Distributed Development Approach for Java Based Enterprise Services in SAP NetWeaver

Applies to:

SAP NetWeaver Composition Environment (CE 7.1) and above.

For more information, visit the SOA Management homepage.

Summary

This article describes an approach for team oriented development of business services implemented as Enterprise Java Beans in SAP NetWeaver. It is assumed that the reader is aware of the basic EJB 3.0 concepts.

Author: Balakrishnan.B

Company: Incture Technologies

Created on: 18 May 2009

Author Bio

Balakrishnan is presently working as a Technical Lead and he is specifically affected with design and architectural issues around the applications developed on SAP NetWeaver platform. He has profound experience in engineering enterprise scale composite applications using the key NetWeaver components (*BPM, BRMS, CAF – Core & GP, EP, WebDynpro, VC*) and open Java EE 5.0 standards (*EJB 3.0, JAX – WS 2.0, JPA 1.0*). His current areas of work include Innovative Product Engineering, Scalability and Performance Optimization.

Table of Contents

Preface

For application developers focusing on developing ESOA service interfaces using EJB 3.0, often the question on how to go about with a distributed model for the development of business services crops up. Well, the obvious answer is to use NWDI as the development infrastructure. That's right but then what the module level dependencies are and how to define it? This article is aimed at providing answers to these questions.

Creating EJB Module(s)

Let's start with the creation of EJB Modules. Under the required development configuration / software component create two DCs of type "*EJB Module*" with suitable names say *ejb/module/one* and *ejb/module/two*. Let's assume that a developer (developer A) has developed an exemplary and fictional re-usable stateless session bean (SSB) under *ejb/module/one* that needs to be invoked from with in another SSB under *ejb/module/two*, of course using *Inversion Of Control | Dependency Injection*.

Let's assume that the DC *ejb/module/one* developed by developer A has the following structure with one domain object definition (*TestObjectOne.java*) and one stateless session bean (*BeanOne.java*) with the following implementation.

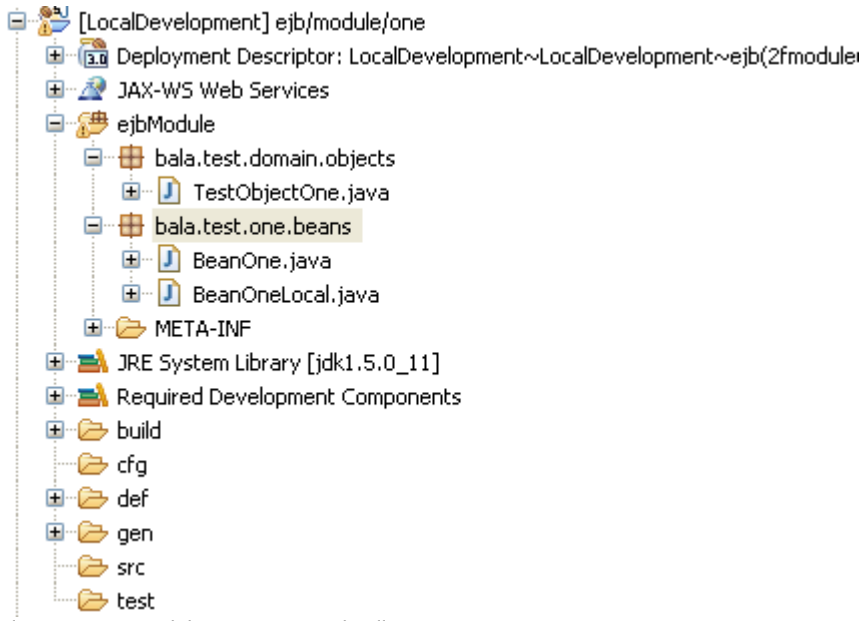

Figure 1: EJB Module DC structure details

//BeanOne.java

```
package bala.test.one.beans;

import javax.ejb.Stateless;

import bala.test.domain.objects.TestObjectOne;


@Stateless
public class BeanOne implements BeanOneLocal {
      public void method1(TestObjectOne one) {
            System.err.println("Invoked method1 of BeanOne.");
            System.err
                        .println("Value of the object passed from method2
of BeanTwo is: "
                                    + one.getValue());
      }
}
```

