

ERNEST SCHEIBER

**PROGRAMARE
DISTRIBUITĂ ÎN JAVA**

Volumul II

Braşov

Prefață

Cursul *Programare distribuită în Java 2* este continuarea cursului *Programare distribuită în Java 1*. Minimal, pentru înțelegerea cursului este nevoie de cunoașterea capitolelor *Servlet* și *JSP* din partea întâi.

Tematica cursului cuprinde:

- WebSocket
- Cadre de dezvoltare (framework) a aplicațiilor Web:
 - *Struts2*
 - *Java Server Faces*
 - *Google Web Toolkit*
- Servicii Web cu:
 - Servicii RPC
 - * *JAX-WS (Java API for XML Web Services)*
 - Servicii REST
 - * *JAX-RS (Java API for RESTful Web Services)*
- Modelul OSGi (Open Source Gateway initiative)

Sursele programelor din curs sunt disponibile prin **git**:

<https://github.com/e-scheiber/DistributedProgramming2.git>

Probleme

- Aplicație GWT dezvoltată prin maven
- Aplicație restygwt-2.2.0

Cuprins

I	WEBSOCKET	11
1	WebSocket	13
1.1	Protocolul WebSocket	13
1.2	Interfața de programare HTML5 de client WebSocket	14
1.3	WebSocket în Java	16
1.3.1	Programare prin adnotări	16
1.3.2	Programare fără adnotări	20
1.3.3	Client Java pentru WebSocket	22
1.3.4	Transmiterea datelor prin adnotarea <code>PathParam</code>	24
1.3.5	Conversie și deconversie prin JSON / XML	27
1.3.6	Streaming	34
II	APLICAȚII WEB - CADRE DE LUCRU	41
2	Aplicații Web	43
2.1	Modelul MVC	43
2.2	Struts 2	46
2.2.1	Aplicații <i>Struts2</i> prin modelul descriptiv	46
2.2.2	Marcaje Struts	49
2.2.3	Aplicații Struts2	51
2.2.4	Interceptori	63
2.2.5	Aplicații <i>Struts2</i> prin modelul programat	70
2.2.6	<i>Struts 2</i> prin <i>maven</i>	74
2.3	Java Server Faces	77
2.3.1	Structura unei aplicații JSF	77
2.3.2	Marcaje JSF	80
2.3.3	Aplicații JSF cu pagini Facelets	82
2.3.4	Aplicații JSF cu pagini JSP	89

2.3.5	Componente grafice <i>Primefaces</i>	101
2.3.6	<i>JSF</i> dezvoltat prin <i>maven</i>	106
3	Google Web Toolkit (GWT)	111
3.1	Dezvoltarea unei aplicații GWT prin <i>ant</i>	111
3.2	Aplicație GWT fără apel de procedură la distanță	113
3.3	Aplicație GWT cu apel de procedură la distanță	120
3.4	Crearea unui widget client	136
3.5	Încărcarea unui fișier - GWT Upload	140
3.6	GWT prin <i>Google AppEngine</i>	143
III	SERVICII WEB	145
4	Servicii RPC	147
4.1	Descrierea unui serviciu JAX-WS	148
4.1.1	XML Schema	149
4.1.2	WSDL	152
4.1.3	Mesaje SOAP	159
4.2	Servicii JAX-WS	164
4.2.1	Serviciu Web ca servlet în <i>apache-tomcat</i> prin <i>ant</i>	164
4.2.2	Componentă EJB sesiune stateless ca serviciu Web	172
4.2.3	Servicii <i>jaxws</i> dezvoltate prin <i>maven</i>	174
5	Servicii JAX-RS	179
5.1	Representational State Transfer	179
5.2	Server <i>Jersey</i> I	180
5.2.1	Generarea resurselor	180
5.2.2	Preluarea parametrilor	185
5.2.3	Aplicație cu server asincron	192
5.2.4	<i>Jersey</i> în <i>JEE</i>	193
5.3	Client <i>Jersey</i>	196
5.4	Server <i>Jersey</i> II	203
5.4.1	Date prin componentă Java	203
5.4.2	Dezvoltare prin <i>maven</i>	209
IV	MODELUL OSGi	213
6	Modelul <i>OSGi</i>	215

6.1	Cadre de lucru OSGi	216
6.2	Programare imperativă - Crearea unui modul OSGi	219
6.3	Exemple	221
6.4	Dezvoltare OSGi prin <i>apache-maven</i>	224
6.5	Programare declarativă	229
6.5.1	Programare declarativă prin <i>Declarative Service</i>	229
6.5.2	Programare declarativă prin <i>Blueprint</i>	232
6.5.3	Programare declarativă prin <i>apache-iPOJO</i>	234
6.6	Serviciul OSGi de jurnalizare	237
6.7	<i>Apache-karaf</i>	238
7	<i>OSGi</i> distribuit	241
7.1	Medii OSGi pentru aplicații distribuite	241
7.2	Servlet ca modul OSGi	242
7.3	Servicii JAXRS cu jersey	246
V	JAVA MANAGEMENT EXTENSIONS	249
8	Java Management Extensions	251
8.1	Standard MBean	252
8.1.1	Crearea unui Standard MBean	252
8.1.2	Crearea unui MBeanServer	254
8.1.3	Notificări	258
8.1.4	Agent MBean	261
8.1.5	Invocarea la distanță	262
	Bibliografie	277

Produsele informatice utilizate

Pe durata existenței, produsele informatice evoluează prin versiunile pe care producătorii ni le pun la dispoziție. Nu de puține ori o versiune nouă nu este compatibilă cu versiunea anterioară, fapt care necesită adaptarea programelor *client*.

Lista următoare precizează versiunile produselor utilizate în lucrare, indicate în majoritatea cazurilor prin resursa de instalare. Lista conține doar produsele care completează pe cele utilizate în *Programare distribuită 1*.

Versiunile produselor informatice utilizate în lucrare		
No.	Produsul informatic	Resursa/versiunea
1	apache-karaf	apache-karaf-4.1.3.tar.gz
2	equinox (OSGi)	org.eclipse.osgi_3.12.0.v20170512-1932.jar
3	Equinox Bridge Servlet	bridge.war
4	felix	org.apache.felix.main.distribution-5.6.8.tar.gz
5	Google Web Toolkit	gwt-2.8.2.zip
6	JAX-RS / jersey	jaxrs-ri-2.26.zip
7	Java Server Faces (JSF)	javax.faces-2.3.2.jar
8	knopflerfish	knopflerfish_osgi_sdk_6.1.1.jar
9	JAX-WS / metro	jaxws-ri-2.2.10, metro-standalone-2.3.1.zip
10	struts	struts2-2.5.13-all.zip

Partea I

WEBSOCKET

Capitolul 1

WebSocket

HTML5 introduce protocolul *WebSocket* pentru comunicații bidireționale între un client HTML5 dintr-un navigator și un server Web. Utilizarea protocolului WebSocket conduce la reducerea semnificativă a traficului în rețea.

Toate navigatoarele importante suportă protocolul WebSocket.

Problematica programării în Java cuprinde:

- Structura unei aplicații server, care revepționează și transmite date;
- Metode de programare a transmițerii și recepționării datelor;
- Aplicații client Java și Web.

1.1 Protocolul WebSocket

Protocolul websocket se inițiază dintr-o comunicație bazată pe protocolul `http` prin mecanismul *upgrade*, introdus de *servlet-api* 3.1: Într-un mesaj `http` trimis de client se indică solicitarea de trecere de la protocolul `http` la protocolul WebSocket prin prezența antetului `Upgrade: websocket`. Exemplul unui asemenea mesaj este¹

```
GET /HelloWebSocket/hello HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: localhost:9090
Origin: null
Sec-WebSocket-Key: tLZ8VGZ8Cw8kt0BvhuV6Vw==
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: x-webkit-deflate-frame
Cookie: JSESSIONID=2BCFF666164139524DD92D573C3859F7;
JSESSIONID=bf12d2417e8eb1bcd6da6137af9d;
treeForm_tree-hi=treeForm:tree:applications
```

¹Mulțimea antetelor depinde de navigatorul utilizat.

Mesajul de răspuns afirmativ este

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ZDE5NmS4T9spIby8/vo/V+rbNAs=
```

Dacă între cei doi parteneri se stabilește acordul (*handshake*) atunci restul comunicațiilor au loc prin intermediul unui soclu TCP pe portul 80.

Astfel ciclul de viață al procesului de comunicație este:

1. Un client solicită printr-un mesaj `http` acordul pentru trecerea la protocolul WebSocket.
2. Serverul răspunde acceptând acordul.
3. Odată stabilită conexiunea, acesta devine bidirecțională (simetrică), clientul și serverul transmit și recepționează mesaje.
4. Una din părți închide conexiunea.

1.2 Interfața de programare HTML5 de client WebSocket

Interfața de programare este definită de un obiect JavaScript:

Constructor

- `WebSocket(in String uri)`
unde *uri* este de forma `ws://host:port/context/numeApel`

Funcții

- `attribute function onopen(evt)`
- `attribute function onmessage(evt)`
- `attribute function onerror(evt)`
- `attribute function onclose(evt)`
- `boolean send(in String data)`
- `close()`

Atribute

- **readyState**

Starea conexiunii:

Valoare	Semnificația
0	Nu s-a stabilit conexiunea
1	Conexiune pregătită pentru comunicații
2	Conexiune în pragul <i>confirmării</i> (<i>handshake</i>)
3	Conexiune închisă și nu mai poate fi redeschisă

- **bufferedAmount**

Numărul octeților trimiși de funcția **send**. Datele sunt codificate UTF-8.

Un șablon de utilizare poate fi

```
<script language="javascript" type="text/javascript">
  var wsUri = "ws://host:8080/context/numeApel";
  var websocket = new WebSocket(wsUri);
  websocket.onopen = function(evt) { . . . };
  websocket.onmessage = function(evt) { . . . };
  websocket.onerror = function(evt) { . . . };
  websocket.onclose = function(evt) { . . . };
  . . .
</script>
```

Expedierea datelor.

Considerăm formularul HTML

```
<form name="myform">
  <input type="text" name="xyz" . . . />
  . . .
  <input type="button" onclick="send"/>
</form>
```

Șablonul funcției *send* este

