Generalitat de Catalunya
Centre de Telecomunicacions
i Tecnologies de la Informació

Framework Corporatiu J2EE

Servei d'integració amb WebServices

Versió 1.2

Històric de modificacions

Data	Autor	Comentaris	Versió
13/01/2006	Atos Origin, sae openTrends	Versió inicial del document	1.0
02/10/2006	Atos Origin	Versió 1.1 openFrame	1.1
13/02/2007	Atos Origin, SAE	Versió 1.2 openFrame	1.2

Llegenda de Marcadors

Índex

1. Introducció

1.1. Propósit

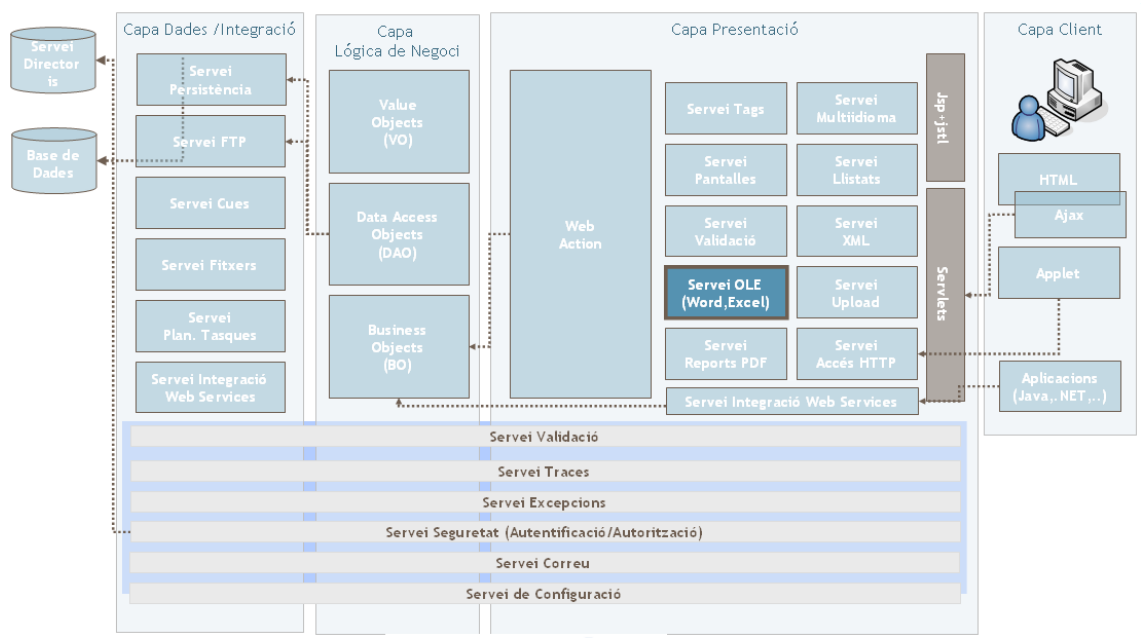
Aquest servei permet configirar i usar de forma senzilla la infraestructura de Web Services en dues modalitats:

- Exportació de serveis Java mitjançant Web Services.
- Importació de Web Services externs, generació si cal de classes Java de invocació.

L'enfoc d'aquest servei és el de simplificar tant la definició de Web Services a partir de serveis Java simples (que no tindran dependències amb la implementació particular de Web Services) així com la de facilitar la invocació a Web Services externs.

1.2. Context i Escenaris d'Ús

El Servei d'Integració de WebServices es troba ubicat dins els serveis continguts a la capa de Dades/Integració de openFrame.



1.3. Versions i Dependències

1.3.1. Versions

No hi ha canvis respecte versió 1.0

1.3.2. Dependències Bàsiques

En el present apartat es mostren quines són les versions i dependències necessàries per fer ús del Servei.

Dins la llista de dependències es mostren diferenciades:

- Dependències bàsiques. Llibreries necessàries per fer ús del servei
- Dependències adicionals. Aquestes dependències són necessàries per poder fer ús de característiques concretes del servei o per l'ús dels tests unitaris proporcionats amb el servei

Nom	Tipus	Versió	Descripció
soap	jar	2.3.1	
spring	jar	1.2.5	http://www.springframework.org
openFrame-services-configuration	jar	1.1	
openFrame-services-exceptions	jar	1.1	
openFrame-services-logging	jar	1.1	
activation	jar	1.0.2	
wsdl4j	jar	1.4	
saaj-api	jar	1.0	
saaj-impl	jar	1.0	
xercesImpl	jar	2.7.1	
xfire-all	jar	1.0-M5	
xmlbeans	jar	2.0-20041214	
xmlbeans-xmlpublic	jar	2.0-20041214	
yom	jar	1.0-alpha-2	

1.3.3. Dependències Adicionals

- Axis
  Dependències per generar classes automàtiques per WDSL i Servidor Web Services

Nom	Tipus	Versió	Descripció
axis	jar	1.3	
axis-ant	jar	1.3	
axis-jaxrpc	jar	1.3	

Nom	Tipus	Versió	Descripció
axis-saaj	jar	1.3	
axis-wsdl4j	jar	1.3	

1.4. A qui va dirigit

Aquest document va dirigit als següents perfils:

- Programador. Per conéixer l'ús del servei
- Arquitecte. Per conéixer quins són els components i la configuració del servei
- Administrador. Per conéixer cóm configurar el servei en cadascun dels entorns en cas de necessitat

1.5. Documents i Fonts de Referència

[1] Spring Web Services http://static.springframework.org/spring/docs/1.2.x/reference/webservices.html