//BeanOneLocal.java

```
package bala.test.one.beans;

import javax.ejb.Local;

import bala.test.domain.objects.TestObjectOne;

@Local
public interface BeanOneLocal {
      public void method1(TestObjectOne one);
}
```

//TestObjectOne.java

```
package bala.test.domain.objects;

import java.io.Serializable;

public class TestObjectOne implements Serializable {

private static final long serialVersionUID = 6953881962079640726L;

String value;

public String getValue() {
      return value;
}

public void setValue(String one) {
      this.value = one;
}


    }
```

Now, for the developer B to use this service in the DC *ejb/module/two,* suitable entities (classes) have to exposed from *ejb/module/one* in the form of public parts. Under the *Component Properties* view of the DC *ejb/module/one,* add the classes (`TestObjectOne`, `BeanOne` and `BeanOneLocal`) to the default public part *client* which is of type compilation.

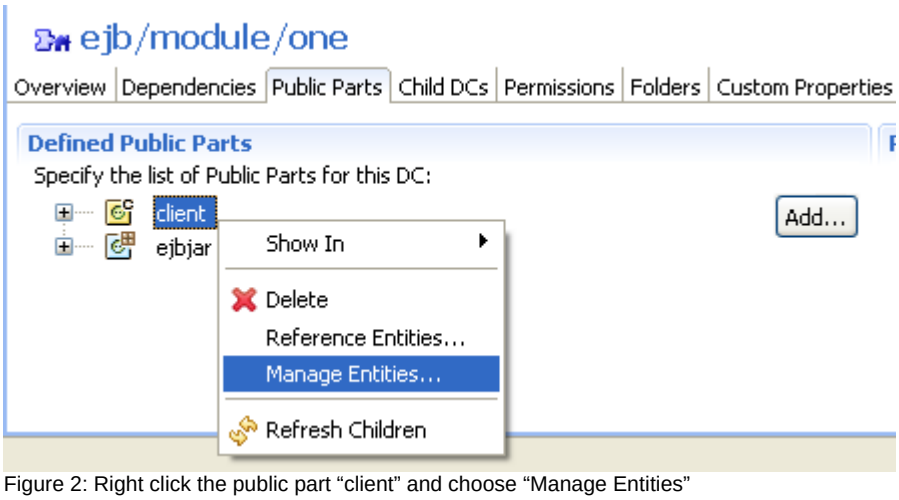

Figure 2: Right click the public part "client" and choose "Manage Entities"

