DOSGi Spring-DM Demo page

This page describes the CXF Distributed OSGi with Spring-DM demo.

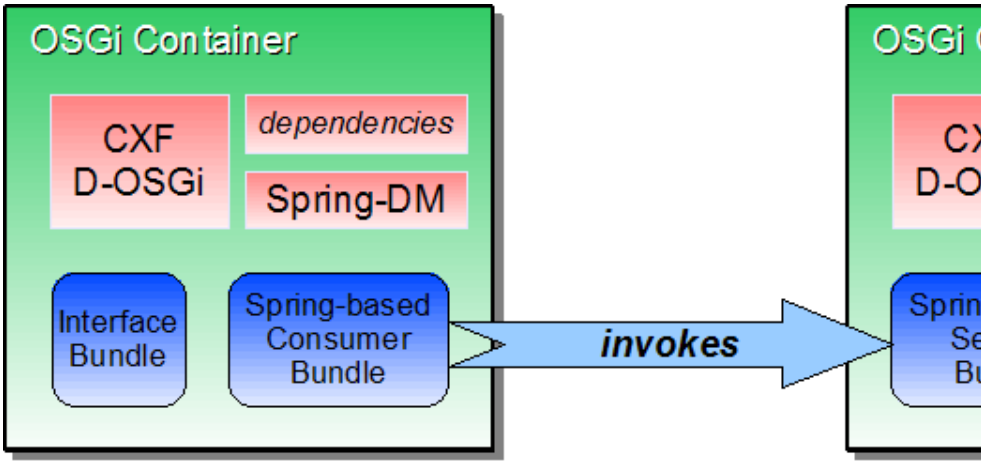
The Spring-DM demo uses Spring-DM to create a remoted OSGi service from a Spring Bean. The consumer side uses Spring-DM to create a consumer to a remote OSGi service. By using Spring-DM you don't need to write code to interact with the OSGi Service Registry. That's all handled through injection which hugely simplifies the code. Besides, you can use all the features of Spring together with OSGi.

It uses the CXF/DOSGi multi bundle distribution since that provides Spring-DM as a dependency. By installing the multi bundle distribution, you have all the prerequisites to run this demo. See here for instructions on installing the multi bundle distribution.

DEMO DESIGN

From a high level the demo is very similar to the greeter demo. It comprises of 3 bundles:

- The demo interface bundle providing the Dinner Service interface.
- The Dinner Service implementation bundle.
- The Dinner Service consumer bundle.



The Dinner Service interface is as follows:

```
public interface DinnerService {
    List<Restaurant> findRestaurants(String searchQuery);
}
```

THE DINNER SERVICE PROVIDER

The service implementation provides a simplistic implementation of the DinnerService interface which is instantiated via Spring as a bean.

The interesting bit is in the `META-INF/spring/spring.xml` file. This file creates the DinnerServiceImpl bean and registers it with the OSGi Service Registry. It also sets the `osgi.remote.interfaces` property on the service to mark is as suitable for remoting. Here are the important parts of the `spring.xml` file:

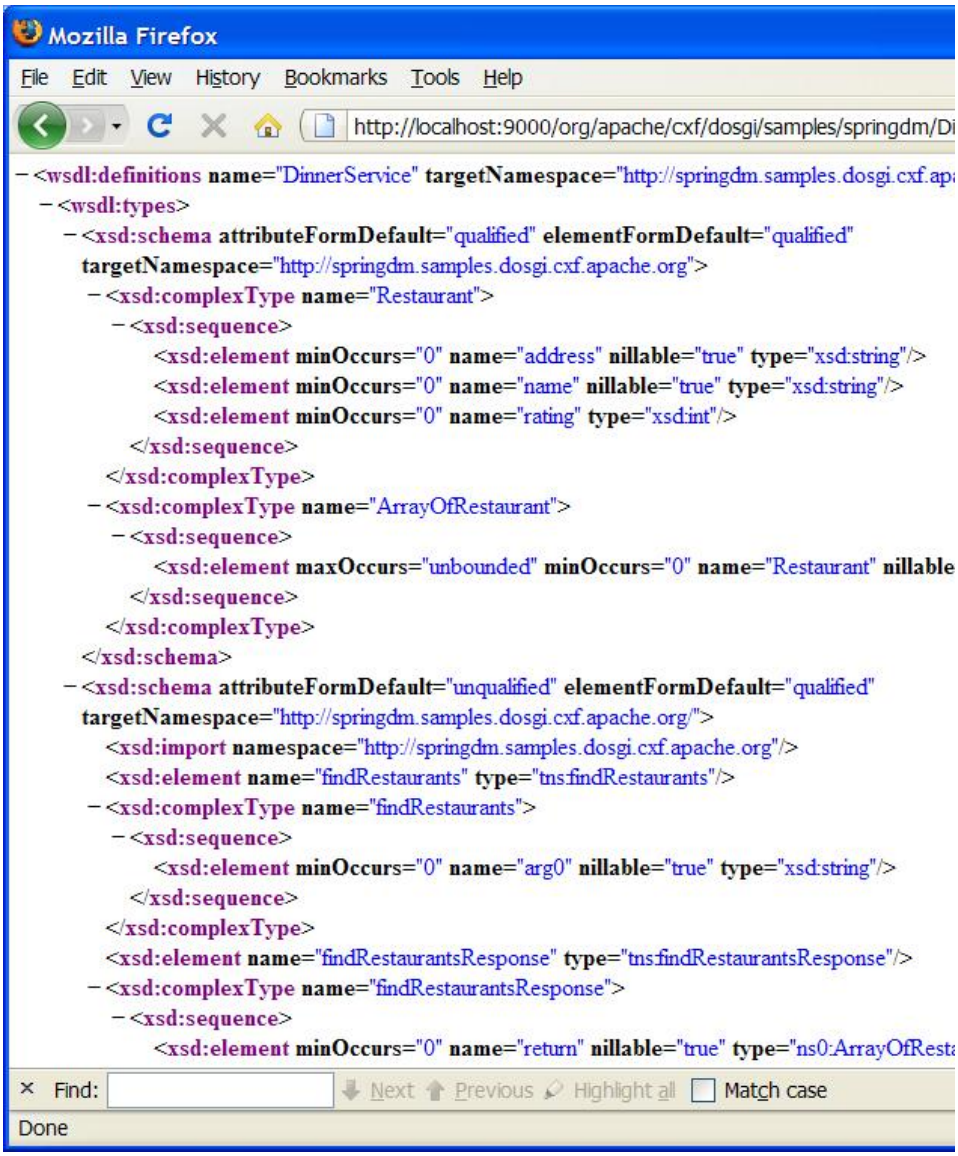
```
<beans xmlns:osgi="...">
  <osgi:service interface="org.apache.cxf.dosgi.samples.springdm.DinnerService"
>
    <osgi:service-properties>
      <entry key="service.exported.interfaces" value="*" />
    </osgi:service-properties>

    <bean class="org.apache.cxf.dosgi.samples.springdm.impl.DinnerServiceImpl"
/>
  </osgi:service>
</beans>
```

