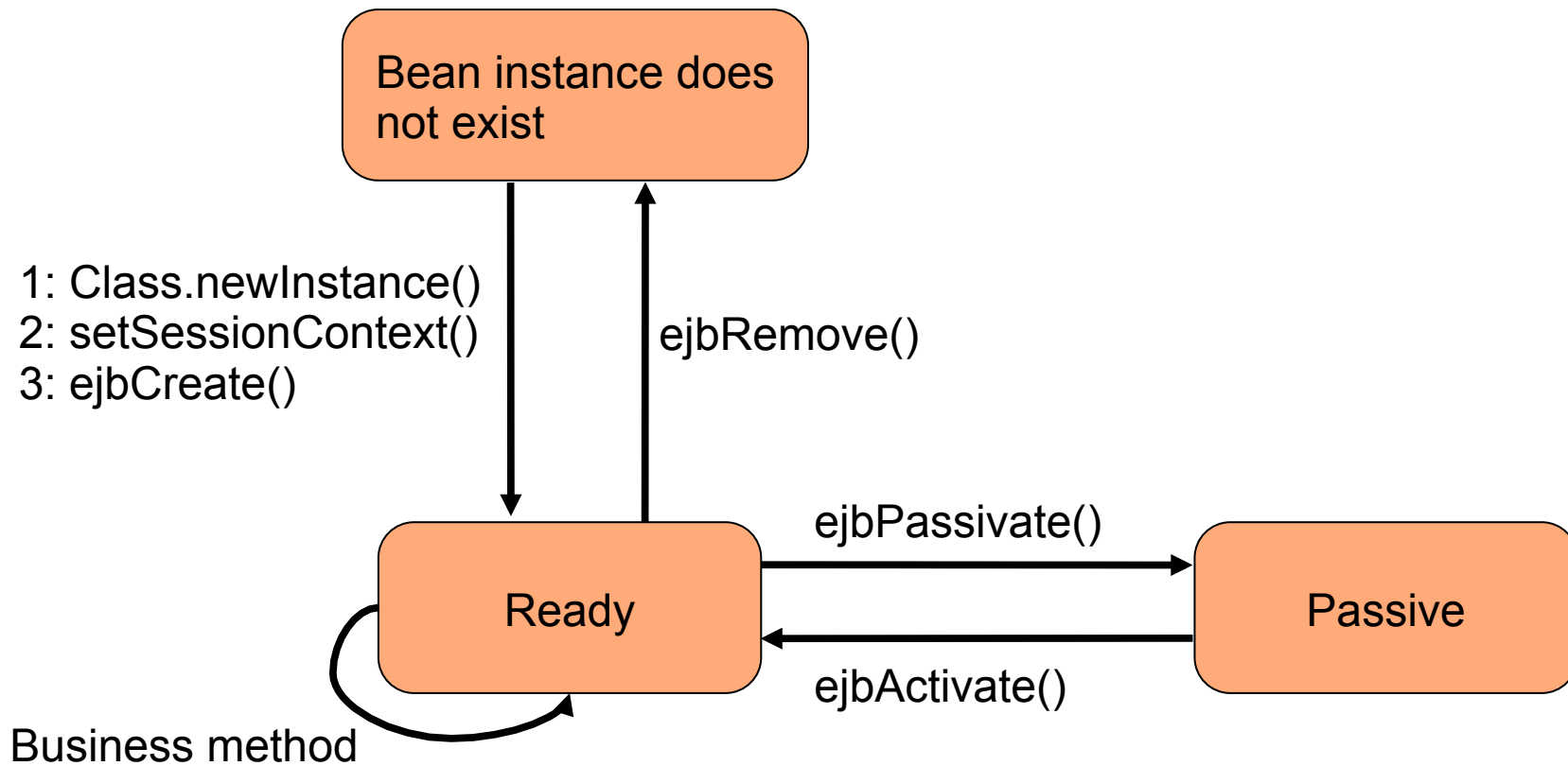


Activation of a Stateful Session Bean





Life Cycle of a Stateful Session Bean

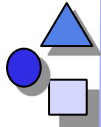




Message-Driven Beans

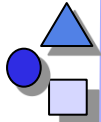
▶ Why?

- Performance
 - Asynchronous process means that clients don't have to wait for the bean to finish
- Reliability
 - With RMI-IIOP the server has to be up when the client is calling it.
 - With a message-oriented middleware (MOM) that supports guaranteed delivery, the message is delivered when the server gets back online
- Support for multiple senders and receivers
 - RMI-IIOP is limited to one client talking to one server



