Introduction to Session beans

EJB - continued

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();
```

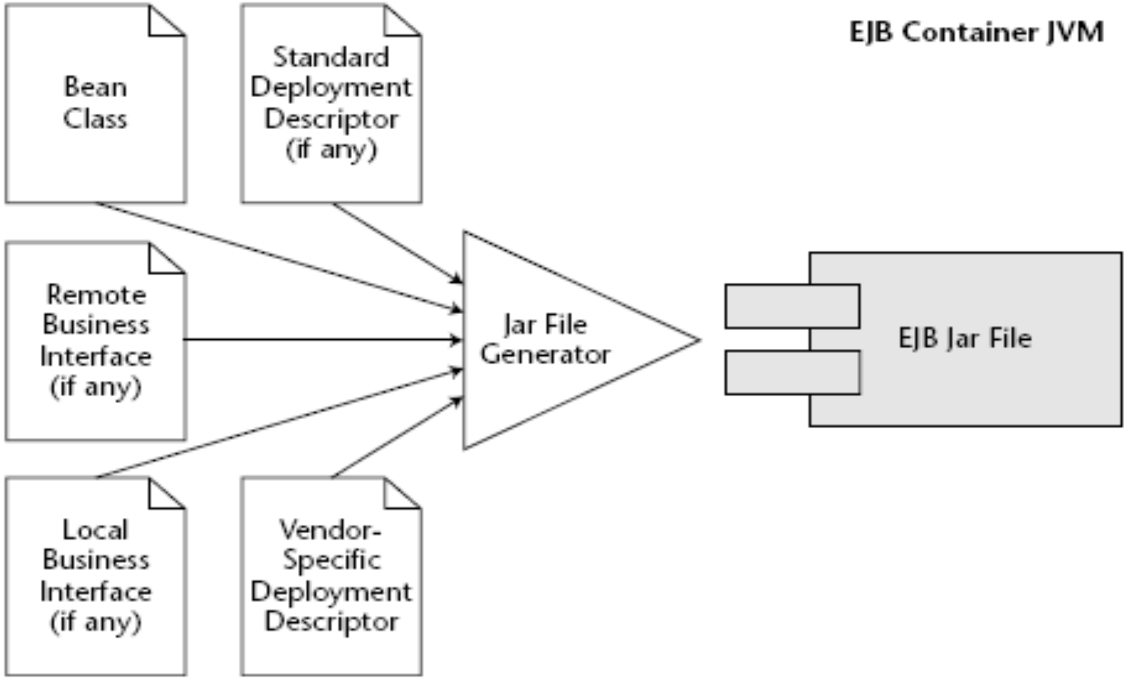
We looked up a bean in *java:comp/env/ejb*.
This is the JNDI location that the EJB specification recommends
(but does not
require) you put beans that are referenced from other beans.

Bean Implementation

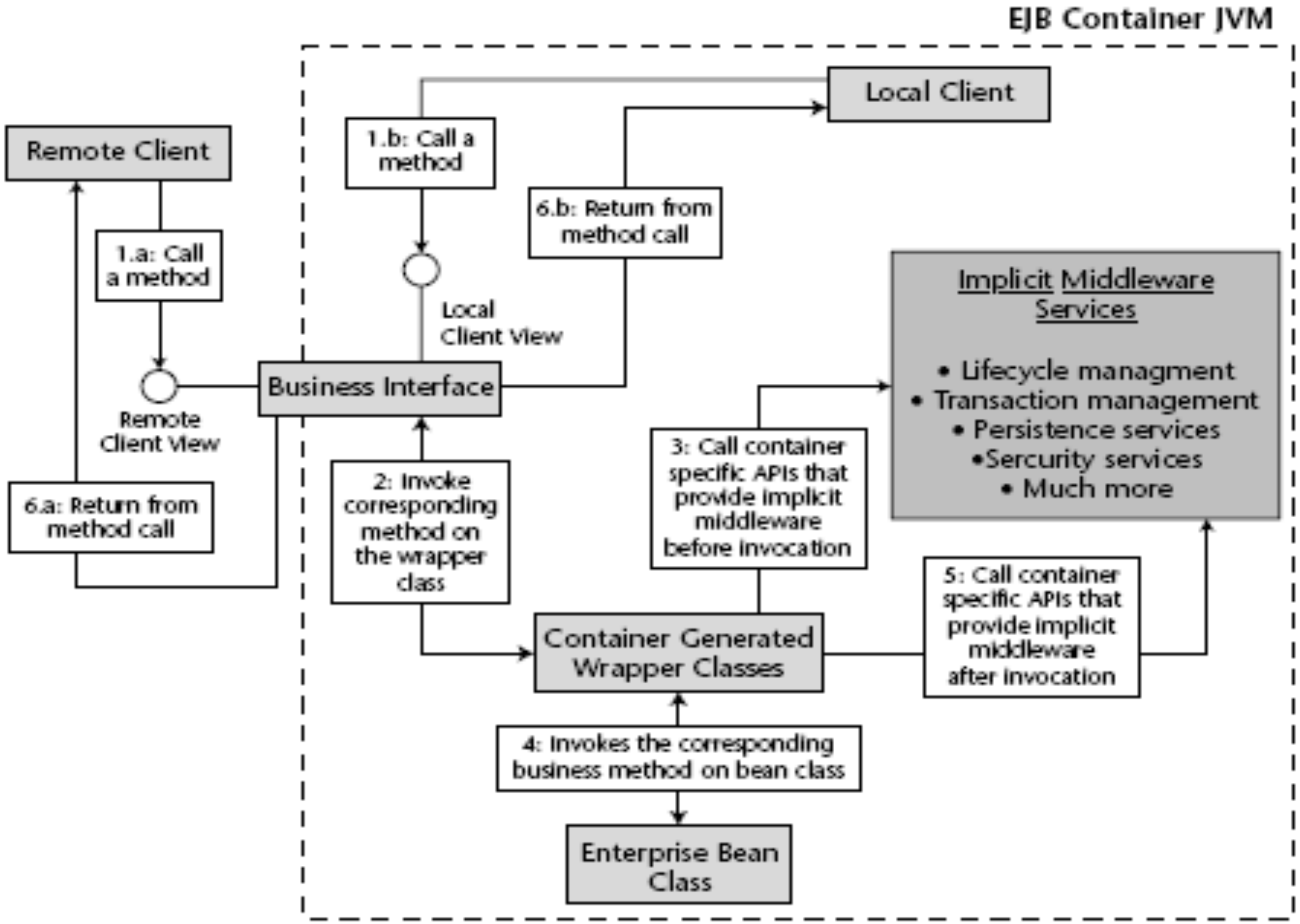
```
EJB 3.0 ==============================================================
package examples.session.stateless;
import javax.ejb.Local; import javax.ejb.Stateless;
@Stateless
@Local(Hello.class)
public class HelloBean implements Hello {
    public String hello() {
        System.out.println(“hello()”); return “Hello, World!”;
