

Aplicații Integrate pentru Întreprinderi

Laborator 7

15.11.2011

Construirea unui serviciu web folosind tehnologia JAX-WS

Scopul laboratorului îl reprezintă înțelegerea arhitecturii pe care tehnologia JAX-WS o oferă pentru dezvoltarea unui serviciu web, precum și mecanismul în care sunt transmise obiectele spre a fi transmise prin infrastructura de comunicație.

1. Ce este un serviciu web ? Tipuri de servicii web și implementarea lor în Java
2. Ce este JAX-WS ?
3. Proiectarea unui serviciu web
4. Utilizarea JAXB pentru conversia claselor Java în scheme XML
5. Interogarea unui serviciu web prin intermediul unei client și folosind un server de aplicații

1. Ce este un serviciu web ? Tipuri de servicii web și implementarea lor în Java

Un serviciu web e o funcționalitate implementată în cadrul unei aplicații, accesibilă printr-o rețea de calculatoare¹, oferind interoperabilitate între aplicații care rulează pe platforme diferite. O caracteristică a serviciilor web este extensibilitatea (fiind descrise în XML), ceea ce face ca acestea să poată fi procesate cu ușurință. Asocierea mai multor servicii web se urmărește pentru rezolvarea unor probleme complexe. Comunicația între entitățile implicate în procesul de dezvoltarea și interogarea serviciilor web² este realizată prin mesaje care nu presupun cunoașterea capacităților de care dispun platformele pe care rulează acestea.

Implementarea serviciilor web se poate face în mai multe moduri, distingându-se serviciile web de dimensiuni „mari” și serviciile web fără stare (*eng.* RESTful³ web services).

Serviciile web de dimensiuni „mari”, implementate în Java prin tehnologia JAX-WS, folosesc pentru comunicație mesaje XML care respectă standardul SOAP (Simple Object Access Protocol) – pentru a descrie arhitectura și formatul mesajelor. Operațiile pe care le pun la dispoziție serviciile pot fi procesate în mod automat, căci definirea interfețelor (care le conțin) este scrisă în limbajul WSDL⁴ (Web Services Description Language). Atât standardul SOAP folosit pentru formatul mesajelor cât și limbajul de definire a interfețelor WSDL au fost adoptate pe scară largă.

¹ Cel mai frecvent este folosit Internetul, iar ca protocol de transfer al informațiilor HTTP.

² Arhitectura după care sunt modelate serviciile web este cea de client-server.

³ REST = Representational State Transfer. Fiind vorba despre servicii web independente de stare, informații cu privire la aceasta sunt reprezentate în conținutul mesajelor care sunt schimbate între client și server.

⁴ Acest limbaj – bazat pe XML – este folosit pentru a defini din punct de vedere sintactic interfețele (care conțin operațiile implementate de serviciile web).

Un serviciu web modelat după standardul SOAP trebuie să conțină următoarele elemente:

- o interfață care descrie operațiile pe care serviciul web le pune la dispoziția utilizatorilor; se poate folosi WSDL pentru detalii cu privire la mesaje, operații, corespondențe și locația de la care se poate accesa serviciul web⁵;
- arhitectura serviciului web trebuie să răspundă unor cerințe complexe nonfuncționale⁶, stabilind un vocabular pentru acestea;
- atât procesarea cât și invocarea operațiilor trebuie să se facă asincron; există standarde care oferă suport pentru această infrastructură, cum ar fi WSRM (Web Services Reliable Messaging) sau API-uri ca JAX-WS.

Serviciile web fără stare⁷ sunt adecvate unor probleme ad-hoc, folosind standarde W3C sau IETF (ca HTTP, XML, URI, MIME) – deci o infrastructură mai accesibilă – mai degrabă decât servicii bazate pe SOAP⁸, ceea ce scade costul pentru dezvoltarea lor, astfel încât adoptarea lor impune mai puține restricții.

Situațiile în care este utilă construirea unor servicii web fără stare este:

- performanța poate fi îmbunătățită printr-o infrastructură de stocare (*eng.* caching system) de vreme ce informațiile oferite de serviciul web nu sunt generate în mod dinamic⁹;
- atât clientul cât și serverul au o înțelegere (comună) cu privire la contextul și conținutul care este transmis în procesul de interacțiune dintre ele, întrucât nu există o interfață care să descrie (standardizat) informațiile care sunt transmise¹⁰;
- lățimea de bandă este un aspect care trebuie limitat întrucât serviciile web fără stare sunt mai frecvent folosite în sisteme încorporate cum ar fi telefoane mobile sau PDA-uri; în această situație surplusul de informații conținut de mesajele bazate pe XML (obiectele SOAP) limitează performanța, implicând și costuri mai ridicate;
- integrarea se poate realiza cu ușurință folosind tehnologii precum JAX-RS sau AJAX (Asynchronous JavaScript with XML) precum și utilitare ca DWR (Direct Web Remoting) pentru a invoca serviciile din contextul unor aplicații web; fiind expuse prin XML, serviciile web pot fi utilizate fără modificarea semnificativă a arhitecturii unei pagini de Internet, funcționalitatea fiind adăugată cu ușurință.

Serviciile web de dimensiuni mari sunt folosite în probleme specifice aplicațiilor integrate pentru întreprinderi, având cerințe mai mari în privința calității serviciilor, în timp ce serviciile web fără stare sunt folosite pentru probleme de interfațare prin rețea.

⁵ Mesajele SOAP pot fi procesate prin tehnologia JAX-WS fără a pune la dispoziția utilizatorilor specificația WSDL.

⁶ Exemple de astfel de cerințe nonfuncționale ar putea fi: procesarea unor tranzacții, rezolvarea unor aspecte ce vizează securitatea, probleme de sincronizare.

⁷ Un test pentru a determina dacă un serviciu web este independent de stare constă în verificarea comportamentului acestuia în situația repornirii serverului care îl găzduiește.

⁸ În această situație, nu sunt necesare mesaje în format XML sau descrieri ale unor servicii bazate pe WSDL.