Unlike in the Greeter demo `osgi.remote.configuration...` properties are not set in the configuration, this means that the service is exposed on the default location of `http://localhost:9000/org/apache/cxf/dosgi/samples/springdm/DinnerService`. The default location is based on the interface name of the service being remoted.

In this example, the bundles are installed in Felix.

```
-> start http://repo2.maven.org/maven2/org/apache/cxf/dosgi/samples/cxf-dosgi-
ri-samples-spring-dm-interface/1.2/cxf-dosgi-ri-samples-spring-dm-interface-
1.2.jar
-> start http://repo2.maven.org/maven2/org/apache/cxf/dosgi/samples/cxf-dosgi-
ri-samples-spring-dm-impl/1.2/cxf-dosgi-ri-samples-spring-dm-impl-1.2.jar
... log messages may appear ...
-> ps
START LEVEL 32
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.8.0)
... bundles ...
[  35] [Active     ] [    1] CXF Distributed OSGi Spring-DM Sample Interface
Bundle
[  36] [Active     ] [    1] CXF Distributed OSGi Spring-DM Sample
Implementation Bundle
```

At this point the service should be available. You can check this by obtaining the WSDL:



THE DINNER SERVICE CONSUMER

As on the remote service provider side, the service consumer is also created using spring. Spring creates a DinnerServiceConsumer bean which is injected with the a proxy to the remote DinnerService. The injection is all done by Spring, which makes the code extremely simple. When Spring is done injecting, it calls the `start()` method where the remote service is used.

```
public class DinnerServiceConsumer {
    DinnerService dinnerService;

    public void setDinnerService(DinnerService ds) {
        dinnerService = ds;
    }

    public void start() {
        System.out.println("Found the following restaurants:");
        for (Restaurant r : dinnerService.findRestaurants("nice and not too
expensive!")) {
            System.out.format("  %s (%s) Rating: %d\n", r.getName(), r.
getAddress(), r.getRating());
        }
    }
}
```

The client side `META-INF/spring/spring.xml` file is also really simple. It simply declares a dependency on the OSGi DinnerService, which is injected into the DinnerServiceConsumer bean.

```
<beans>
  <osgi:reference id="dinnerServiceRef" interface="org.apache.cxf.dosgi.
samples.springdm.DinnerService"/>

  <bean class="org.apache.cxf.dosgi.samples.springdm.client.
DinnerServiceConsumer"
        init-method="start">
    <property name="dinnerService" ref="dinnerServiceRef"/>
  </bean>
</beans>
```

Like in the Greeter demo, the client side needs to be configured to know where the remote service actually is. When using a Discovery system this configuration is provided dynamically via Discovery, see the DOSGi Discovery Demo page. In this demo this information is provided statically in a `OSGI-INF/remote-service/remote-services.xml` file.

```
<endpoint-descriptions xmlns="http://www.osgi.org/xmlns/rsa/v1.0.0">
  <endpoint-description>
    <property name="objectClass">
      <array>
        <value>org.apache.cxf.dosgi.samples.springdm.DinnerService</value>
      </array>
    </property>
    <property name="endpoint.id">http://localhost:9000/org/apache/cxf/dosgi
/samples/springdm/DinnerService</property>
    <property name="service.imported.configs">org.apache.cxf.ws</property>
  </endpoint-description>
</endpoint-descriptions>
```

Install and run the consumer side of the demo in a separate Felix instance:

```
-> start http://repo2.maven.org/maven2/org/apache/cxf/dosgi/samples/cxf-dosgi-
ri-samples-spring-dm-interface/1.2/cxf-dosgi-ri-samples-spring-dm-interface-
1.2.jar
-> start http://repo2.maven.org/maven2/org/apache/cxf/dosgi/samples/cxf-dosgi-
ri-samples-spring-dm-client/1.2/cxf-dosgi-ri-samples-spring-dm-client-1.2.jar
... log messages may appear, at some point the consumer will make an
invocation on the remote service, you will see:
Found the following restaurants:
  Jojo's (1 food way) Rating: 3
  Boohaa's (95 forage ave) Rating: 1
  MicMac (Plastic Plaza) Rating: 1
```

And on the service provider side, you can see that it has been invoked as the following message appears:

```
-> Hey! Someone's using the Dinner Service! Query: nice and not too expensive!
```