```
function send(){
  var xyz=document.myform.xyz.value;
  websocket.send(xyz);
}
```

Recepția unui rezultat furnizat de server. Funcția `onmessage` permite recuperarea rezultatului din `evt.data`.

1.3 WebSocket în Java

Interfața de programare Java pentru WebSocket declară clase atât pentru server cât și pentru client.

Servet-API 3.1 conține specificațiile JSR 356 (*Java Specification Request*) pentru WebSocket. O implementare de referință a fost realizată de *Oracle* în pachetul *tyrus* inclusă în *glassfish 4*.

Pentru server două moduri de programare sunt definite :

- Prin utilizarea adnotărilor (*Annotation driven*);
- Fără utilizarea adnotărilor (*Interface driven*).

1.3.1 Programare prin adnotări

Șablonul de programare este

```
@ServerEndpoint(value="/urlPattern")
public class EndpointWebSocketServer {
    private static Set<Session> sessions =
        Collections.synchronizedSet(new HashSet<Session>());

    @OnMessage
    public void myTask(String msg, Session session)
        throws IOException, EncodeException{
        . . .
    }

    @OnOpen
    public void onOpen(Session session){
        sessions.add(session);
    }

    @OnClose
    public void onClose(Session session){
        sessions.remove(session);
    }
}
```

Metoda cu adnotarea `@OnMessage` asigură accesul la datele cererii unui client.

Interfața `javax.websocket.Session`

Metode

- `void addMessageHandler(MessageHandler handler)`
- `RemoteEndpoint.Basic getBasicRemote()`

Prin intermediul unui obiect de tip `RemoteEndpoint.Basic` se programează expedierea răspunsului către client.

- `RemoteEndpoint.Async getAsyncRemote()`
- `void close()`

Interfața `RemoteEndpoint` are subinterfețele

`RemoteEndpoint.Async`
`RemoteEndpoint.Basic`.

Interfața `javax.websocket.RemoteEndpoint.Basic`

Metode

- `void sendObject(Object data) throws IOException, EncodeException`
- `void sendText(String data) throws IOException`
- `void sendBinary(ByteBuffer data) throws IOException`
- `OutputStream getSendStream() throws IOException`

Structura aplicației Web cu desfășurare în *glassfish* este

AppSerser

```
|--> WEB-INF
|      |--> classes
|      |      |      *.class
|      index.html
```

Exemplul 1.3.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicația *server* are codul

```
1 package websocket.cmmdc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.OnOpen;
6 import javax.websocket.OnClose;
7 import javax.websocket.EncodeException;
8 import java.io.IOException;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import java.util.stream.Stream;
13 import javax.websocket.RemoteEndpoint;
14 import java.util.function.BiFunction;
15 //import java.util.concurrent.Future;
```

```

17 @ServerEndpoint(value="/cmmdc")
18 public class CmmdcWebSocketServer{
19     public class CmmdcWebSocketServer{
20         static public long cmmdc(long a,long b){
21             BiFunction<Long,Long,Long> f=(m,n)->{
22                 long r,c;
23                 do{
24                     c=n;
25                     r=m%n;
26                     m=n;
27                     n=r;
28                 }
29                 while(r!=0);
30                 return Long.valueOf(c);
31             };
32             return f.apply(a,b).longValue();
33         }

34     private static Set<Session> sessions =
35         Collections.synchronizedSet(new HashSet<Session>());

36     @OnMessage
37     public void myTask(String msg,Session session)
38         throws IOException,EncodeException{
39         String [] elem=msg.split(":");
40         long m=Long.parseLong(elem[0]);
41         long n=Long.parseLong(elem[1]);
42         System.out.println(m+" : "+n);
43         long r=CmmdcWebSocketServer.cmmdc(m,n);
44         String rez=Long.valueOf(r).toString();
45         /*
46         // Varianta 1
47         for (Session peer : sessions) {
48             if(peer.equals(session)){
49                 peer.getBasicRemote().sendText(rez);
50                 //Future<Void> future=peer.getAsyncRemote().sendText(rez);
51             }
52         }
53         */
54         // Varianta 2
55         Stream<Session> stream=sessions.stream();
56         stream
57             .filter(s->s.equals(session))
58             .forEach(s->{
59                 RemoteEndpoint.Basic endpoint=s.getBasicRemote();
60                 try{
61                     endpoint.sendText(rez);
62                 }
63                 catch(IOException e){};
64             });
65         /*
66         .forEach(s->{
67             Future<Void> future=s.getAsyncRemote().sendText(rez);
68         });
69         */
70     }

71     @OnOpen
72     public void onOpen(Session session){

```

```

76     sessions.add(session);
77 }

79 @OnClose
80 public void onClose(Session session){
81     sessions.remove(session);
82 }
83 }

```

Clientul HTML5/Javascript este

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5     <title>Cmmdc WebSocket</title>

7     <script language="javascript" type="text/javascript">
8       var wsUri = "ws://localhost:8080/CmmdcWebSocket/cmmdc";
9       var websocket = new WebSocket(wsUri);
10      websocket.onopen = function(evt) { onOpen(evt) };
11      websocket.onmessage = function(evt) { onMessage(evt) };
12      websocket.onerror = function(evt) { onError(evt) };
13      websocket.onclose = function(evt) { onClose(evt) };

15      function init() {
16        output = document.getElementById("result");
17      }

19      function send() {
20        var sm=document.cmmmc.m.value;
21        var sn=document.cmmmc.n.value;
22        var msg=sm+":"+sn;
23        websocket.send(msg);
24        writeToScreen("SENT: " + msg);
25      }

27      function onOpen(evt) {
28        writeToScreen("CONNECTED");
29      }

31      function onMessage(evt) {
32        writeToScreen("RECEIVED: " + evt.data);
33      }

35      function onError(evt) {
36        writeToScreen('<span style="color: red;">ERROR:</span> ' +
37          evt.data);
38      }

40      function onClose(evt) {
41        writeToScreen("CLOSED");
42      }

44      function myclose(){
45        websocket.close();
46      }

48      function writeToScreen(message) {

```

```

49     var pre = document.createElement("p");
50     pre.style.wordWrap = "break-word";
51     pre.innerHTML = message;
52     output.appendChild(pre);
53 }

55     window.addEventListener("load", init, false);
56 </script>
57 </head>
58 <body>
59     <h1>CMMDC with WebSocket!!</h1>
60     <div style="text-align: center;">
61         <form name="cmmdc">
62             <table border="1">
63                 <tr>
64                     <td> Primul num&#259;r </td>
65                     <td> <input type="number" name="m" value="1"
66                         required min="1" /> </td>
67                 </tr>
68                 <tr>
69                     <td> Al doilea num&#259;r </td>
70                     <td> <input type="number" name="n" value="1"
71                         required min="1" /> </td>
72                 </tr>
73                 <tr>
74                     <td>
75                         <input type="button" value="Calculeaza" onclick="send()" />
76                     </td>
77                 <td></td>
78                 </tr>
79                 <tr>
80                     <td>
81                         <input type="button" value="Inchide" onclick="myclose()" />
82                     </td>
83                     <td></td>
84                 </tr>
85             </table>
86         </form>
87     </div>
88     <div id="result"></div>
89 </body>
90 </html>

```

1.3.2 Programare fără adnotări

Clasa `javax.websocket.Endpoint`

Metode

- `void onClose(Session session, CloseReason closeReason)`
- `void onError(Session session, Throwable th)`
- `abstract void onOpen(Session session, EndpointConfig config)`

Interfața `javax.websocket.MessageHandler.Whole<T>`**Metode**

- `void onMessage(T message)`

În funcție de tipul mesajului `T` acesta poate fi

Mesaj text	Mesaj binar
String	ByteBuffer
Reader	byte[]
	InputStream

Pentru același exemplu, fără modificarea clientului componenta server va fi alcătuită din două clase:

- *MyApplicationConfig*

în care se declară clasa care tratează solicitarea clientului și *numele de apel* (urlPattern);

- *CmmdcWebSocketEndpoint*

clasa declarată anterior și care rezolvă solicitarea clientului.

Clasa extinde clasa `javax.websocket.Endpoint`, programatorul implementează metodele `onOpen`, `onClose`, `onError`. În metoda `onOpen`, se utilizează o clasă care implementează interfața `MessageHandler.Whole` prin care avem acces la parametrii cererii.

Codurile sunt:

```

1 import java.util.Set;
2 import javax.websocket.Endpoint;
3 import javax.websocket.server.ServerApplicationConfig;
4 import javax.websocket.server.ServerEndpointConfig;

6 public class MyApplicationConfig implements ServerApplicationConfig {

8     @Override
9     public Set<ServerEndpointConfig> getEndpointConfigs(
10         Set<Class<? extends Endpoint>> set) {
11         return new HashSet<ServerEndpointConfig>() {
12             {
13                 add(ServerEndpointConfig.Builder
14                     .create(CmmdcWebSocketEndpoint.class, "/cmmdc")
15                     .build());
16             }
17         };
18     }

20     @Override
21     public Set<Class<?>> getAnnotatedEndpointClasses(Set<Class<?>> set){

```

```

22     return Collections.emptySet();
23 }
24 }

```

și

```

1 package websocket.cmmdc;
2 import javax.websocket.CloseReason;
3 import javax.websocket.RemoteEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.Endpoint;
6 import javax.websocket.EndpointConfig;
7 import javax.websocket.MessageHandler;
8 import javax.websocket.EncodeException;
9 import java.io.IOException;

11 public class CmmdcWebSocketEndpoint extends Endpoint {

13     public void onOpen (Session session, EndpointConfig config) {
14         final RemoteEndpoint.Basic remote = session.getBasicRemote();
15         session.addMessageHandler (new MessageHandler.Whole<String>() {
16             public void onMessage(String msg) {
17                 try {
18                     String[] elem=msg.split(":");
19                     long m=Long.parseLong(elem[0]);
20                     long n=Long.parseLong(elem[1]);
21                     System.out.println(m+" : "+n);
22                     long r=cmmdc(m,n);
23                     remote.sendText(Long.valueOf(r).toString());
24                 }
25                 catch (Exception ioe) { }
26             }
27         });
28     }

30     public void onClose(Session session, CloseReason closeReason) {
31         System.out.println("Closing: " + closeReason.getReasonPhrase());
32     }

34     public void onError (Session session, Throwable throwable) {
35         System.out.println("Error: " + throwable.getLocalizedMessage());
36     }

38     public long cmmdc(long m, long n){. . .}
39 }