⁹ Implementarea unei structuri de stocare temporară trebuie să se facă ținând cont de limitările metodei HTTP GET (în cazul majorității serverelor).

¹⁰ De regulă, producătorii de servicii web fără stare pun la dispoziție și unelte care descriu utilizatorilor interfața acestora în majoritatea limbajelor de programare folosite pe scară largă.

Tehnologia JAX-WS implementează cerințe mai complexe referitoare la calitatea serviciilor (QoS – Quality of Service), mai frecvent întâlnite în aplicațiile integrate pentru întreprinderi, oferind suport pentru mai multe protocoale care oferă standarde pentru securitate și fiabilitate, asigurând interoperabilitatea între servere și clienți conforme cu aceste protocoale pentru servicii web.

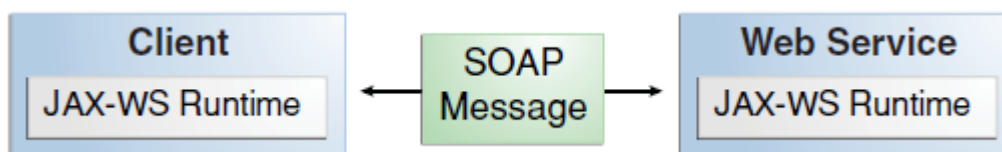
Tehnologia JAX-RS facilitează procesul de dezvoltare a unor aplicații web conforme arhitecturii REST pentru a integra anumite proprietăți în acestea precum: cuplarea slabă (dezvoltarea serverului se poate face fără afectarea clienților existenți) sau scalabilitate, bazându-se pe o arhitectură simplă. Clienții pot folosi o parte sau toate serviciile web puse la dispoziție în timp ce serverul poate fi dezvoltat, integrându-le și cu alte servicii web.

2. Ce este JAX-WS ?

JAX-WS (Java API for XML Web Services) este o tehnologie Java folosită pentru dezvoltarea de servicii web și de aplicații care le invocă, definit de JSR224 comunicația bazându-se pe XML. Utilizatorii pot dezvolta servicii web orientate pe mesaje sau pe apeluri la distanță (RPC – Remote Procedure Call).

O invocare a unui serviciu web este realizată prin protocolul SOAP, care este bazat pe XML. Specificația SOAP include structura antetelor, regulile de codificare și convențiile pentru reprezentarea invocărilor și răspunsurilor serviciului web. Astfel, invocările și răspunsurile sunt transmise ca mesaje SOAP prin protocolul HTTP. Deși mesajele SOAP sunt complexe, acest fapt este ascuns prin API-ul JAX-WS. Pe server sunt definite operațiile serviciului web în cadrul unei interfețe precum și implementarea lor. Un client crează un obiect delegat (obiect local reprezentând serviciul) și invocă metodele puse la dispoziție de serviciul web prin intermediul acestuia. Programatorul nu trebuie să genereze mesaje SOAP și nici să le analizeze, întrucât sistemul de execuție JAX-WS convertește invocările și răspunsurile în și din mesaje SOAP.

Avantajele JAX-WS constau în independența de platformă a limbajului de programare Java și în faptul că entitățile implicate în dezvoltarea și accesarea serviciului web nu trebuie ruleze pe o platformă Java, întrucât sunt folosite tehnologii definite de W3C: HTTP, SOAP și WSDL¹¹.



Comunicația între un serviciu web și un client folosind tehnologia JAX-WS [1]



Exemplu

În cele ce urmează, crearea unui serviciu web folosind tehnologia JAX-WS va fi ilustrată pe cazul particular al aplicației implementând funcționalitatea unui serviciu de programare, a cărui funcționalitate a fost descrisă pe larg în cadrul laboratorului anterior.

¹¹ WSDL specifică un format XML pentru descrierea unui serviciu ca un set de noduri din rețea care realizează anumite operații asupra mesajelor.

3. Proiectarea unui serviciu web

Un serviciu web dezvoltat cu JAX-WS este o clasă Java adnotată cu indicația `@WebService` din pachetul `javax.jws.WebService` aceasta fiind desemnată ca entitatea care deservește serviciul web respectiv (*eng.* endpoint). Așadar, structura prin care este descris un serviciu web (SEI – Service Endpoint Interface / Implementation) este o interfață¹² sau o clasă Java unde sunt declarate metodele pe care un client le poate invoca pentru un serviciu.

```
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import java.util.ArrayList;

@WebService(serviceName = "Programare")
public class Programare
{
    ArrayList<Interval> orar;
    ArrayList<Interval> intervaleOcupate;

    public Programare() {    }

    @WebMethod(operationName = "obțineOrar")
    public ArrayList<Interval> obțineOrar() {    return orar;    }

    @WebMethod(operationName = "obțineIntervaleOcupate")
    public ArrayList<Interval> obțineIntervaleOcupate()
    {    return intervaleOcupate;    }

    @WebMethod(operationName = "obțineIntervaleLibere")
    public ArrayList<Interval> obțineIntervaleLibere()
    {
        ArrayList<Interval> intervaleLibere = new ArrayList<Interval>();
        return intervaleLibere;
    }

    @WebMethod(operationName = "ocupaInterval")
    public boolean ocupaInterval
        (@WebParam(name = "student") int identificador,
         @WebParam(name = "perioada") Interval interval)
    {
        boolean rezultat = false;
        return rezultat;
    }

    @WebMethod(operationName = "elibereazaInterval")
    public boolean elibereazaInterval
        (@WebParam(name = "student") int identificador,
         @WebParam(name = "perioada") Interval interval)
    {
        boolean rezultat = true;
        return rezultat;
    }
}
```

¹² Nu este obligatoriu să se definească o interfață (explicit) atunci când se serviciul web este construit cu JAX-WS, întrucât funcționalitatea este descrisă prin implementarea operațiilor. Totuși, folosind elementul `endpointInterface` în adnotarea `@WebService`, se poate defini explicit interfața, definindu-se metodele la care utilizatorul are acces.