1.6. Glossari

2. Descripció Detallada

2.1. Arquitectura i Components

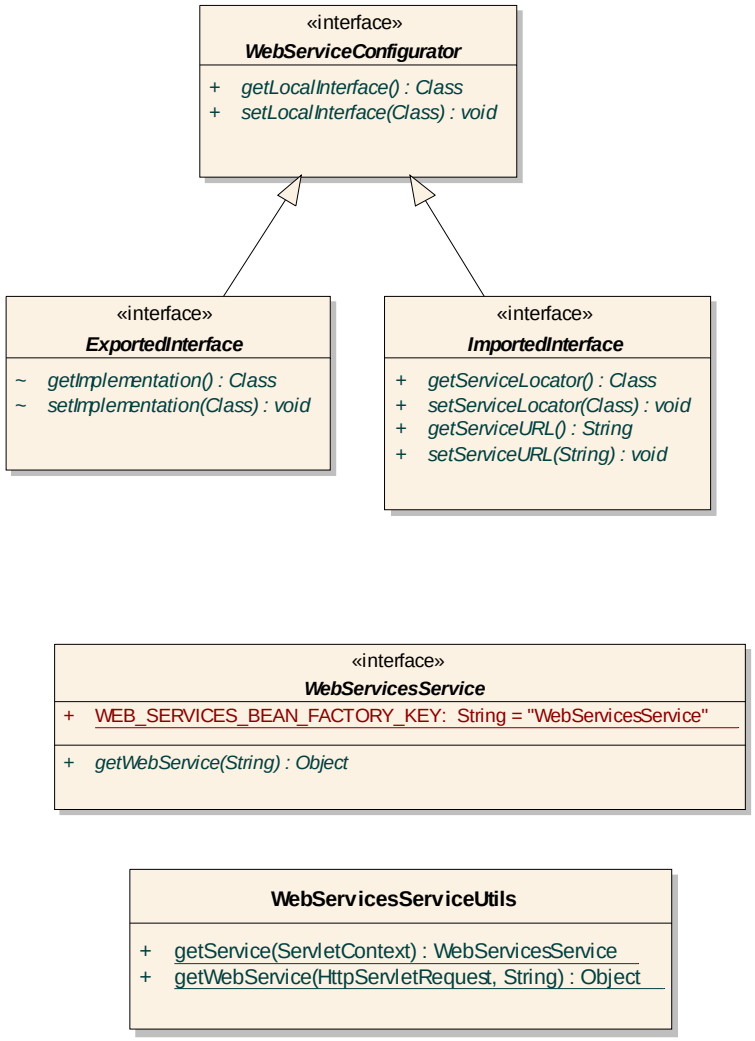
Els components podem classificar-los en:

- Interfícies i Components Genérics. Interfícies del servei i components d'ús general amb independència de la implementació escollida.
- Implementació basada en Spring

2.1.1. Interfícies i Components Genérics

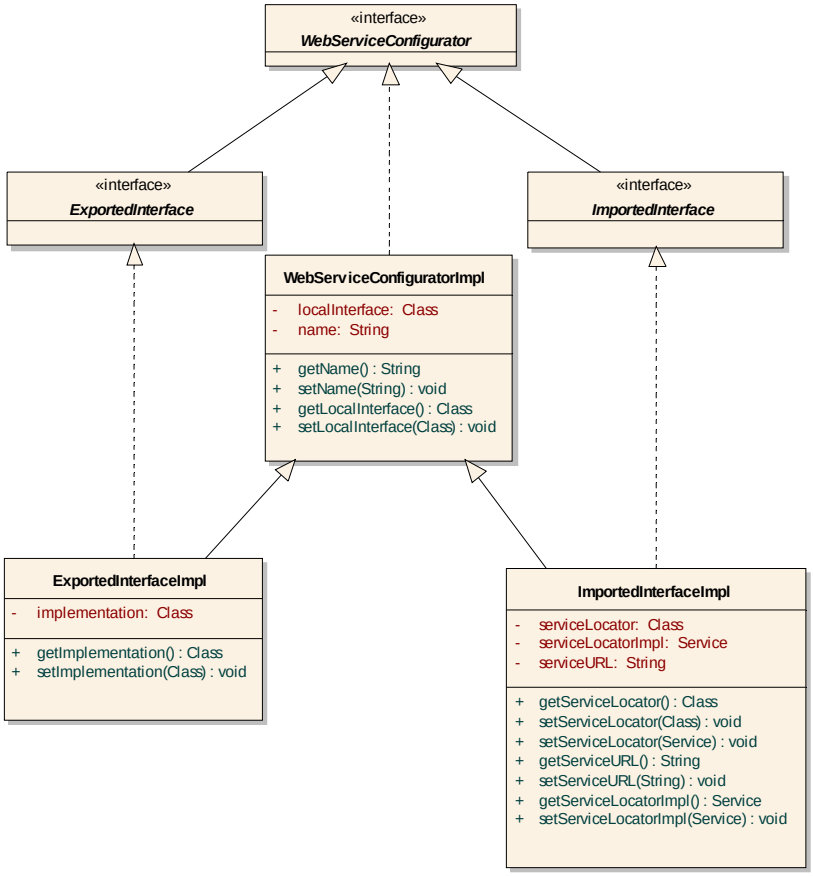
El servei defineix les següents interfícies i components genérics:

Component	Package	Descripció
WebServiceConfigurator	net.opentrends.openframe.services.webservices	Interfície bàsica que defineix la configuració de qualsevol web service, tant exportat com importat
ExportedInterface	net.opentrends.openframe.services.webservices	Extén la interfície 'WebServiceConfigurator' per definir les propietats necessàries per exportar un servei
ImportedInterface	net.opentrends.openframe.services.webservices	Extén la interfície 'WebServiceConfigurator' per definir les propietats necessàries per importar un servei
WebServicesService	net.opentrends.openframe.services.webservices	Interfície del Servei. Permet obtenir la referència a un webservice com si d'una interfície local es tractés
WebServicesServiceUtils	net.opentrends.openframe.services.webservices	Classe estàtica d'obtenció de referències a un web service segons el request de presentació
WebServicesServiceException	net.opentrends.openframe.services.webservices	Excepció llençada pel servei



2.1.2. Implementació basada en Spring

Component	Package	Descripció
WebServiceConfiguratorImpl	net.opentrends.openframe.services.webservices.impl	Implementació de la interfície 'WebServiceConfigurator' basada en Spring
ExportedInterfaceImpl	net.opentrends.openframe.services.webservices.impl	Implementació de la interfície 'ExportedInterface' basada en Spring
ImportedInterfaceImpl	net.opentrends.openframe.services.webservices.impl	Implementació de la interfície 'ImportedInterface' basada en Spring
WebServicesServiceImpl	net.opentrends.openframe.services.webservices.impl	Implementació de la interfície 'WebServicesService' basada en Spring


«interface»
WebServiceConfigurator
«interface»
ExportedInterface
«interface»
ImportedInterface
WebServiceConfiguratorImpl
- localInterface: Class
- name: String
+ getName() : String
+ setName(String) : void
+ getLocalInterface() : Class
+ setLocalInterface(Class) : void
ExportedInterfaceImpl
- implementation: Class
+ getImplementation() : Class
+ setImplementation(Class) : void
ImportedInterfaceImpl
- serviceLocator: Class
- serviceLocatorImpl: Service
- serviceURL: String
+ getServiceLocator() : Class
+ setServiceLocator(Class) : void
+ setServiceLocator(Service) : void
+ getServiceURL() : String
+ setServiceURL(String) : void
+ getServiceLocatorImpl() : Service
+ setServiceLocatorImpl(Service) : void

2.2. Instal·lació i Configuració

2.2.1. Instal·lació

La instal·lació del servei requereix de la utilització de la llibreria 'openFrame-services-webservices' i les dependències indicades a l'apartat 'Introducció-Versions i Dependències'.

