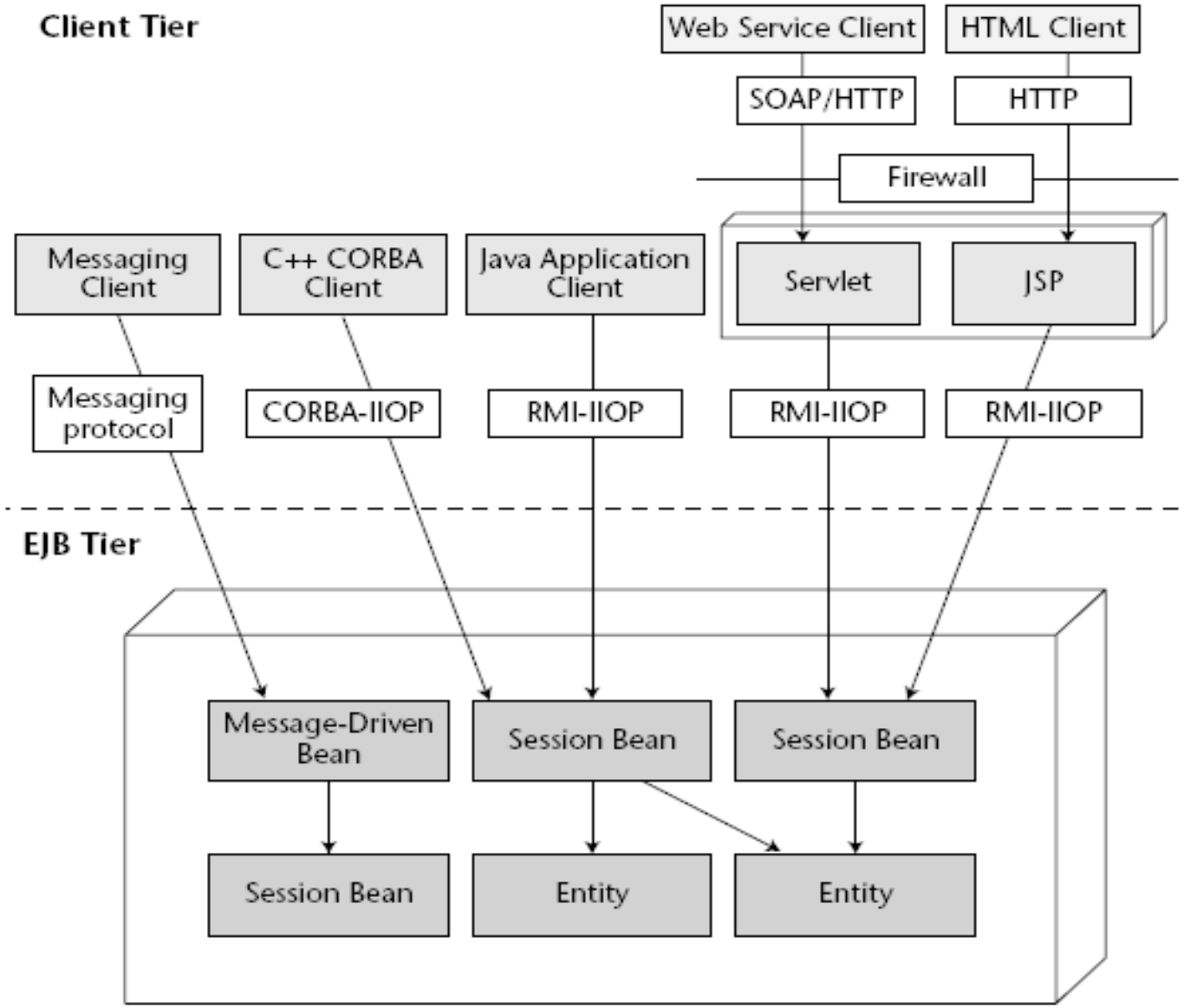
Introduction to Session beans

EJB

Architecture



Stateless session Beans

- **A stateless session bean does not maintain a conversational state for a particular client.**

When a client invokes the method of a stateless bean, the bean's instance variables may contain a state, but only for the duration of the invocation. When the method is finished, the state is no longer retained.

Stateless vs. stateful session Beans

- **All instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client.**

  => Stateless session beans can support multiple clients, and offer better scalability for applications that require large numbers of clients.

  Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.

EJB ingredients

- **Interfaces:** The remote and home interfaces are required for remote access. For local access, the local and local home interfaces are required.

- **Enterprise bean class:** Implements the methods defined in the interfaces.

- **Helper classes:** Other classes needed by the enterprise bean class, such as exception and utility classes.

- **Deployment descriptor:** see later

Remote Interface

```
/**
* This is the HelloBean remote interface.
*
* This interface is what clients operate on when
* they interact with EJB objects. The container
* vendor will implement this interface; the
* implemented object is the EJB object, which
* delegates invocations to the actual bean.
*/
public interface Hello extends javax.ejb.EJBObject
{
/**
* The one method - hello - returns a greeting to the client.
*/
   public String hello() throws java.rmi.RemoteException;
}
```

Must throw RemoteException

Home Interface

```
/**
* This is the home interface for HelloBean. This interface
* is implemented by the EJB Server's tools - the
* implemented object is called the Home Object, and serves
* as a factory for EJB Objects.
*
* One create() method is in this Home Interface, which
* corresponds to the ejbCreate() method in HelloBean.
*/
public interface HelloHome extends javax.ejb.EJBHome
{
/*
* This method creates the EJB Object.
*
* @return The newly created EJB Object.
*/
  Hello create() throws java.rmi.RemoteException,
                        javax.ejb.CreateException;
}
```