Characteristics of Message-Driven Beans

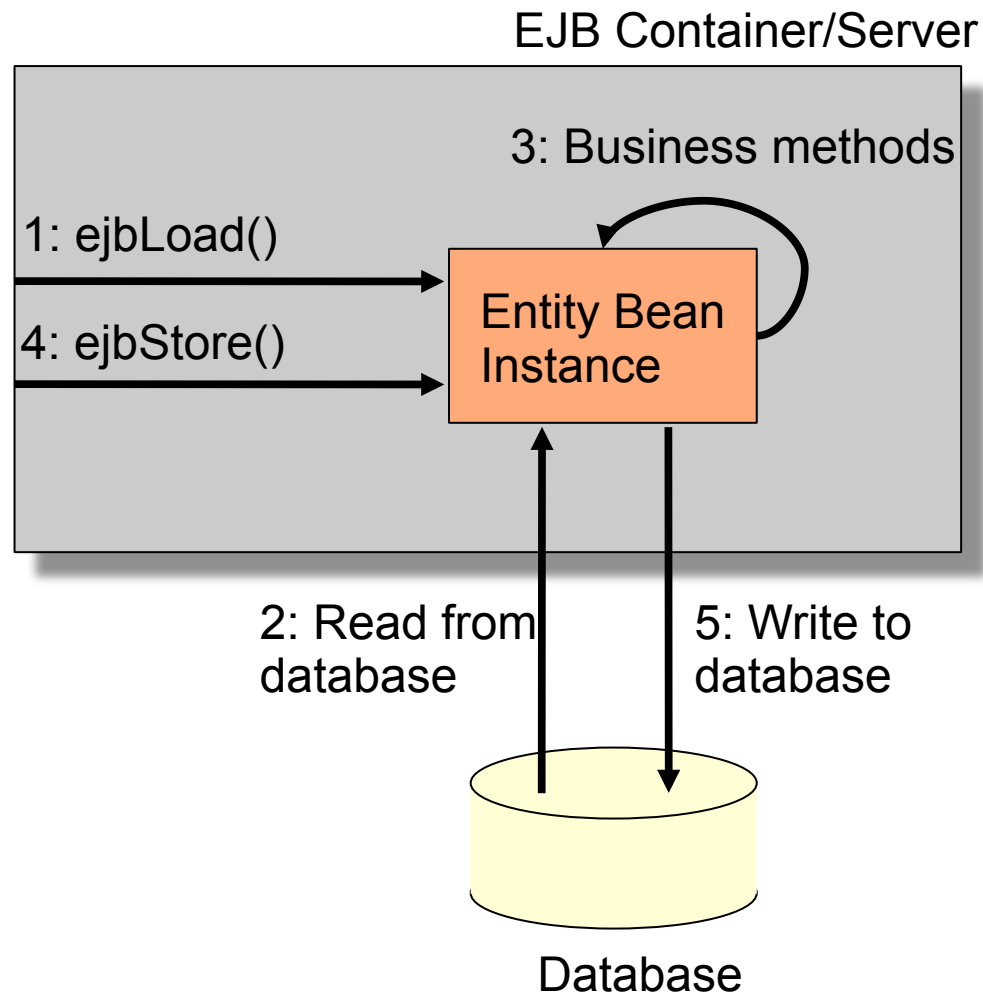
- ▶ MDBs don't have a home, local home, remote or local interface
- ▶ MDBs have a single, weakly typed business method
 - `onMessage()` is used to process messages
 - MDBs don't have any return values
 - However, it is possible to send a response to the client
 - MDBs cannot send exceptions back to clients
- ▶ MDBs are stateless
- ▶ MDBs can be durable or nondurable subscribers
 - durable means that the subscriber receives all messages, even if it is inactive



Entity Beans Overview

- ▶ Entity beans are persistent objects that can be stored in permanent storage
 - Live on the entity or database layer of the 3-tier architecture
 - The entity bean data is the physical set of data stored in the database
- ▶ An entity bean consists of the same files as a session bean
 - remote/local interface
 - home/local home interface
 - the enterprise bean class
 - the deployment descriptor
- ▶ Two kinds of entity beans
 - *Bean-managed persistent or container-managed persistent*

Loading and Storing an Entity Bean



12.3 Entity Beans in 2.0

(for your own study)





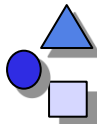
Features of Entity Beans

- ▶ Entity bean instances are a view into a database
 - The bean and the data in the database are conceptually the same
 - Entity beans survive failures: persistent
- ▶ Several entity bean instances may represent the same underlying data
 - An entity bean has a primary key to uniquely identify the database data
 - Entity bean instances can be pooled
 - must implement `ejbActivate()` and `ejbPassivate()`
- ▶ Entity beans are found with special finder methods

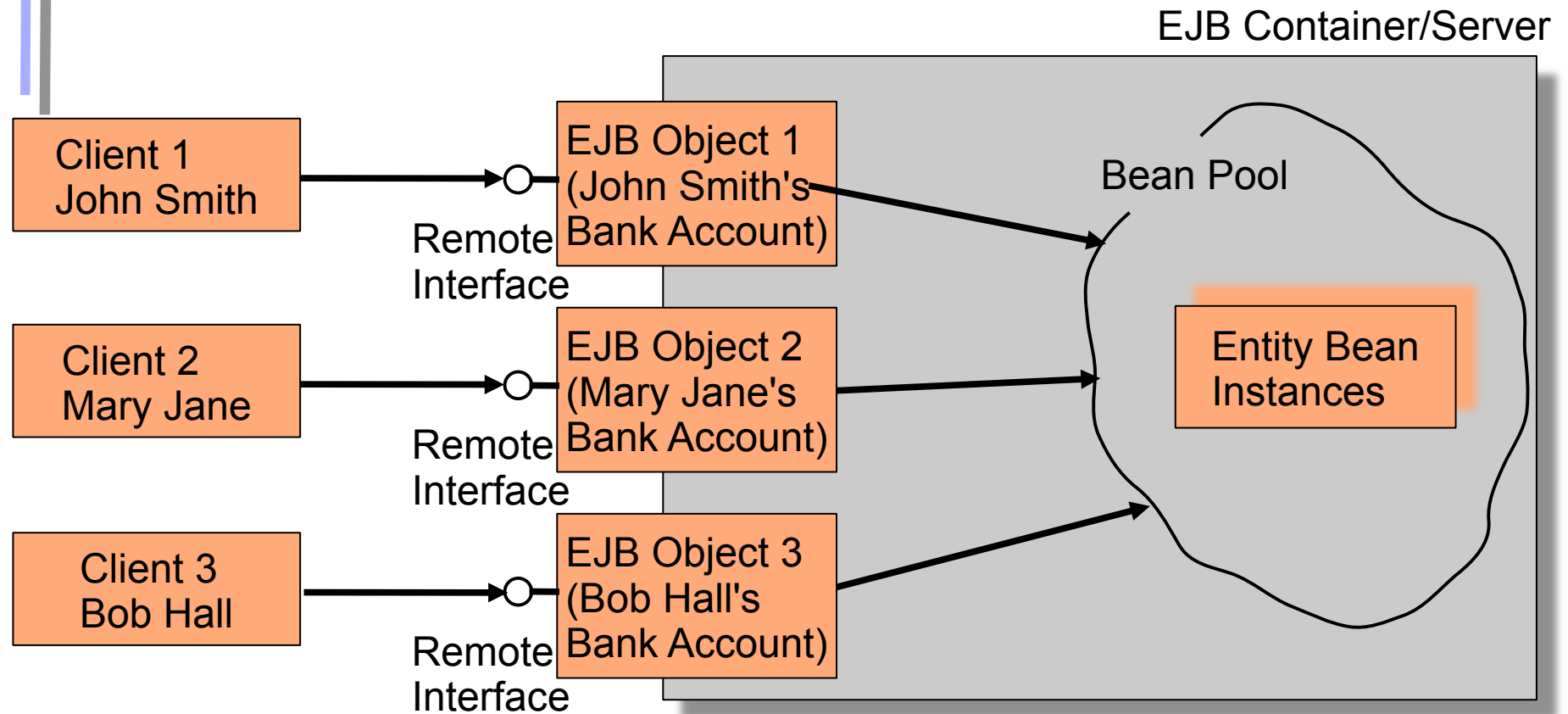


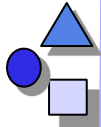
How is Persistence Achieved?