Structurile (clasele, interfețele) care definesc un serviciu web folosind tehnologia JAX-WS trebuie să îndeplinească mai multe condiții:

- clasa care implementează serviciul web trebuie să fie adnotată cu însemnările `javax.jws.WebService` sau `javax.jws.WebServiceProvider`;
- clasa care implementează serviciul web poate referi în mod explicit interfața care descrie acest serviciu web prin elementul `endpointInterface` al însemnării `@WebService`, însă acest lucru nu este obligatoriu: dacă nu este întâlnit nici un element `endpointInterface` în însemnarea `@WebService`, interfața care descrie serviciul web este definită implicit;
- metodele ce implementează funcționalitatea serviciului web trebuie să fie publice și nu trebuie să fie declarate ca fiind `static` sau `final`;
- metodele ce implementează funcționalitatea serviciului web, ce sunt accesibile utilizatorilor trebuie să fie adnotate cu însemnarea `javax.jws.WebMethod`;
- metodele ce implementează funcționalitatea serviciului web trebuie să aibă parametrii și rezultate întoarse de tipuri compatibile cu JAXB;
- clasa care implementează serviciul web nu trebuie să fie declarată ca fiind `final` și nu trebuie să fie abstractă;
- clasa care implementează serviciul web trebuie să implementeze un constructor public implicit;
- clasa care implementează serviciul web nu trebuie să conțină metoda `finalize`;
- clasa care implementează serviciul web poate folosi adnotațiile `javax.annotation.PostConstruct` sau `javax.annotation.PreDestroy` pentru evenimente legate de ciclul de viață al serviciului web:
 - metoda `@PreConstruct` va fi invocată înainte ca structura care implementează serviciul web să răspundă la cereri din partea clienților;
 - metoda `@PreDestroy` va fi invocată înainte ca structura care definește serviciul web să fie își încheie activitatea.

În exemplul de mai sus, clasa care implementează serviciul web, `Programare`, este adnotată ca fiind descriind operațiile acestuia prin însemnarea `@WebService(serviceName="Programare")`. Clasa `Programare` declară mai multe metode (`obțineOrar`, `obțineIntervaleOcupate`, `obțineIntervaleLibere`, `ocupaInterval`, `elibereazaInterval`) cu însemnarea `@WebMethod(operationName="")` prin care acestea sunt făcute vizibile către clienți. Parametrii metodelor accesibile la nivelul clienților trebuie să fie adnotate cu însemnare `@WebParam(name="")`, putându-se specifica un nume sugestiv pentru semnificația argumentelor. Totodată, este necesar ca implementarea să conțină și un constructor implicit¹³ care va fi apelat atunci când un client face o invocare pentru o metodă asociată serviciului web respectiv.

¹³ Pentru exemplul de față, este necesar ca în constructor să se construiască obiectul orar. Preluarea informațiilor dintr-o sursă externă (fișier) pentru obținerea acestui element trebuie să țină cont de faptul că în urma operației de dezvoltare (*eng.* deployment) a serviciului web, aplicația este plasată într-un context al serverului de aplicații care va pune la dispoziție respectivul serviciu web, astfel încât amplasarea sursei externe trebuie să fie cunoscută ca locație (server web sau sistem de fișiere) în mod absolut și nu relativ la adresa unde se găsesc clasele.

4. Utilizarea JAXB pentru conversia claselor Java în scheme XML

JAXB (Java Architecture for XML Binding) oferă o modalitate de conversie între aplicații dezvoltate în limbajul de programare Java și scheme XML. Poate fi folosit independent sau împreună cu JAX-WS, oferind o modalitate de transmitere a mesajelor în cazul utilizării unor servicii web. Astfel, JAX-WS delegă către JAXB maparea tipurilor din limbajul de programare Java¹⁴ (pentru parametri și valori întoarse) cu definiții XML. Dezvoltatorii serviciilor web nu trebuie să cunoască detaliile acestei corespondențe, însă trebuie să cunoască faptul că nu orice clasă din limbajul de programare Java poate fi folosită drept parametru sau valoare întoarsă în JAX-WS.

Conversia scheme XML – limbajul Java

tip schemă XML	tip Java
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

Conversia limbajul Java – scheme XML

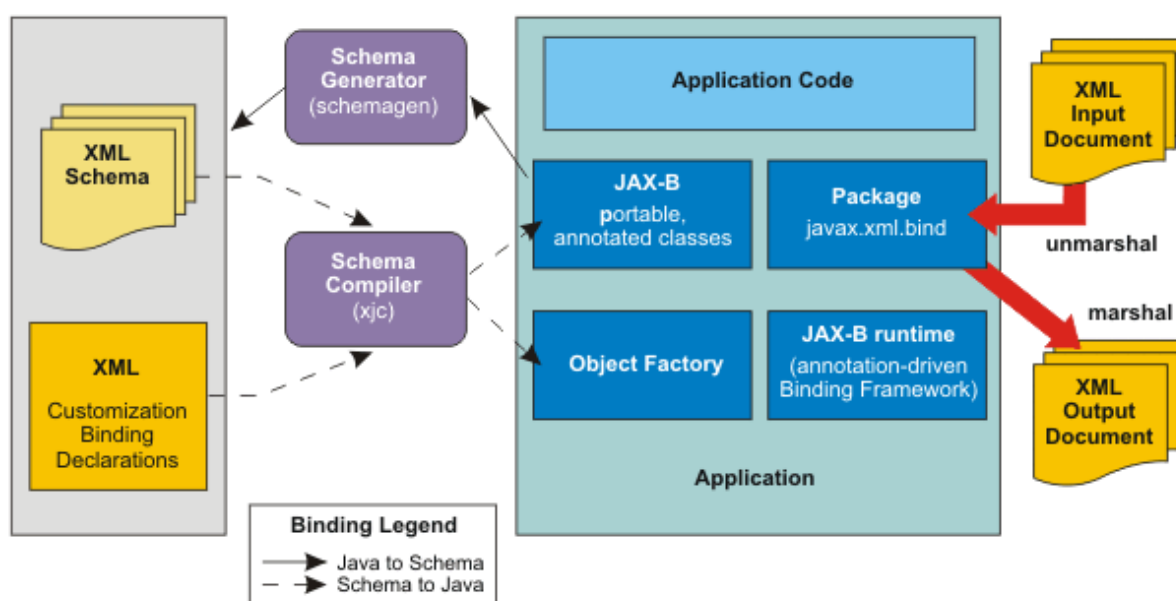
tip Java	tip schemă XML
java.lang.String	xs:string
java.math.BigInteger	xs:integer
java.math.BigDecimal	xs:decimal
java.util.Calendar	xs:dateTime
java.util.Date	xs:dateTime
javax.xml.namespace.QName	xs:QName
java.net.URI xs:string	xs:string
javax.xml.datatype.XMLGregorianCalendar	xs:anySimpleType
javax.xml.datatype.Duration	xs:duration
java.lang.Object	xs:anyType
java.awt.Image	xs:base64Binary
javax.activation.DataHandler	xs:base64Binary
javax.xml.transform.Source	xs:base64Binary
java.util.UUID	xs:string

