Aplicații Integrate pentru Întreprinderi

Laborator 7

18.11.2013

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 şi mecanismul în care sunt transmise obiectele spre a fi transmise prin infrastructura de comunicație.

1. Ce este un serviciu web ? Tipuri de servicii web şi 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 unui client şi folosind un server de aplicații

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

Un serviciu web reprezintă o funcționalitate implementată în cadrul unei aplicații, accesibilă printr-o rețea de calculatoare[1], oferind interoperabilitate între programe ce rulează pe platforme diferite. O caracteristică a serviciilor web este extensibilitatea şi descrierea funcționalității într-un limbaj ce poate fi procesat automat, astfel încât să poată fi procesate cu uşurință. Asocierea (slab cuplată) mai multor servicii web se urmăreşte pentru rezolvarea unor probleme complexe. Astfel, programe ca oferă funcționalități de bază pot interacționa pentru a oferi servicii sofisticate, cu valoare adăugată. Comunicația între entitățile implicate în procesul de implementare şi interogare a serviciilor web[2] este realizată prin mesaje care nu implică cunoaşterea capabilităților 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" şi serviciile web fără stare (*eng.* RESTful[3] 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) – spre a descrie arhitectura şi 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 WSDL[4] (Web Services Description Language). Atât standardul SOAP folosit pentru formatul mesajelor cât şi 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 şi server.
[4] Acest limbaj – bazat pe XML – este folosit spre a defini din punct de vedere sintactic interfețele (ce 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 şi locația de la care se poate accesa serviciul web[5];
- arhitectura serviciului web trebuie să răspundă unor cerințe complexe nonfuncționale[6], stabilind o sintaxă pentru acestea;
- atât procesarea cât şi 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 stare[7] sunt adecvate unor scenarii de integrare ad-hoc, folosind standarde W3C sau IETF (ca HTTP, XML, URI, MIME) – deci o infrastructură mai accesibilă – ceea scade costul pentru dezvoltarea lor astfel că adoptarea lor impune mai puține restricții. Sunt mai bine integrate cu HTTP decât serviciile bazate pe SOAP[8].

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

- performanța poate fi îmbunătățită printr-o infrastructură de stocare (*eng.* caching system) atâta timp cât informațiile oferite de serviciul web nu sunt generate în mod dinamic[9];
- atât clientul cât şi serverul au o înțelegere (comună) cu privire la contextul şi conținutul care este transmis în procesul de interacțiune dintre ele, întrucât nu există o interfață care să descrie (standardizat) informațiile care sunt transmise[10];
- 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 şi costuri mai ridicate;
- integrarea cu diferite aplicații Internet se poate realiza cu uşurință folosind tehnologii precum JAX-RS sau AJAX (Asynchronous JavaScript with XML) ca şi utilitare ca DWR (Direct Web Remoting) spre a invoca serviciile dezvoltate; fiind expuse prin XML, serviciile web pot fi utilizate fără modificarea semnificativă a arhitecturii paginii Internet.

Serviciile web de dimensiuni mari sunt folosite în probleme specifice aplicațiilor integrate pentru întreprinderi, având cerințe mai complexe 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 şi 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 adresare, coordonare, încredere.

[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 găzduieşte.

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

[9] Implementarea unei astfel de structură de stocare 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 şi 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), frecvent întâlnite în aplicațiile integrate pentru întreprinderi, oferind suport pentru mai multe protocoale care oferă standarde pentru securitate şi fiabilitate, asigurând interoperabilitatea între servere şi clienți conforme cu aceste protocoale pentru servicii web.

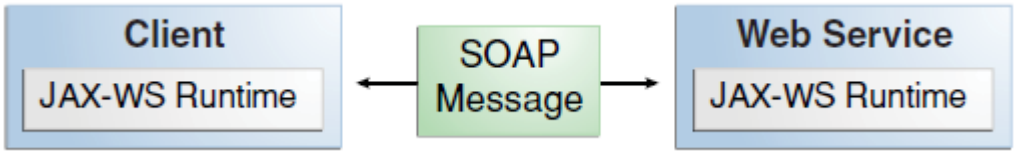
Tehnologia JAX-RS facilitează procesul de dezvoltare a unor aplicații web conforme arhitecturii REST pentru a integra anumite proprietăți în acestea precum: cuplarea slabă (dezvoltarea serverului se poate face fără afectarea clienților existenți) sau scalabilitate, bazându-se pe o arhitectură simplă. Clienții pot folosi o parte sau toate serviciile web puse la dispoziție în timp ce serverul poate fi dezvoltat, integrându-le şi 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 şi 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 şi convențiile pentru reprezentarea invocărilor şi răspunsurilor serviciului web. Astfel, invocările şi răspunsurile sunt transmise ca mesaje SOAP prin protocolul HTTP. Deşi 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 şi implementarea lor. Un client crează un obiect delegat (obiect local reprezentând serviciul) şi invocă metodele puse la dispoziție de serviciul web prin intermediul acestuia. Programatorul nu trebuie să genereze mesaje SOAP şi nici să le analizeze, întrucât sistemul de execuție JAX-WS converteşte invocările şi răspunsurile în şi din mesaje SOAP.