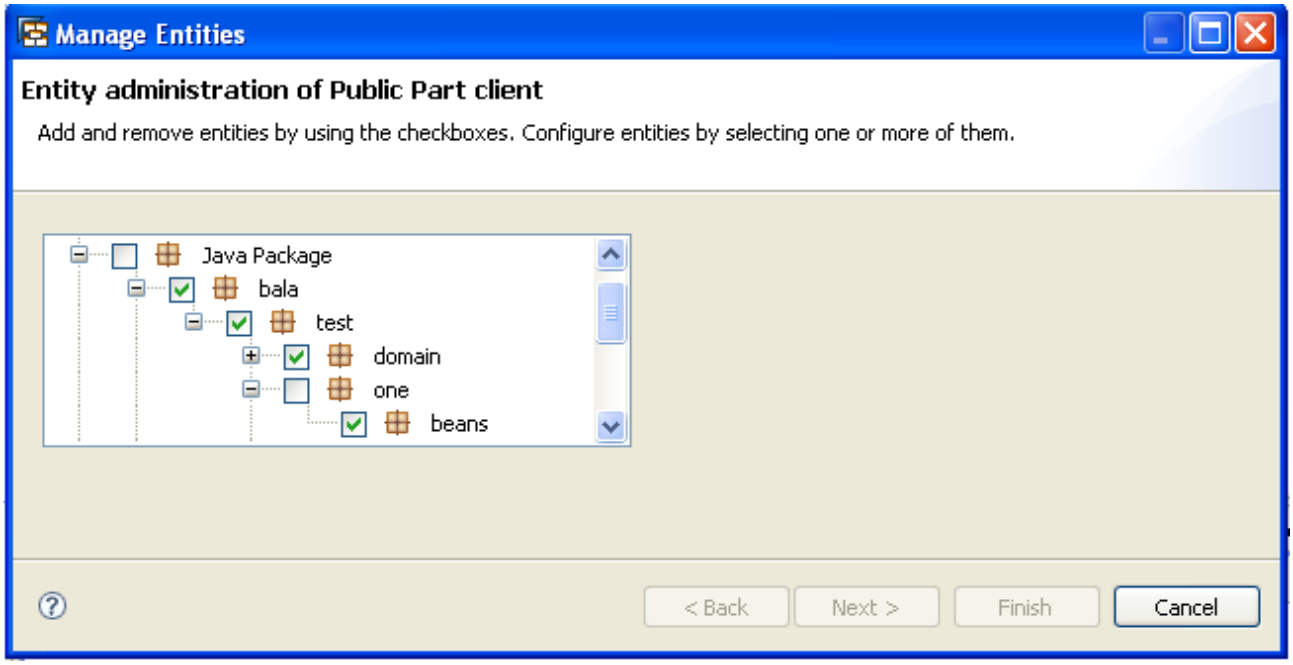

Figure 3: Select the entities (packages) that are to be used in the service implementation

A DC build is required after adding the entities to the public part. With this, the services defined under *ejb/module/one* can be readily consumed in the DC developed by developer B. Let's assume that the DC *ejb/module/two* has the below mentioned structure (Figure 4.a). For the operations exposed by *BeanOneLocal* to be accessible in *BeanTwo* the exposed public part definition should be added as a dependency to the *ejb/module/two* as shown in Figure 4.b below.

