Enterprise

In dit artikel behandelt de auteur XFire, een open source initiatief voor Java webservices. De nadruk ligt op de opzet, architectuur en het toepassen van XFire in applicaties. XFire is een Java SOAP framework, dat een indrukwekkende lijst met specificaties ondersteunt. Daarnaast is dit product gericht op snelheid, interoperabiliteit en flexibiliteit qua omgeving.

Java webservices met XFire

Open source initiatief

Je zou verwachten dat XFire in de lijn ligt van Axis of JBossWS, maar er is een essentieel verschil met deze producten. XFire is niet alleen een container voor webservices maar is in eerste instantie ontwikkeld om gebruikt te worden binnen bestaande containers. Het is daarom mogelijk om het gemakkelijk binnen verschillende omgevingen toe te passen zoals een standaard J2EE-container of Spring framework. Het is daarnaast ook mogelijk om XFire als standalone container te gebruiken. Afgezien van dit belangrijke punt biedt XFire de functionaliteit die je ook vindt in andere bekende webservice engines. Denk hierbij aan ondersteuning van JAX-WS, code-of WSDL-generatie, WS-Security, WS-Addressing en verschillende binding technieken voor POJO's.

Wanneer we XFire vergelijken met de andere SOAP stacks die beschikbaar zijn dan valt op dat het XFire team zich richt op het implementeren van de belangrijke WS-specificaties en dat het gemak van ontwikkelen van webservices zoals nu ook in EE 5 is gepositioneerd hoog in het vaandel heeft staan. Ontwikkelaars die affiniteit hebben met het werken met IOC (inversion of control) containers zoals Spring zullen gemakkelijk XFire in hun bestaande configuraties kunnen gebruiken om willekeurige Java-klassen tentoon te stellen als webservices. Het gaat dan ook meer om configuratie dan het fysieke programmeren. Dat kan de productiviteit verhogen wanneer je de configuratie goed in de vingers hebt.

Het voordeel dat voortvloeit uit de containertransparantie van XFire maakt het voor ons ontwikkelaars gemakkelijk om flexibel gebruik te maken van verschillende security-frameworks en andere eigenschappen die worden geboden door de container die we gebruiken. Uiteraard brengt deze flexibiliteit ook de nodige complexiteit met zich mee. Bij het overwegen van het toepassen van XFire zou dit laatste punt in mijn optiek zwaar moeten meewegen. Frameworks zoals Axis of JBoss kunnen gemak brengen door het totale scala aan functionaliteit wat zij leveren, zij bieden echter minder flexibiliteit wanneer je een bestaande applicatie gemakkelijk wilt uitrusten met webservice technologie.

Het maken van een keuze in het aanbod van verschillende oplossingen zoals Axis, XFire en JBoss hangt sterk af van de niet-functionele randvoorwaarden die gelden voor de applicaties. Zaken als robuustheid, flexibiliteit en het toepassen van beveiliging verschillen per oplossing. Ook zal de aanwezigheid van bestaande applicatieservers hier een belangrijke rol in spelen. Het is verstandig om je eerst te oriënteren welke standaarden je nodig hebt op het gebied van webservices. Dit maakt de keuze uit het beschikbare aanbod gemakkelijker. Een goed overzicht van de eigenschappen en standaarden die de verschillende oplossingen bieden kun je vinden op de website van XFire: http://xfire.codehaus.org/Stack+Comparison.

Wanneer een organisatie op het punt staat webservice-functionaliteit te gaan gebruiken, zou ik aanraden eerst een proof of concept te doen en daarin verschillende producten te gebruiken voor

Ronald van Aken
is werkzaam als consultant bij Sirius ICT solutions BV te Amsterdam

een optimale beoordeling. Dit zorgt ervoor dat de ontwikkelaars goed ervaren hoe het werken met de API's en configuraties van de verschillende oplossingen aanvoelt.

Ontwikkelen webservice

Met XFire kunnen we in principe elke Java-klasse als webservice beschikbaar stellen. Dit gebeurt door een configuratiebestand te specificeren, waarin wordt aangegeven welke services beschikbaar zijn. De services worden vervolgens beschikbaar gemaakt door de XFire servlet-klasse die in de web.xml wordt geconfigureerd. Deze aanpak komt overeen met het interceptor pattern (Figuur 1).