Avantajele JAX-WS constau în independența de platformă a limbajului de programare Java şi în faptul că entitățile implicate în dezvoltarea şi accesarea serviciului web nu trebuie ruleze pe o platformă Java, întrucât sunt folosite tehnologii definite de W3C: HTTP, SOAP şi WSDL[11].



Comunicația între un serviciu web şi 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 aplicații implementând funcționalitatea unui serviciu de rezervare a locurilor la un restaurant, al cărui comportament a fost descris 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 deserveşte serviciul web respectiv (*eng.* endpoint). Aşadar, structura prin care este descris un serviciu web (SEI – Service Endpoint Interface / Implementation) este o intefață[12] sau o clasă Java unde sunt declarate metodele pe care un client le poate invoca pentru un serviciu.

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService(serviceName = "Reservation")
public class Reservation {
    public int                      numberOfSeats;
    public PersistentInterval       timetable;
    public PersistentReservationData reservationData;

    public Reservation() { }

    @WebMethod(operationName = "getTimeTable")
    public ArrayList<Interval> getTimeTable() {
        return persistentInterval2IntervalArrayList(timetable);
    }

    @WebMethod(operationName = "getAvailableSeats")
    public int getAvailableSeats(
            @WebParam(name = "interval") Interval interval) {
        int result = numberOfSeats;
        // ...
        return result;
    }

    @WebMethod(operationName = "makeReservation")
    public boolean makeReservation(
            @WebParam(name = "customerId") int customerId,
            @WebParam(name = "interval") Interval interval,
            @WebParam(name = "noOfSeatsRequired") int noOfSeatsRequired) {
        boolean result = false;
        // ...
        return result;
    }

    @WebMethod(operationName = "cancelReservation")
    public boolean cancelReservation(
            @WebParam(name = "customerId") int customerId,
            @WebParam(name = "interval") Interval interval) {
        boolean result = false;
        // ...
        return result;
    }
}
```

[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. Totuşi, 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 şi 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 şi rezultate întoarse de tipuri compatibile cu JAXB;
- clasa care implementează serviciul web nu trebuie să fie declarată ca fiind `final` şi 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 defineşte serviciul web să fie îşi încheie activitatea.

În exemplul de mai sus, clasa care implementează serviciul web, `Reservation`, este adnotată ca fiind descriind operațiile acestuia prin însemnarea `@WebService(serviceName="Reservation")`. Clasa `Reservation` declară mai multe metode (`getTimetable`, `getAvailableSeats`, `makeReservation`, `cancelReservation`, `getReservation`) fiecare fiind adnotate cu `@WebMethod(operationName="")` prin care acestea sunt făcute vizible către clienți. Parametrii acestor metode accesibile la nivelul clienților trebuie să fie adnotate cu însemnarea `@WebParam(name="")`, putându-se specifica un nume sugestiv pentru semnificația argumentelor. Totodată, este necesar ca implementarea să conțină şi un constructor implicit[13] 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 `timetable`. Preluarea informațiilor dintr-o sursă externă (fişier) 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 fişiere) în mod absolut şi 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 şi 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 deleagă către JAXB maparea tipurilor din limbajul de programare Java[14] (pentru parametri şi 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

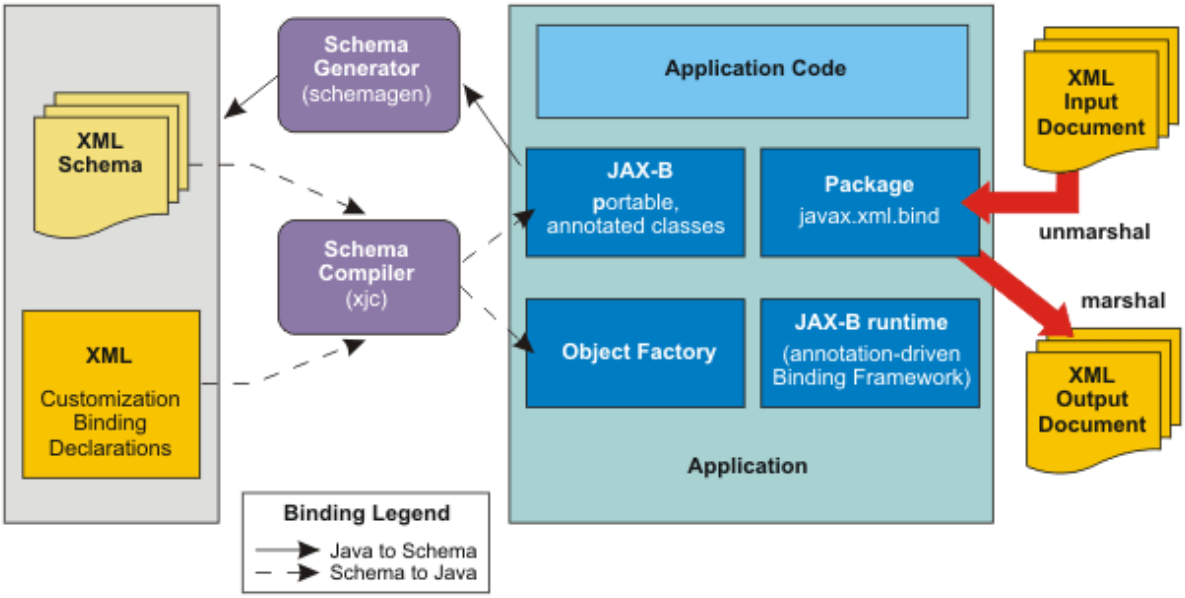
tip Java	tip schemă XML
java.lang.String	xs:string
java.math.BigInteger	xs:integer
java.math.BigDecimal	xs:decimal
java.util.Calendar	xs:dateTime
java.util.Date	xs:dateTime
javax.xml.namespace.QName	xs:QName
java.net.URI xs:string	xs:string
javax.xml.datatype.XMLGregorianCalendar	xs:anySimpleType
javax.xml.datatype.Duration	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 defineşte un set de adnotări astfel încât pentru clase definite de utilizator să se poată realiza conversia la un document XML:

Adnotare	Descriere
`@XmlRootElement(namespace="...")`	defineşte elementul rădăcină din cadrul schemei XML
`@XmlElement`	face conversia între o proprietate JavaBeans sau un câmp al unei clase şi un atribut XML
`@XmlAccesorType(XMLAccessType.PACKAGE|FIELD)`	este aplicat unui pachet sau unei clase Java, specificând daca proprietățile JavaBeans sau atributele vor fi serializate
`@XmlType(propOrder={"atr1","atr2",...,"atrn"})`	permite stabilirea structurii documentului XML prin indicarea ordinii în care atributele 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` şi un cadru general de rulare[15]. După stabilirea corespondenței dintre schema XML şi clasele Java, conversia între ele este realizată în mod automat prin intermediul JAXB[16]. 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 şi 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. Aşadar, JAXB reprezintă tehnologia standard utilizată pentru conversia datelor în cadrul serviciilor web de dimensiuni „mari" implementate prin JAX-WS. Totuşi, JAXB poate fi folosit şi independent de JAX-WS, atunci când se doreşte gestiunea conversiei dintre documente XML şi 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 şi pentru a asigura un mecanism de persistență (prin documente XML), fără a face apel la JPA (Java Persistance API), adecvat stabilirii unei corespondențe între clase Java şi baze de date relaționale. Deşi 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 şi `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 {
      // ...
}
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name="ReservationData")
public class ReservationData {
      // ...
}
```