JAXB definește un set de adnotări astfel încât pentru clase definite de utilizator să se poată realiza conversia la un document XML:

Adnotare	Descriere
@XmlElement(namespace="spatiudenum")	definește elementul rădăcină din cadrul arborelui XML
@XmlAttribute	face conversia între o proprietate JavaBeans sau un câmp al unei clase și un atribut XML
@XmlAccessorType(XMLAccessType.PACKAGE FIELD)	este aplicat unui pachet sau unei clase Java, specificând dacă proprietățile JavaBeans sau attributele vor fi serializate
@XmlType(propOrder={"atr2", "atr1", ... })	permite stabilirea structurii documentului XML prin indicarea ordinii în care attributele clasei vor fi copiate

¹⁴ Trebuie avut în vedere faptul că limbajul de programare Java oferă un set de tipuri de date mult mai bogat decât schemele XML.

JAXB pune la dispoziție un utilitar pentru compilarea unei scheme XML într-o clasă Java – intitulat `xjc` –, un utilitar de generare a unei scheme XML dintr-o clasă Java – denumită `schemagen` și un cadru general de rulare¹⁵. După stabilirea corespondenței dintre schema XML și clasele Java, conversia între ele este realizată în mod automat prin intermediul JAXB¹⁶. Adnotările conțin toate datele necesare pentru ca JAXB să realizeze conversia dintr-un format într-altul, astfel încât obiectele să poată fi transmise prin infrastructura de comunicație. JAXB asigură împachetarea obiectelor Java spre documente XML și despachetarea documentelor XML în instanțe ale claselor Java. Totodată, JAXB poate oferi validarea documentelor XML potrivit schemei generate, astfel încât acestea respectă constrângerile care au fost stabilite. Așadar, JAXB reprezintă tehnologia standard utilizată pentru conversia datelor în cadrul serviciilor web de dimensiuni „mari” implementate prin JAX-WS. Totuși, JAXB poate fi folosit și independent de JAX-WS, atunci când se dorește gestiunea conversiei dintre documente XML și obiecte Java în cadrul unor aplicații informatice.



Arhitectura JAXB

Prin urmare, JAXB este folosit în implementarea serviciilor web prin tehnologia JAX-WS pentru a asigura transportul obiectelor Java (parametrii sau valori întoarse ale metodelor), utilizând un format universal (XML) – independent de platformă, dar și pentru a asigura un mecanism de persistență (prin documente XML), fără a face apel la JPA (Java Persistence API), adecvat stabilirii unei corespondențe între clase Java și baze de date relaționale. Deși este mai rudimentar, această soluție este folosită atunci când starea obiectelor trebuie să fie reținută, iar dimensiunea acestora nu este foarte voluminoasă.

¹⁵ Astfel, se poate porni de la o definiție a unei scheme XML (XSD – XML Schema Definition), creându-se obiecte JavaBeans corespunzătoare (folosind compilatorul pentru scheme XML `xjc`) sau se poate porni de la obiecte JavaBeans creându-se definițiile unei scheme XML corespunzătoare (folosind utilitarul `schemagen`). Atât `xjc` cât și `schemagen` fac parte din distribuția Java, începând cu versiunea 6.

¹⁶ Informațiile reținute în cadrul documentelor XML pot fi accesate fără a fi necesară înțelegerea structurii lor. Clasele Java rezultate pot fi folosite pentru accesarea serviciului web dezvoltat.

În cazul exemplului de mai sus, este necesar ca obiectele cu tipul `Interval` să fie transportate prin infrastructura de comunicație, motiv pentru care definiția clasei trebuie să indice acest lucru:

```
import javax.xml.bind.annotation.XmlAccessType;  
import javax.xml.bind.annotation.XmlAccessorType;  
import javax.xml.bind.annotation.XmlType;  
  
@XmlAccessorType(XmlAccessType.FIELD)  
@XmlType(name="Interval")  
public class Interval  
{  
    ...  
}
```

Prin folosirea adnotării `@XmlAccessorType`, se indică faptul că toate attributele clasei vor fi serializate¹⁷, motiv pentru care nu mai este necesară implementarea interfeței `java.io.Serializable`. Serializarea este realizată prin intermediul unor documente XML, tipul de date (specificat prin adnotarea `@XmlType`) fiind `Interval`. Se asigură totodată faptul că descrierea în limbaj WSDL va cuprinde definițiile metodelor din această clasă, astfel încât ele vor putea fi folosite corespunzător în cadrul aplicațiilor ce accesează funcționalitatea oferită de serviciul web, fiind cuprinse în clasele generate la nivelul clientului.