Bean Implementation

```
/**
* Demonstration stateless session bean.
*/
public class HelloBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext ctx;
    //
    // EJB-required methods
    //
    public void ejbCreate() { System.out.println("ejbCreate()"); }
    public void ejbRemove() { System.out.println("ejbRemove()"); }
    public void ejbActivate() { System.out.println("ejbActivate()");}
    public void ejbPassivate() {System.out.println("ejbPassivate()");}
    public void setSessionContext(javax.ejb.SessionContext ctx) {
      this.ctx = ctx; }
    //
    // Business methods
    //
    public String hello() {
        System.out.println("hello()");
        return "Hello, World!";
    }
}
```

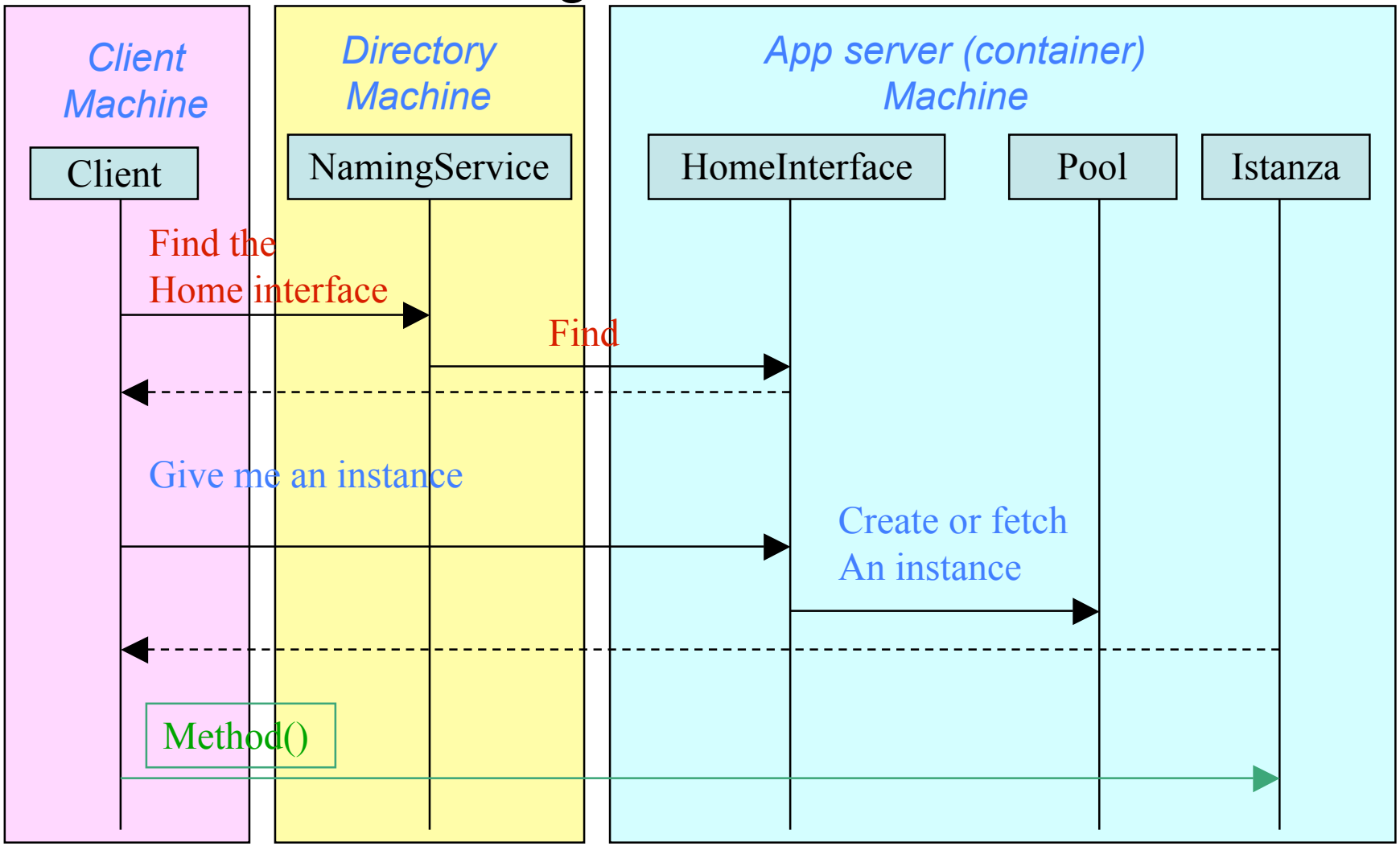
Client Implementation

```
import javax.naming.Context;
import javax.naming.InitialContext;
import java.util.Properties;
/**
* This class is an example of client code that invokes
* methods on a simple stateless session bean.
*/
public class HelloClient {
    public static void main(String[] args) throws Exception {
        /*
        * Setup properties for JNDI initialization.
        * These properties will be read in from the command line.
        */
        Properties props = System.getProperties();
        /*
        * Obtain the JNDI initial context.
        * The initial context is a starting point for
        * connecting to a JNDI tree. We choose our JNDI
        * driver, the network location of the server, etc.
        * by passing in the environment properties.
        */
        Context ctx = new InitialContext(props);
```

