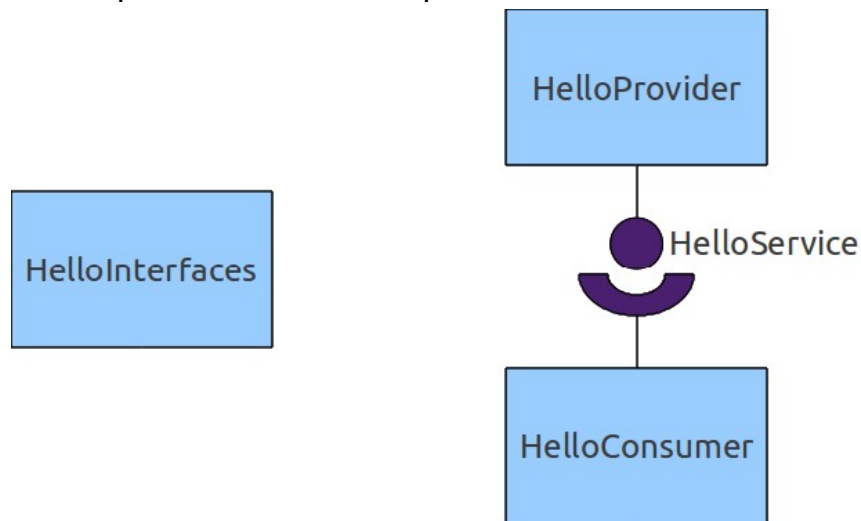


Remote Services using ECF

After a few weeks of work, I finally got an example of OSGi remote services running. This code is based on ECF framework for remote services. It uses Spring-DM configuration for creation of bundle and services. It has been tested in Virgo as OSGi runtime.

One of our main concern was to be able to deploy our applications either on one single machine, or in a distributed way, but without having to modify the code. We want this to be done only by configuration.

Let's consider a simple HelloWorld example :



The export of service in HelloProvider is done by the following lines in the spring configuration file :

```
<bean name="helloProvider" class="org.eclipse.ecf.examples.remoteservices.hello.impl.Hello">
<osgi:service interface="org.eclipse.ecf.examples.remoteservices.hello.IHello"
ref="networkService"/>
```

On the other side, the import in HelloConsumer is done this way:

```
<osgi:reference id="helloService"
interface="org.eclipse.ecf.examples.remoteservices.hello.IHello" />

<bean id="helloClient"
class="org.eclipse.ecf.examples.internal.remoteservices.hello.dm.consumer.HelloClientThread"
init-method="run">
<property name="hello" ref="helloService" />
</bean>
```

One simple solution to be able to deploy this example either locally or remotely is to create adapters bundle that will handle the remote stuff for you. The hello service will still be imported or exported locally by the consumer and provider, but it will be exposed or retrieved remotely by the adapters bundles.

For this purpose, we have created two classes : RemoteServiceExporter, and RemoteServiceImporter. See code below

RemoteServiceExporter

```
public class RemoteServiceExporter {
    private static Logger LOGGER = LoggerFactory.getLogger(RemoteServiceExporter.class);

    private IContainerManager    containerManager;
    private String                serviceInterface;
    private Object                serviceImplementation;

    public void start() throws Exception {
        Preconditions.checkNotNull(containerManager);

        ContainerFactory containerFactory =
(ContainerFactory)containerManager.getContainerFactory();
        Preconditions.checkNotNull(containerFactory);

        // Get or create container
        IContainer container = null;
        IContainer containers[] = containerFactory.getAllContainers();
        for (IContainer iContainer : containers) {
            if (iContainer instanceof TCPServerSOContainer) {
                container = iContainer;
                LOGGER.info("Container was found: class={},
connectedId={},", iContainer.getClass(), iContainer.getConnectedID());
                break;
            }
        }
        if (container == null) {
            container = containerFactory.createContainer("ecf.generic.server");
            LOGGER.info("Create new container");
        }
        Preconditions.checkNotNull(container);

        // Get remote service container adapter
        IRemoteServiceContainerAdapter containerAdapter = (IRemoteServiceContainerAdapter)
container
            .getAdapter(IRemoteServiceContainerAdapter.class);
        Preconditions.checkNotNull(containerAdapter);

        // Register remote service
        IRemoteServiceRegistration serviceRegistration =
containerAdapter.registerRemoteService(
            new String[] { serviceInterface }, serviceImplementation, null);
        LOGGER.info("{} registered remotely ({})", serviceInterface, serviceRegistration);
    }

    public void setContainerManager(IContainerManager aContainerManager) {
        containerManager = aContainerManager;
    }
    public IContainerManager getContainerManager() {
        return containerManager;
    }
    public String getServiceInterface() {
        return serviceInterface;
    }
    public void setServiceInterface(String serviceInterface) {
        this.serviceInterface = serviceInterface;
    }
    public Object getServiceImplementation() {
        return serviceImplementation;
    }
    public void setServiceImplementation(Object serviceImplementation) {
        this.serviceImplementation = serviceImplementation;
    }
}
```