- ▶ **Serialization**
 - Very expensive to query objects stored using serialization
 - consider getting all accounts with a specific amount of money
- ▶ **Object-relational mapping (relational databases)**
 - Map the object to a relational database when it is stored
 - Allows advanced queries and visualization
 - The mapping is either hand-coded or achieved by finished products
- ▶ **Object databases**
 - Persistent store that holds entire objects, no mapping step
 - Queries possible by using an object query language (OQL)
 - Supports relationships between objects
 - Predictable scalability and performance
 - Strong integrity and security
 - Object databases haven't taken off so Object-relational mappings are normally used



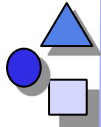
Pooling Entity Beans





Bean-Managed Persistent Entity Beans (BMP Beans)

- ▶ The developer is required to provide the implementation to map the instances to and from storage
 - Java Database Connectivity (JDBC)
- ▶ BMP beans have to implement `javax.ejb.EntityBean`:
 - `setEntityContext (javax.ejb.EntityContext)`
 - The context can be queried of information regarding the container
 - `unsetEntityContext ()`
 - `ejbRemove ()`
 - Removes the data from the persistent storage
 - `ejbActivate ()`
 - Lets the bean allocate resources after being swapped in
 - `ejbPassivate ()`
 - Called before the bean is swapped out so it can release resources
 - `ejbLoad ()`
 - Loads database data into the bean
 - `ejbStore ()`
 - Stores the data in the bean to the database



Bean-Managed Persistent Entity Beans

- ▶ BMP beans also have to other kinds of methods relating to storage
 - `ejbCreate()`
 - Used to create new entries in the database (optional)
 - Finder methods
 - `ejbFindXXX()`
 - Must have at least one: `ejbFindByPrimaryKey()`
 - Normally contains database queries
 - e.g., `SELECT id FROM accounts WHERE balance > 3000`
 - `ejbHomeXXX()` methods
 - Performs simple services over a set of beans
- ▶ A BMP entity bean consists of
 - Bean-managed state fields, persistable fields that are loaded from the database
 - Business logic methods: Performs services for clients
 - EJB-required methods: Required methods that the container calls to manage the bean



Example - Bean-Managed State Fields

▶ AccountBean.java

```
import java.sql.*;
import javax.naming.*;
import javax.ejb.*;
import java.util.*;

public class AccountBean implements EntityBean {
    protected EntityContext context;

    // Bean-managed state fields
    private String accountID;
    private String ownerName;
    private double balance;

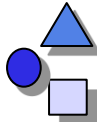
    public AccountBean() { }

    ...
}
```

```
public void deposit(double amount) {
    balance += amount;
}

public void withdraw(double amount) {
    if (amount < balance) {
        balance -= amount;
    }
}

public void getBalance() {
    return balance;
}
```



Example

- Business Logic Methods

```
...cont...
public void ejbHomeGetTotalBankValue() {
    PreparedStatement pStatement = null;
    Connection connection = null;
    try {
        connection = getConnection();
        pStatement = connection.prepareStatement(
            "select sum(balance) as total from accounts");
        ResultSet rs = pStatement.executeQuery();
        if (rs.next()) { return rs.getDouble("total"); }
    } catch (Exception e) { ... }
    finally {
        try { if (pStatement != null) pStatement.close(); }
        catch (Exception e) { ... }
        try { if (connection != null) connection.close(); }
        catch (Exception e) { ... }
    }
}
```

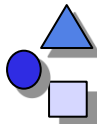
...cont...



Example - Required Methods

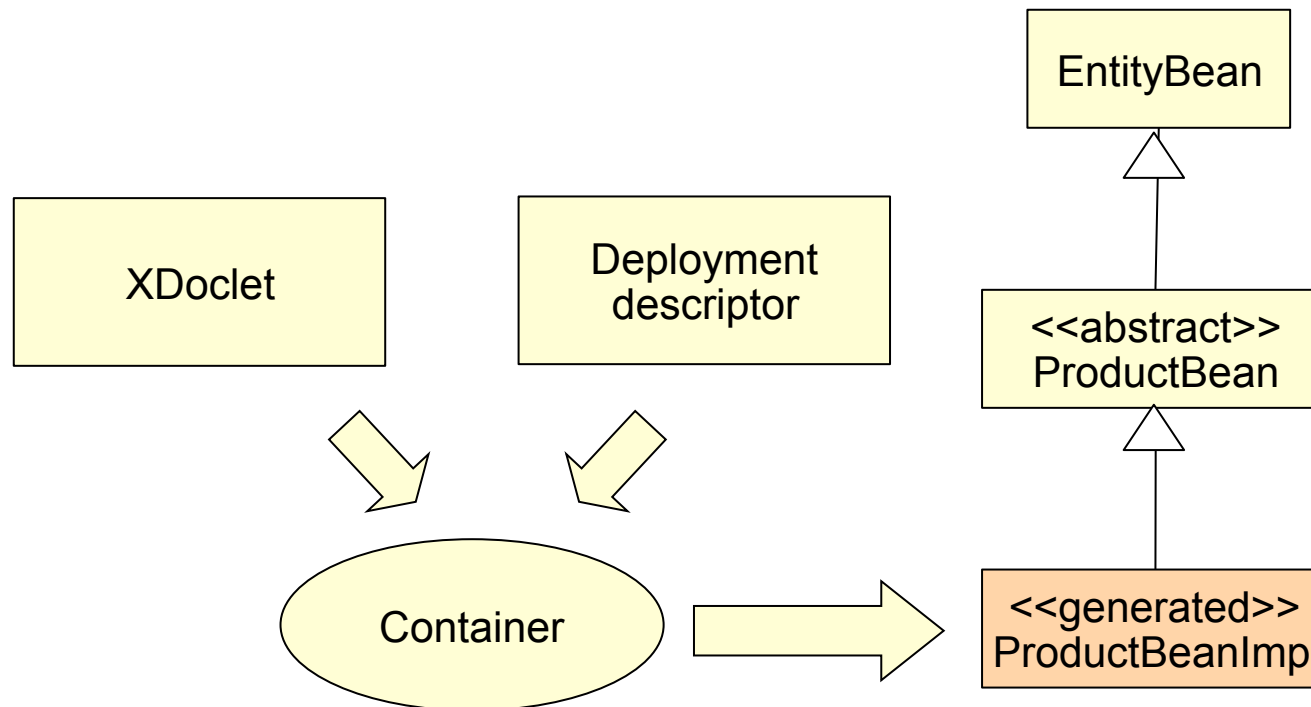
```
...cont...  
public void ejbRemove {  
    PreparedStatement pStatement = null;  
    Connection connection = null;  
    AccountPK pk = (AccountPK) context.getPrimaryKey();  
    String id = pk.accountID;  
    try {  
        connection = getConnection();  
        pStatement = connection.prepareStatement(  
            "delete from accounts where id = ?1");  
        pStatement.setString(1, id);  
        pStatement.executeQuery();  
    } catch (Exception e) { ... }  
    finally {  
        try { if (pStatement != null) pStatement.close(); }  
        catch (Exception e) { ... }  
        try { if (connection != null) connection.close(); }  
        catch (Exception e) { ... }  
    }  
}
```