Client Implementation

```
        /* Get a reference to the home object - the
        * factory for Hello EJB Objects
        */
        Object obj = ctx.lookup("HelloHome");
        /* Home objects are RMI-IIOP objects, and so they must be cast
        * into RMI-IIOP objects using a special RMI-IIOP cast.
        */
        HelloHome home = (HelloHome)
        javax.rmi.PortableRemoteObject.narrow(obj, HelloHome.class);
        /* Use the factory to create the Hello EJB Object
        */
        Hello hello = home.create();
        /*Call the hello() method on the EJB object. The
        * EJB object will delegate the call to the bean,
        * receive the result, and return it to us.
        * We then print the result to the screen.
        */
        System.out.println(hello.hello());
        /*
        * Done with EJB Object, so remove it.
        * The container will destroy the EJB object.
        */
        hello.remove();
    }
}
```

The logical architecture



Deployment Descriptor

Deployment descriptor: An XML file that specifies information about the bean such as its transaction attributes.

- **You package the files in the preceding list into an EJB JAR file, the module that stores the enterprise bean.**
- **To assemble a J2EE application, you package one or more modules--such as EJB JAR files--into an EAR file, the archive file that holds the application.**

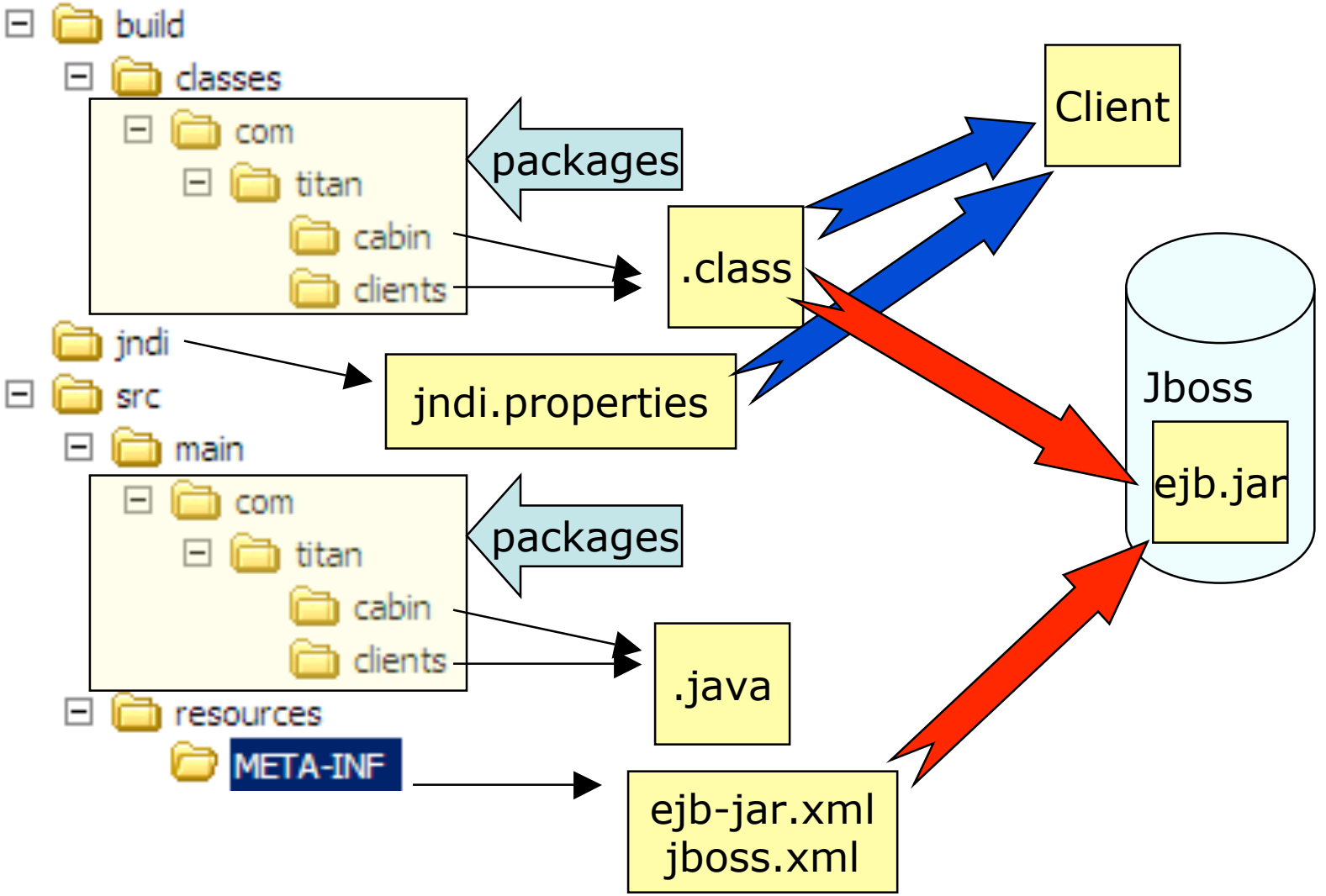
ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
    version="2.1">
    <enterprise-beans>
        <session>
            <ejb-name>HelloWorldEJB</ejb-name>
            <home>examples.ejb21.HelloHome</home>
            <remote>examples.ejb21.Hello</remote>
            <local-home>examples.ejb21.HelloLocalHome</local-home>
            <local>examples.ejb21.HelloLocal</local>
            <ejb-class>examples.ejb21.HelloBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    ...
</ejb-jar>
```

ejb-jar.xml (continued)

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

The file structure



Introduction to Session beans

LOCAL BEANS

Local Interface