This service exporter will get (or create if it doesn't exist) an ECF generic server container, and publish the service on this container.

RemoteServiceImporter

```
public class RemoteServiceImporter {
    private static Logger LOGGER = LoggerFactory.getLogger(RemoteServiceImporter.class);

    private IContainerManager    containerManager;
    private Object               remoteService;
    private String               remoteServiceHostAddress;
    private String               serviceInterface;

    public void start() throws Exception {
        Preconditions.checkNotNull(containerManager);

        // 1. Create Container
        IContainer container =
containerManager.getContainerFactory().createContainer("ecf.generic.client");
        Preconditions.checkNotNull(container);

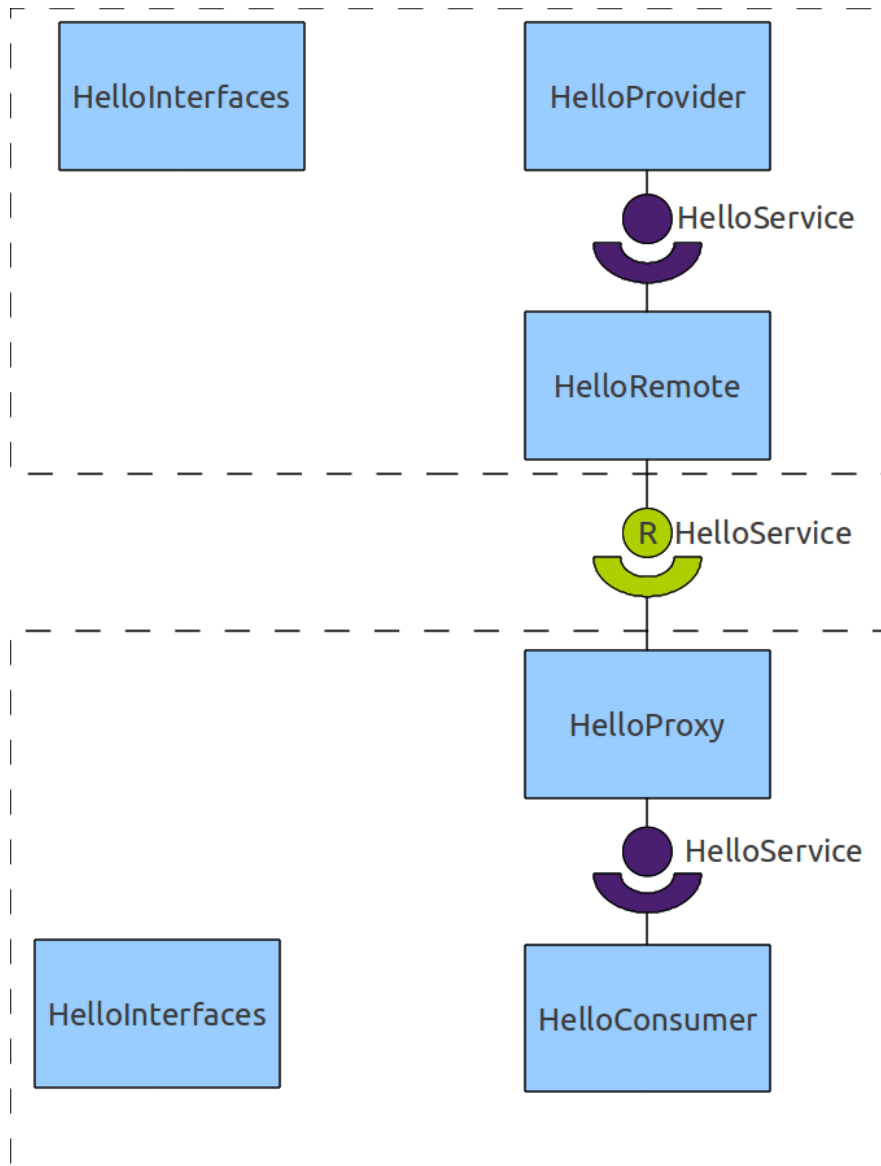
        // 2. Get remote service container adapter
        IRemoteServiceContainerAdapter containerAdapter = (IRemoteServiceContainerAdapter)
container.getAdapter(IRemoteServiceContainerAdapter.class);
        Preconditions.checkNotNull(containerAdapter);

        // 3. Lookup IRemoteServiceReference
        IRemoteServiceReference[] networkServiceReferences = containerAdapter
        .getRemoteServiceReferences(
remoteServiceHostAddress,
        IDFactory.getDefault().createID(container.getConnectNamespace(),
        serviceInterface, null));
        Preconditions.checkNotNull(networkServiceReferences);
        Preconditions.checkState(networkServiceReferences.length > 0);
        // 4. Get remote service for reference
        IRemoteService remoteServiceProxy =
containerAdapter.getRemoteService(networkServiceReferences[0]);
        // 5. Get the proxy
        remoteService = remoteServiceProxy.getProxy();
        LOGGER.info("Remote service found  {}", remoteService);
    }

    public void setContainerManager(IContainerManager aContainerManager) {
        containerManager = aContainerManager;
    }
    public Object getRemoteService() {
        return remoteService;
    }
    public void setRemoteServiceHostAddress(String remoteServiceHost) {
        this.remoteServiceHostAddress = remoteServiceHost;
    }
    public void setServiceInterface(String serviceInterface) {
        this.serviceInterface = serviceInterface;
    }
}
}
```