```

1.3.3 Client Java pentru WebSocket

Programarea unui client Java necesită

- clasa `javax.websocket.Endpoint`
- interfața `javax.websocket.Session`

și în plus

Clasa `javax.websocket.ContainerProvider`

Metode

- `public static WebSocketContainer getWebSocketContainer()`

Interfața `javax.websocket.WebSocketContainer`

Metode

- `Session connectToServer(Endpoint endpointInstance, ClientEndpointConfig cec, URI path)` throws `DeploymentException`, `IOException`

Codul unui client Java pentru una din aplicațiile anterioare este

```

1 import java.io.IOException;
2 import java.net.URI;
3 import java.util.Scanner;
4 import java.net.URISyntaxException;
5 import javax.websocket.ContainerProvider;
6 import javax.websocket.DeploymentException;
7 import javax.websocket.WebSocketContainer;
8 import javax.websocket.Session;
9 import javax.websocket.Endpoint;
10 import javax.websocket.EndpointConfig;
11 import javax.websocket.MessageHandler;

12
13 public class WebSocketClient extends Endpoint{
14     private static boolean sfarsit=false;
15     private static String SERVER =
16         "ws://localhost:8080/CmmdcWebSocketAD/cmmdc";

17
18     public static void main(String[] args){
19         Scanner scanner=new Scanner(System.in);
20         System.out.println("m=");
21         long m=scanner.nextLong();
22         String sm=Long.valueOf(m).toString();
23         System.out.println("n=");
24         long n=scanner.nextLong();
25         String sn=Long.valueOf(n).toString();
26         String data=sm+":"+sn;
27         WebSocketContainer container =
28             ContainerProvider.getWebSocketContainer();

29
30         try {
31             Session session = container.connectToServer(
32                 WebSocketClient.class, null, new URI(SERVER));
33             session.getBasicRemote().sendText(data);
34             while(! sfarsit){System.out.println(" ");};
35             session.close();
36         }
37         catch(Exception ex){
38             System.out.println("LocalEndPoint Exception : "+ex.getMessage());
39         }

```

```

40 }
42 public void onOpen(Session session, EndpointConfig config) {
43     session.addMessageHandler(new MessageHandler.Whole<String>() {
44         public void onMessage(String text){
45             System.out.println("Cmmdc : "+text);
46             sfarsit=true;
47         }
48     });
49 }
50 }

```

1.3.4 Transmiterea datelor prin adnotarea PathParam

În varianta cu adnotare utilizând `ServerEndpoint` de forma

```
@ServerEndpoint("/urlPattern/{param}")
```

un parametru dat este accesibil prin

```

@OnOpen
public void onOpen(Session session,
    @PathParam("param") String param) {
    . . .
}

```

Datele fiind disponibile odată cu lansarea cererii clientului acțiunea pentru rezolvarea solicitării se face după adnotarea `@OnOpen` și nu după adnotarea `@OnMessage`.

Exemplul 1.3.2

```

1 package websocket.cmmmc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.OnOpen;
6 import javax.websocket.OnClose;
7 import javax.websocket.OnError;
8 import javax.websocket.RemoteEndpoint;
9 import javax.websocket.EncodeException;
10 import javax.websocket.server.PathParam;
11 import java.io.IOException;
12 import java.util.Set;
13 import java.util.Collections;
14 import java.util.HashSet;
15 import java.util.function.BiFunction;
16
17 @ServerEndpoint(value="/cmmmc/{msg}")

```



```

18 public class CmmdcWebSocketServer{
19     private static Set<Session> sessions =
20         Collections.synchronizedSet(new HashSet<Session>());

22     static public long cmmdc(long a,long b){
23         BiFunction<Long,Long,Long> f=(m,n)->{
24             long r,c;
25             do{
26                 c=n;
27                 r=m % n;
28                 m=n;
29                 n=r;
30             }
31             while(r!=0);
32             return Long.valueOf(c);
33         };
34         return f.apply(a,b).longValue();
35     }

37     @OnMessage
38     public void onMessage(String message,Session session){}

40     @OnOpen
41     public void onOpen(Session session , @PathParam("msg") String msg)
42         throws IOException ,EncodingException{
43         sessions.add(session);
44         String [] elem=msg.split(" ");
45         long m=Long.parseLong(elem[0]);
46         long n=Long.parseLong(elem[1]);
47         long r=CmmdcWebSocketServer.cmmdc(m,n);
48         String rez=Long.valueOf(r).toString();
49         sessions.stream()
50             .filter(s->s.equals(session))
51             .forEach(s->{
52                 RemoteEndpoint.Basic endpoint=s.getBasicRemote();
53                 try{
54                     endpoint.sendText(rez);
55                 }
56                 catch(IOException e){};
57             });
58     }

60     @OnClose
61     public void onClose(Session session){
62         sessions.remove(session);
63     }
64 }

```

cu apelarea din

```

1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5         <title>Cmmdc WebSocket</title>

7         <script language="javascript" type="text/javascript">
8             var websocket;

```

```

10     function init() {
11         output = document.getElementById("result");
12     }

14     function send() {
15         var sm=document.cmmdc.m.value;
16         var sn=document.cmmdc.n.value;
17         var msg=sm+" "+sn;
18         var wsUri = "ws://localhost:8080/CmmdcWebSocketAD/cmmdc/"+msg;
19         websocket = new WebSocket(wsUri);
20         websocket.onopen = function(evt) { onOpen(evt) };
21         websocket.onmessage = function(evt) { onMessage(evt) };
22         websocket.onerror = function(evt) { onError(evt) };
23         websocket.onclose = function(evt) { onClose(evt) };
24     }

26     function myclose(){
27         websocket.close();
28     }

30     function onOpen(evt) {
31         writeToScreen("CONNECTED");
32     }

34     function onMessage(evt) {
35         writeToScreen("RECEIVED: " + evt.data);
36     }

38     function onError(evt) {
39         writeToScreen('<span style="color: red;">ERROR:</span> ' +
40             evt.data);
41     }

43     function onClose(evt) {
44         writeToScreen("CLOSED");
45     }

47     function writeToScreen(message) {
48         var pre = document.createElement("p");
49         pre.style.wordWrap = "break-word";
50         pre.innerHTML = message;
51         output.appendChild(pre);
52     }

54     window.addEventListener("load", init, false);
55 </script>
56 </head>
57 <body>
58 <h1>CMMDC with WebSocket!!</h1>
59 <div style="text-align: center;">
60     <form name="cmmdc">
61         <table border="1">
62             <tr>
63                 <td> Primul num&#259;r </td>
64                 <td> <input type="number" name="m" value="1"
65                     required min="1"/> </td>
66             </tr>
67             <tr>
68                 <td> Al doilea num&#259;r </td>

```

```

69         <td> <input type="number" name="n" value="1"
70             required min="1" /> </td>
71     </tr>
72     <tr>
73         <td>
74             <input type="button" value="Calculeaza" onclick="send()" />
75         </td>
76     </tr>
77 </tr>
78 <tr>
79     <td>
80         <input type="button" value="Inchide" onclick="myclose()" />
81     </td>
82 </tr>
83 </tr>
84 </table>
85 </form>
86 </div>
87 <div id="result"></div>
88 </body>
89 </html>

```

1.3.5 Conversie și deconversie prin JSON / XML

Interfața de programare pentru WebSocket oferă posibilitatea conversiei unei componente Java într-un obiect JSON sau document XML² și transformarea inversă.

Resursele suplimentare sunt

JSON	XML
javax.json.jar	activation-*.jar
	jaxb-api.jar
	jaxb-impl-*.jar

Programatorul trebuie să

- Implementeze interfețele `Encoder.Text<T>` și `Decoder.Text<T>`;
- Declare clasele de conversie și deconversie în adnotarea `@ServerEndpoint`, sub forma

```

@ServerEndpoint(value="/numeApel",
    encoders={ClasaConversie.class, . . .},
    decoders={ClasaDeconversie.class, . . .})

```

Șablonul de implementare al interfeței `Encoder.Text` este

²Parcurgerea acestei secțiuni necesită familiarizarea cu *Java Script Object Notation* JSON și JAXB.

```

import javax.websocket.Encoder;
import javax.websocket.EndpointConfig;
import javax.websocket.EncodeException;

public class MyEncoder implements Encoder.Text<T> {
    @Override
    public void init(EndpointConfig ec) { }

    @Override
    public void destroy() { }

    @Override
    public String encode(T obj) throws EncodeException{
        // Cod de conversie al campurilor obiectului obj in JSON
        return objJsonString;
    }
}

```

Șablonul de implementare al interfeței `Decoder.Text` este

```

import javax.websocket.Decoder;
import javax.websocket.EndpointConfig;
import javax.websocket.DecodeException;

public class MyDecoder implements Decoder.Text<T>{
    @Override
    public void init(EndpointConfig ec) { }

    @Override
    public void destroy() { }

    @Override
    public T decode(String string) throws DecodeException{
        // Instantierea unui obiect de tip T cu datele din string.
    }

    @Override
    public boolean willDecode(String string) {
        return true;
    }
}

```

Exemplul 1.3.3

Fie componenta Java

```

1 import javax.xml.bind.annotation.XmlRootElement;
2
3 @XmlRootElement(name="date")
4 public class CmmdcBean{
5     private long m;
6     private long n;
7     public void setM(long m){
8         this.m=m;
9     }
10
11     public long getM(){
12         return m;

```

```

13 }
14
15 public void setN(long n){
16     this.n=n;
17 }
18
19 public long getN(){
20     return n;
21 }
22 }

```

Aplicația server utilizează doar decodorul - transformarea din JSON / XML în obiect de tip *CmmdcBean*. Transformarea este realizată de serverul Web. Codul serverului este

```

1 package websocket.cmmdc;
2 import javax.websocket.OnMessage;
3 import javax.websocket.server.ServerEndpoint;
4 import javax.websocket.Session;
5 import javax.websocket.OnOpen;
6 import javax.websocket.OnClose;
7 import javax.websocket.EncodeException;
8 import java.io.IOException;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import javax.websocket.CloseReason;
13
14 @ServerEndpoint(value="/cmmdc",
15     decoders={CmmdcBeanJSONDecoder.class})
16 //
17 // Pentru XML se va utiliza
18 // decoders={CmmdcBeanXMLDecoder.class}
19 //
20 public class CmmdcWebSocketServer{
21     private static Set<Session> sessions =
22         Collections.synchronizedSet(new HashSet<Session>());
23
24     @OnMessage
25     public void myTask(Session session, CmmdcBean obj)
26         throws IOException, EncodeException{
27         long m=obj.getM();
28         long n=obj.getN();
29         long r=cmmdc(m,n);
30         for (Session peer : sessions) {
31             if(peer.equals(session)){
32                 peer.getBasicRemote().sendText(Long.valueOf(r).toString());
33             }
34         }
35     }
36
37     @OnOpen
38     public void onOpen(Session session){
39         sessions.add(session);
40     }
41
42     @OnClose
43     public void onClose(Session session){
44         try{