2.2.2. Configuració

La configuració del Servei d'Integració de Web Services implica els següents pasos:

1) Definir la implementació del Servei que s'usarà
2) Definir les interfícies pare d'exportació de serveis

3) Definir quins elements volem exportar com a WebServices i/o
4) Definir quins elements volem importar com a WebServices

Definició del Servei

```
<bean id="webServicesService"
      ...>
```

Fitxer de configuració: openFrame-services-webservices.xml

Ubicació proposada: <PROJECT_ROOT>/src/main/resources/spring

En aquest pas indicarem el bean del servei de Web Services de **openFrame** i la implementació que es farà servir. En l'actualitat s'ofereix la implementació 'net.opentrends.openframe.services.webservices.impl. WebServicesServiceImpl'.

Podem definir les següents propietats:

Propietat	Requerit	Descripció
exportedInterfaces	No	Llista de serveis a exportar. Veure 'Definició dels Serveis a Exportar'
importedInterfaces	No	Llista de serveis a importar. Veure 'Definició dels Serveis a Importar'

Exemple:

```
<bean name="webServicesService" class="net.opentrends.openframe.services.
  webservices.impl.WebServicesServiceImpl"
  ...
</bean>
```

És a dir, si ens volem comunicar amb un Webservice ja existent farem ús de la propietat 'exportedInterfaces', mentre que si volem publicar un WebService usarem la propietat 'importedInterfaces'.

Definició de les Interfícies Pare d'Exportació i Importació de Serveis

```
<bean abstract="true" id="ExportedInterfaceDefinition"
      ...>

<bean abstract="true" id="ImportedInterfaceDefinition"
      ...>
```

Fitxer de configuració: openFrame-services-webservices.xml

Ubicació proposada: <PROJECT_ROOT>/src/main/resources/spring

Usar el següent codi:

```
<bean abstract="true"
   id="exportedInterfaceDefinition"     class="net.opentrends.openframe.services.
          webservices.impl.ExportedInterfaceImpl"
/>
<bean abstract="true"
   id="importedInterfaceDefinition"     class="net.opentrends.openframe.services.
   webservices.impl.ImportedInterfaceImpl"
/>
```

Amb aquesta secció de declaracions es pretén establir les classes que permeten fer la definició dels WebServices, i estalviar així haver de repetir la declaració de la classe sencera (els noms de package són llargs) per cada bean d'importació/exportació que declarem. L'atribut "abstract=true" implica que aquests beans no s'instancien, només serveixen per fer-ne redefinicions.

NOTA: Només definirem un i/o l'altre segons ens comuniquem amb un webservice i/o publiquem un webservice.

Definició dels Serveis a Exportar

```
<bean name="webServicesService">
   <property name="exportedInterfaces">
```

Fitxer de configuració: openFrame-services-webservices.xml

Ubicació proposada: <PROJECT_ROOT>/src/main/resources/spring

Definir els serveis a exportar dins la propietat 'exportedInterfaces' del bean definició del servei. Aquesta propietat és una llista dels beans que exportarem. Per cada bean a exportar farem que el seu parent sigui la definició abstracta ('exportedInterfaceDefinition') i es definiran les següents propietats:

Propietat	Requerit	Descripció
name	Sí	Nom amb el que es publicarà al servlet el Web Service resultant (per generar la URL de publicació)
localInterface	Sí	La interfície Java convencional que ha d'implementar el servei. És aquesta interfície la que openFrame s'encarrega de fer accessible externament, generant de forma automàtica el WSDL que descriu la interfície.

Propietat	Requerit	Descripció
implementation	Sí	Classe concreta que realitza el servei i implementa la interfície

Exemple:

```
<bean name="webServicesService"
      parent="WebServiceDefinition">
   <property name="exportedInterfaces">
       <list>
           <bean parent="exportedInterfaceDefinition">                  <property
name="name" value="testService"/>                        <property
name="implementation"
                    value="net.opentrends.samples.
                                webservices.TestServiceImpl"/>
             <property name="localInterface"
           value="net.opentrends.samples.
                                webservices.TestService"/>
           </bean>
      </list>
    </property>
   ...
</bean>
```

Només especificant les tres propietats s'aconsegueix que un servei Java sigui accessible per Web Services.

En aquest exemple, es podria obtenir el WDSL (Web Services Description Language) generat a una URL semblant a:

http://localhost:8080/openFrame-samples-webservices/testService?wsdl

On testService es correspon al nom definit en el xml del servlet.

Per últim haurem d'afegir al fitxer 'web.xml' de l'aplicació un param-value per a que carregui el fitxer 'xfire.xml' del jar de xfire:

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
            ...
            classpath:org/codehaus/xfire/spring/xfire.xml
       </param-value>
</context-param>
```

Definició dels Serveis a Importar

```
<bean name="webServicesService">
   <property name="importedInterfaces">
```

Fitxer de configuració: openFrame-services-webservices.xml

Ubicació proposada: <PROJECT_ROOT>/src/main/resources/spring

Definir els serveis a importar dins la propietat 'importedInterfaces' del bean definició del servei. Per cada bean a importar farem que el seu parent sigui la definició abstracta ('importedInterfaceDefinition').

Existeixen 2 possibilitats d'integració amb serveis externs:

1) Disposem de la URL de definició del servei remot i de una interfície Java correcta que equival al WDSL. El servei web és de tipus "document" i no "rpc"

   En aquest cas, definirem les següents propietats:

Propietat	Requerit	Descripció
name	Sí	Nom amb el que s'identificarà el Web Service importat, per tal de poder demanar-ne una instància al Servei de Web Services.
serviceURL	Sí	Url remota on s'exposa el servei. Si l'adreça del servei és per exemple: http://localhost:8080/openFrame-samples-webservices/testService llavors el descriptor WSDL del web service s'ha de poder trobar a: http://localhost:8080/openFrame-samples-webservices/testService ?WSDL Això succeeix de forma automàtica amb els serveis exportats mijançant openFrame.
localInterface	Sí	Interfície Java convencional que permet accedir al servei extern

   Exemple:

```
<bean parent="importedInterfaceDefinition">                    <property
name="name" value="GoogleSearch" />                  <property name="serviceURL"
                                        value="http://localhost:8080/
                openFrame-samples-webservices/testService"/>
   <property name="localInterface"
   value="net.opentrends.samples.
                    webservices.TestService" />
</bean>
```

2) Volem generar de forma automàtica les classes de comunicació a partir d'un WDSL extern. Veure l'apartat 'Eines de Suport-Generació de Classes a partir WDSL' per

consultar el procediment de generació d'aquestes classes. Definirem les següents propietats:

Propietat	Requerit	Descripció
name	Sí	Nom amb el que s'identificarà el Web Service importat, per tal de poder demanar-ne una instància al Servei de Web Services.
serviceLocator	Sí	Classe encarregada de localitzar el servei remot. En cas de fer ús de les indicacions de l'apartat 'Eines de Suport' aquesta classes correspon a la que conté el sufix 'locator' dins les classes generades.
localInterface	Sí	Interfície Java convencional que permet accedir al servei extern

2.3. Utilització del Servei

openFrame exposa una interfície d'utilització, "net.opentrends.openframe.webservices.WebServicesService", que amaga la implementació real utilitzada. D'aquesta forma diverses implementacions són fàcilment configurables en funció de les necessitats de l'aplicació. Aquesta interfície permet obtenir un webservice i treballar-hi posteriorment com si d'una classe normal es tractés.

La interfície té el següent aspecte:

```
package net.opentrends.openframe.services.webservices;

/**
 * Interface which allows the user to retrieve a Web Service interface from a
given configuration
 *
 */
public interface WebServicesService {
...
   public Object getWebService(String serviceName);

}
```

Aquest interfície exposa un únic métode 'getWebService' que permet obtenir una instància del Web Service definit amb el nom del paràmetre passat. Aquest nom correspon al definit a la propietat 'name' de la definició dels serveis importats o exportats (veure apartat 'Configuració').

És responsabilitat de programador identificar correctament la interfície de servei tant a la configuració com a l'hora de fer ús del servei, fent el corresponent "cast". Una vegada realitzat el cast, podem treballar-hi com si d'un servei local es tractés.

Adicionalment, s'ofereix una classe 'net.opentrends.openframe.services.webservices.WebServicesServiceUtils' que proporciona el mètode 'getWebService(HttpServletRequest request, String serviceName)'. Aquest mètode

permet que des de les classes de presentació s'obtingui de forma directa una referència a una instància de webservice.

2.4. Eines de Suport

2.4.1. Generació de Classes a partir WDSL

Per a generar les classes necessàries per importar un servei extern a partir d'un WDSL (veure apartat 'Configuració-Definició dels Serveis a Importar'), podem crear un 'goal' de Maven amb el següent contingut:

```
  <goal name="opentrends:generate-from-wsdl"          description="generate
client classes from wsdl"       prereqs="opentrends:init">

    <ant:fileScanner var="wsdlFiles">
      <ant:fileset dir="${axis.url}">
        <ant:include name="**/*.wsdl" />
      </ant:fileset>
    </ant:fileScanner>
    <j:if test="${wsdlPresent == 'true'}">
        <ant:mkdir
                 dir="${maven.src.dir}
                        /webservices/generated"/>
    </j:if>
           <j:forEach var="wsdlFile"
                       items="${wsdlFiles.iterator()}">
            <axis-wsdl2java
                    output="${maven.src.dir}
                        /webservices/generated"
                    verbose="true"
                    testcase="${basedir}/src/main/test}"
                    noimports="true"
                    url="${wsdlFile}">
            </axis-wsdl2java>
     </j:forEach>
 </goal>

  <goal name="opentrends:init">
   <taskdef resource="axis-tasks.properties"
classpathref="maven.dependency.classpath" />
    <ant:available property="wsdlPresent"
         file="${axis.url}" />
    <ant:available property="generatedPresent"
         file="${maven.gen.src}" />
    <j:if test="${wsdlPresent == 'true'}">
         <j:set var="maven.gen.src"
   value="${maven.src.dir}/webservices"/>
        <ant:path id="my.other.src.dir"
      location="${maven.src.dir}/
                                        webservices/generated"/>
          <maven:addPath id="maven.compile.src.set"
      refid="my.other.src.dir"/>
    </j:if>
   </goal>
```

En executar aquest goal es realitzarà de forma automàtica el següent procés:

1) S'examina el directori pre-establert on es troben els fitxers *.WSDL (en general, a src/main/wsdl)

2) Es generen les classes Java basades en Axis que equivalen als objectes remots descrits en el WSDL. Aquestes classes són generades al directori 'src/main/java/webservices/generated'

2.5. Integració amb Altres Serveis

2.5.1. Definició de les Excepcions Internacionalitzades

El servei defineix vàries excepcions amb diferents claus de missatges. Aquestes són:

openFrame.services.WebServices.lookup_failed=lookup failed for **{0}**
openFrame.services.WebServices.bad_bean_configuration=bad configuration for **{0}**
openFrame.services.WebServices.remote_method_invocation_failed=remote method **{0}** failed for bean **{1}**

Per tant, han d'existir en el fitxer de recursos.

2.6. Preguntes Freqüènts

Quan arranquem l'aplicació ens apareix al log el missatge: java.lang.NoSuchMethodError: javax.xml.namespace.QName.<init>(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String)

```
[10/03/06 17:18:32:130 CET] 2f75d693 ContextLoader E
org.springframework.web.context.ContextLoader  TRAS0014I: The following exception
was logged java.lang.NoSuchMethodError: javax.xml.namespace.QName: method
<init>(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V not found
   at     at
org.codehaus.xfire.aegis.type.DefaultTypeMappingRegistry.<clinit>(DefaultTypeMappi
ngRegistry.java:54).null(Unknown Source)
   at java.lang.Class.forName0(Native Method)
   at java.lang.Class.forName(Class.java(Compiled Code))
   at org.springframework.util.ClassUtils.forName(ClassUtils.java:88)
   at
org.springframework.beans.factory.support.BeanDefinitionReaderUtils.createBeanDefi
nition(BeanDefinitionReaderUtils.java:65)
   at
org.springframework.beans.factory.xml.DefaultXmlBeanDefinitionParser.parseBeanDefi
nitionElement(DefaultXmlBeanDefinitionParser.java:369)
...
```

Si es fa exportació de WebServices mitjançant XFire, hem de tenir cura de quina és la versió del Servidor d'Aplicacions que es fa servir. Els Servidors d'Aplicacions tenen per defecte algunes llibreries ja incorporades que poden afectar a les aplicacions que vulguin utilitzar noves versions d'aquestes llibreries. La política per defecte dels Servidors d'Aplicacions sol ser que en cas de que l'aplicació importi una classe es carregui primer la disponible al Servidor d'Aplicacions. D'aquí que en el cas mostrat, la classe 'QName' és trobada al Servidor d'Aplicacions i no es té en compte la que es troba a la llibreria més nova de l'aplicació.

Per a canviar aquest comportament, existeix la possibilitat de canviar la precedència de la càrrega de classes per aplicació, de forma que si una classe està en una llibreria del servidor i en una llibreria del módul de l'aplicació, sigui prioritària aquesta última enlloc de la primera.

Comprovacions prèvies

Abans de fer cap canvi de configuració ens hem d'assegurar que es troba en el fitxer de dependències la llibreria 'stax-api-1.0.jar' i que s'afegirà al war creat.