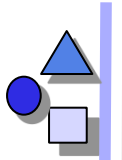
...



Container-Managed Persistency in 2.0

- ▶ TemplateMethod design pattern with generated hook class implementation





Container-Managed Persistent Entity Beans (CMB)

- ▶ The container performs the storage operations
 - This gives a clean separation between the entity bean and its persistent representation
 - The container generates the persistence logic
- ▶ The CMP entity bean is always abstract
 - The container generates a concrete subclass
- ▶ The CMP entity beans have no declared fields
 - Also the get/set method implementations are generated by the container from the deployment descriptor
- ▶ CMP beans get an abstract persistence schema
 - An abstract persistence schema is declared in the deployment descriptor so the container will know what to generate
- ▶ There is a query language, EJB Query Language (EJB-QL)
 - `SELECT OBJECT(a) FROM Account AS a WHERE a.balance > ?1`

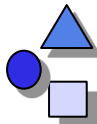


Example: TemplateMethod Pattern

```
import javax.ejb.*;
public abstract class ProductBean implements EntityBean {
    protected EntityContext context;
    public abstract String getName();
    public abstract void setName(String name);
    public abstract String getDescription();
    public abstract void setDescription(String description);
    public abstract double getBasePrice();
    public abstract void setBasePrice(double price);
    public abstract String getProductID();
    public abstract void setProductID(String productID);

    public void ejbActivate() { }
    public void ejbRemove() { }
    public void ejbPassivate() { }
    public void ejbLoad() { }
    public void ejbStore() { }
    public void setEntityContext(EntityContext ctx) { context = ctx; }
    public void unsetEntityContext() { context = null; }
    public void ejbPostCreate(String productID, String name,
        String description, double basePrice) { }
    public String ejbCreate(String productID, String name,
        String description, double basePrice) {
        setProductID(productID);      setName(name);
        setDescription(description);  setBasePrice(basePrice);
        return productID;
    }
}
```

Hook methods



CMP Entity Beans – Deployment Descriptor

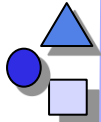
- ▶ You have to declare how the container should generate methods and fields

...declarations of interfaces, etc

```
<cmp-field>
  <field-name>productID</field-name>
</cmp-field>
<cmp-field>
  <field-name>name</field-name>
</cmp-field>
<cmp-field>
  <field-name>description</field-name>
</cmp-field>
<cmp-field>
  <field-name>basePrice</field-name>
</cmp-field>
```

.. queries ...

```
<query>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql>
    <![CDATA(SELECT OBJECT(a) FROM ProductBean AS a WHERE name=?1)]>
  </ejb-ql>
</query>
```



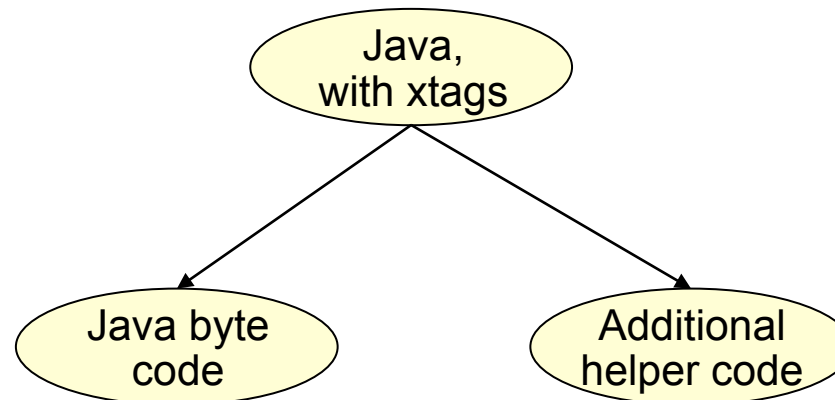
EJB and Others

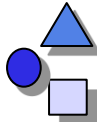
- ▶ Interceptors and Decorators
 - The Interceptor of a bean is like a decorator
 - It can be overwritten and extended from outside the EJB
 - User can write filters for EJB
 - JBoss uses this for aspect-oriented EJB (see later)
- ▶ EJB was formed after Microsoft's MTS (now COM+)
 - COM+ is in .NET
 - Models are somewhat similar
- ▶ Corba Component Model (CCM) is also similar



XDoclets

- ▶ An XDoclet is a plugin into the XDoclet framework
- ▶ The XDoclet framework is a doclet, i.e., a Javadoc extension
- ▶ XDoclets define new tags (xtags), used for metadata
 - Tags can have attribute lists
 - `/* @ejb.bean type = "CMP" name="client" view-type="local" */`
- ▶ Tags steer code generation
 - XDoclet compiler reads the Java source files, evaluates commented tags and generates additional code