De asemenea, este necesară asigurarea persistenței pentru două obiecte (având tipul `ArrayList<Interval>`): `orar` și `intervaleOcupate`. Prin JAXB, starea acestor obiecte poate fi reținută prin intermediul unor documente XML pe server, specificându-se un mecanism de transformare. În acest scop, se va defini clasa `IntervalPersistenta`, conținând două attribute `ArrayList<GregorianCalendar>` (pentru care este cunoscut mecanismul de transformare în document XML), adnotările în cadrul acestei clase (`@XmlRootElement`, `@XmlAttribute`) indicând modul în care informațiile din obiecte vor fi transpuse în documentul XML. Totodată, se vor defini metode setter / getter corespunzătoare pentru accesarea informațiilor necesare.

```
@XmlRootElement  
public class IntervalPersistenta  
{  
    @XmlAttribute  
    public ArrayList<GregorianCalendar> intervaleInceput;  
    @XmlAttribute  
    public ArrayList<GregorianCalendar> intervaleSfarsit;  
    ...  
}
```

Astfel, un obiect `ArrayList<Interval>` pentru care nu era cunoscut modul de transformare în document XML¹⁸ este împărțit în două obiecte `ArrayList<GregorianCalendar>`, funcționalitatea fiind asemănătoare.

¹⁷ Attributele clasei sunt de tipul `GregorianCalendar`, care sunt de tip serializabil și pentru care există și corespondența în scheme XML.

¹⁸ Deși modalitatea de transformare este cunoscută pentru clasa `Interval`, trebuie specificat mecanismul de transformare pentru tipul de date `ArrayList<Interval>`, astfel încât operațiile de împachetare și despachetare să fie realizate pentru obiecte de acest fel. Pentru ușurința accesului la datele conținute în obiect, au fost definite două attribute corespunzătoare celor din clasa `Interval`.

Operațiile asupra obiectelor `orar` și `intervaleOcupate` vor implica încărcarea, respectiv descărcarea lor din fișierele care le rețin starea.

```
JAXBContext context;

public Programare()
{
    try
    {
        context = JAXBContext.newInstance(IntervalPersistenta.class);
    }
    catch (Exception e) { }

    if (new File("orar.xml").exists())
        orar = despachetare("orar.xml");
    else
    {
        orar = new IntervalPersistenta();
        impachetare(orar, "orar.xml");
    }

    if (new File("intervaleOcupate.xml").exists())
        intervaleOcupate = despachetare("intervaleOcupate.xml");
    else
    {
        intervaleOcupate = new IntervalPersistenta();
        impachetare(intervaleOcupate, "intervaleOcupate.xml");
    }
}
```

Contextul JAXB referitor la persistență va fi inițializat cu tipul de date pentru care se dorește conversia. Obiectele `orar` și `intervaleOcupate` sunt încărcate din fișier în constructor (în condițiile în care fișierul corespunzător există, altfel aceste obiecte sunt create împreună cu fișierele care le rețin starea), descărcarea stării în fișier urmând a fi realizată în cadrul operațiilor `ocupaInterval` și `elibereazaInterval`, care acționează asupra lor.

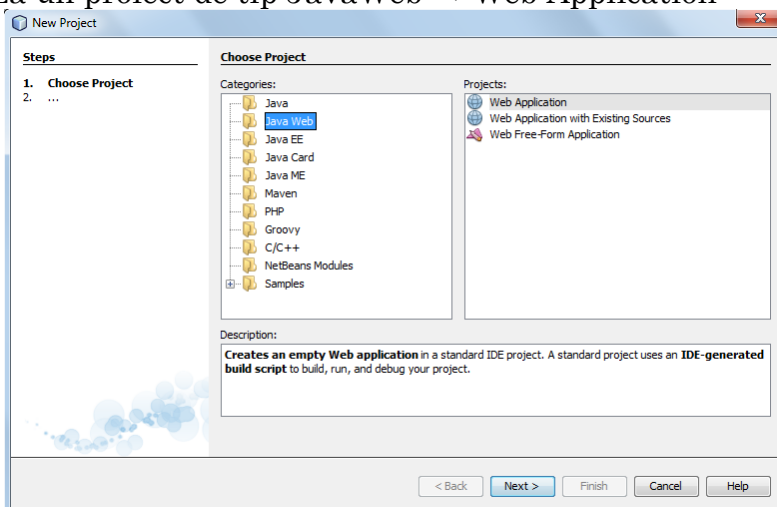
Pentru aceasta, se vor implementa mecanismele corespunzătoare de împachetare și despachetare puse la dispoziție de contextul JAXB.

```
public final void impachetare(IntervalPersistenta obiect, String fisier)
{
    try
    {
        Marshaller conversie = context.createMarshaller();
        conversie.marshal(obiect, new FileOutputStream(fisier));
    }
    catch (Exception e) { }
}

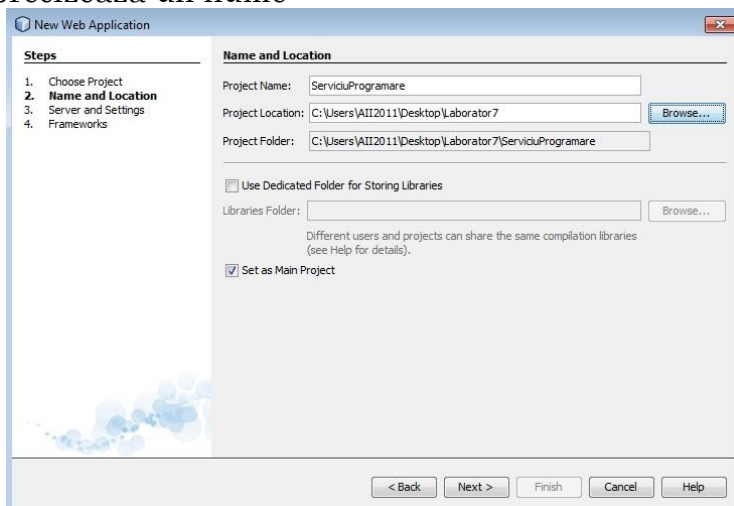
public final IntervalPersistenta despachetare(String fisier)
{
    try
    {
        Unmarshaller conversie = context.createUnmarshaller();
        return (IntervalPersistenta)
            conversie.unmarshal(new FileInputStream(fisier));
    }
    catch (Exception e) { }
    return null;
}
```