```
/**
* This is the HelloBean local interface.
*
* This interface is what local clients operate
* on when they interact with EJB local objects.
* The container vendor will implement this
* interface; the implemented object is the
* EJB local object, which delegates invocations
* to the actual bean.
*/
public interface HelloLocal extends javax.ejb.EJBLocalObject
{
/**
* The one method - hello - returns a greeting to the client.
*/
   public String hello();
}
```

May throw EJBException instead of RemoteException

Local Home Interface

```
/**
* This is the home interface for HelloBean. This interface
* is implemented by the EJB Server's tools - the
* implemented object is called the Home Object, and serves
* as a factory for EJB Objects.
*
* One create() method is in this Home Interface, which
* corresponds to the ejbCreate() method in HelloBean.
*/
public interface HelloLocalHome extends javax.ejb.EJBLocalHome
{
/*
* This method creates the EJB Object.
*
* @return The newly created EJB Object.
*/
  HelloLocal create() throws javax.ejb.CreateException;
}
```

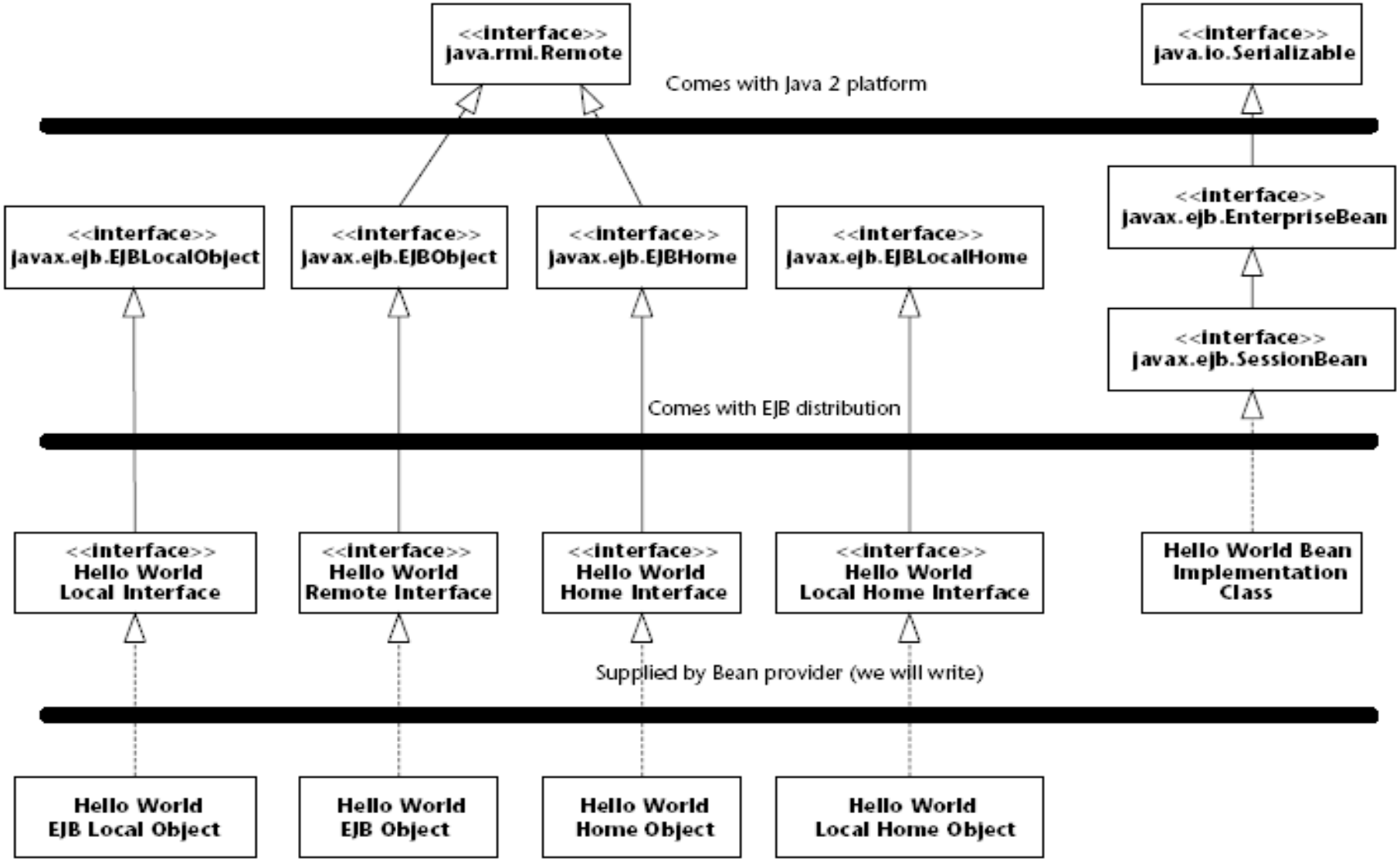
Local Client

```
Object ref = jndiContext.lookup("HelloHome");
HelloHome home = (HelloHome)
PortableRemoteObject.narrow(ref,HelloHome.class);
...
HelloHome cabin_1 = home.create();
```

```
HelloLocalHome home = (HelloLocalHome )
jndiContext.lookup("java:comp/env/ejb/ HelloLocalHome ");
...
HelloLocalHome cabin_1 = home.create();
```

We looked up a bean in *java:comp/env/ejb*.
This is the JNDI location that the EJB specification recommends (but does not
require) you put beans that are referenced from other beans.

Hierarchy of HelloWorld



Introduction to Session beans

EJB 3.0

Remote Interface