```

```

45         session.close(new CloseReason(CloseReason.CloseCodes.NORMAL_CLOSURE,
46                                     "sesison close"));
47     }
48     catch(IOException e){
49         e.printStackTrace();
50     }
51     sessions.remove(session);
52 }
54 public long cmmdc(long m, long n){. . .}
55 }

```

Codul decodurului din JSON este

```

1 package websocket.cmmdc;
2 import javax.websocket.Decoder;
3 import javax.websocket.EndpointConfig;
4 import javax.websocket.DecodeException;
5 import javax.json.Json;
6 import javax.json.JsonArray;
7 import javax.json.JsonObject;
8 import javax.json.JsonReader;
9 import java.io.StringReader;

11 public class CmmdcBeanJSONDecoder implements Decoder.Text<CmmdcBean>{
12     @Override
13     public void init(EndpointConfig ec) { }

15     @Override
16     public void destroy() { }

18     @Override
19     public CmmdcBean decode(String string) throws DecodeException{
20         StringReader sr=new StringReader(string);
21         JsonReader jsonReader = Json.createReader(sr);
22         JsonObject obj=jsonReader.readObject();
23         jsonReader.close();
24         long m=0,n=0;
25         String sm=obj.getString("m");
26         m=Long.parseLong(sm);
27         String sn=obj.getString("n");
28         n=Long.parseLong(sn);
29         CmmdcBean bean=new CmmdcBean();
30         bean.setM(m);
31         bean.setN(n);
32         return bean;
33     }

35     @Override
36     public boolean willDecode(String string) {
37         return true;
38     }
39 }

```

iar codul decodurului din XML este

```

1 package websocket.cmmdc;
2 import javax.websocket.Decoder;
3 import javax.websocket.EndpointConfig;

```

```

4 import javax.websocket.DecodeException;
5 import javax.xml.bind.JAXBContext;
6 import javax.xml.bind.Unmarshaller;
7 import java.io.StringReader;

9 public class CmmdcBeanXMLDecoder implements Decoder.Text<CmmdcBean>{
10     @Override
11     public void init(EndpointConfig ec) { }

13     @Override
14     public void destroy() { }

16     @Override
17     public CmmdcBean decode(String string) throws DecodeException{
18         StringReader sr=new StringReader(string);
19         CmmdcBean bean=null;
20         JAXBContext jaxbContext;
21         try{
22             jaxbContext=JAXBContext.newInstance(CmmdcBean.class);
23             Unmarshaller unmarshaller=jaxbContext.createUnmarshaller();
24             bean=(CmmdcBean) unmarshaller.unmarshal(sr);
25         }
26         catch(Exception e){
27             e.printStackTrace();
28         }
29         return bean;
30     }

32     @Override
33     public boolean willDecode(String string) {
34         return (string!=null);
35     }
36 }

```

Clasa clientului utilizează doar conversia datelor în obiect JSON sau XML. Codorul în JSON este

```

1 import javax.websocket.Encoder;
2 import javax.websocket.EndpointConfig;
3 import javax.websocket.EncodeException;
4 import javax.json.Json;
5 import javax.json.JsonObject;

7 public class CmmdcBeanJSONEncoder implements Encoder.Text<CmmdcBean> {
8     @Override
9     public void init(EndpointConfig ec) { }

11     @Override
12     public void destroy() { }

14     @Override
15     public String encode(CmmdcBean obj) throws EncodeException{
16         long m=obj.getM();
17         long n=obj.getN();
18         JsonObject jsonObject=Json.createObjectBuilder()
19             .add("m",Long.valueOf(m).toString())
20             .add("n",Long.valueOf(n).toString())
21             .build();
22         String objJsonString=jsonObject.toString();

```

```

23     return objJsonString;
24 }
25 }

```

iar codorul în XML este

```

1 import javax.websocket.Encoder;
2 import javax.websocket.EndpointConfig;
3 import javax.websocket.EncodeException;
4 import javax.xml.bind.JAXBContext;
5 import javax.xml.bind.Marshaller;
6 import java.io.StringWriter;

8 public class CmmdcBeanXMLEncoder implements Encoder.Text<CmmdcBean> {
9     @Override
10    public void init(EndpointConfig ec) { }

12    @Override
13    public void destroy() { }

15    @Override
16    public String encode(CmmdcBean obj) throws EncodeException{
17        JAXBContext jaxbContext=null;
18        StringWriter st=null;
19        try{
20            jaxbContext=JAXBContext.newInstance(CmmdcBean.class);
21            Marshaller marshaller=jaxbContext.createMarshaller();
22            st=new StringWriter();
23            marshaller.marshal(obj,st);
24            System.out.println("Results : "+st.toString());
25        }
26        catch(Exception e){
27            e.printStackTrace();
28        }
29        return st.toString();
30    }
31 }

```

Codul clasei client este

```

1 import java.net.URI;
2 import javax.websocket.ContainerProvider;
3 import javax.websocket.WebSocketContainer;
4 import javax.websocket.Session;
5 import java.util.Scanner;
6 import javax.websocket.Endpoint;
7 import javax.websocket.EndpointConfig;
8 import javax.websocket.MessageHandler;

10 public class WebSocketClient extends Endpoint{
11     private static boolean sfarsit=false;

13     public static void main(String[] args){
14         Scanner scanner=new Scanner(System.in);
15         System.out.println("m=");
16         long m=scanner.nextLong();
17         System.out.println("n=");
18         long n=scanner.nextLong();
19         CmmdcBean obj=new CmmdcBean();

```



```

20     obj.setM(m);
21     obj.setN(n);
22     int tip=1;
23     String server="";
24     do{
25         System.out.println("Encoder Type");
26         System.out.println("1: JSON");
27         System.out.println("2: XML");
28         tip=scanner.nextInt();
29     }
30     while((tip!=1) && (tip!=2));

32     WebSocketContainer container =
33         ContainerProvider.getWebSocketContainer();
34     String request=null;
35     try {
36         if (tip==1){
37             server="ws://localhost:8080/JsonCmmdcWebSocket/cmmdc";
38             CmmdcBeanJSONEncoder encoderJSON=new CmmdcBeanJSONEncoder();
39             request=encoderJSON.encode(obj);
40         }
41         else{
42             server="ws://localhost:8080/XmlCmmdcWebSocket/cmmdc";
43             CmmdcBeanXMLEncoder encoderXML=new CmmdcBeanXMLEncoder();
44             request=encoderXML.encode(obj);
45         }
46         System.out.println(request);
47         Session session=
48             container.connectToServer(WebSocketClient.class, null,
49                 new URI(server));
50         session.getBasicRemote().sendText(request);
51         while(! sfarsit){};
52     }
53     catch(Exception ex){
54         System.out.println("LocalEndPoint Exception : "+ex.getMessage());
55     }
56 }

58 public void onOpen(Session session, EndpointConfig config) {
59     session.addMessageHandler(new MessageHandler.Whole<String>() {
60         public void onMessage(String text){
61             System.out.println("Cmmdc : "+text);
62             sfarsit=true;
63             System.exit(0);
64         }
65     });
66 }
67 }

```

Verificare prin *tyrus-client*

Verificarea funcționării unui server WebSocket se poate realiza cu aplicația *tyrus-client-cli-*.jar* prin

```
java -jar tyrus-client-cli-*.jar ws://host:8080/context/numeApel
```

Comenzile care se pot utiliza se obțin prin comanda `help`.

În cazul aplicației *CmmdcWebSocketAD*, dacă $m = 56$ și $n = 42$ atunci se va cere

```
send 56:42
close
quit
```

1.3.6 Streaming

Prin socluri se pot transmite fluxuri de date (*streaming*). Detalii de programare depind de natura datelor ce trebuie vehiculate (text, sunet, grafică).

Expedierea datelor de către server se realizează prin intermediul unui flux dat de metoda

`OutputStream getSendStream()`

a clasei `javax.websocket.RemoteEndpoint.Basic`.

Exemplul 1.3.4 *Clientul solicită serverului transmisia unui fișier text, cerere care va fi satisfăcută de server. Clientul cunoaște lista fișierelor pe care le poate trimite serverul.*

Aplicația server este pentru sistemul de operare Windows.
Clasa server *TextStreaming*

```
1 import javax.websocket.OnMessage;
2 import javax.websocket.server.ServerEndpoint;
3 import javax.websocket.Session;
4 import javax.websocket.OnOpen;
5 import javax.websocket.OnClose;
6 import javax.websocket.EncodeException;
7 import java.io.IOException;
8 import java.io.OutputStream;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import java.nio.file.Paths;
13 import java.nio.file.Path;
14 import java.nio.file.Files;

16 @ServerEndpoint(value="/text")
17 public class TextStreaming{
18     private static Set<Session> sessions =
19         Collections.synchronizedSet(new HashSet<Session>());

21     @OnMessage
22     public void myTask(String msg,Session session)
23         throws IOException,EncodeException{
24         String fs=System.getProperty("file.separator");
25         int noFile=Integer.parseInt(msg);
26         String fileName="";
```

```

27     switch(noFile){
28     case 1:
29         fileName="capitol.txt";
30         break;
31     case 2:
32         fileName="junit.tex";
33         break;
34     }
35     // Tomcat
36     String path="webapps"+fs+"TextStreaming"+fs+"WEB-INF"+fs+"text"+fs;
37     // Glassfish
38     //String path=".." + fs + "applications" + fs + "TextStreaming" +
39     fs + "WEB-INF" + fs + "text" + fs;
40     Path cale=Paths.get(path+fileName);
41     for (Session peer : sessions) {
42         if(peer.equals(session)){
43             OutputStream out=peer.getBasicRemote().getSendStream();
44             Files.copy(cale,out);
45             out.close();
46         }
47     }
48 }

50 @OnOpen
51 public void onOpen(Session session){
52     sessions.add(session);
53 }

55 @OnClose
56 public void onClose(Session session){
57     sessions.remove(session);
58 }
59 }