Prin folosirea adnotării `@XmlAccessorType`, se indică faptul că toate atributele clasei vor fi serializate[17], 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` / `ReservationData`. 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, e necesară asigurarea persistenței pentru obiectele de tipul `ArrayList<Interval>` / `ArrayList<ReservationData>`: `timetable` / `reservationData` Prin JAXB, starea acestor obiecte poate fi reținută prin intermediul unor documente XML pe server, specificându-se un mecanism de transformare. Astfel, se vor defini clasele `PersistentInterval` / `PersistentReservationData`, conținând două atribute `ArrayList<GregorianCalendar>`, respectiv două atribute `ArrayList<Integer>` şi `ArrayList<GregorianCalendar>` (pentru care este cunoscut mecanismul de transformare în document XML), adnotările în cadrul acestei clase (`@XmlRootElement`, `@XmlElement`) indicând modul cum informațiile din obiecte vor fi transpuse în documentul XML. Trebuie definite metode setter / getter corespunzătoare pentru accesarea informațiilor necesare.

```
@XmlRootElement
public class PersistentInterval {
    public ArrayList<GregorianCalendar> startingIntervals;
    public ArrayList<GregorianCalendar> endingIntervals;
    @XmlElement
    public ArrayList<GregorianCalendar> getStartingIntervals() {
        return startingIntervals;
    }
    @XmlElement
    public ArrayList<GregorianCalendar> getEndingIntervals() {
        return endingIntervals;
    }
}
```

[17] Atributele clasei sunt de tipul `GregorianCalendar`, care sunt de tip serializabil şi pentru care există şi corespondența în scheme XML.

```
@XmlRootElement
public class PersistentReservationData {
    private ArrayList<Integer> customerIds;
    private ArrayList<GregorianCalendar> startingIntervals;
    private ArrayList<GregorianCalendar> endingIntervals;
    private ArrayList<Integer> numberOfSeats;
    @XmlElement
    public ArrayList<Integer> getCustomerIds() {
        return customerIds;
    }
    @XmlElement
    public ArrayList<GregorianCalendar> getStartingIntervals() {
        return startingIntervals;
    }
    @XmlElement
    public ArrayList<GregorianCalendar> getEndingIntervals() {
        return endingIntervals;
    }
    @XmlElement
    public ArrayList<Integer> getNumberOfSeats() {
        return numberOfSeats;
    }
}
```

Astfel un obiect de tipul `ArrayList<Interval>` / `ArrayList<ReservationData>` pentru care nu era cunoscut modul de transformare în document XML[18] este împărțit în obiecte de tipul `ArrayList<GregorianCalendar>`, funcționalitatea acestora fiind asemănătoare.

De remarcat faptul că adnotarea `@XmlElement` va fi indicată pentru metoda getter asociată atributului respectiv şi nu pentru atributul în sine[19].