```
EJB 2.1 ================================================

public interface Hello extends javax.ejb.EJBObject
{
/**
* The one method - hello - returns a greeting to the client.
*/
   public String hello() throws java.rmi.RemoteException;
}

EJB 3.0 ================================================

package examples.session.stateless;
public interface Hello {
    public String hello();
}
```

business interface

Bean Implementation

```
EJB 2.1 ============================================================
public class HelloBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext ctx;
    public void ejbCreate() { System.out.println("ejbCreate()"); }
    public void ejbRemove() { System.out.println("ejbRemove()"); }
    public void ejbActivate() { System.out.println("ejbActivate()");}
    public void ejbPassivate() {System.out.println("ejbPassivate()");}
    public void setSessionContext(javax.ejb.SessionContext ctx) {
      this.ctx = ctx; }
    public String hello() {
        System.out.println("hello()");  return "Hello, World!";
    }
}
EJB 3.0 ============================================================
package examples.session.stateless;
import javax.ejb.Remote; import javax.ejb.Stateless;
@Stateless
@Remote(Hello.class)
public class HelloBean implements Hello {
    public String hello() {
        System.out.println("hello()"); return "Hello, World!";
    }
}
```

enterprise bean instance

The remote client – 3.0

```
package examples.session.stateless;
import javax.naming.Context;
import javax.naming.InitialContext;
public class HelloClient {
    public static void main(String[] args) throws Exception {
    /*
    * Obtain the JNDI initial context.
    *
    * The initial context is a starting point for
    * connecting to a JNDI tree.
    */
        Context ctx = new InitialContext();
        Hello hello = (Hello)
        ctx.lookup("examples.session.stateless.Hello");
    /*
    * Call the hello() method on the bean.
    * We then print the result to the screen.
    */
        System.out.println(hello.hello());
    }
}
```

ejb-jar.xml – 3.0