```
<dependency>
      <groupId>stax</groupId>
      <artifactId>stax-api</artifactId>
      <version>1.0</version>
      <properties>
                  <war.bundle>true</war.bundle>
            </properties>
    </dependency>
```

Aquesta llibreria conté la versió de QName que necessita XFire.

Adicionalment comprovar que hem definit les següents dependències:

- openFrame-services-logging-1.0
- openFrame-services-exceptions-1.0
- openFrame-services-webservices-1.0.1
- xfire-all-1.0-M5 (grup xfire)
- stax-api-1.0 (grup stax)
- wstx-asl-2.9.1 (grup woodstox)
- yom-1.0-alpha-2 (grup yom)
- jaxen-1.1-beta-8 (grup jaxen)
- jdom-1.0 (jdom)
- xbean-spring (xbean)
- wsdl4j-1.5.2 (wsdl4j)
- commons-beanutils-1.7.0
- spring-1.2.5

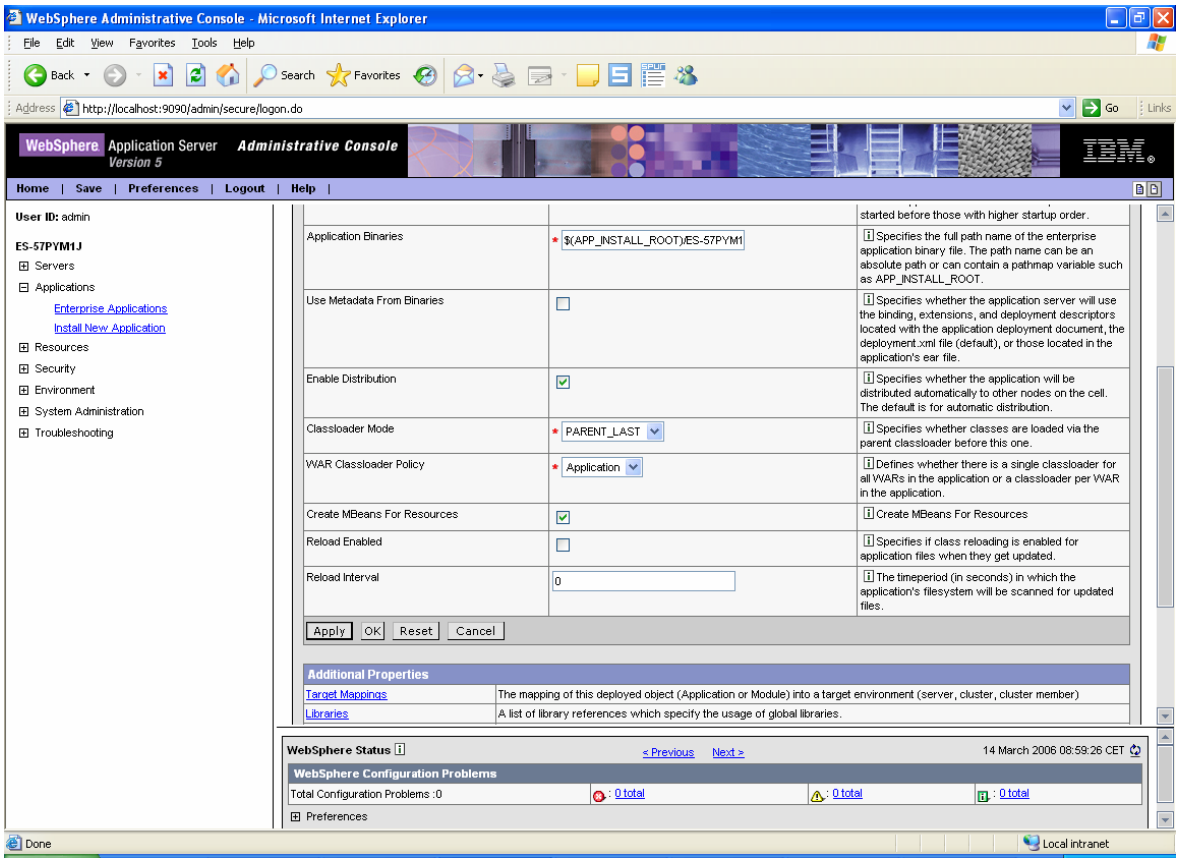
Per totes elles assegurar-se de que s'incorporaran al war amb el tag '<war.bundle>true</war.bundle>'.

Websphere

A Websphere, una vegada fet el desplegament de l'aplicació tornar a seleccionar 'Applications>Enterprise Applications' i seleccionar l'aplicació. Des de la pestanya 'Configuration' realitzar els següents canvis:

- Canviar el valor del camp 'Classloader Mode' a 'PARENT_LAST':
- Canviar el valor del camp 'WAR Classloader Policy' a 'Application'

Aplicar els canvis. En aquest moment iniciar l'aplicació i comprovar que no es dóna el missatge previ de conflicte.



WebLogic

En WebLogic podem fer el canvi directament al fitxer 'weblogic.xml' introduint el valor 'true' en el tag 'prefer-web-inf-classes':

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE weblogic-web-app
PUBLIC "-//BEA Systems, Inc.//DTD Web Application 8.1//EN"
"http://www.bea.com/servers/wls810/dtd/weblogic810-web-jar.dtd">

<weblogic-web-app>
<container-descriptor>
   <prefer-web-inf-classes>true</prefer-web-inf-classes>
```

```
</container-descriptor>

</weblogic-web-app>
```

Apareix l'Error 'javax.xml.stream.FactoryConfigurationError: Provider null could not be instantiated: java.lang.NullPointerException'

Si aquest error es dona en una aplicació desplegada al Servidor d'Aplicacions, tenim 2 possibles solucions:

a) Crear un fitxer per definir la factoria 'XMLInputFactory' (opció més recomanada)

Crear un directori 'META-INF/services' (dins webapp si és una aplicació Web) i en aquest directori crear:

- Fitxer 'javax.xml.stream.XMLInputFactory' (sense extensió)

  El contingut d'aquest fitxer ha de ser 'com.ctc.wstx.stax.WstxInputFactory'.