Operațiile asupra obiectelor `timetable` şi `reservationData` vor implica încărcarea, respectiv descărcarea lor din fişierele care le reține starea.

```
JAXBContext contextPersistentInterval;
JAXBContext contextPersistentReservationData;
public Reservation() {
    try {
        contextPersistentInterval =
        JAXBContext.newInstance(PersistentInterval.class);
        contextPersistentReservationData =
        JAXBContext.newInstance(PersistentReservationData.class);
    }
    catch (Exception exception) {
        System.out.println ("exceptie: "+exception.getMessage());
    }
    if (new File("timetable.xml").exists()) {
        timetable = unpackPersistentInterval("timetable.xml");
    } else {
        timetable = new PersistentInterval();
        packPersistentInterval(timetable,"timetable.xml");
    }
```

[18] Deşi modalitatea de transformare este cunoscută pentru clasa `Interval`, trebuie specificat mecanismul de transformare pentru tipul de date `ArrayList<Interval>`, astfel încât operațiile de împachetare şi despachetare să fie realizate pentru obiecte de acest fel. Pentru uşurința accesului la datele conținute în obiect, au fost definite două atribute corespunzătoare celor din clasa `Interval`. Similar pentru `ReservationData` / `ArrayList<ReservationData>`.

[19] Adnotarea `@XmlElement` se precizează pentru metodele getter întrucât acestea sunt publice, spre diferență de atributele în sine care au modul de acces privat. Într-o distribuție anterioară a JAX-WS, adnotarea `@XmlAttribute` se definea pentru atributele clasei.

```
    if (new File("reservationData.xml").exists()) {
        reservationData =
        unpackPersistentReservationData("reservationData.xml");
    } else {
        reservationData = new PersistentReservationData();
        packPersistentReservationData
                    (reservationData,"reservationData.xml");
        }
    }
```

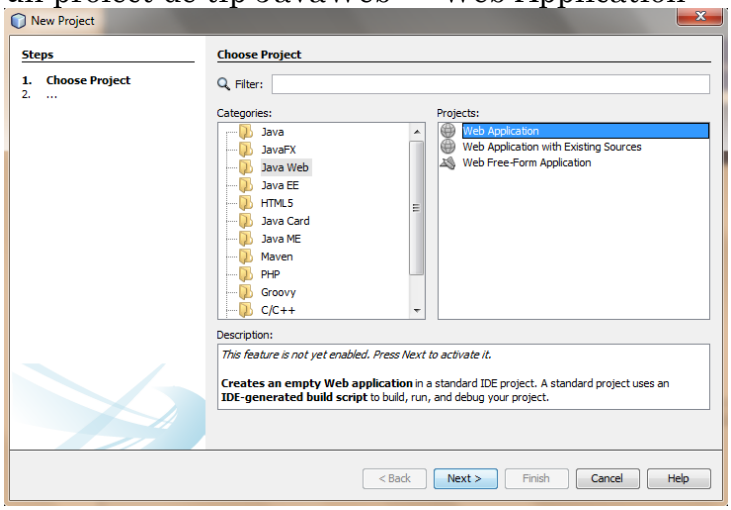
Contextul JAXB referitor la persistență va fi inițializat cu tipul de date pentru care se doreşte conversia. Obiectele `timetable` şi `reservationData` sunt încărcate din fişier în constructor (în condițiile în care fişierul corespunzator există, altfel aceste obiecte sunt create împreună cu fişierele care le reține starea), descărcarea stării în fişier urmând a fi realizată în cadrul operațiilor `makeReservation` şi `cancelReservation`, care acționează asupra lor.

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

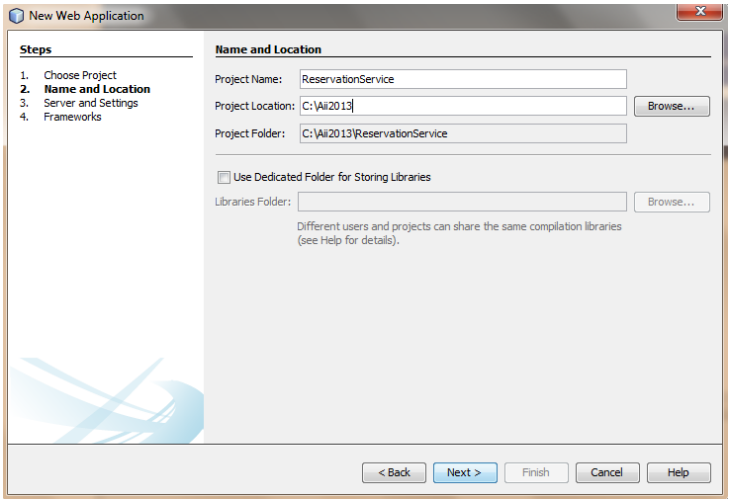
```
    public final void packPersistentInterval
                    (PersistentInterval object, String file) {
        try {
            Marshaller conversion =
            contextPersistentInterval.createMarshaller();
            conversion.marshal(object, new FileOutputStream(file));
        }
        catch (Exception exception) { }
    }
    public final void packPersistentReservationData
                    (PersistentReservationData object, String file) {
        try {
            Marshaller conversion =
            contextPersistentReservationData.createMarshaller();
            conversion.marshal(object, new FileOutputStream(file));
        }
        catch (Exception exception) { }
    }
    public final PersistentInterval unpackPersistentInterval(String file) {
        try {
            Unmarshaller conversion =
            contextPersistentInterval.createUnmarshaller();
            return (PersistentInterval)
                   conversion.unmarshal(new FileInputStream(file));
        }
        catch (Exception exception) { }
        return null;
    }
    public final PersistentReservationData unpackPersistentReservationData
                                          (String file) {
        try {
            Unmarshaller conversion =
            contextPersistentReservationData.createUnmarshaller();
            return (PersistentReservationData)
                   conversion.unmarshal(new FileInputStream(file));
        }
        catch (Exception exception) { }
        return null;
    }
```