Studiu de Caz: Dezvoltarea unui serviciu web folosind NetBeans 7.0.1 și serverul de aplicații GlassFish 3.1

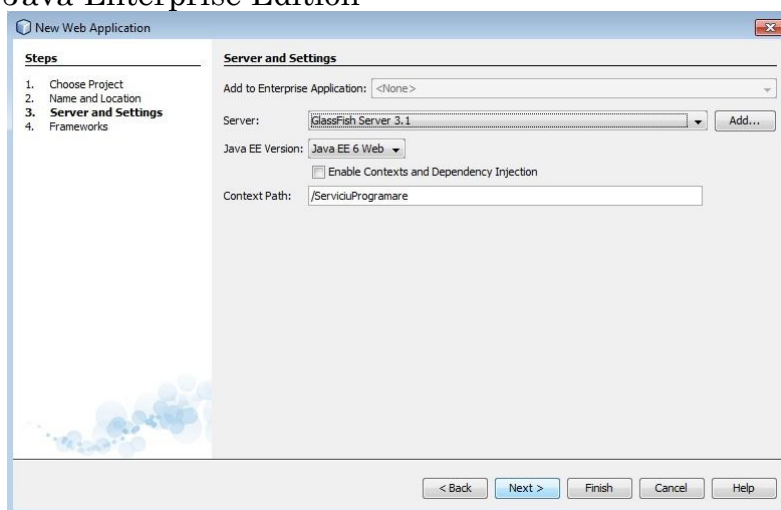
Se crează un proiect de tip JavaWeb → Web Application¹⁹



pentru care se precizează un nume



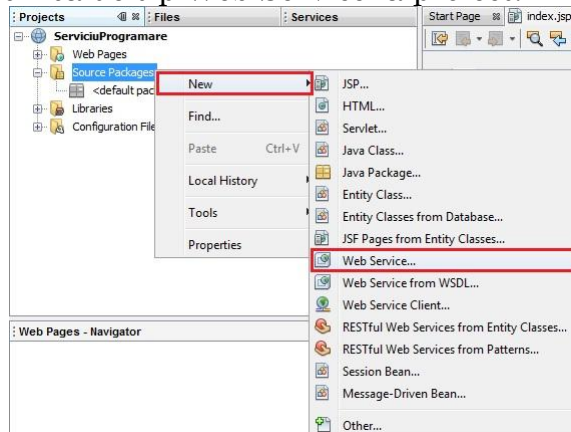
și care va fi dezvoltat folosind ca server de aplicații GlassFish Server 3.1 și versiunea 6 a Java Enterprise Edition



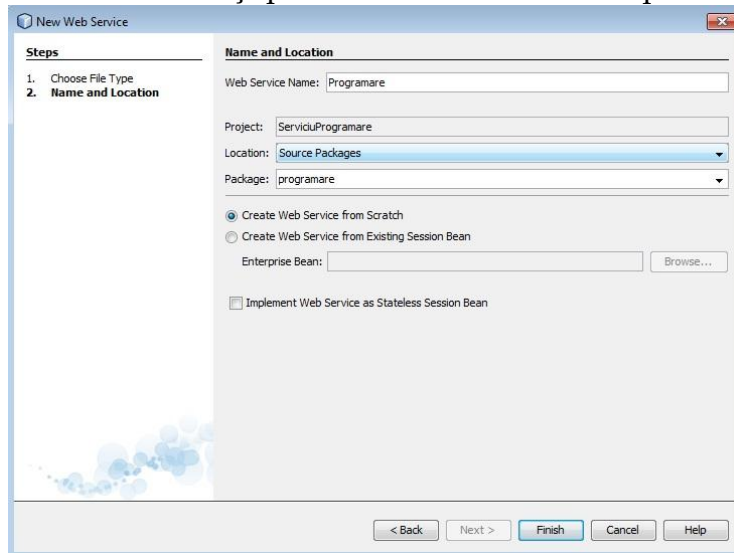
¹⁹ Inițializarea Java Enterprise Edition, realizată în acest moment, poate dura mai mult timp.

Se pot specifica și alte tehnologii cu care serviciul web poate fi integrat (Spring Web MVC, JavaServer Faces, Struts sau Hibernate).

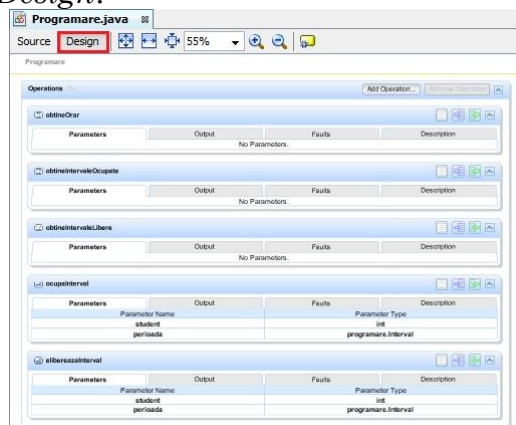
Se adaugă o resursă de tip Web Service la proiect:



pentru care se indică un nume și pachetul din care va face parte



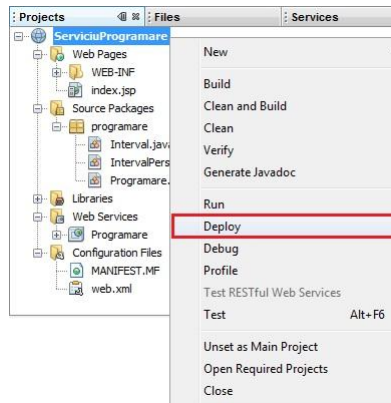
După implementarea funcționalității puse la dispoziție, aceasta poate fi vizualizată din modul *Design*:



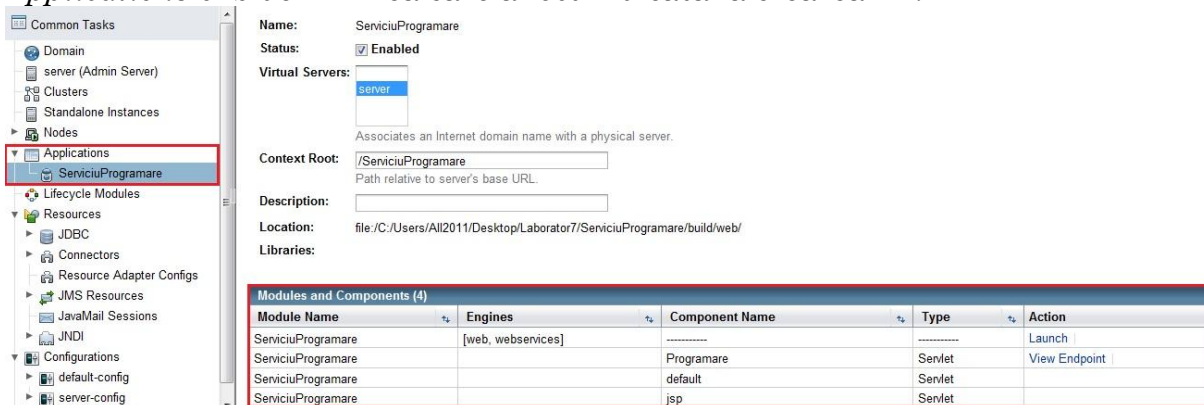
Sunt indicate aici numele și tipul parametrilor acceptați de metodele serviciului web, tipul rezultatului întors, excepțiile care pot fi generate precum și o descriere a funcționalității respective.

În momentul în care este lansat în execuție serviciul web (opțiunea *Deploy* corespunzătoare proiectului), este pornit și serverul de aplicații GlassFish.

Ulterior, modificările asupra codului sursă sunt reflectate pe serverul de aplicații în mod automat atunci când acesta este salvat.



Accesând consola serverului de aplicații GlassFish (la adresa <http://localhost:4848>), poate fi vizualizat serviciul web dezvoltat în secțiunea *Applications* sub denumirea care a fost indicată la crearea lui.



Legătura *ViewEndpoint* corespunzătoare componenteii serviciului web trimite către fișierul WSDL care conține descrierea serviciului web (la adresa <http://localhost:8080/ServiciuProgramare/Programare?wsdl>), permițând și testarea funcționalității acestuia²⁰.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions ... targetNamespace="http://programare/" name="Programare">
<types>
<xsd:schema><xsd:import namespace="http://programare/"
schemaLocation="http://localhost:8080/ServiciuProgramare/Programare?xsd=1" />
</xsd:schema>
</types>
<message name="ocupaInterval">
<part name="parameters" element="tns:ocupaInterval" />
</message>
<message name="ocupaIntervalResponse">
<part name="parameters" element="tns:ocupaIntervalResponse" />
</message>
<portType name="Programare">
<operation name="ocupaInterval">
<input wsam:Action="http://programare/Programare/ocupaIntervalRequest"
message="tns:ocupaInterval" />
<output wsam:Action="http://programare/Programare/ocupaIntervalResponse"
message="tns:ocupaIntervalResponse" />
</operation>
</portType>
```

²⁰ Pentru exemplul de mai sus, testarea funcționalității nu este foarte relevantă întrucât obiectele transmise ca parametrii sau ca valori întoarse nu pot fi vizualizate decât prin intermediul referinței către ele și nu sub aspectul structurii lor.

```
<binding name="ProgramarePortBinding" type="tns:Programare">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="ocupaInterval">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
</binding>
<service name="Programare">
<port name="ProgramarePort" binding="tns:ProgramarePortBinding">
<soap:address location="http://localhost:8080/ServiciuProgramare/Programare" />
</port>
</service>
</definitions>
```

5. Interogarea unui serviciu web prin intermediul unei client și folosind un server de aplicații

Pe baza fișierului WSDL care descrie serviciul web sunt generate²¹ clasele ce permit invocarea funcționalității puse la dispoziție de acesta. Astfel, pentru fiecare metodă implementată de serviciul web (definită ca `operation` în secțiunea `binding` a fișierului WSDL) sunt generate clasele `Operation` și `OperationResponse`. Totodată, sunt generate și clasele corespunzătoare obiectelor transmise ca parametrii sau rezultate întoarse, în care tipul atributelor sunt transformate astfel încât să poată fi transmise prin infrastructura de comunicație²². Invocările metodelor implementate în cadrul serviciului web se fac printr-un obiect local care funcționează ca un delegat pentru serviciul la distanță. Acesta este obținut ca `port`²³ al unei alte clase generate pe baza componentei serviciului web unde metodele sunt definite, `Componenta_Service`.

```
public class Client
{
    public static void main(String[] args)
    {
        programare.Programare_Service serviciu =
            new programare.Programare_Service();
        programare.Programare port = serviciu.getProgramarePort();

        programare.Interval interval = new programare.Interval();