Use of Xdoclets in EJB 2.0

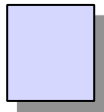
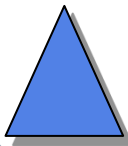
- ▶ Generation of
 - Deployment descriptors
 - Default interfaces
 - Implementation stubs
- ▶ Example [from XDoclet documentation]

```
/** Account
    @see Customer
    @ejb.bean name="bank/Account" type="CMP"
             jndi-name="ejb/bank/Account"
             primkey-field="id"
    @ejb.finder signature="java.util.collection findAll()"
             unchecked="true"
    @ejb.transaction type="required"
    @ejb.interface remote-class="test.interfaces.Account"
    @version 1.5
*/
```

12.4. EJB 3.0

heavily uses metadata markup to generate all dependent interfaces and code

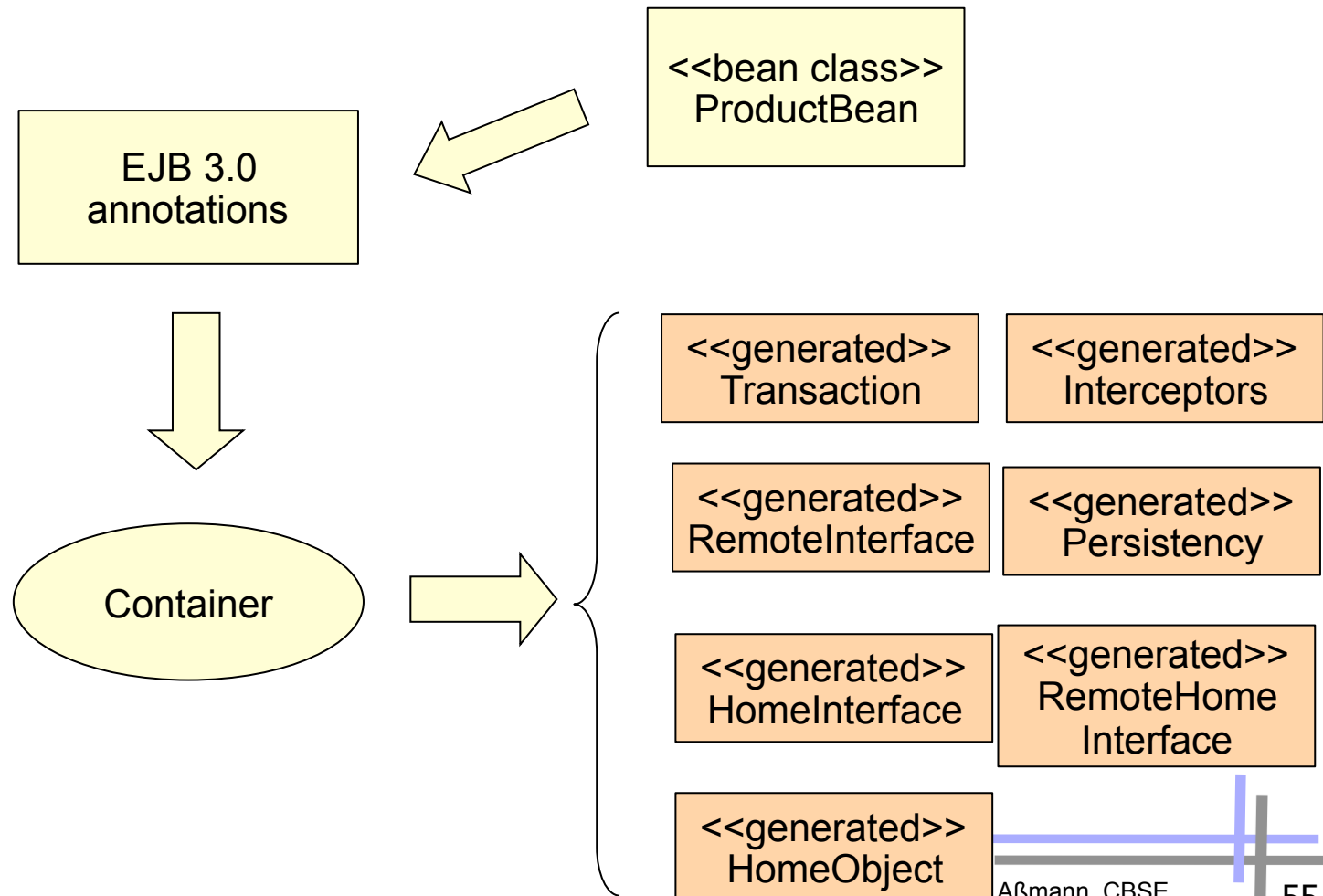
.. Replaces XDoclet





EJB 3.0

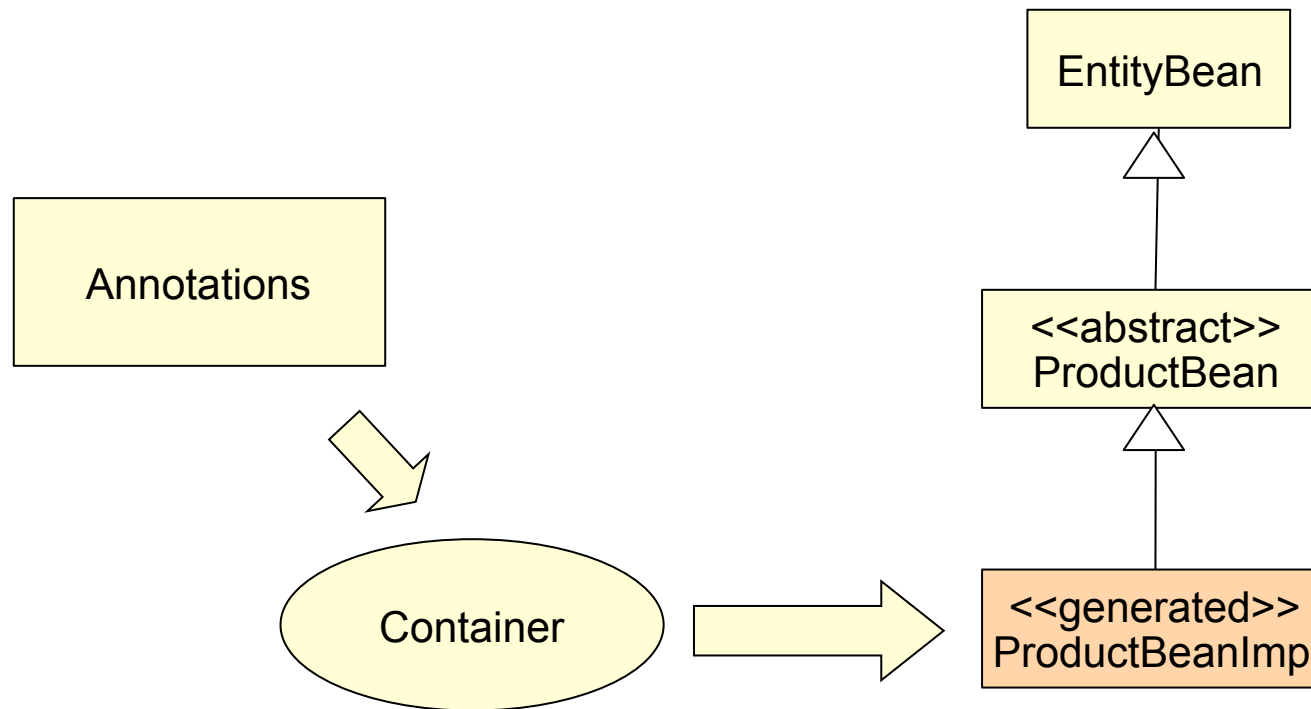
- ▶ Only the bean class is specified
 - Rest of the classes is generated from metadata annotations