Studiu de Caz: Dezvoltarea unui serviciu web folosind NetBeans 7.4 şi serverul de aplicații GlassFish 4.0

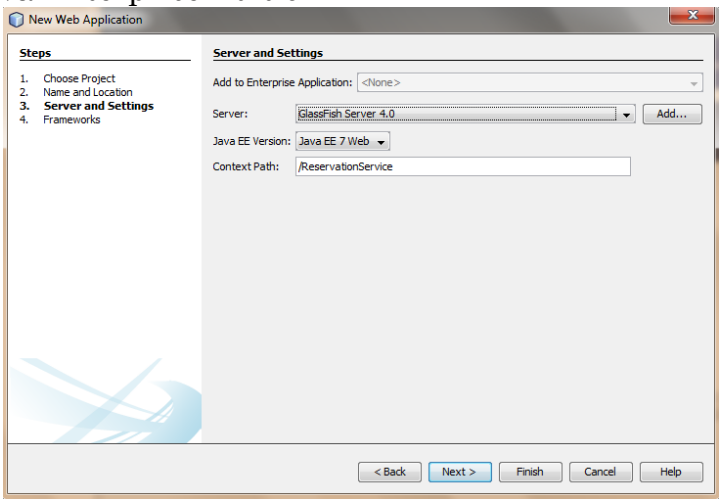
Se crează un proiect de tip JavaWeb → Web Application[20]



pentru care se precizează un nume



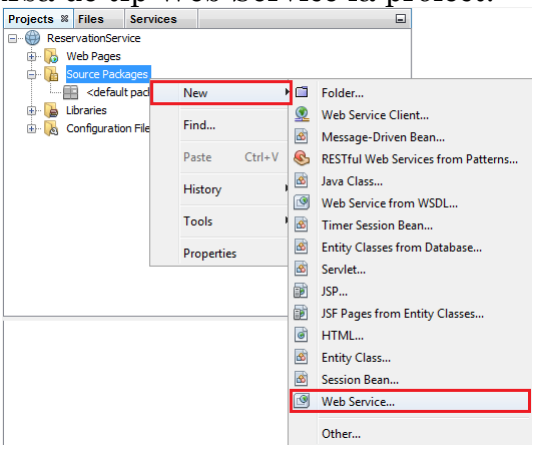
şi care va fi dezvoltat folosind ca server de aplicații GlassFish Server 4.0 şi versiunea 7 a Java Enterprise Edition



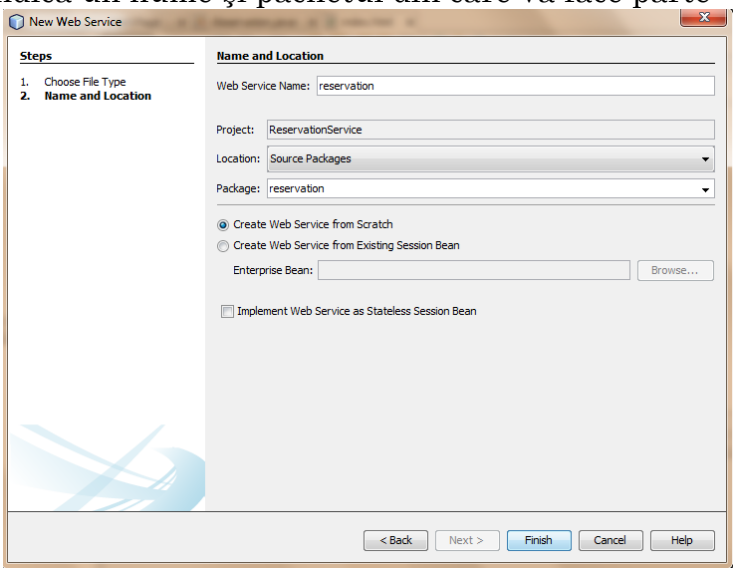
[20] Inițializarea Java Enterprise Edition, realizată în acest moment, poate dura mai mult timp.