The service importer creates an ECF generic client container, look for the remote service on this container, and creates a proxy for the remote service. This proxy is then made available through an accessor getRemoteService, that will be used in the following as a factory method for spring bean creation.

When installed on two physical machines, the HelloWorld example will look like this:



The new bundles HelloRemote, and HelloProxy doesn't contain any code at all. The "common" classes RemoteServiceImporter, and RemoteServiceExporter being included in the HelloInterfaces bundle.

The bundles simply contain the following Spring configuration files:

HelloRemote:

```
<!-- Retrieve service locally -->
<osgi:reference interface="org.eclipse.ecf.examples.remoteservices.hello.IHello"
    id="helloService" />

<!-- Publish service remotely -->
<osgi:reference id="containerManager" interface="org.eclipse.ecf.core.IContainerManager"/>

<bean id="helloServiceExporter"
    class="org.eclipse.ecf.examples.remoteservices.adapter.RemoteServiceExporter" init-method="start">
    <property name="containerManager" ref="containerManager" />
    <property name="serviceInterface"
        value="org.eclipse.ecf.examples.remoteservices.hello.IHello"/>
    <property name="serviceImplementation" ref="helloService"></property>
</bean>
```

This will create a container manager, and use the RemoteServiceExporter to expose the service remotely.

HelloProxy:

```
<osgi:reference id="containerManager" interface="org.eclipse.ecf.core.IContainerManager"/>

<bean id="helloServiceImporter" class="org.eclipse.ecf.examples.remoteservices.adapter
.RemoteServiceImporter" init-method="start">
    <property name="containerManager" ref="containerManager" />
    <property name="serviceInterface"
        value="org.eclipse.ecf.examples.remoteservices.hello.IHello" />
    <property name="remoteServiceHostAddress" value="${hello-service.address}" />
</bean>

<bean id="helloService" factory-bean="helloServiceImporter" factory-method="getRemoteService" />

<!-- publish the service locally -->
<osgi:service interface="org.eclipse.ecf.examples.remoteservices.hello.IHello" ref="helloService" />
```

This will create a container manager, and use the RemoteServiceImporter to retrieve the service reference remotely. Then, the RemoteServiceImporter instance is used as a bean factory to create the local service bean, to be able to publish it into the local OSGi service repository.

The two adapters bundles can be reused for any application. The only limitation for now is that they use ECF generic provider, and don't offer the possibility to configure this. This can be improved in the future by adding a configuration property for these beans.