Aquest comportament només funcionarà si no existeix el fitxer 'jaxp.properties' en el subdirectori 'jre\lib' de Java (veure següent alternativa). Així doncs, cal assegurar-se de que aquest fitxer (explicat en l'altra alternativa) no existeixi.

b) Editar un fitxer 'jaxp.properties' a la localització del subdirectori 'jre\lib' amb la següent informació:

```
javax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFacto
ryImpl
javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
javax.xml.parsers.DocumentBuilderFactory=org.apache.xerces.jaxp.DocumentBuilderFac
toryImpl
javax.xml.stream.XMLInputFactory=com.ctc.wstx.stax.WstxInputFactory
```

El subdirectori 'jre\lib' l'haurem de cercar en el directori del JRE que s'estigui fent servir. Així, per exemple en Websphere accedirem al directori 'AppServer\jre\lib' del directori d'instal·lació.

3. Exemples

Veure l'apartat 'Configuració'. En un futur s'afegiran més exemples.

3.1. Exemple de Test

Classe: 'net.opentrends.openframe.services.webservices.test.WebServicesServiceTest'

Per tractar-se d'un Test Unitari l'obtenció del servei es realitza de forma directa amb 'ClassPathXMLApplicationContext'. Cal recordar que podrem definir en els nostres beans de l'aplicació el servei sense necessitar d'accedir-hi amb 'ClassPathXmlApplicationContext'. En aquest cas és necessari per tractar-se d'un test unitari.

Exemple d'accés directe:

```
//Obtenim el context Spring de test, aixo  no cal fer-ho explícitament
//en una aplicació web

BeanFactory beanFactory = new
   ClassPathXmlApplicationContext("webServicesContext.xml");

//Obtenim el servei pròpiament dit

WebServicesService service = (WebServicesService)
beanFactory.getBean(WebServicesService.WEB_SERVICES_BEAN_FACTORY_KEY);



//ara es pot fer servir l'interface  WebServicesService per accedir
   a un Web Serice remot ...

GoogleSearchPort google = (GoogleSearchPort)
                 service.getWebService("GoogleSearch");

google.doGoogleSearch(...);
```

Exemple d'accés des d'un Action:

```
GoogleSearchPort google =
   WebServicesServiceUtils.getWebService(request,"GoogleSearch");

google.doGoogleSearch(...);
```

3.2. Exemple de Publicació d'un Servei de Codis Postals

En aquest exemple veurem cóm podem publicar un servei de forma senzilla mitjançant openFrame i XFire.

Suposem que volem publicar un servei que permet obtenir les localitats associades a un codi postal. Com a exemple pràctic i bastant real farem servir un servei extern ja existent de GeoNames.

3.2.1. Creació de la Interfície

En primer lloc creem una interfície Java en la que definim quins mètodes oferirem.

```
package net.opentrends.samples.geo;

import java.util.Collection;

public interface GeoNamesService {
    public Collection getLocationsPostalCode(String aPostalCode);
}
```

En aquest cas trobem un mètode que a partir d'un codi postal ens retornarà una col·lecció de poblacions coincidents amb el codi postal.

3.2.2. Implementació de la Interfície

A continuació definim la implementació de la interfície:

```
public class GeoNamesServiceImpl implements GeoNamesService {


    public GeoNamesServiceImpl() {
        super();
        // TODO Auto-generated constructor stub
    }


    public Collection getLocationsPostalCode(String aPostalCode) {

        URL url;
        ArrayList list = new ArrayList();

        try {
            url = new URL("http://ws.geonames.org/postalCodeSearch?postalcode=" +
aPostalCode + "&country=ES&maxRows=10");
            try {

                SAXReader reader = new SAXReader();
                Document document = reader.read(url);
                // XES: Access with XPath
                List listNodes = document.selectNodes( "//geonames/code/name" );

                for ( Iterator i = listNodes.iterator(); i.hasNext(); ) {
                    DefaultElement element = (DefaultElement)i.next();
                    String text = element.getText();
                    list.add(text);
                    System.out.println(text);
```

```
                }

            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        } catch (MalformedURLException e2) {
            // TODO Auto-generated catch block
            e2.printStackTrace();
        }

        return list;
    }
```

En l'exemple mostrat s'obté la informació a partir d'un servei proporcionat per GeoNames i es parseja el resultat rebut per crear una col·lecció. No és propósit d'aquest exemple mostrar cóm obtindrem en les nostres aplicacions la geocodificació dels codis postals, però hem cregut convenient oferir un exemple molt real. Per tant hauria estat suficient amb deixar al lector una implementació de la interfície que fes un càlcul simple.

Una vegada tenim implementada la classe podem publicar-la fàcilment com un WebService:

3.2.3. Publicació del Servei

```
<bean abstract="true" id="exportedInterfaceDefinition"
   class="net.opentrends.openframe.services.webservices.impl.ExportedInterfaceImpl
"/>

<bean name="webServicesService"
class="net.opentrends.openframe.services.webservices.impl.WebServicesServiceImpl">
   <property name="exportedInterfaces">
    <list>
          <bean parent="exportedInterfaceDefinition">
            <property name="name" value="geoService"/>
            <property name="implementation"
value="net.opentrends.samples.geo.GeoNamesServiceImpl"/>
            <property name="localInterface"
value="net.opentrends.samples.geo.GeoNamesService"/>
          </bean>
      </list>
    </property>
</bean>
```

La publicació del servei és tan senzilla com el codi a dalt mostrat, on dins la propietat 'exportedInterfaces' del bean 'webServicesService' s'ha definit una exportació amb la següent informació:

- name: Nom del servei publicat. Corresponent a cóm es publicarà el servei dins el WSDL
- implementation: Nom de la classe implementació
- localInterface: Nom de la interfície

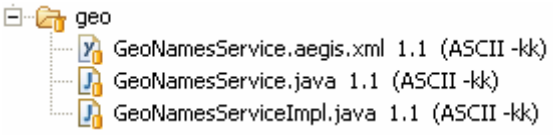
Adicionalment ens hem d'assegurar d'afegir al fitxer 'web.xml' de l'aplicació un param-value per a que carregui el fitxer 'xfire.xml' del jar de xfire:

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
            ...
            classpath:org/codehaus/xfire/spring/xfire.xml
       </param-value>
</context-param>
```

Tipus de Dades

En la utilització dels WebServices cal tenir en compte que hi han tipus de dades que no poden ser enviats i rebuts si no s'especifica de forma adicional cóm el Servidor pot tractar les peticions.

Si definim una col·lecció per exemple (com en aquest cas exemple), haurem d'especificar quin tipus de dades contindrà internament per saber com passar les dades de retorn. Per a aconseguir-ho només cal que definim un fitxer amb el nom de la interfície i a continuació l'extensió 'aegis.xml' i ubicar-lo en el mateix lloc que la interfície.



En l'exemple, definim el següent contingut:

```
<?xml version="1.0" encoding="UTF-8" ?>

<mappings>
    <mapping>
        <method name="getLocationsPostalCode">
            <return-type componentType="java.lang.String"/>
        </method>
    </mapping>
</mappings>
```

Això vol dir que la col·lecció de retorn del mètode 'getLocationsPostalCode' conté elements de tipus String.

Per a informació d'altres mapejos consultar la url 'http://xfire.codehaus.org/Aegis+Binding'.

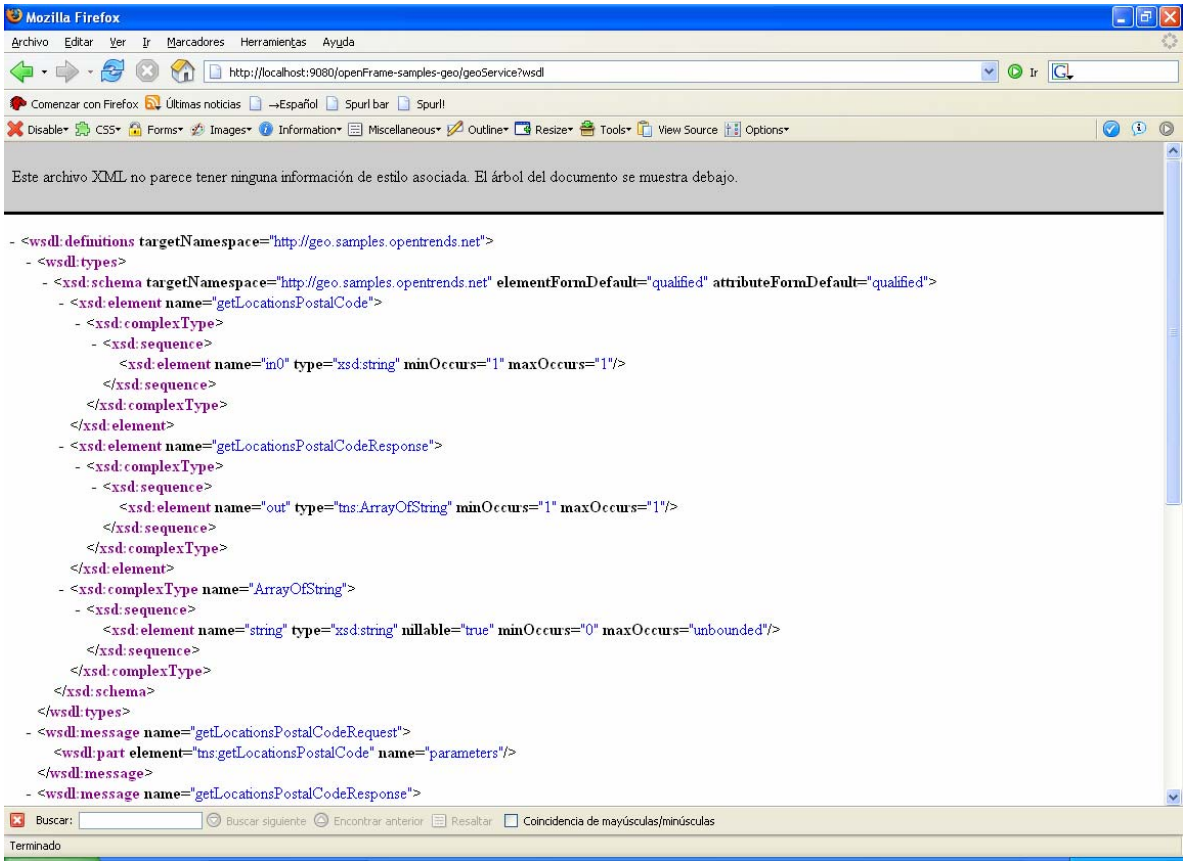
3.2.4. Desplegament del Servei i Prova d'Accés al WSDL

A continuació, podem crear un war i desplegar-lo en un servidor d'aplicacions.

Una vegada desplegat podem accedir al seu wsdl amb la següent url:

http://localhost:9080/<nom context app> /geoService?wsdl

El nom 'geoService' correspon al valor de l'atribut 'name' que hem especificat prèviament al fitxer de configuració.



Cóm es pot veure el WSDL s'ha generat de forma automàtica.

3.2.5. Accés des d'un Client

Si volem accedir al servei publicat podem fer ús de la importació seguint els següents pasos:

1) Definició en el fitxer de configuració de la importació

```
<bean abstract="true" id="importedInterfaceDefinition"
   class="net.opentrends.openframe.services.webservices.impl.ImportedInterfaceImpl
"/>



<bean name="webServicesService"
class="net.opentrends.openframe.services.webservices.impl.WebServicesServiceImpl">
    <property name="importedInterfaces">
       <list>
           <bean parent="importedInterfaceDefinition">
                  <property name="name" value="geoNamesService" />
                  <property name="serviceURL"
   value="http://localhost:9080/openFrame-samples-geo/geoService"/>
                  <property name="localInterface"
   value="net.opentrends.samples.geo.GeoNamesService" />
           </bean>

     </list>
    </property>
```

On dins la propietat 'importedInterfaces' s'ha definit una importació amb la següent informació:

- name: Nom que ens permetrà a partir del mètode 'getService' del bean 'webServicesService' obtenir una referència a la interfície (definida a la propietat 'localInterface' com si d'una classe local es tractés
- localInterface: Nom de la interfície
  serviceUrl: Url del servei corresponent a la url publicada prèviament. Exemple: http://localhost:9080/<nom context app> /geoService.

2) Accés des d'una classe

Una vegada definit això l'accés seria tan simple com l'exemple mostrat a continuació:

```
GeoNamesService geoService =
(GeoNamesService)webServicesService.getWebService("geoNamesService");
Collection locations = geoService.getLocationsPostalCode("08031");
```

S'ha obtingut des del bean 'webServicesService' la interfície 'GeoNamesService' a partir del nom que havíem definit a la propietat 'name' en el pas 1). A partir d'aquest moment es pot fer ús de la interfície com si d'una classe local es tractés.

Encara millor, podem definir el servei sense que es sabés que fem ús del web service definint una injecció de tipus factoria:

```
<bean id="geoNamesService" factory-bean="webServicesService"
```

```
        factory-method="getWebService">
           <constructor-arg><value>geoNamesService</value></constructor-arg>
    </bean>
```

En aquest cas estem especificant que el bean amb id 'geoNamesService' s'obtindrà a partir de la crida "getWebService('geoNamesService')".

3.2.6. Exemple d'ús de Ajax amb el Servei Publicat

NOTA: Tot i no ser objectiu del present document es mostra en aquest apartat cóm podem utilitzar el servei 'geoNamesService' (que permet comunicar-se amb el WebService) per presentar les poblacions a l'usuari segons el codi postal introduit usant Ajax:

En primer lloc, publicarem el servei definit en el pas anterior (<bean id="geoNamesService" factory-bean="webServicesService" ...) introduint al fitxer 'src/main/resources/dwr/dwr.xml' que volem des del client Web accedir a aquest servei:

```
<create creator="spring" javascript="geoNamesService">
          <param name="beanName" value="geoNamesService"/>
</create>
```

En el camp 'javascript' s'indica com a valor el nom del fitxer javascript que es generarà de forma automàtica i que serà usat des del client. Així, en la pàgina JSP hem d'afegir una referència tal i com es mostra a continuació:

```
<script src="<c:url value="/AppJava/dwr/interface/geoNamesService.js"/>">
</script>
```

Una vegada definida aquesta referència, usarem la llibreria 'prototype' per definir una classe que controlarà l'event de canvi de valor en el camp d'introducció del codi postal:

```
❶var PostalCodeWatcher = Class.create();

PostalCodeWatcher.prototype = {

     initialize: function(field) {❷
                 this.field = $(field);
                 this.field.onchange =
this.getLocationsPostalCode.bindAsEventListener(this)❸;
                 },


          getLocationsPostalCode: function(evt)❹ {
                 if ($F(this.field)) {
```

```
                        ❺geoNamesService.getLocationsPostalCode(
                                        $F(this.field),
                                {
                                        ❻callback:function(dataFromServer) {
                                                updateLocations(dataFromServer);
                                        },
                                        timeout:5000
                                }
                        );
                }

        }

};

❻var watcher = new PostalCodeWatcher('zip');
```

És important anotar (consultar la llibreria prototype per a més referència) els següents punts:

❶ Definició d'una classe on s'especificarà el comportament
❷ El mètode initialize defineix els valors de la instància de la classe.
❸ En el valor del camp 'onchange' utilitzem la funció 'bindAsEventListener' per lligar la funció 'getLocationsPostalCode de la classe' com a listener de l'event (en el moment que canvii el valor del component)

```
this.field.onchange = this.getLocationsPostalCode.bindAsEventListener(this);
```

❹ El mètode 'getLocationsPostalCode' defineix el tractament de l'event i és on es farà la crida a la funció del servidor mitjançant Ajax.
❺ Crida a la instància publicada amb DWR (segons s'havia definit al fitxer dwr.xml com 'javascript="geoNamesService"') a un mètode concret passant com a paràmetre el valor del camp

NOTA: La funció $F() de prototype permet obtenir el valor de qualsevol tipus de input (enlloc d'haver d'usar diferents funcions segons el tipus de component).

❻ De forma adicional als paràmetres de la funció cridada podem incorporar més paràmetres. El primer paràmetre 'callback' ens permet especificar quina funció serà cridada quan la crida a la funció finalitzi. El segon paràmetre permet especificar quin timeout de la crida volem utilitzar. Dins el callback s'especifica la crida a la funció 'updateLocations(dataFromServer)' on el paràmetre 'dataFromServer' seran les dades de retorn de la funció.

```
<script>
var locations;
```

```
    function closeSuggestBox() {
      $('suggestBoxElement').innerHTML = '';
      $('suggestBoxElement').style.visibility = 'hidden';
    }


    // remove highlight on mouse out event
    function suggestBoxMouseOut(obj) {
      document.getElementById('pcId'+ obj).className = 'suggestions';
    }

    // the user has selected a place name from the suggest box
    function suggestBoxMouseDown(obj) {
      closeSuggestBox();
      var placeInput = $('city');
      placeInput.value = locations[obj];
    }

    // function to highlight places on mouse over event
    function suggestBoxMouseOver(obj) {
      document.getElementById('pcId'+ obj).className = 'suggestionMouseOver';
    }


    function updateLocations(dataFromServer) {
            $('suggestBoxElement').style.visibility = 'visible';
            $('suggestBoxElement').innerHTML = '<small><i>loading ...</i></small>';

            locations = dataFromServer;
            //alert(DWRUtil.toDescriptiveString(dataFromServer,1));
            if (locations.length > 1) {
                    $('suggestBoxElement').style.visibility = 'visible';
                var suggestBoxHTML  = '';
                // iterate over places and build suggest box content
                for (i=0;i< locations.length;i++) {
                  suggestBoxHTML += "<div class='suggestions' id=pcId" + i + "
onmousedown='suggestBoxMouseDown(" + i + ")' onmouseover='suggestBoxMouseOver(" +
i +")' onmouseout='suggestBoxMouseOut(" + i +")'> " + locations[i] +'</div>';
                }
                   $('suggestBoxElement').innerHTML = suggestBoxHTML;
            } else {

                if (dataFromServer.length == 1) {
                  $("city").value=locations[0];
                }
              closeSuggestBox();
            }
    }
</script>
```

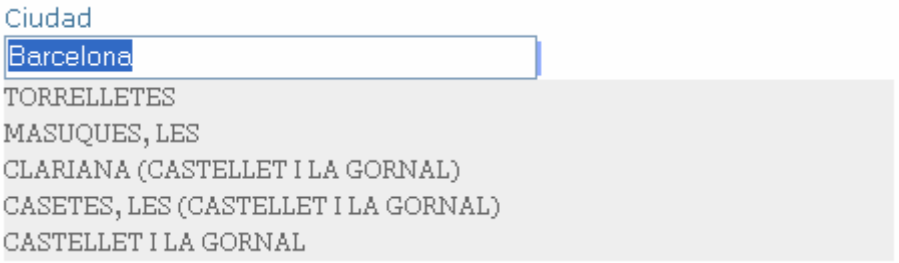
Degut a que la funció ens retorna una col·lecció de poblacions es tracten els casos d'un valor retornat -es copiarà directament al component destí- o de més d'un valor –es mostraran vàries capes per simular una selecció a l'usuari.

Per últim, podem definir l'estil d'aquestes seleccions:

```
<style>
  #suggestBoxElement {border: 1px solid #8FABFF; visibility:hidden; text-align:
left;  white-space: nowrap; background-color: #eeeeee;}
```

```
    .suggestions { font-size: 14;background-color: #eeeeee;  }
    .suggestionMouseOver { font-size: 14;background: #3333ff; color: white;  }
  </style>
```

Exemple de resultat mostrat en introduir un codi postal de múltiples poblacions:



En seleccionar un valor aquest és copiat al component.

4. Annexos