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

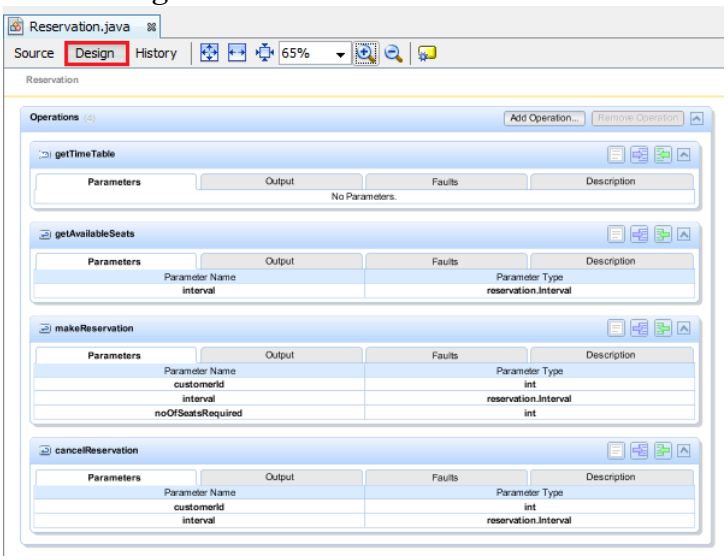
Se adaugă o resursă de tip Web Service la proiect:



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

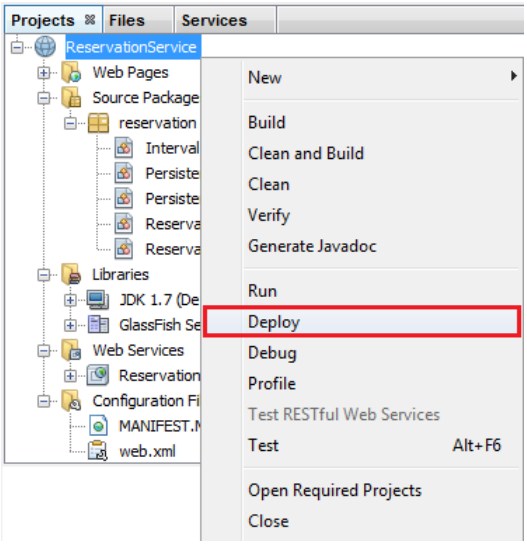


După implementarea funcționalității puse la dispoziție, aceasta poate fi vizualizată din modul *Design*:



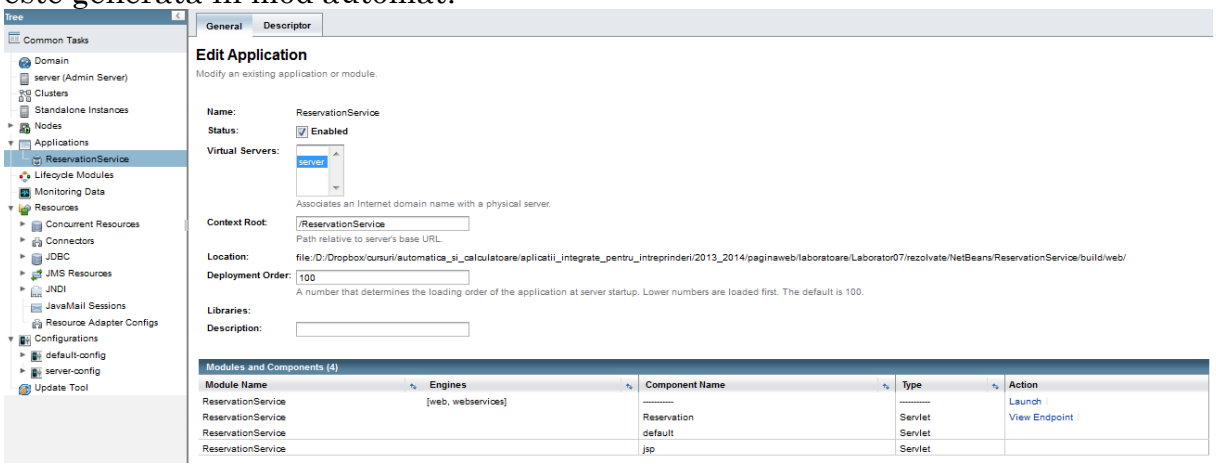
Sunt indicate aici numele şi tipul parametrilor acceptați de metodele serviciului web, tipul rezultatului întors, excepțiile care pot fi generate precum şi o descriere a funcționalității respective.

În momentul în care este lansat în execuție serviciul web (opțiunea *Deploy* corespunzătoare proiectului), este pornit şi 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.

Datele de autentificare pot fi preluate din mediul de dezvoltare NetBeans, accesând Tools → Servers → Glassfish Server 4.0 (tab-ul Common). De regulă, utilizatorul care deține toate drepturile este admin, iar parola asociată acestuia este generată în mod automat.



Legătura *ViewEndpoint* corespunzătoare componentei serviciului web trimite către fişierul WSDL care conține descrierea serviciului web (la adresa http://localhost:8080/ReservationService/Reservation?wsdl), permițând şi testarea funcționalității acestuia[21].