De XFire servlet zal standaard het bestand 'services.xml' zoeken om de configuratie te bepalen voor Java klasses die als webservices beschikbaar moeten worden gemaakt. Afhankelijk van de container zul je een keuze moeten maken welke klasse we als XFire servlet configureren in de web.xml. Het meest voorkomend is het gebruik van de standaard of Spring variant van de XFire servlet. In de volgende code zien we een voorbeeld van een web.xml met daarin een configuratie voor XFire in combinatie met Spring:

```
<!--  start XFire -->
<servlet>
    <servlet-name>XFireServlet</servlet-name>
    <servlet-class>
                        org.codehaus.XFire.spring.
XFireSpringServlet
    </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
      <url-pattern>/servlet/XFireServlet/*</url-pat-
tern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>XFireServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
</servlet-mapping>
<!--  end XFire configuration -->
```

Figuur 2. Web.xml met XFire servlet configuratie.

Java-klasse als webservice

Met gebruik van XFire kunnen we gemakkelijk een Java-klasse als webservice benaderen. Het voorbeeld in dit artikel toont de stappen die nodig zijn om XFire te gebruiken binnen een standaard J2EE-webapplicatie. Voor het gemak hanteer ik hier een bottom-up aanpak, wat inhoudt dat ik bij de klasse begin. We beginnen met het definiëren van een interface voor onze webservice: zie Figuur 3.

```
public interface Registration {
   public String registerMember(Member member);
}
```

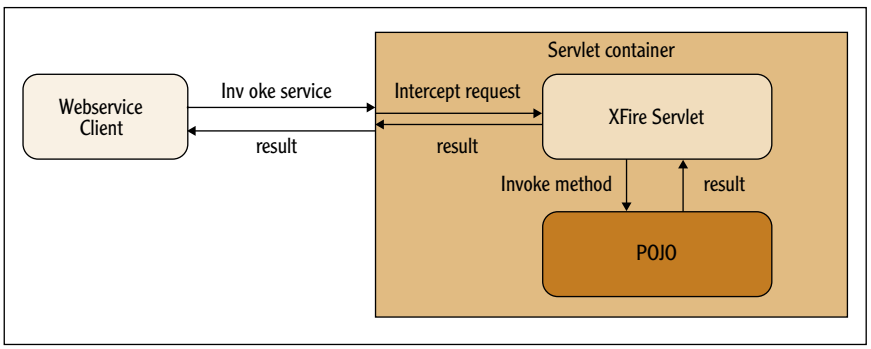
Figuur 3. Het definiëren van een interface voor onze webservice

```
Vervolgens hebben we een implementatie voor deze
interface nodig. De implementatie is eveneens een
normale Java-klasse.

<< begin kader met computercode >>
public class RegistrationImpl implements Registration
{

  public String registerMember(Member member) {
     // RegistrationBO is the class holding the
     // actual business logic         for   regis-
tering a new member
           RegistrationBO  registrationBO  =  new
RegistrationBO();
            String  result  =  registrationBO.
register(member);
     return result;
  }
}
```

Figuur 4. Voorbeeld implementatie voor webservice.

Dit voorbeeld is erg simpel en de Member klasse die als input wordt gegeven is niets meer dan een Java Bean met wat accessors voor naam, e-mail en andere gegevens. Het is echter mogelijk om complexe objecten als input en als output te gebruiken. Eventuele exceptions worden netjes vertaald naar SOAP Faults in de WSDL en aanroep van de service. In dit voorbeeld zorgt XFire ervoor dat het bericht dat binnenkomt wordt vertaald in de gewenste klasse voor de input en zal het resultaat verwerken in het SOAP-bericht wat wordt teruggestuurd.