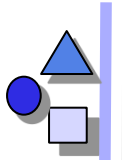




Container-Managed Persistency in 3.0

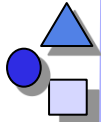
- ▶ TemplateMethod design pattern with generated hook class implementation





Container-Managed Persistent Entity Beans (CMB)

- ▶ The container performs the storage operations
 - This gives a clean separation between the entity bean and its persistent representation
 - The container generates the persistence logic
- ▶ The CMP entity bean is always abstract
 - The container generates a concrete subclass
- ▶ The CMP entity beans have no declared fields
 - Also the get/set method implementations are generated by the container from the deployment descriptor
- ▶ CMP beans get an abstract persistence schema
 - An abstract persistence schema is declared in the deployment descriptor so the container will know what to generate



Marking in EJB 3.0 – Annotation Types

- ▶ Bean class annotations refer to classes and create interfaces with adapters:

```
@Entity
```

```
public class AccountBean implements Account {  
    public void deposit (double money) {...}  
}
```

```
@Stateless
```

```
@Stateful
```

```
@MessageDriven
```

```
@Local
```

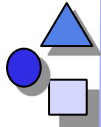
```
@Remote
```

```
@RemoteHome
```

```
@LocalHome
```



From [EJB 3.0 Features]



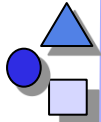
Method Callback Annotations

- ▶ The default methods can be adorned with user-written *filters* (*before*, *after*, and *around fragments* (*advices*))
- ▶ Filter methods are part of Interceptor objects

```
@PrePassivate
void signalPassivation() {
    System.out.println("passivating bean now...");
}
```

[from EJB 3.0 Features]

```
@PreDestroy
/* Callback method defined inside a Listener class*/
@PrePersist
@PostPersist
public class AccountListener{
    @PostPersist
    insertAccountDetails(AccountDetails accountDetails){}
}
@PrePassivate
@PostPassivate
@CallbackListener
```



Custom Interceptors

```
[from EJB 3.0 Features]
// Provides profiling logic in a business method (with interceptors)
/* The interceptor class */
public class ProfilingInterceptor {
    @AroundInvoke // indicates that this is the interceptor method
    public Object profile(InvocationContext invocation) throws Exception {
        long start = System.currentTimeMillis();
        try {
            return invocation.proceed(); // this statement would call the withdraw method
        } finally {
            long time = start - System.currentTimeMillis();
            Method method = invocation.getMethod();
            System.out.println(method.toString() + "took" + time + " (ms)");
        }
    }
}
/* The bean class */
@Stateless
public class BankAccountBean implements BankAccount {
    @PersistenceContext EntityManager entityManager;
    @Interceptors(ProfilingInterceptor.class)
    public void withdraw(int acct, double amount) { ... }
    public void deposit(int acct, double amount) { ... }
}
```



Transaction Control with Attributes

- ▶ Classes and methods may receive transaction attributes
 - **Required:** bean joins the client's transaction
 - **RequiresNew:** bean starts new transaction
 - **NotSupported:** interrupt transaction, execute without it
 - **Supported:** bean joins the client's transaction, otherwise executes without transaction
 - **Mandatory:** bean joins the client's transaction, otherwise signals error

[The Java 2 EE tutorial]