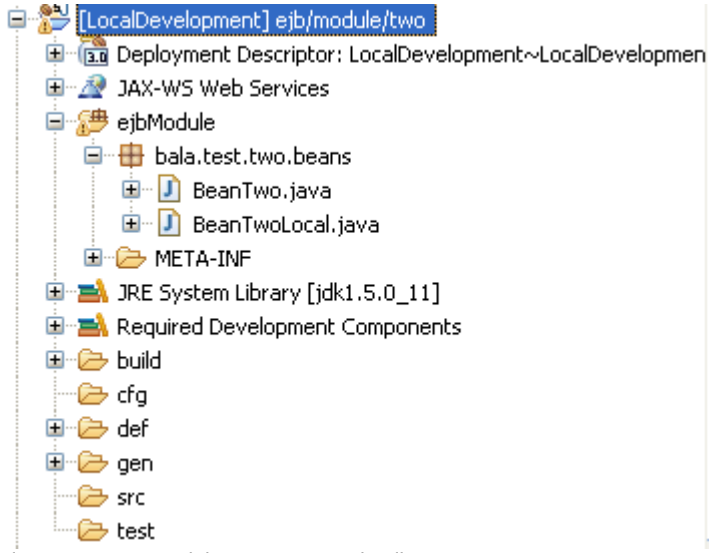

Figure 4.a: EJB Module DC structure details

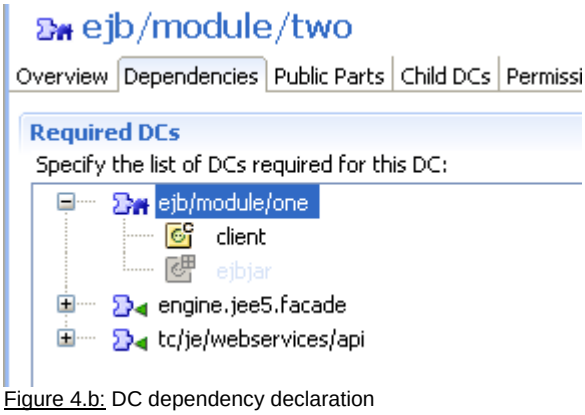

Figure 4.b: DC dependency declaration

Let's assume that developer B wants to consume the re-usable operation *method1* in the session bean *BeanTwo* with the following implementation. For readability reasons, all the example implementations shown in the article have been kept as simple as possible.

```
package bala.test.two.beans;

import javax.ejb.EJB;
import javax.ejb.Stateless;

import bala.test.domain.objects.TestObjectOne;
import bala.test.one.beans.BeanOneLocal;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;


@WebService(name="BeanTwo", portName="BeanTwoPort", serviceName="BeanTwoService",
targetNamespace="http://test.bala/two/beans/")
@Stateless
public class BeanTwo implements BeanTwoLocal {
   @EJB
   BeanOneLocal beanOne;
```

```
    @WebMethod(operationName="method2", exclude=false)
    public void method2(@WebParam(name="o")
    TestObjectOne o) {
          System.err
                      .println("Trying to invoke method1 on injected bean
BeanOne.");
          beanOne.method1(o);
    }
}
```

Please note that this service has been web service enabled to allow easy testing of the business interfaces.

Creating Enterprise Application Module

So far so simple and straight forward… The details were almost "*no brainer*" for the services developer. However, the key point now is to deploy the EJB Modules in the right way so that the runtime injection of the declared dependent EJBs works with out any problems. Here comes the importance of using "*Only one application DC"* for deploying both the modules. The image shown below depicts the same.

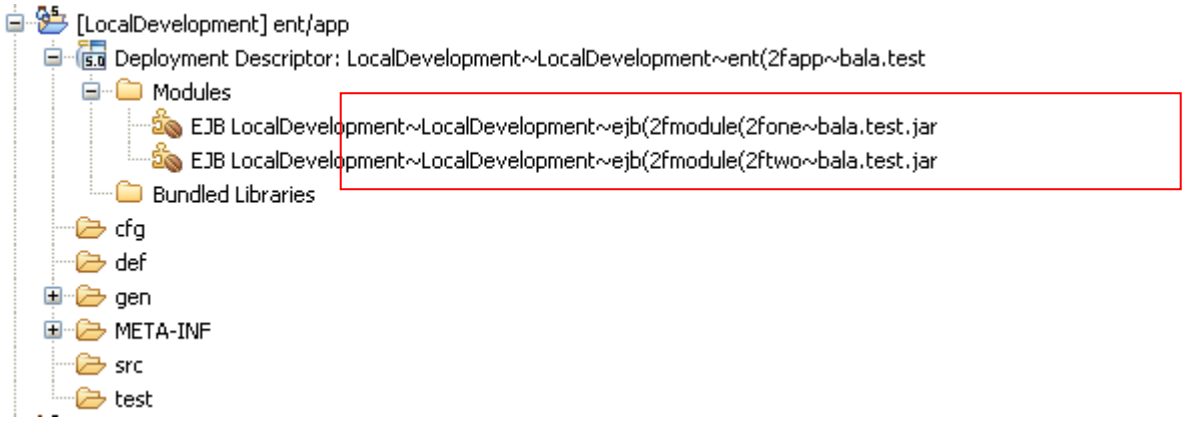

Figure 5: Application DC structure details

Note: If two different applications one each for of the EJB modules *ejb/module/one* and *ejb/module/two* are used then a *java.lang.NoClassDefFoundError:* will occur during the runtime as the relevant class loader will not be able to load the suitable classes required for the services execution.

Once deployed the services (operation *method2*) can be directly executed / tested from web services navigator (http://server:port/wsnavigator). Since the operation simply logs certain messages in the server side, to check the correctness of the execution, relevant messages could be located in Log Viewer. (http://server:port/nwa/logs)

Figure 6: The messages logged by EJB Modules on successful execution.

Related Content

Service Implementation in Java

EJB 3.0: Interceptors and Callbacks Made Easy – Part I

EJB 3.0: Interceptors and Callbacks Made Easy – Part II

For more information, visit the SOA Management homepage.

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.