```

Clasa clientului Java *WebSocketClient*

```

1  import java.io.IOException;
2  import java.net.URI;
3  import javax.websocket.ContainerProvider;
4  import javax.websocket.DeploymentException;
5  import javax.websocket.WebSocketContainer;
6  import javax.websocket.Session;
7  import java.util.Scanner;
8  import javax.websocket.Endpoint;
9  import javax.websocket.EndpointConfig;
10 import javax.websocket.MessageHandler;
11 import java.io.DataInputStream;
12 import java.io.InputStream;
13 import java.io.ByteArrayInputStream;

15 public class WebSocketClient extends Endpoint{
16     private static boolean sfarsit=false;
17     private static String SERVER="ws://localhost:8080/TextStreaming/text";

19     public static void main(String[] args){
20         Scanner scanner=new Scanner(System.in);
21         System.out.println("Fisierul de accesat:");
22         System.out.println("1: capitol.txt");
23         System.out.println("2: junit.tex");

```

```

24     int noFile=scanner.nextInt();
25     String data=Integer.valueOf(noFile).toString();
26     WebSocketContainer container =
27         ContainerProvider.getWebSocketContainer();
28     try {
29         Session session=
30             container.connectToServer(WebSocketClient.class,null,
31                 new URI(SERVER));
32         session.getBasicRemote().sendText(data);
33         while(! sfarsit){};
34     }
35     catch(Exception ex){
36         System.out.println("LocalEndPoint Exception : "+ex.getMessage());
37     }
38 }

40 // Varianta 1
41 /*
42 public void onOpen(Session session, EndpointConfig config){
43     session.addMessageHandler(new MessageHandler.Whole<byte[]>() {
44         public void onMessage(byte[] msg){
45             try(InputStream is=new ByteArrayInputStream(msg);
46                 DataInputStream in = new DataInputStream(is)){
47                 String s;
48                 while (!(s=in.readUTF()).equals("endOfFile"))
49                     System.out.println(s);
50             }
51             catch(IOException e){
52                 e.printStackTrace();
53             }
54             sfarsit=true;
55         }
56     });
57 }
58 */

60 // Varianta 2
61 public void onOpen(Session session, EndpointConfig config){
62     session.addMessageHandler(new MessageHandler.Whole<byte[]>() {
63         public void onMessage(byte[] msg){
64             System.out.println(new String(msg));
65             sfarsit=true;
66         }
67     });
68 }
69 }

```

Exemplul 1.3.5 *Clientul solicită serverului transmiterea unui fișier grafic, cerere care va fi satisfăcută de server. Clientul cunoaște lista fișierelor pe care le poate trimite serverul.*

Aplicația server este pentru sistemul de operare Windows.
Clasa server *ImageStreaming*

```

1 import javax.websocket.OnMessage;
2 import javax.websocket.server.ServerEndpoint;

```

```

3 import javax.websocket.Session;
4 import javax.websocket.OnOpen;
5 import javax.websocket.OnClose;
6 import javax.websocket.EncodeException;
7 import java.io.IOException;
8 import java.io.OutputStream;
9 import java.util.Set;
10 import java.util.Collections;
11 import java.util.HashSet;
12 import java.nio.file.Paths;
13 import java.nio.file.Path;
14 import java.nio.file.Files;

16 @ServerEndpoint(value="/image")
17 public class ImageStreaming{
18     private static Set<Session> sessions =
19         Collections.synchronizedSet(new HashSet<Session>());

21     @OnMessage
22     public void myTask(String msg, Session session)
23         throws IOException, EncodeException{
24         String fs=System.getProperty("file.separator");
25         int noFile=Integer.parseInt(msg);
26         String fileName="";
27         switch(noFile){
28             case 1:
29                 fileName="brasov.jpg";
30                 break;
31             case 2:
32                 fileName="xml-pic.jpg";
33                 break;
34         }
35         // Tomcat
36         String path="webapps"+fs+"ImageStreaming"+fs+"WEB-INF"+fs+"images"+fs;
37         // Glassfish
38         //String path=".." + fs + "applications" + fs + "ImageStreaming" + fs +
39         "WEB-INF" + fs + "images" + fs;
40         Path cale=Paths.get(path+fileName);
41         for (Session peer : sessions) {
42             if(peer.equals(session)){
43                 OutputStream out=peer.getBasicRemote().getSendStream();
44                 Files.copy(cale, out);
45                 out.close();
46             }
47         }
48     }

50     @OnOpen
51     public void onOpen(Session session){
52         sessions.add(session);
53     }

55     @OnClose
56     public void onClose(Session session){
57         sessions.remove(session);
58     }
59 }

```

Clasa client *WebSocketClient.java*

```

1 import java.io.IOException;
2 import java.net.URI;
3 import javax.websocket.ContainerProvider;
4 import javax.websocket.DeploymentException;
5 import javax.websocket.WebSocketContainer;
6 import javax.websocket.Session;
7 import java.util.Scanner;

9 import javax.websocket.Endpoint;
10 import javax.websocket.EndpointConfig;
11 import javax.websocket.MessageHandler;

13 import java.awt.image.BufferedImage;
14 import javax.imageio.ImageIO;
15 import java.io.InputStream;
16 import java.io.ByteArrayInputStream;
17 import java.awt.Image;

19 public class WebSocketClient extends Endpoint{
20     private static boolean sfarsit=false;
21     private static String SERVER =
22         "ws://localhost:8080/ImageStreaming/image";

24     public static void main(String[] args){
25         Scanner scanner=new Scanner(System.in);
26         System.out.println("Fisierul de descarcata:");
27         System.out.println("1: Imagine din Brasov");
28         System.out.println("2: pic-xml");
29         int noFile=scanner.nextInt();
30         String data=Integer.valueOf(noFile).toString();
31         WebSocketContainer container =
32             ContainerProvider.getWebSocketContainer();

34         try {
35             Session session=
36                 container.connectToServer(WebSocketClient.class, null,
37                     new URI(SERVER));
38             session.getBasicRemote().sendText(data);
39             while(! sfarsit){};
40         }
41         catch(Exception ex){
42             System.out.println("LocalEndPoint Exception : "+ex.getMessage());
43         }
44     }

46     public void onOpen(Session session, EndpointConfig config){
47         session.addMessageHandler(new MessageHandler.Whole<byte[]>() {
48             public void onMessage(byte[] msg){
49                 try{
50                     InputStream in=new ByteArrayInputStream(msg);
51                     BufferedImage bi=ImageIO.read(in);
52                     Image image=(Image)bi;
53                     ShowImage s=new ShowImage(image);
54                     s.show();
55                 }
56                 catch(IOException e){
57                     e.printStackTrace();
58                 }

```

```
59         sfarsit=true;
60     }
61 }));
62 }
63 }
```

Clasa *ShowImage* afișează imaginea descărcată.

Partea II

APLICAȚII WEB CADRE DE LUCRU

Capitolul 2

Aplicații Web

Printre aplicațiile distribuite de tip client server, în care comunicațiile se bazează pe protocolul *http*, se disting

- *Aplicații Web (site)*: Cererea adresată serverului este lansată de o persoană prin intermediul unui site, utilizând un program navigator: *Mozilla Firefox*, *Google Chrome*, *Microsoft InternetExplorer*, *Opera*, *Apple Safari*, etc.
- *Servicii Web*: Cererea către server se face de un program. Aplicația server și client se programează utilizând interfețe de programare specifice.

Sunt cunoscute multe cadre de dezvoltare (framework) a aplicațiilor Web gratuite (<http://java-source.net/open-source/web-frameworks>).

Cadrele de lucru pentru aplicații Web asigură:

- separarea preocupărilor prin modelul **Model–View–Controller** (MVC);
- simplificarea gestiunii legăturilor dintre componentele unei aplicații prin fișiere de configurare sau metode alternative;
- oferirea unor facilități de programare (de exemplu, completarea prin program a unor casete de selecție, încărcarea unui fișier, validarea datelor).

2.1 Modelul MVC

Modelul MVC este o schemă de proiectare a aplicațiilor client-server. Modelul MVC este alcătuită din trei părți:

- Componenta *Model* se referă în principal la datele de intrare și ieșire care determină problema dar conține de obicei și clasele ce rezolvă cererile clienților asigurând funcționalitatea aplicației (business classes).
- Componenta *View* asigură interfața grafică a clientului.
- Componenta *Controller* realizează comunicarea / legătura între clasele și / sau resursele din *View* cu clasele din *Model*.

Bazate pe folosirea tehnologiei servlet-urilor și a JSP, materializarea modelului MVC este bine reprezentată de cadrele de dezvoltare (framework) pentru aplicații Web:

- *Struts*
- *Java Server Faces (JSF)*
- *Google Web Toolkit (GWT)*

care sunt prezentate în continuare.

În fiecare caz, gestiunea activităților se face de către cadrul de lucru, deci are loc inversarea controlului.

În legătură cu MVC sunt modelele *Model-View-ViewModel* - *MVVM* și *Model-View-Presenter* - *MVP*.

Diferența dintre modelul MVP și MVVM constă în faptul că în modelul MVP interacțiunile sunt executate programat iar în modelul MVVM acestea sunt realizate automat prin mecanismul de mapare/legare a datelor ¹.

Diferența față de MVC este pus în evidență în Fig. 2.1.

În Java modelul MVVM este exemplificat de produsul *Dukescript* - *knock-out.js*.

¹<http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/>.