[21] 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 şi nu sub aspectul structurii lor.

```
<definitions targetNamespace="http://reservation/" name="Reservation">
<types>
    <xsd:schema><xsd:import namespace="http://reservation/"
    schemaLocation="http://localhost:8080/ReservationService/Reservation?xsd=1"/>
    </xsd:schema>
</types>
<message name="getTimeTable">
    <part name="parameters" element="tns:getTimeTable"/>
</message>
<message name="getTimeTableResponse">
    <part name="parameters" element="tns:getTimeTableResponse"/>
</message>
<portType name="Reservation">
    <operation name="getTimeTable">
        <input wsam:Action="http://reservation/Reservation/getTimeTableRequest"
        message="tns:getTimeTable"/>
        <output wsam:Action="http://reservation/Reservation/getTimeTableResponse"
        message="tns:getTimeTableResponse"/>
    </operation>
</portType>
<binding name="ReservationPortBinding" type="tns:Reservation">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document"/>
    <operation name="getTimeTable">
        <soap:operation soapAction=""/>
        <input><soap:body use="literal"/></input>
        <output><soap:body use="literal"/></output>
    </operation>
</binding>
<service name="Reservation">
    <port name="ReservationPort" binding="tns:ReservationPortBinding">
        <soap:address
        location="http://localhost:8080/ReservationService/Reservation"/>
    </port>
</service>
</definitions>
```

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

Pe baza fişierului WSDL care descrie serviciul web sunt generate[22] 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 fişierului WSDL) sunt generate clasele `Operation` şi `OperationResponse`. Totodată, sunt generate şi clasele corespunzătoare obiectelor transmise ca parametrii sau rezultate întoarse, în care tipul atributelor sunt transformate astfel încât să poată fi transmise prin infrastructura de comunicație[23]. 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[24] al unei alte clase generate pe baza componentei serviciului web unde metodele sunt definite, `Componenta_Service`.

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

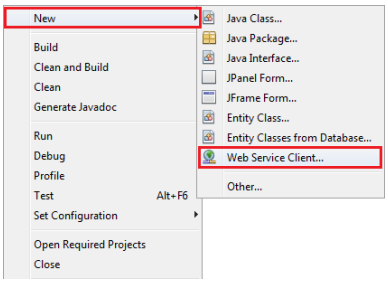
[23] În cazul claselor `Interval / ReservationData`, obiectele de tip `GregorianCalendar` sunt convertite în mod automat la tipul de date echivalent `XMLGregorianCalendar`.

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

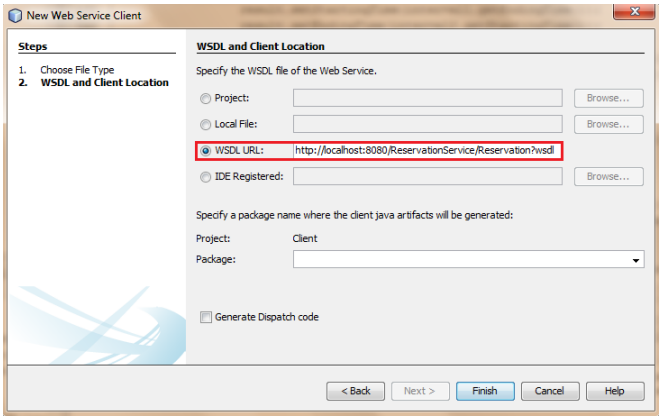
```
public class Client {
    public static void main(String[] args) {
        reservation.Reservation_Service service =
                    new reservation.Reservation_Service();
        reservation.Reservation port = service.getReservationPort();
        List<Interval> timetable = port.getTimeTable();
    }
}
```

Studiu de Caz: Dezvoltarea unui client ce utilizează un serviciu web folosind NetBeans 7.4 şi serverul de aplicații GlassFish 4.0

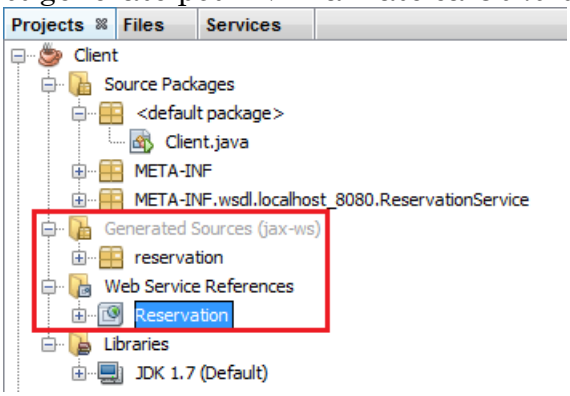
Se crează o aplicație *Java Desktop Application* la care se adaugă un client pentru un serviciu web:



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



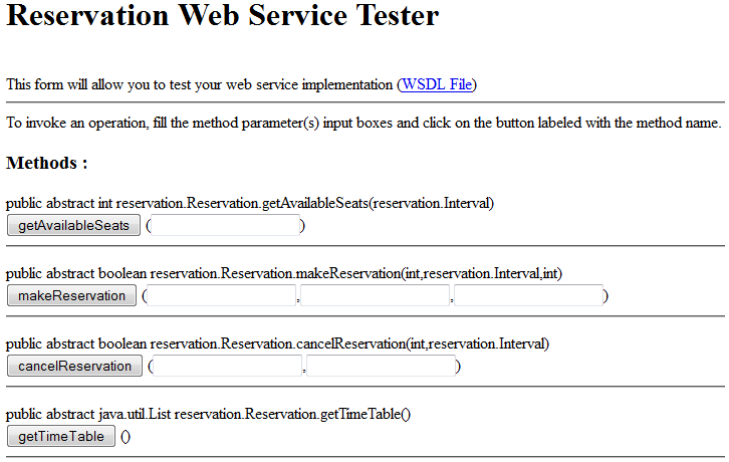
Clasele ce au fost generate pot fi vizualizate ca *Generated Sources (jax-ws)*.