```
<?xml version="1.0" encoding="UTF-8" ?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_3_0.xsd"
version="3.0">
<enterprise-beans>
</enterprise-beans>
</ejb-jar>
```

Keep in mind these terms…

- The ***enterprise bean instance*** is a plain old Java object instance of an enterprise bean class. It contains business method implementations of the methods defined in the remote/local business interface, for session beans.

- The ***business interface*** is a plain old Java interface that enumerates the business methods exposed by the enterprise bean. Depending on the client view supported by the bean, the business interface can be further classified into a local business interface or a remote business interface.

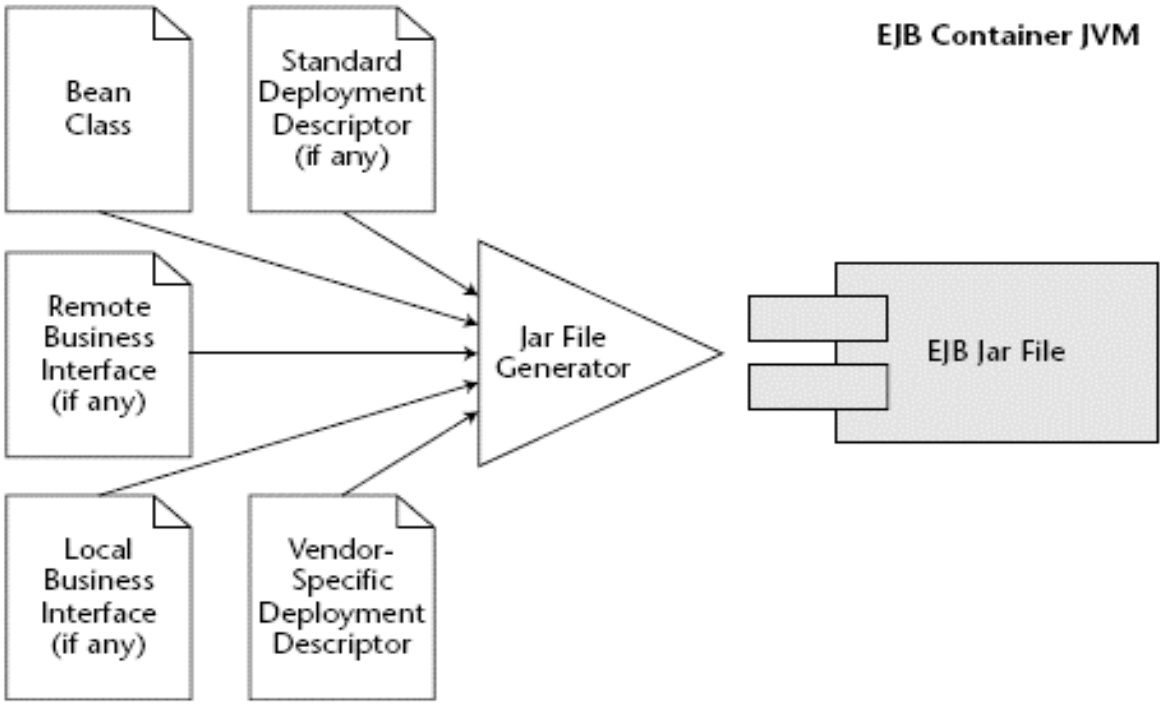
- The ***deployment descriptor*** is an XML file that specifies the middleware requirements for your bean. You use the deployment descriptor to inform the container about the services you need for the bean, such as transaction services, security, and so on. Alternatively, you can specify the middleware requirements using deployment annotations within the bean class as well.

Keep in mind these terms…

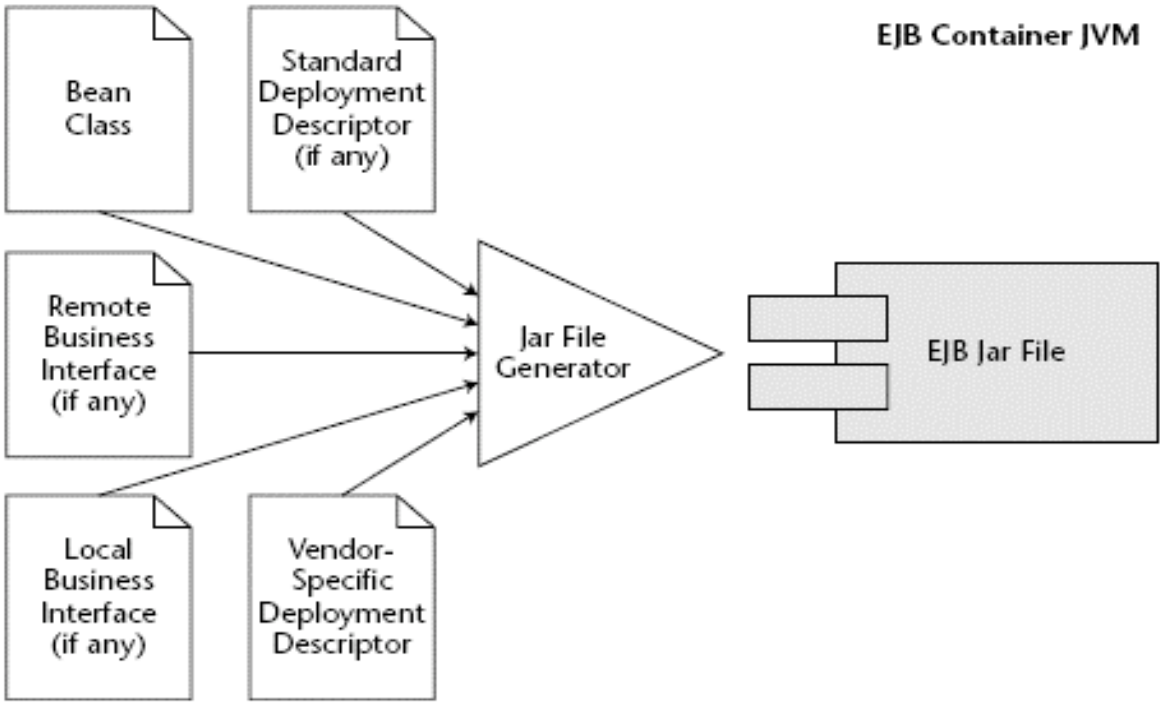