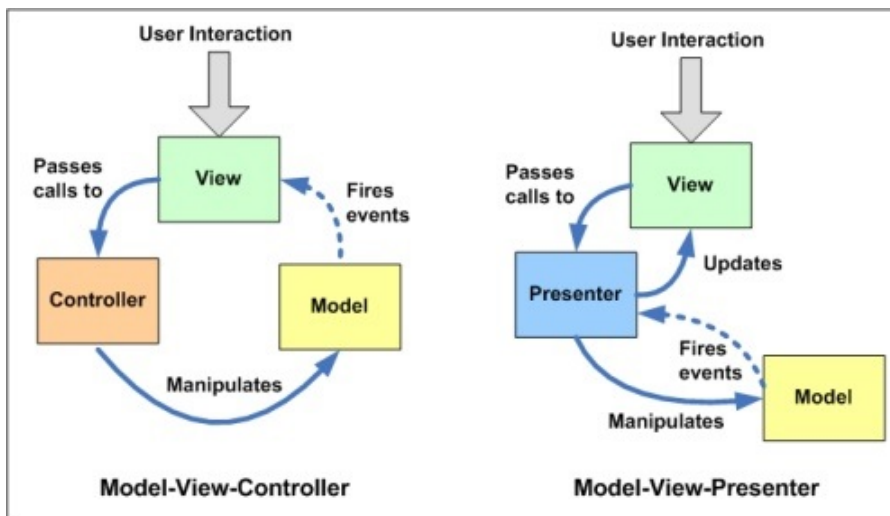


Figure 2.1: Comparația dintre MVC și MVP

2.2 Struts 2

Struts2 este un cadru de dezvoltare (framework) a aplicațiilor bazat pe modelul MVC dezvoltat în cadrul proiectului *apache*.

Struts2 utilizează atât modelul descriptiv de aplicație, prin intermediul unui fișier `xml`, de configurare, `struts.xml`, cât și modul programat prin adnotări.

Instalarea produsului. Resursele necesare sunt cuprinse în catalogul `lib` obținut în urma dezarhivării arhivei de distribuție a produsului.

Resursele minimale sunt

<code>asm-*.jar</code>	<code>asm-commons-*.jar</code>
<code>asm-tree-*.jar</code>	<code>commons-fileupload-*.jar</code>
<code>commons-io-*.jar</code>	<code>commons-lang3-*.jar</code>
<code>commons-logging-*.jar</code>	<code>freemarker-*.jar</code>
<code>javaassist-*.jar</code>	<code>log4j-api-*.jar</code>
<code>log4j-core-*.jar</code>	<code>ognl-*.jar</code>
<code>struts2-core-2.5.*.jar</code>	

Uzual aceste fișiere se copiază în catalogul `WEB-INF\lib` al aplicației Web.

2.2.1 Aplicații *Struts2* prin modelul descriptiv

Modelul descriptiv este caracterizat print-un fișier de configurare `struts.xml`. Componentele unei aplicații *Struts2* sunt:

- Componenta *View* este alcătuită din documente `html` sau pagini `JSP`, conținând formulare prin care clientul introduce date și prin care se afișează rezultatele.

Apelarea din exteriorul aplicației *Struts2* se face prin

`http://host:port/context/NumeAcțiune.action`

iar din interiorul aplicației prin `NumeAcțiune.action`.

- Componenta *Model* este alcătuită dintr-o clasă ce extinde clasa

`com.opensymphony.xwork2.ActionSupport`

care implementează interfața `Action`. Această interfață declară metoda

```
public String execute() throws Exception
```

Programatorul ori suprascrie această metodă ori indică metoda care îndeplinește activitățile necesare satisfacerii cererii. Această metodă se declară prin atributul `method` a elementului `action` din componenta *Controller* `struts.xml`.

În ambele cazuri se returnează un `String` fixat de clasa `ResultNames`: `SUCCESS`, `ERROR`, `INPUT`, `LOGIN`, `NONE`.

Parametrilor preluați dintr-un formular li se asociază în componenta Java câmpuri *private*. Pentru fiecare asemenea câmp se definesc metodele `set` și / sau `get`.

Fiecare clasă trebuie să facă parte dintr-un pachet.

- Componenta *Controller* este reprezentat de fișierul de configurare *struts.xml*, plasat în catalogul `WEB-INF\classes` al aplicației Web. Prin datele acestui fișier se face asocierea dintre numele acțiunii, clasa care implementează acțiunea și fișierele cu rezultatele prelucrărilor.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="pachet" extends="struts-default">
    <action name="NumeActiune"
      class="pachet.Clasa"
      method="numeMetoda">
      <result>/jsp/Rezultat1.jsp</result>
      <result name="input">/jsp/Rezultat2.jsp</result>
      . . .
    </action>
    . . .
  </package>
</struts>
```

Totodată, utilizarea lui *Struts2* necesită fișierul `web.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app id="WebApp_9" version="2.4"
3   xmlns="http://java.sun.com/xml/ns/j2ee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
7
8   <display-name>Struts Blank</display-name>
9
10  <filter>
11    <filter-name>struts2</filter-name>
12    <filter-class>
```

```

13     org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter
14     </filter-class>
15 </filter>

17 <filter-mapping>
18     <filter-name>struts2</filter-name>
19     <url-pattern>/*</url-pattern>
20 </filter-mapping>

22 <welcome-file-list>
23     <welcome-file>index.html</welcome-file>
24 </welcome-file-list>
25 </web-app>

```

Desfășurarea aplicației Web este

webapps

```

|--> catalogul aplicatiei (context)
    |--> html
    |--> jsp
    |--> WEB-INF
        |--> classes
            |--> pachetul aplicatiei
                |      |      *.class
                |      |      struts.xml
                |      |      log4j2.xml
            |--> lib
                |      *.jar
                |      web.xml
            index.html

```

Dacă se completează fișierele sursă cu clasa

```

1 package action;
2 import ognl.OgnlRuntime;
3 import javax.servlet.ServletContextEvent;
4 import javax.servlet.ServletContextListener;
5 import java.util.logging.Logger;

7 /**
8  * To allow works Struts 2 with Google App Engine
9  */
10 public class InitListener implements ServletContextListener {

12     public InitListener() {
13     }

15     public void contextInitialized(ServletContextEvent sce) {
16         OgnlRuntime.setSecurityManager(null);
17     }

```



```

19     public void contextDestroyed(ServletContextEvent sce) {
20     }
21 }

```

atunci se poate utiliza fișierul **web.xml**

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app id="struts_blank" version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
7     <display-name>Struts Blank Convention</display-name>
8
9     <listener>
10         <listener-class>action.InitListener</listener-class>
11     </listener>
12
13     <filter>
14         <filter-name>struts2</filter-name>
15         <filter-class>
16             org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter
17         </filter-class>
18     </filter>
19
20     <filter-mapping>
21         <filter-name>struts2</filter-name>
22         <url-pattern>/*</url-pattern>
23     </filter-mapping>
24
25     <welcome-file-list>
26         <welcome-file>index.html</welcome-file>
27     </welcome-file-list>
28 </web-app>

```

2.2.2 Marcaje Struts

Marcajele Struts sunt definite într-o bibliotecă care se declară într-o pagină JSP prin

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

Nu este nevoie de specificarea lor în fișierul de configurare **web.xml**.

Amintim marcajele:

- **s:form** Pentru marcarea formularului.

Atribute ale marcajului:

Atribut	Fel	Descriere
action	obligatoriu	definește acțiunea de executat.

- **s:textfield** Declară un câmp de introducere date de tip string.

Atribute ale marcajului:

Atribut	Fel	Descriere
name	obligatoriu	Numele câmpului.
label	opțional	Textul explicativ al informației de introdus.

- **s:password** Declară un câmp de introducere a parolei.

Atribute ale marcajului:

Atribut	Fel	Descriere
name	obligatoriu	Numele câmpului.
label	opțional	Textul explicativ al informației de introdus.

- **s:submit** Declară un buton de comandă.

Atribute ale marcajului:

Atribut	Fel	Descriere
value	opțional	Textul explicativ al acțiunii

- **s:property** Afișează valoarea proprietății (atribut, câmp).

Atribute ale marcajului:

Atribut	Fel	Descriere
value	obligatoriu	Numele proprietății (al câmpului)

- **s:select**

Atribute ale marcajului:

Atribut	Fel	Descriere
label	opțional	Textul explicativ al controlului grafic
name	opțional	Numele proprietății
list	obligatoriu	Lista opțiunilor sau numele listei <i>Exemplu:</i> "{'C2F','F2C'}"
listKey	opțional	Valorile cheii. Proprietatea ca avea valoarea selectată.
listValue	opțional	Valorile afișate

- **s:hidden**

Atribute ale marcajului:

Atribut	Fel	Descriere
name	obligatoriu	Numele proprietății / câmpului
value	obligatoriu	Valoarea transmisă

- **s:if**

Atribute ale marcajului:

Atribut	Fel	Descriere
test	obligatoriu	Test

- **s:else**

```
<s:if test="%{incercari > 0}">
  Incercari <s:property value="%{incercari+1}" />
</s:if>
<s:else>
  Prima incercare
</s:else>
```

Marcajele *Strut2* folosesc limbajul *Object Graph Navigation Language* (OGNL) care

- leagă elementele grafice de control de obiecte;
- crează dinamic liste și tabele de proprietăți pentru elementele grafice de control;
- încarcă metodele necesare rezolvării cererii clientului;
- Realizează conversia datelor din șirul de caractere recepționat în tipurile Java (`String`, `boolean` / `Boolean`, `int` / `Integer`, `long` / `Long`, `float` / `Float`, `double` / `Double`, `Date`, `List`, `Map`, `array`).

La fiecare apelare *Struts2* dar înaintea executării solicitării cerute *struts2* crează un obiect *ValueStack* care conține *Object Stack* - stiva obiectelor - și *Context Map* cu atributele cererii, ale sesiunii, etc.

2.2.3 Aplicații Struts2

Exemplificăm prin aplicația simplă de calcul a celui mai mare divizor comun a două numere naturale.

Exemplul 2.2.1

Aplicația *Struts2* poate conține mai multe acțiuni / activități. Alegerea se face dintr-o pagină *AlegeApp.html*, care ține de partea *view* a aplicației Web. Această pagină **html** reprezintă punctul de intrare în aplicație.

- Fișierul *struts.xml* al componentei *Controller* este

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5 <struts>
6   <package name="cmmdc" extends="struts-default">
7
8     <action name="Cmmdc" class="cmmdc.Cmmdc"
9       method="computeCmmdc">
10      <result>/jsp/ResultCmmdc.jsp</result>
11    </action>
12
13    <action name="*" >
14      <result>/html/AlegeApp.html</result>
15    </action>
16
17  </package>
18 </struts>

```

- *AlegeApp.html*

```

1 <html>
2 <head>
3   <title> Alege Aplicatia </title>
4 </head>
5 <body bgcolor="#aaceaa">
6   <h1> Alege&#355;i aplica&#355;ia </h1>
7   <ul>
8     <li>
9       <a href="/mystruts2-app/jsp/Cmmdc.jsp">
10        Calculul celui mai mare divizor comun a dou&#259;
11        numere naturale
12      </a>
13    </li>
14  </ul>
15 </body>
16 </html>

```

În cazul de față este o singură opțiune: calculul celui mai mare divizor comun.