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements Transaction {
    ...
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}
    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}
    public void thirdMethod() {...}
    public void fourthMethod() {...}
}
```

12.5 Evaluation of EJB

as composition system





Component Model

- ▶ Mechanisms for secrets and transparency: very good
 - Interface and implementation repository
 - Location, transaction, persistence transparency
 - Life-time of service hidden, states hidden
 - Communication protocol can be replaced (RMI-IIOP, CORBA-IIOP)
- ▶ (Limited) local parameterization by deployment descriptors
 - The services to use are specified
 - The storage mechanisms for CMP entity beans can be modified
- ▶ Deployment of EJB supported
 - Code generation of stubs
- ▶ Standardization: Good
 - Technical vs. application specific vs. business components
- ▶ EJB 2.0 is quite heavy; 3.0 is slimmer
 - Not a universal technique for everything
 - The goal is to make enterprise systems easier to implement and maintain



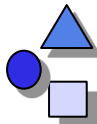
Composition Technique

- ▶ Mechanisms for connection
 - Mechanisms for locating
 - JNDI
 - Mechanisms for adaptation
 - RMI – stubs, skeletons
 - Mechanisms for glueing
 - Container producing glue code
- ▶ Mechanisms for aspect separation
 - Middleware services declared in the deployment descriptor
- ▶ Mechanisms for Meta-modeling
 - with Java reflection and metadata annotations
- ▶ Scalability
 - Pooling ensures scaling

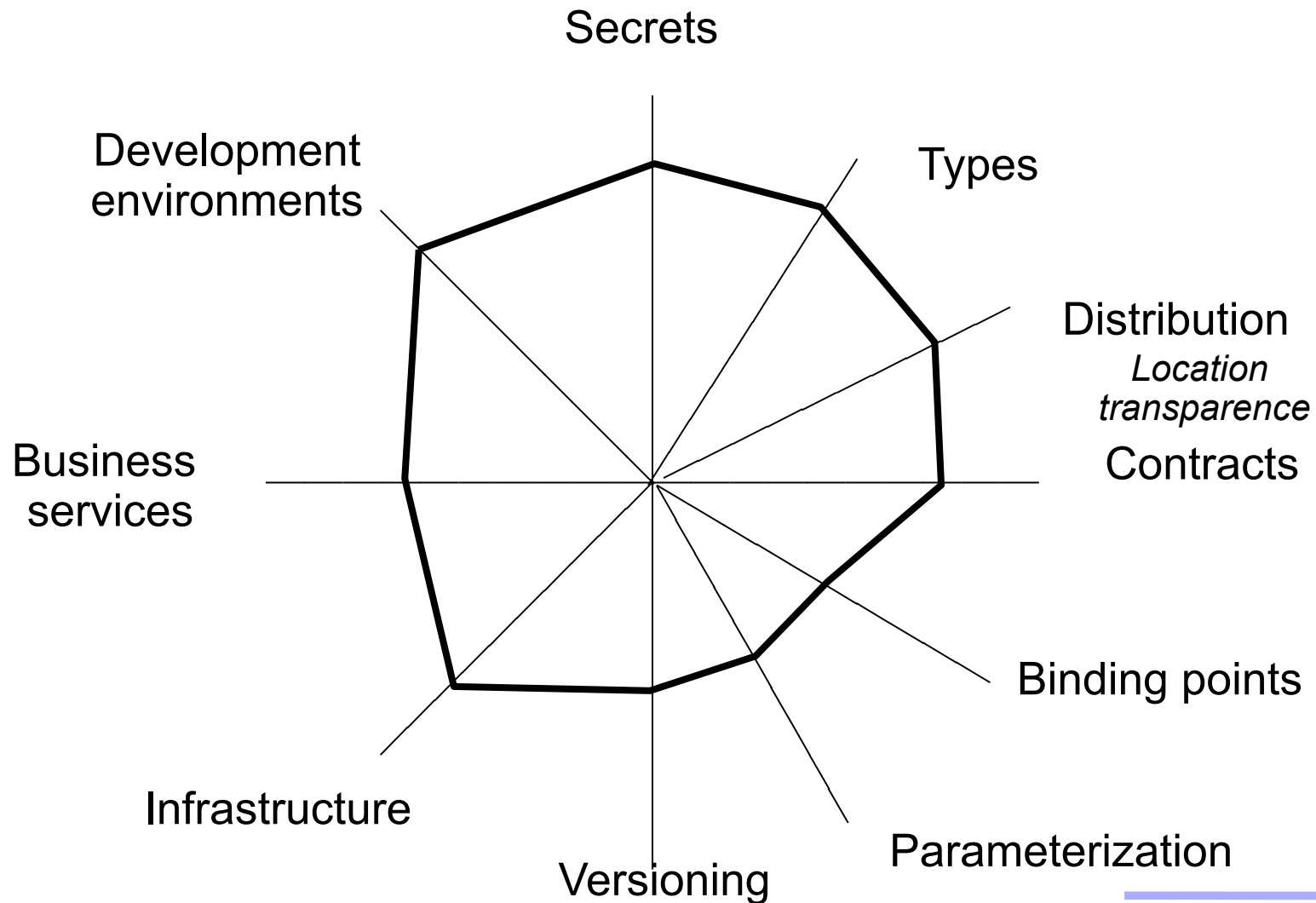


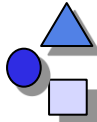
Composition Language

- ▶ The deployment descriptor language is a simple composition language
- ▶ Limited:
 - Glue code is provided by the container
 - Services can be added/removed/modified by changing the deployment descriptor
 - CMP entity beans can be customized by changing the deployment descriptor

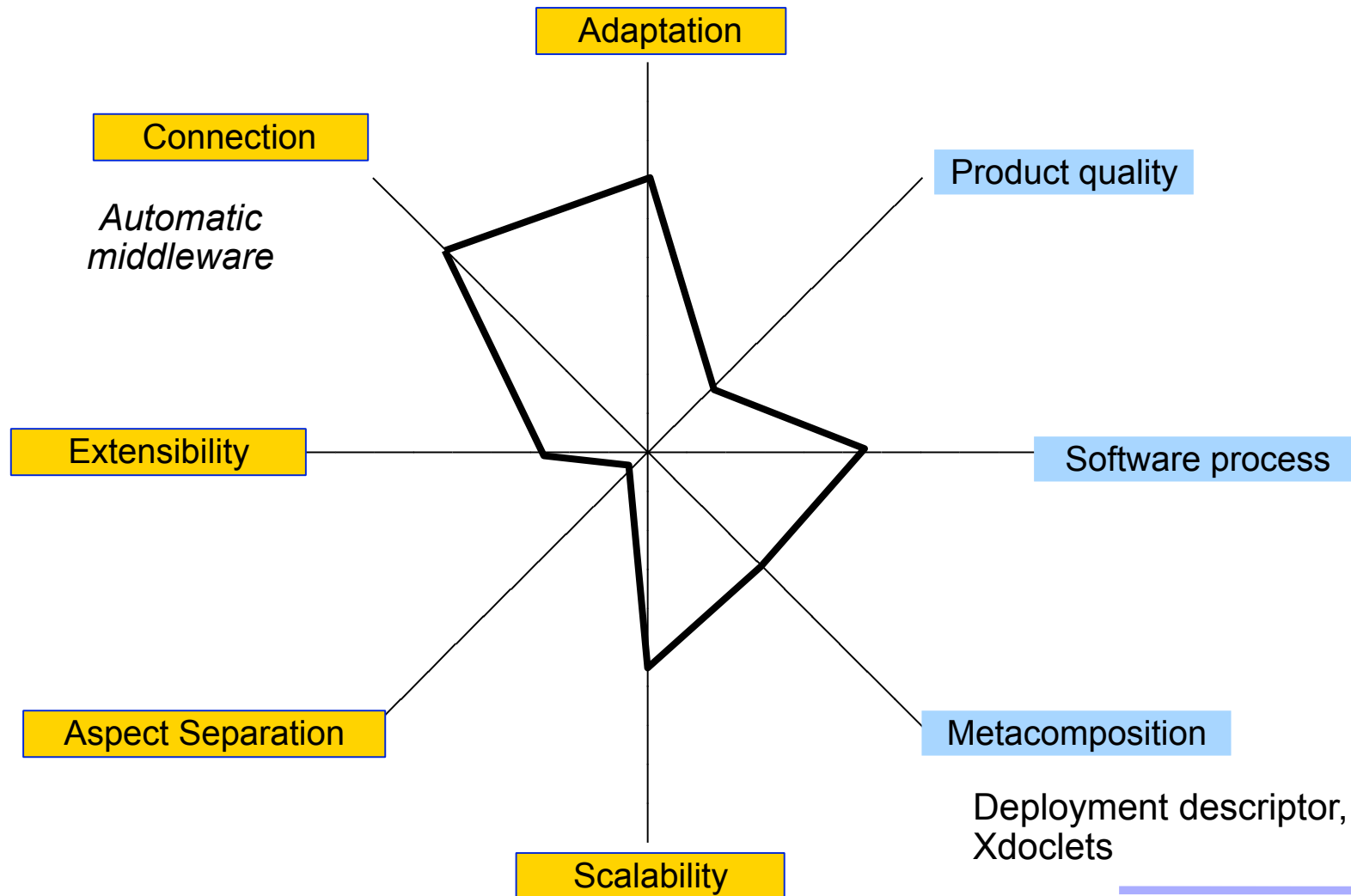


EJB - Component Model



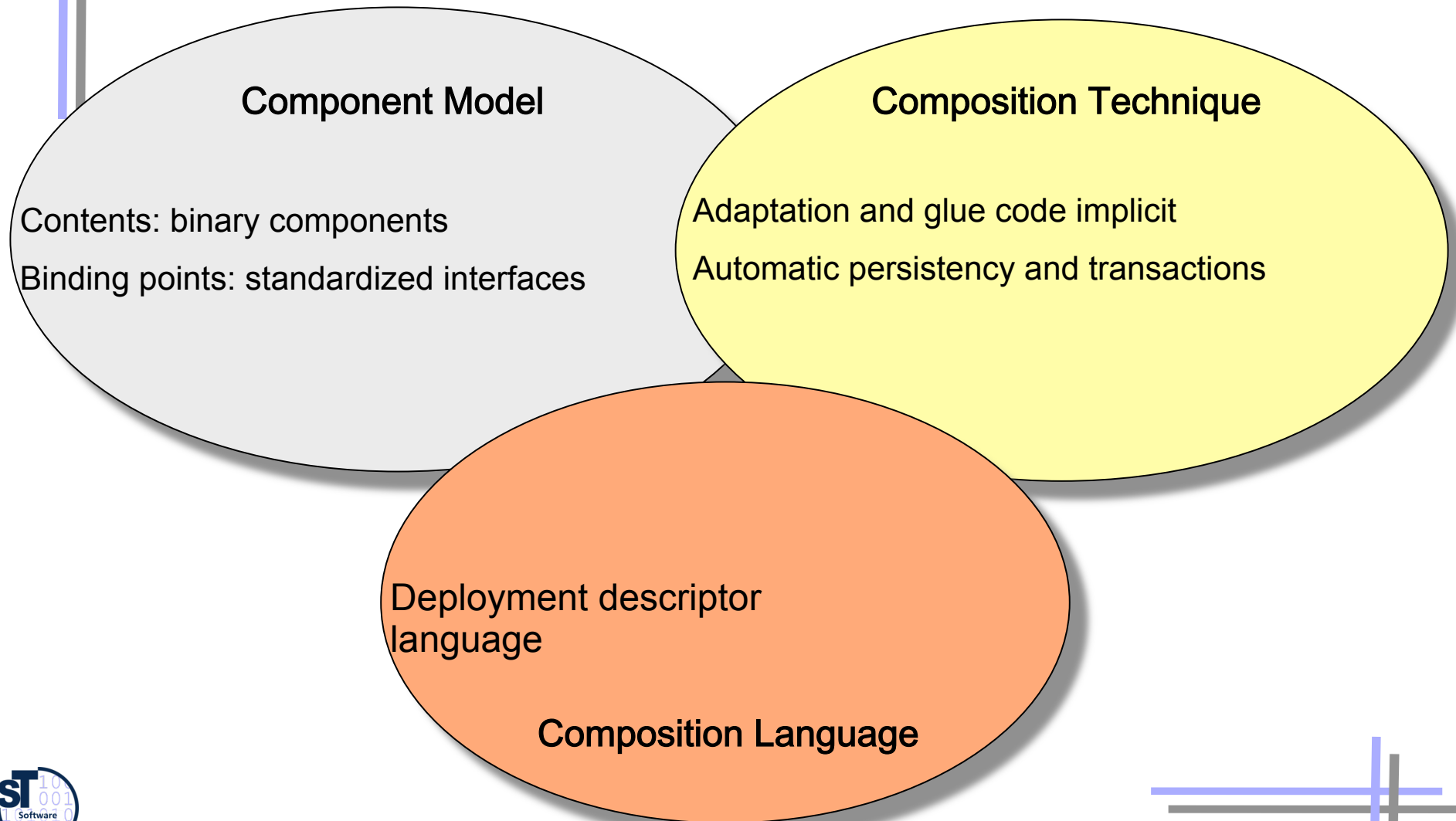


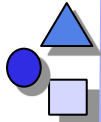
EJB – Composition Technique and Language





EJB as Composition Systems





What Have We Learned

- ▶ EJB is big, not for everything
 - Allows the developer to focus on business logic
 - Provides very useful services, like transparency, persistence, security, networking independence, etc
 - Can interoperate with CORBA
- ▶ It is a well-defined standard by SUN/Oracle
- ▶ It works in symbiosis with several other APIs
 - JNDI, RMI, JDBC, JMS, etc



The End