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/ReservationService/Reservation?Tester

Invocarea unei funcționalități în acest caz este întotdeauna însoțită de mesajele SOAP care sunt schimbate între client şi server, fiind descrise cererea (invocarea metodei din cadrul serviciului web) precum şi răspunsul (rezultatul execuției operației respective în contextul serverului unde rulează serviciul web). Pentru metodele `getTimetable`, `getReservation` pot fi vizualizate referințe către obiecte de tip `Interval / ReservationData`, în timp ce metodele `getAvailableSeats` şi `makeReservation / cancelReservation` 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.



Pentru operația `getTimetable`, formatul mesajelor SOAP interschimbate între client şi server este următorul:

SOAP Request

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

SOAP Response

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

Activitate de Laborator

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

Metoda va fi implementată în constructorul clasei `Reservation`. Atunci când este apelată metoda la distanță pusă la dispoziție de serviciul web, mai întâi va fi invocat acest constructor. Orarul va fi încărcat dintr-un fișier de pe server (`timetable.xml`) prin care se asigură persistența datelor – dacă acesta există, iar în caz contrar poate fi generat aleator sau preluat dintr-un alt fișier (resursă) existent pe sever[25]. Acelaşi comportament va fi implementat şi pentru rezervările realizate de clienți, acestea fiind reținute în `reservationData.xml`. În cazul în care acest fișier nu există pe server, el va fi creat fără nici un conținut (nu există încă clienți care să fi invocat metoda `makeReservation`).

Obiectele create în cadrul serverului au un ciclu de viață ce se desfăşoară exclusiv pe perioada în care sunt apelate metodele dezvoltate de serviciul web din contextul clientului, ulterior acestea fiind distruse. De aceea, pentru a asigura persistența informațiilor, este necesar ca starea acestora să fie salvată / încărcată de fiecare dată când se realizează aceste operații. Cum constructorul serviciului web este apelat înainte ca metodele invocate să fie executate, operațiile care vizează asigurarea persistenței sunt realizate aici.

[1p] 2. Să se implementeze metodele de conversie între tipurile de date `ArrayList<Interval>` şi `PersistentInterval`, respectiv `ArrayList<ReservationData>` şi `PersistentReservationData`.

Este suficient să se implementeze conversiile `PersistentInterval` → `ArrayList<Interval>` şi `PersistentReservationData` → `ArrayList<ReservationData>`.

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

[4p] 4. Să se implementeze metoda `getAvailableSeats`.

[1p] 5. Să se implementeze metoda `makeReservation`.

[1p] 6. Să se implementeze metoda `cancelReservation`.

[0p] 7. Să se lanseze în execuție `ReservationService` prin operația *Deploy*.

Să se lanseze în execuție `ReservationService` prin operația *Run as...* → *Java Application*.

[0p] 8. Să se genereze sursele pentru client, accesând *Web Service References* → *Reservation* (right-click) → *Refresh*.

Se va bifa check-box-ul *Also replace local wsdl file with original wsdl located at:* http://localhost:8080/ReservationService/Reservation?wsdl.

Să se genereze sursele pentru client, prin intermediul scriptului `client.[bat|sh]`.

[1p] 9. Să se apeleze metodele `makeReservation` şi `cancelReservation` din client.

Un obiect de tip GregorianCalendar din cadrul clasei Interval va putea fi afişat apelând metoda `toXMLFormat()`.

Un obiect de tip GregorianCalendar care va fi stabilit ca limită a unui interval, va fi creat prin metoda

```
DatatypeFactory.newInstance().newXMLGregorianCalendar(
    new GregorianCalendar(year, month, day_of_month, hour_of_day, minute).
```

[1p] 10. Să se implementeze o nouă metodă în serviciul web `getReservations` care întoarce rezervările realizate de un anumit client, al cărui identificator este precizat ca parametru al metodei.

---

[25] În cazul în care orarul este preluat dintr-un fişier de pe server, acesta trebuie să se găsească:
- în cazul NetBeans:
Windows: C:\Users\Aii2013\AppData\Roaming\NetBeans\7.4\config\GF_4.0\domain1\config
Linux: /home/aii2013/glassfish-4.0/glassfish/domains/domain1/config (dacă mediul de dezvoltare a fost instalat folosit utilizatorul root, el va fi plasat în /usr/local)
- în cazul Eclipse: în rădăcina proiectului.

Bibliografie

[1] Eric Jendrock, Ricardo-Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa, *The Java EE 7 Tutorial, Release 7 for Java EE Platform*, 2013
[2] *Getting started with JAX-WS Web Services*, http://netbeans.org/kb/docs/websvc/jax-ws.html
[3] JAXB – http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Finfo%2Fae%2Fae%2Fcwbs_jaxb.html