- Componenta *View* a acestei aplicații este alcătuită din două fișiere:

- Introducerea datelor (*Cmmdc.jsp*)

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>

```

```

4      <title>Calculul celui mai mare divizor comun a dou&#259;
5      numere naturale</title>
6  </head>
7  <body>
8      <h3> Introduce&#355;i </h3>
9      <s:form action="Cmmdc.action">
10         <s:textfield label="Primul numar" name="m" />
11         <s:textfield label="Al doilea numar" name="n" />
12         <s:submit value="Calculeaza" />
13     </s:form>
14 </body>
15 </html>

```

– Afişarea rezultatelor (*RezultCmmdc.jsp*)

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>
4     <title>Cmmdc</title>
5   </head>
6   <body>
7     <h2>Cmmdc: </h2>
8     <!--
9       <s:property value="message" />
10      -->
11     <s:property value="#session.cmmdc"/>
12   </body>
13 </html>

```

- Componenta *Model* corespunzătoare acţiunii *Cmmdc* este

```

1 package cmmdc;
2 import com.opensymphony.xwork2.ActionSupport;
3 import com.opensymphony.xwork2.ActionContext;
4 import java.util.Map;
5
6 public class Cmmdc extends ActionSupport {
7
8     public String computeCmmdc(){
9         long c=cmmdc(m,n);
10        // varianta de transmitere a rezultatului printr-un camp
11        //
12        // setMessage(Long.valueOf(c).toString());
13        //
14        // varianta de transmitere a rezultatului prin session
15        Map attr=ActionContext.getContext().getSession();
16        attr.put("cmmdc",Long.valueOf(c).toString());
17        return SUCCESS;
18    }
19
20    public long cmmdc(long m, long n){. . .}
21
22    private long m;
23    public long getM() {
24        return m;
25    }
26    public void setM(long m) {
27        this.m = m;
28    }
29 }

```

```

28     }
30     private long n;
31     public long getN() {
32         return n;
33     }
34     public void setN(long n) {
35         this.n = n;
36     }
38     /*
39     private String message;
40     public void setMessage(String message){
41         this.message = message;
42     }
43     public String getMessage() {
44         return message;
45     }
46     */
47 }

```

Apelarea aplicației se face prin *index.html*

```

1 <html>
2 <head>
3   <META HTTP-EQUIV="Refresh"
4   CONTENT="0;URL=http://localhost:8080/mystruts2-app/html/AlegeApp.html">
5 </head>
6 <body>
7   <p>Apelarea aplicatiei...</p>
8 </body>
9 </html>

```

Accesul la un obiect *session* se poate obține prin

```

Map attr=ActionContext.getContext().getSession();
attr.put(key,object);

```

Aici *attr* este un identificator definit de programator.

În componenta *view* referința la sesiune se poate face prin sintaxa **#session.key** sau **#attr.key**.

Validarea de bază. Clasa *ActionSupport* implementează interfața *com.opensymphony.xworks2.Validateable* cu metoda *public void validate()*, care permite realizarea de verificări asupra datelor încărcate.

Metoda *void addFieldError(String numeCamp, String mesaj)* afișează mesajul pe ieșirea *ResultName.INPUT*.

În cazul clasei *Cmmdc.java* completarea pentru verificarea câmpurilor formularului este

```

public void validate(){
    if(m==0){
        addFieldError("m","Camp necompletat");
    }
    if(n==0){
        addFieldError("n","Camp necompletat");
    }
}

```

Verificarea caracterului numeric al datelor convertite de *Struts* în tipuri numerice este făcută de OGNL.

O facilitate interesantă oferită de *Struts* este completarea automată a unei liste de opțiuni (select).

Exemplul 2.2.2 *Un fișier text conține informațiile {nume județ, capitala județului, abrevierea}, separate printr-un spațiu. Se cere construirea unei aplicații Web care pentru un județ indicat, afișează informațiile corespunzătoare din fișierul menționat.*

Aplicația este alcătuită din două acțiuni:

1. Completarea listei de opțiuni în cadrul formularului. Această acțiune este lansată la apelarea aplicației dintr-o pagină `html`, de exemplu

```

<a href="http://localhost:8080/mystruts2-app/AlegeJudetul.action">
    Referinte despre judete
</a>

```

- Componenta *Control*:

```

<action name="AlegeJudetul" class="appjud.ListaJudeteAction">
    <result>/jsp/AppJud.jsp</result>
</action>

```

Pagina JSP de ieșie, *AppJud.jsp*, este pagina pe care o utilizează clientul pentru selectarea județului. Această pagină reprezintă componenta *View* - de apelare a acțiunii următoare.

- Componenta *Model* este dată de clasa *ListaJudeteAction.java*

```

1 package appjud;
2 import com.opensymphony.xwork2.ActionSupport;
3 import com.opensymphony.xwork2.ActionContext;
4 import java.util.Map;
5 import java.util.List;
6 import java.util.ArrayList;
7 import java.util.HashMap;
8 import java.io.InputStream;
9 import java.io.InputStreamReader;
10 import java.io.BufferedReader;
11 import java.io.IOException;

```

```

13 public class ListaJudeteAction extends ActionSupport{
14     private HashMap<String,RefJudet> refJudete=
15         new HashMap<String,RefJudet>();

17     public List<RefJudet> getJudeteList(){
18         List<RefJudet> list=new ArrayList(50);
19         try(
20             InputStream fis=
21                 this.getClass().getResourceAsStream("judete.txt");
22             InputStreamReader isr=new InputStreamReader(fis);
23             BufferedReader br=new BufferedReader(isr)){
24             String s="",jud,capit,abrev;
25             do{
26                 s=br.readLine();
27                 if(s!=null){
28                     String[] st=s.split(" ");
29                     jud=st[0];
30                     capit=st[1];
31                     abrev=st[2];
32                     RefJudet bean=new RefJudet();
33                     bean.setJud(jud);
34                     bean.setCapit(capit);
35                     bean.setAbrev(abrev);
36                     list.add(bean);
37                     refJudete.put(jud,bean);
38                 }
39             } while(s!=null);
40         }
41         catch(IOException e){
42             e.printStackTrace();
43         }
44         Map attr=ActionContext.getContext().getSession();
45         attr.put("refJudete",refJudete);
46         return list;
47     }
48 }
49 }

```

unde *RefJudet.java* are codul

```

1 package appjud;
2 public class RefJudet implements java.io.Serializable{
3     private String jud;
4     private String capit;
5     private String abrev;

7     public RefJudet(){

9     public void setJud(String jud){
10         this.jud=jud;
11     }
12     public String getJud(){
13         return jud;
14     }

16     public void setCapit(String capit){
17         this.capit=capit;
18     }
19     public String getCapit(){

```



```

20     return capit;
21 }

23 public void setAbrev(String abrev){
24     this.abrev=abrev;
25 }
26 public String getAbrev(){
27     return abrev;
28 }
29 }

```

Clasa *ListaJudeteAction* generează:

- lista *judeteList* prin metoda *getJudeteList* care va fi utilizată de *Struts* pentru completarea unei liste de opțiuni din formularul aplicației. Lista este alcătuită din componente Java de tip *RefJudet*.
- un obiect de tip `HashMap<String, RefJudet>` care este reținut de obiectul sesiune al aplicației. Acest obiect conține datele fișierului text și este utilizat la satisfacerea cererii clientului.

2. Rezolvarea cererii clientului - adică oferirea datelor cerute pe baza selecției indicate.

- Componenta *Controller*:

```

<action name="RefJudet" class="appjud.JudBean">
    <result>/jsp/ResultJud.jsp</result>
</action>

```

- Componenta *View* (*AppJud.jsp*) este

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3 <head>
4     <title>Referinte Judet</title>
5 </head>
6 <body>
7     <h1> Referinte despre judete </h1>
8     <p/>
9     <s:form
10         action="RefJudet.action">
11         <s:select name="selectat" label="Judete"
12             list="judeteList" listKey="%{jud}" listValue="%{jud}" />
13         <s:submit/>
14     </s:form>
15 </body>
16 </html>

```

Câmpul *selectat* va conține numele județului ales. Lista *judeteList* este alcătuită din componente *RefJudet* iar `%{jud}` se referă ca câmpul *jud* al unei asemenea componente.

- Componenta *View* (*ResultJud.jsp*) de afișare a rezultatelor

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>
4     <title>Referite Judet</title>
5   </head>
6   <body>
7     <h2>Referintele despre judetul </h2>
8     <s:property value="jud"/>
9     <p/>
10    Capitala:
11    <s:property value="capit"/>
12    <p/>
13    Abrevierea:
14    <s:property value="abrev"/>
15  </body>
16 </html>

```

- Componenta *Model* () pentru satisfacerea cererii clientului este dată din

```

1 package appjud;
2 import java.util.Map;
3 import java.util.HashMap;
4 import com.opensymphony.xwork2.ActionSupport;
5 import com.opensymphony.xwork2.ActionContext;

7 public class JudBean extends ActionSupport{
8   private String jud=null;
9   private String capit=null;
10  private String abrev=null;
11  private String selectat;

13  public JudBean(){

15  public String execute() throws Exception {
16    Map attr=ActionContext.getContext().getSession();
17    HashMap<String, RefJudet> refJudete=
18      (HashMap<String, RefJudet>)attr.get("refJudete");
19    RefJudet rj=refJudete.get(selectat);
20    jud=rj.getJud();
21    capit=rj.getCapit();
22    abrev=rj.getAbrev();
23    return SUCCESS;
24  }

26  public String getJud(){
27    return jud;
28  }

30  public String getCapit(){
31    return capit;
32  }

34  public String getAbrev(){
35    return abrev;
36  }

```

```

38     public void setSelectat(String selectat){
39         this.selectat=selectat;
40     }
41     public String getSelectat(){
42         return selectat;
43     }
44 }

```

Cele două părți sunt legate prin

```

<action name="AlegeJudetul" class="appjud.ListaJudeteAction">
    <result>/jsp/AppJud.jsp</result>
</action>

<action name="RefJudet" class="appjud.JudBean">
    <result>/jsp/ResultJud.jsp</result>
</action>

```

Încărcarea unui fișier - Upload

Pentru încărcarea unui fișier, *Strut2* oferă o soluție prefabricată, bazată pe produsul *commons-fileupload* de la *apache*.