- The ***Ejb-jar*** file is the packaging unit for an enterprise bean, consisting of all the above artifacts. An EJB 3.0 Ejb-jar file can also consist of the old-style beans, if your application uses components defined using pre–EJB 3.0 technologies.

- The ***vendor-specific deployment descriptor*** lets you specify your bean's needs for proprietary container services such as clustering, load balancing, and so on. Avendor can alternatively provide deployment metadata for these services, which, like standard metadata, can be used within the bean class to specify the configuration for these services. The vendor-specific deployment descriptor's definition changes from vendor to vendor.

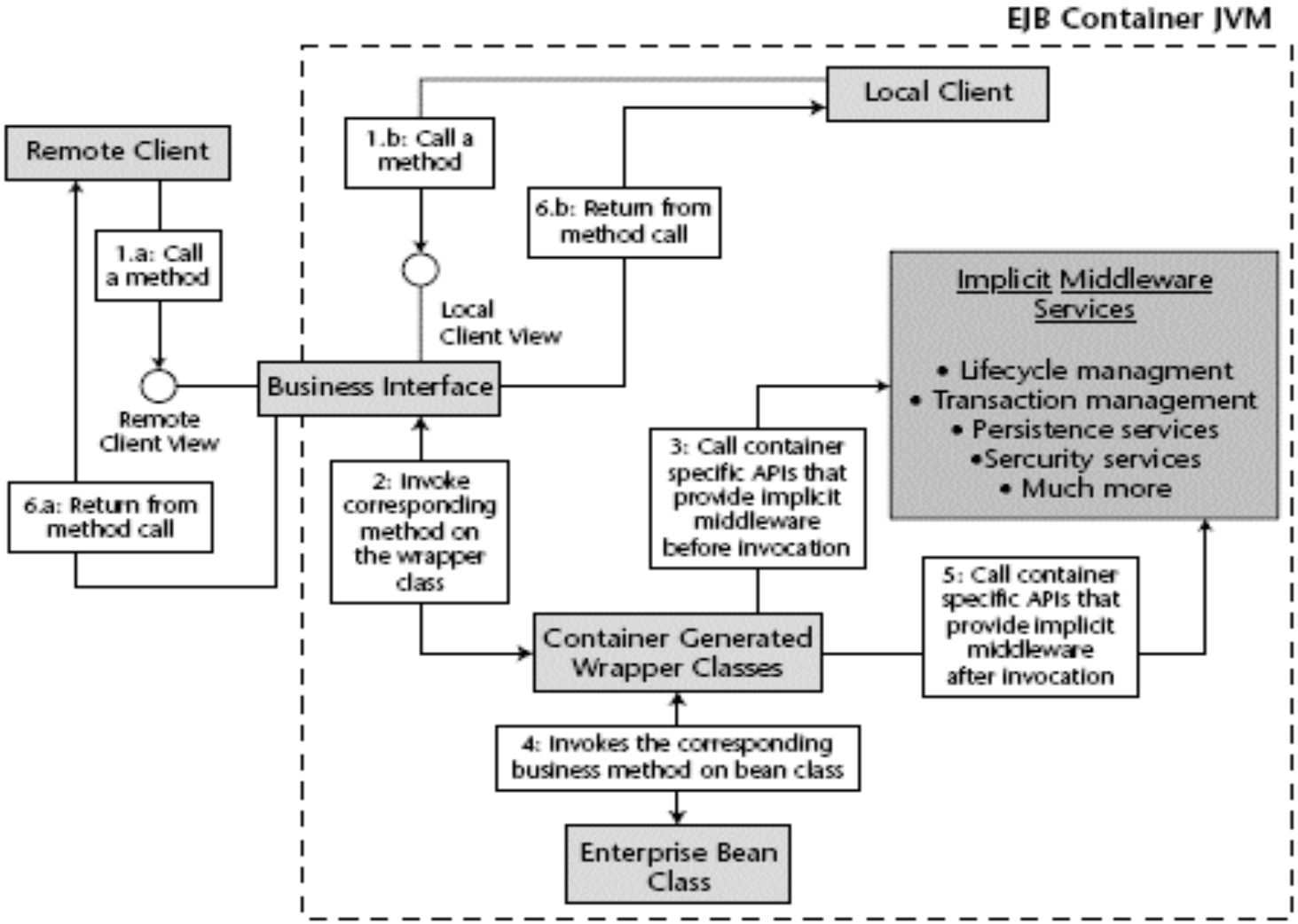
3.0 Packaging



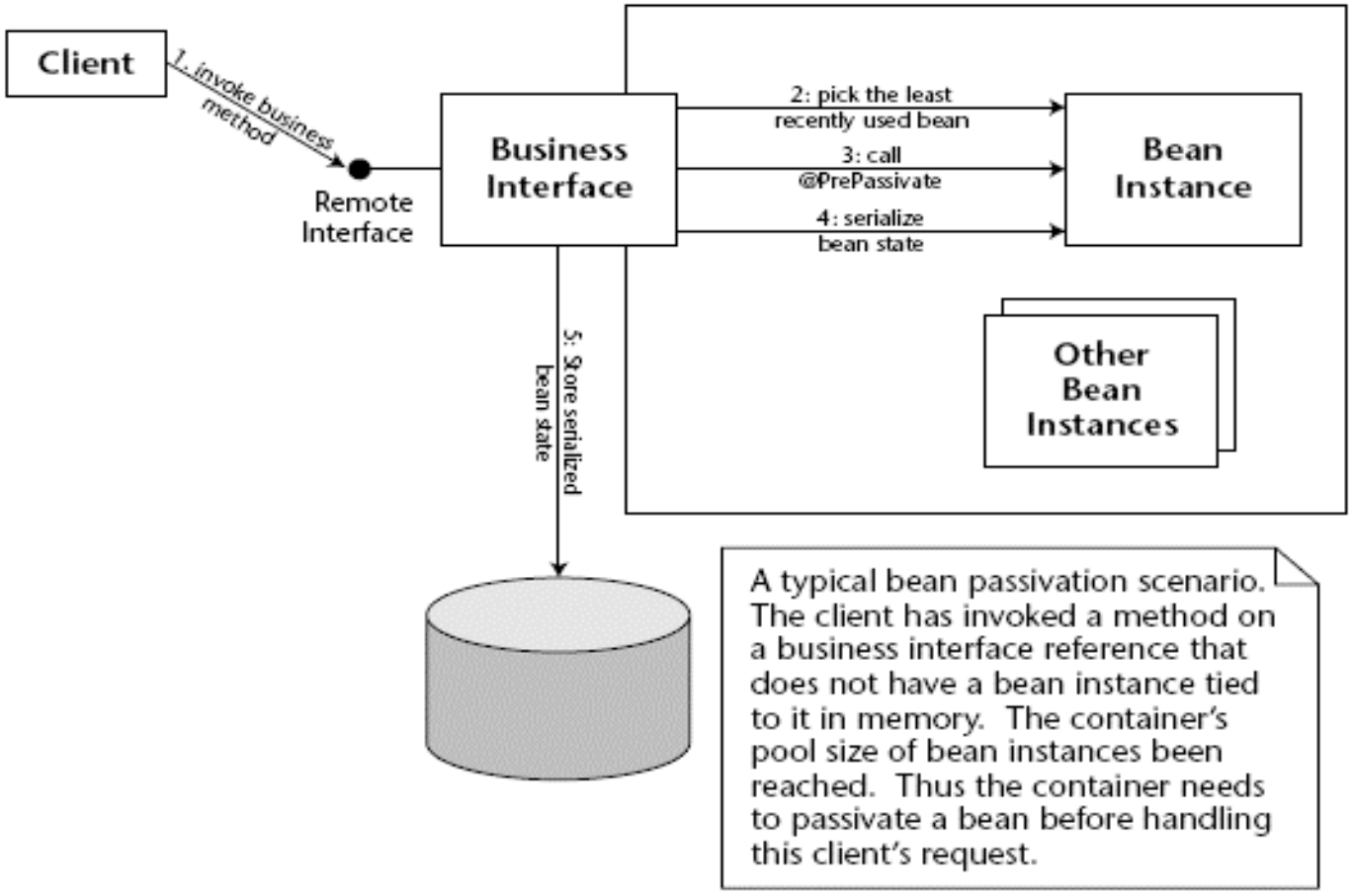
3.0 Packaging



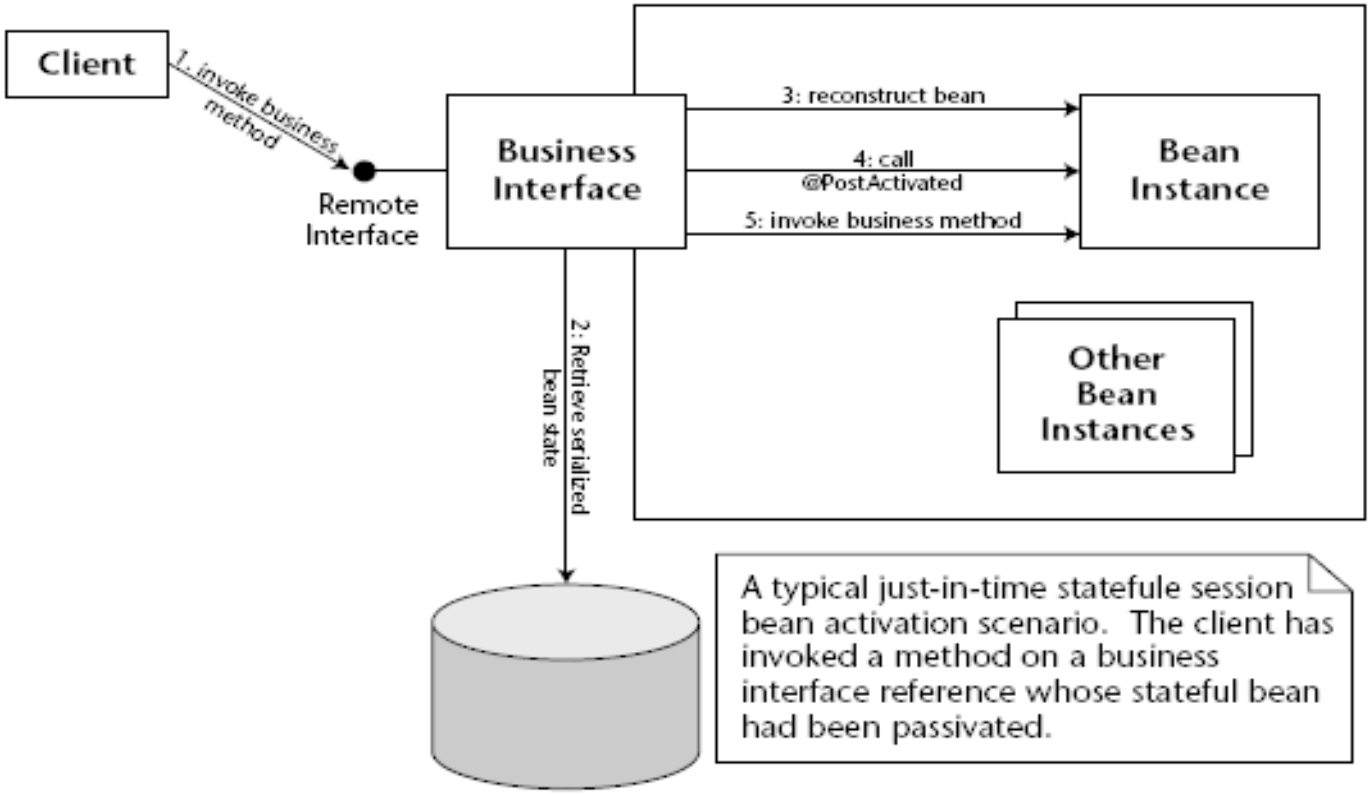
3.0 Lifecycle



Passivation



Activation


Client
1. invoke business method
Remote Interface
Business Interface
3: reconstruct bean
4: call @PostActivated
5: invoke business method
Bean Instance
Other Bean Instances
2: Retrieve serialized bean state
A typical just-in-time statefule session bean activation scenario. The client has invoked a method on a business interface reference whose stateful bean had been passivated.

Managing the lifecycle – 3.0