    }
}
```

enterprise bean instance

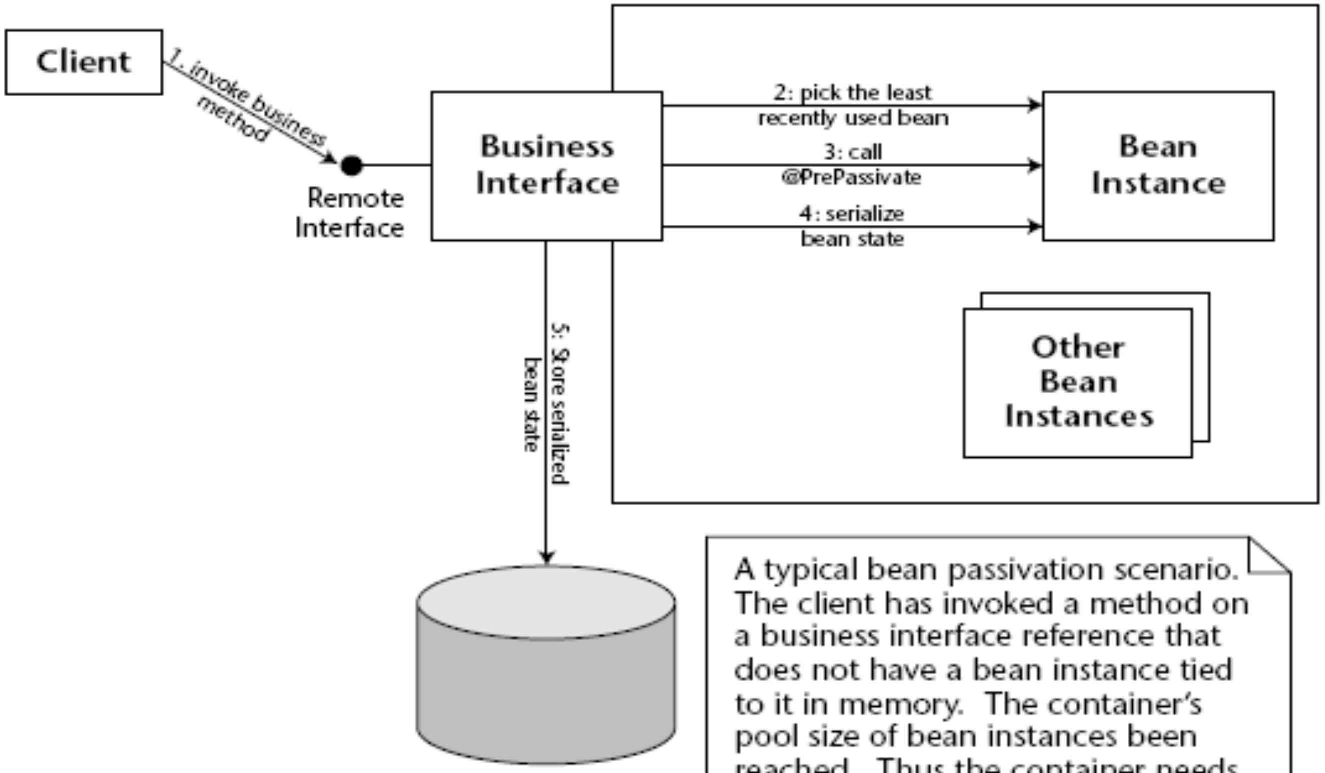
3.0 Packaging


Bean Class
Standard Deployment Descriptor (if any)
EJB Container JVM
Remote Business Interface (if any)
Jar File Generator
EJB Jar File
Local Business Interface (if any)
Vendor-Specific Deployment Descriptor

3.0 Lifecycle

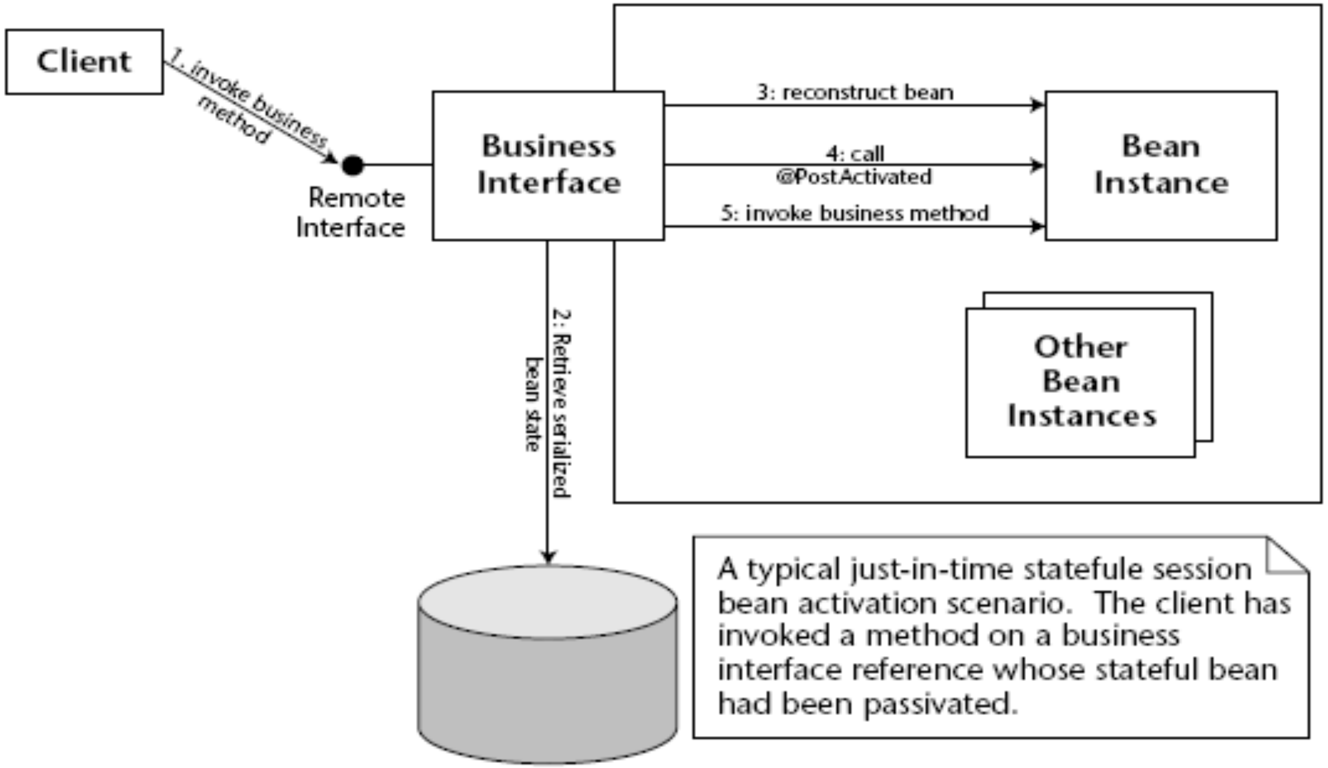

EJB Container JVM
Local Client
Remote Client
1.b: Call a method
6.b: Return from method call
1.a: Call a method
Local Client View
Implicit Middleware Services
• Lifecycle managment
• Transaction management
• Persistence services
•Sercurity services
• Much more
Business Interface
Remote Client View
2: Invoke corresponding method on the wrapper class
3: Call container specific APIs that provide implicit middleware before invocation
6.a: Return from method call
5: Call container specific APIs that provide implicit middleware after invocation
Container Generated Wrapper Classes
4: Invokes the corresponding business method on bean class
Enterprise Bean Class

Passivation



A typical bean passivation scenario. The client has invoked a method on a business interface reference that does not have a bean instance tied to it in memory. The container's pool size of bean instances been reached. Thus the container needs to passivate a bean before handling this client's request.

Activation



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...>
        }
        ...
}
```