We kunnen XFire ook zodanig configureren dat het XML-bericht aan onze service wordt doorgegeven en we het parsen zelf kunnen invullen, dit kan vooral bij grote berichten de performance verbeteren. We hebben dan het voordeel dat we alleen hoeven te parsen wat we nodig hebben en we de volledige controle over dat proces hebben. De meeste ontwikkelaars kiezen echter voor de oplossing om de concrete Java-methode beschikbaar te stellen zoals in het voorbeeld.



Figuur 1. Interceptor pattern.

Wanneer de Java-klassen gereed zijn moeten we de configuratie in het services.xml aanpassen wat door de XFire wordt ingelezen. Figuur 5 toont hoe dit in zijn werk gaat.

```
<bean id="RegistrationBeanTarget"
   class="registration.impl.RegistrationImpl">
</bean>

<bean id="bookService"
  class="org.codehaus.XFire.spring.remoting.
XFireExporter">
   <property name="serviceBean" ref="RegistrationBea
nTarget"/>
    <property name="serviceClass"
 value="registration.Registration"/>
</bean>
```

Figuur 5. Voorbeeld services.xml

Een minpunt voor dit product is het feit dat je de XML-configuratie van XFire moet leren

Hiermee is onze webservice klaar om te worden gedeployed. We zullen een keuze moeten maken of we de benodigde JAR-bestanden in het WAR-bestand stoppen of deze in je applicatieserver beschikbaar stellen. Het nadeel als we de JAR bestanden in de WAR bijvoegen is dat dit bestand erg groot wordt, het voordeel is dat we hem dan wel gemakkelijk op elke server kunnen uitrollen.

Client

Het maken van een client is het eenvoudigste als we de WsGen tool gebruiken die wordt meegeleverd met XFire. Uiteraard kun je ook andere frameworks gebruiken om de client-code te maken of de service volledig dynamisch aanroepen door zelf het SOAP-bericht samen te stellen. Ik zal echter in mijn voorbeeld uitgaan van de WSDL en alle benodigde client-code genereren. Ik gebruik hiervoor een ANT-script (Figuur 6) om de benodigde code te genereren.

```
<taskdef name="wsgen" classname="org.codehaus.xfire.
gen.WsGenTask"
              classpathref="xfire.classpath"/>

<target name="generate-client">
    <echo message="Running wsgen"/>
    <wsgen outputDirectory="../client/src"
           generateServerStubs="false"
           wsdl="../res/wsdl/RegistrationImpl.wsdl"
                          package="client.xfire"
overwrite="true"/>
</target>
```

Figuur 6. Voorbeeld client generatie via ANT script

Voordat je WsGen gaat gebruiken, zou ik adviseren om eerst de verschillende argumenten door te nemen. In het voorbeeld verwijs ik naar een WSDL-document dat ik heb opgeslagen in mijn project-directory, je kunt hier ook de URL opgeven van de service. Ik heb de WSDL echter handmatig gemaakt en maak geen gebruik van de WSDL-generatie van XFire.

Nu kunnen we de client-code die is aangemaakt gebruiken om de service aan te roepen. Het voorbeeld in Figuur 7 toont hoe dit in zijn werk gaat.

```
public static void main(String[] args) {
   // RegistrationClient was generated by wsgen
          RegistrationClient   client   =   new
RegistrationClient();

   // RegistrationPortType was generated by wsgen
   RegistrationPortType serv = client.getRegistrati-
onHttpPort();

   // We use the generated Member class from wsgen
   Member member = new Member();
   member.setName("Duke");

   String reply = serv.registerMember(member);
   System.out.println(reply);

}
```

Figuur 7. Gebruik van client skeleton code.

Conclusie

Ik hoop met dit artikel een idee te hebben gegeven dat het ontwikkelen van webservices in Java niet per definitie complex hoeft te zijn. Webservices zijn echter nog steeds een van de meest lastige gebieden door de hoeveelheid externe specificaties die je moet kennen om echt goed gebruik te maken van de frameworks die beschikbaar zijn. Een minpunt voor dit product is het feit dat je de XML-configuratie van XFire moet leren. Afgezien daarvan levert XFire ons veel gemak. Het sluit aan bij de huidige standaarden zodat de services die je ontwikkelt ook echt interoperabel zijn wat betreft client en server.