        boolean rezultat = port.ocupaInterval(0,interval);
        System.out.println("rezultat="+rezultat);
    }
}
```

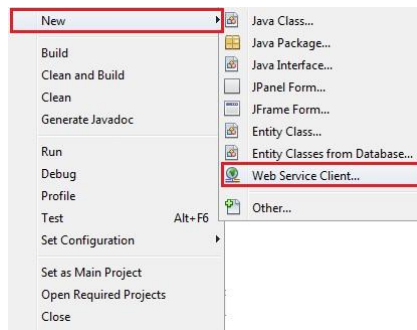
²¹ Pentru generarea unui serviciu web se folosește comanda `wsgen`, în timp ce pentru importarea unui serviciu web (spre a putea fi invocat) se folosește comanda `wsimport`. Utilitățile `wsgen`, respectiv `wsimport` sunt incluse în distribuția standard Java, începând cu versiunea 6.

²² În cazul clasei `Interval`, obiectele de tip `GregorianCalendar` sunt convertite în mod automat la tipul de date echivalent `XMLGregorianCalendar`.

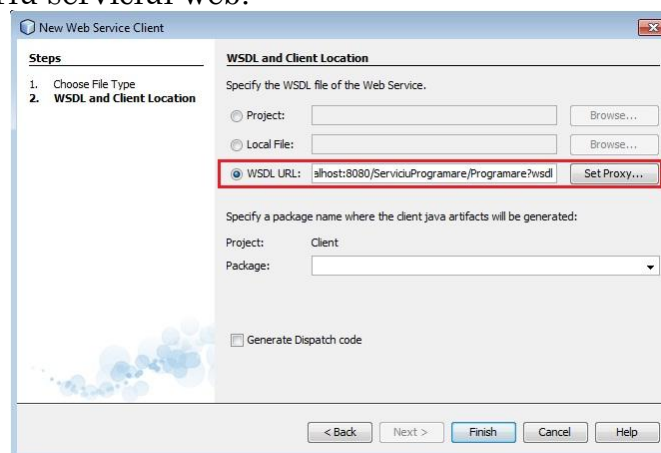
²³ În literatura de specialitate, se folosește denumirea de port pentru obiectul delegat corespunzător serviciului web la distanță, ce conține interfața (SEI – Service Endpoint Interface) definită de acesta.

Studiu de Caz: Dezvoltarea unui client folosind NetBeans 7.0.1 și serverul de aplicații GlassFish 3.1

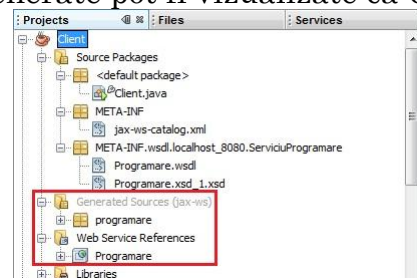
Se crează o aplicație *Java Desktop Application* la care se adaugă un client pentru un serviciu web:



a cărei descriere (locația fișierului WSDL) este specificată pentru generarea claselor care descriu serviciul web:



Clasele ce au fost generate pot fi vizualizate ca *Generated Sources (jax-ws)*.



Dacă se modifică definiția metodelor puse la dispoziție de serviciul web, actualizarea pe client se face prin operația *Refresh* aplicată referinței către serviciul web.

Operațiile pot fi testate din consola serverului de aplicații GlassFish, disponibile la <http://localhost:8080/ServiciuProgramare/Programare?Tester>

Invocarea unei funcționalități în acest caz este întotdeauna însoțită de mesaje SOAP care sunt schimbate între client și server, fiind descrise cererea (invocarea metodei din cadrul serviciului web) precum și răspunsul (rezultatul execuției operației respective în contextul serverului unde rulează serviciul web). Pentru metodele `obțineOrar`, `obțineIntervaleOcupate` și `obțineIntervaleLibere` pot fi vizualizate referințe către obiecte de tip `Interval`, iar metodele `ocupaInterval` și `elibereazaInterval` nu pot fi accesate corespunzător pentru că trebuie specificați ca parametrii referințe către obiecte de tip `Interval`, existente în contextul serverului pe care se află serviciul web.

Programare Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract boolean programare.Programare.ocupaInterval(int,programare.Interval)
ocupaInterval ()

public abstract boolean programare.Programare.elibereazaInterval(int,programare.Interval)
elibereazaInterval ()

public abstract java.util.List programare.Programare.obtineIntervaleOcupate()
obtineIntervaleOcupate ()

public abstract java.util.List programare.Programare.obtineIntervaleLibere()
obtineIntervaleLibere ()

public abstract java.util.List programare.Programare.obtineOrar()
obtineOrar ()

Pentru operația `obtineOrar`, formatul mesajelor SOAP interschimbate între client și server este următorul:

SOAP Request

```
-----  
<?xml version="1.0" encoding="UTF-8"?>  
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
  <S:Header/>  
  <S:Body>  
    <ns2:obtineOrar xmlns:ns2="http://programare/">  
  </S:Body>  
</S:Envelope>
```

SOAP Response

```
-----  
<?xml version="1.0" encoding="UTF-8"?>  
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
  <S:Body>  
    <ns2:obtineOrarResponse xmlns:ns2="http://programare/">  
  </S:Body>  
</S:Envelope>
```



Activitate de Laborator

[1p] 1. Să se genereze un orar în momentul în care este apelată o metodă implementată de `ServiciuProgramare`, în condițiile în care nu există un orar.

[2p] 2. Să se implementeze metodele de conversie între tipurile de date `ArrayList<Interval>` și `IntervalPersistenta`.

[1p] 3. Să se implementeze metoda `obtineIntervaleLibere`.

[2p] 4. Să se implementeze metoda `ocupaInterval`.

[2p] 5. Să se implementeze metoda `elibereazaInterval`²⁴.

Trebuie verificat dacă parametrul `interval` se găsește în obiectul `intervaleOcupate` înainte de a realiza operațiile deja implementate.

[0p] 6. Să se lanseze în execuție `ServiciuProgramare` prin operația *Deploy*.

[1p] 7. Să se apeleze metodele `ocupaInterval` și `elibereazaInterval`.

[1p] 8. Să se afișeze orarul, `intervaleleOcupate` și `intervaleleLibere` prin apelul metodelor `obtineOrar`, `obtineIntervaleOcupate` și `obtineIntervaleLibere`.

²⁴ Operația `remove` pe obiectul `intervaleOcupate` trebuie realizată primind ca parametrul indexul intervalului identificat ca fiind ocupat de student, întrucât referințele obiectelor reținute nu vor fi niciodată aceleași.

Bibliografie

[1] Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, Chinmayee Srivathsa, *The Java EE 6 Tutorial, Basic Concepts*, 4th Edition, Addison Wesley, 2011

[2] *Introducing JAX-WS 2.0 with the Java SE Platform*,
http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/

[3] JAXB –
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Finfo%2Fae%2Fae%2Fcwbs_jaxb.html