```
@Stateful
public class MyBean {
@PrePassivate
        public void passivate() {
        <close socket connections, etc...>
        }
        ...
@PostActivate
        public void activate() {
        <open socket connections, etc...>
        }
        ...
}
```

JBOSS and NetBeans

Download Jboss 4.2.1

http://labs.jboss.com/jbossas/downloads/

Download Netbeans 6.0 beta

http://www.netbeans.org/community/releases/60/

Setting the JNDI properties

```
private HelloBeanRemote lookupHelloBeanBean() {
    Properties props= new Properties();
    props.setProperty("java.naming.factory.initial",
        "org.jnp.interfaces.NamingContextFactory");
    props.setProperty("java.naming.provider.url",
        "jnp://localhost:1099");
    props.setProperty("java.naming.factory.url.pkgs",
        "org.jboss.naming:org.jnp.interfaces");
   try {
     Context c = new InitialContext(props);
     //return (HelloBeanRemote) c.lookup("java:comp/env/HelloBeanBean");
     return (HelloBeanRemote) c.lookup("EnterpriseDemo/HelloBeanBean/remote");
   } catch (NamingException ne) {
     …
  }
 }
```

JBOSS: see the JNDI names

http://localhost:8080/jmx-console/HtmlAdaptor

Click on Service=JNDI

Choose List