Acțiunea - partea de control - poate fi

```

<action name="doUpload" class="upload.UploadAction" method="upload">
    <result name="success">/jsp/ResultUpload.jsp</result>
    <result name="error">/jsp/ErrorUpload.jsp</result>
</action>

```

Componenta *View* este alcătuită din

- Alegerea fișierului care se încarcă (*Upload.jsp*)

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head><title>Upload</title>
4   </head>
5   <body>
6     <h3> Incarcarea unui fisier (upload) </h3>
7     <s:form action="doUpload.action"
8       method="post" enctype="multipart/form-data">
9       <s:file name="myFile" label="File" />
10      <s:submit />
11    </s:form>
12  </body>
13 </html>

```

commons-fileupload permite încărea mai multor fișiere, fapt nefolosit în pagina JSP de mai sus.

- Furnizarea unui răspuns.

La încărcarea unui fișier alături de fișierul propriu-zis sunt preluate numele și tipul fișierului. Șablonul de prelucrare este

```
String dataDir = servletContext.getRealPath("/WEB-INF/");
File savedFile = new File(dataDir, myFileFileName);
myFile.renameTo(savedFile);
```

unde *myFile* este numele câmpului din documentul `jsp` corespunzător fișierului care se încarcă. Încărcarea se face într-o zonă de lucru al serverului Web, fiind ștearsă la finalizarea acțiunii. Este sarcina programatorului să preia fișierul *savedFile* în vederea prelucrării / salvării persistente. *myFileFileName* și *myFileContentType* sunt completate de *Struts* cu numele și respectiv, tipul fișierului încărcat.

Exemplul 2.2.3 *Componenta model care preia într-un String conținutul unui fișier text încărcat.*

Modelul este dat de clasa

```
1 package upload;
2 import org.apache.struts2.ServletActionContext;
3 import com.opensymphony.xwork2.ActionSupport;
4 import com.opensymphony.xwork2.ActionContext;
5 import java.util.Map;
6 import java.io.File;
7 import java.io.FileInputStream;
8 import java.io.InputStreamReader;
9 import java.io.BufferedReader;
10 import java.io.OutputStreamWriter;
11 import java.io.BufferedWriter;
12 import java.io.FileOutputStream;

14 public class UploadAction extends ActionSupport{
15     private File myFile;
16     private String myFileFileName;
17     private String myFileContentType;

19     public File getMyFile() {
20         return myFile;
21     }

23     public void setMyFile(File myFile) {
24         this.myFile = myFile;
25     }

27     public String getMyFileFileName() {
28         return myFileFileName;
29     }

31     public void setMyFileFileName(String myFileFileName) {
32         this.myFileFileName = myFileFileName;
33     }
```

```

35 public String getMyFileContentType() {
36     return myFileContentType;
37 }
38
39 public void setMyFileContentType(String myFileContentType) {
40     this.myFileContentType = myFileContentType;
41 }
42
43 public String upload() throws Exception {
44     String fs=System.getProperty("file.separator");
45     Map attr=ActionContext.getContext().getSession();
46     ServletContext servletContext =
47         ServletActionContext.getServletContext();
48     if (myFile != null) {
49         String dataDir = servletContext.getRealPath("/WEB-INF");
50         File savedFile = new File(dataDir, myFileFileName);
51         myFile.renameTo(savedFile);
52
53         StringBuffer sb=new StringBuffer();
54         FileInputStream fis=new FileInputStream(savedFile);
55         InputStreamReader isr=new InputStreamReader(fis);
56         BufferedReader br=new BufferedReader(isr);
57
58         int n=dataDir.length()-7;
59         String dataDir0=dataDir.substring(0,n);
60         File f=new File(dataDir0+fs+"upload"+fs+myFileFileName);
61         FileOutputStream fos=new FileOutputStream(f);
62         OutputStreamWriter osw=new OutputStreamWriter(fos);
63         BufferedWriter bw=new BufferedWriter(osw);
64         String line;
65         do{
66             line=br.readLine();
67             if(line!=null){
68                 sb.append(line+"\n");
69                 bw.write(line,0,line.length());
70                 bw.newLine();
71             }
72         }
73         while(line!=null);
74         attr.put("files",sb.toString());
75         bw.close();
76         osw.close();
77         fos.close();
78         br.close();
79         isr.close();
80         return SUCCESS;
81     }
82     else{
83         attr.put("error","Upload Error");
84         return ERROR;
85     }
86 }
87 }

```

Descărcarea unui fișier - Download

Și pentru această problemă *Struts2* are o soluție predefinită: este declarat un tip de răspuns **stream**, caz în care răspunsul este implicit un flux **InputStream**. În acest caz, numai pentru rezultat este nevoie de un fișier **jsp**.

Exemplul 2.2.4 *Aplicație Web în care clientul alege dintr-o listă un fișier care este descărcat.*

În componenta de control se introduce

```
<action name="doDownload" class="download.DownloadAction">
  <result type="stream">
    <param name="inputName">fileInputStream</param>
    <param name="contentDisposition">attachment;filename=${fileName}</param>
    <param name="bufferSize">2048</param>
    <param name="contentType">application/octet-stream</param>
  </result>
</action>
```

Lansarea aplicației se face din (*Download.jsp*)

```
1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head><title>Download</title>
4   </head>
5   <body>
6     <h3> Descarcarea unui fisier (download) </h3>
7     <s:form
8       action="doDownload.action">
9       <s:select name="file" label="Alege"
10         list="{'capitol.txt','xml-pic.jpg','TomJones.mp3','clock.avi'}"/>
11       <s:submit value="Descarca"/>
12     </s:form>
13   </body>
14 </html>
```

Struts injectează în codul acțiunii numele fișierului de descărcat - în cazul exemplului, numele este conținut de câmpul *file*.

Clasa modelului (acțiunii) este

```
1 package download;
2 import javax.servlet.ServletContext;
3 import com.opensymphony.xwork2.ActionSupport;
4 import org.apache.struts2.ServletActionContext;
5 import java.io.InputStream;
6 import java.io.File;
7 import java.io.FileInputStream;
8 import java.nio.file.Paths;

10 public class DownloadAction extends ActionSupport{
11   private String file;
12   public void setFile(String file) {
13     this.file=file;
14   }
```

```

16 public String getFile(){
17     return file;
18 }
19
20 private InputStream fileInputStream;
21
22 public InputStream getFileInputStream() {
23     return fileInputStream;
24 }
25 public String execute() throws Exception {
26     String fs=System.getProperty("file.separator");
27     ServletContext servletContext =
28         ServletActionContext.getServletContext();
29     String path=servletContext.getRealPath("/") +
30         fs+"resources"+fs;
31     fileInputStream = new FileInputStream(new File(path+file));
32     return SUCCESS;
33 }
34 }

```

2.2.4 Interceptori

Interceptorii sunt componente *Struts2* care execută sarcini înainte sau după procesarea unei cereri. Interceptorii fixează fluxul de prelucrare al unei aplicații *Struts2* și asigură realizarea de sarcini transversale (*cross-cutting tasks*).

Pachetul **struts-default** declară o familie de interceptori **defaultStack** necesară pentru îndeplinirea sarcinilor uzuale.

Exemplul 2.2.5 Interceptorii predefiniți *timer* și *logger* într-o aplicație *Struts2*.

Rezultatele sunt vizibile în fereastra serverului Web. Aplicația constă din:
Componenta *control*

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC
3 "http://Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4 "http://struts.apache.org/dtds/struts-2.0.dtd">
5 <struts>
6     <package name="primul" extends="struts-default">
7         <action name="MyAction" class="exemplu.MyAction">
8             <interceptor-ref name="timer" />
9             <interceptor-ref name="logger" />
10            <interceptor-ref name="defaultStack" />
11            <result>success.jsp</result>
12        </action>
13    </package>
14 </struts>

```

Componenta *model*

```

1 package exemplu;
2 import com.opensymphony.xwork2.ActionSupport;

4 public class MyAction extends ActionSupport{
5     public String execute(){
6         return "success";
7     }
8 }

```

Componenta *view* conține fișierele

- *index.jsp*

```

1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <%@ taglib prefix="s" uri="/struts-tags"%>
3 <html>
4     <head>Exemplu</head>
5     <body>
6         <s:form action="MyAction">
7             <s:submit />
8         </s:form>
9     </body>
10 </html>

```

- *success.jsp*

```

1 <html>
2     <body>
3         Success
4     </body>
5 </html>

```

Se pot defini interceptori proprii. Considerăm exemplul

Exemplul 2.2.6 *Aplicație Struts2 pentru ghicirea unui număr întreg cuprins între 1 și 10 din 3 încercări.*

Pornind de la aplicația *Struts* simplă vom modifica programul prin introducerea un interceptor.

Componenta *control*

```

1 ?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC
3     "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4     "http://struts.apache.org/dtds/struts-2.0.dtd">

6 <struts>
7     <package name="default" extends="struts-default">
8         <action name="guess" class="exemplu.GuessNumber">
9             <result name="input">/jsp/index.jsp</result>
10            <result>/jsp/success.jsp</result>
11            <result name="error">/jsp/error.jsp</result>
12        </action>
13    </package>
14 </struts>

```


Componenta *model*

```

1 package exemplu;

3 import java.util.Map;
4 import java.util.Random;
5 import org.apache.struts2.interceptor.SessionAware;
6 import com.opensymphony.xwork2.ActionSupport;

8 public class GuessNumber extends ActionSupport implements SessionAware{

10     private int numar;
11     private int incercari=1;
12     private Map session;
13     private int maxIncerari=4;

15     public int getNumar() {
16         return numar;
17     }
18     public void setNumar(int numar) {
19         this.numar = numar;
20     }

22     public int getIncerari() {
23         return incercari;
24     }
25     public void setIncerari(int incercari) {
26         this.incerari = incercari;
27     }

29     public void setMaxIncerari(int maxIncerari) {
30         this.maxIncerari = maxIncerari;
31     }
32     public int getMaxIncerari(){
33         return maxIncerari;
34     }

36     @Override
37     public void setSession(Map session) {
38         this.session = session;
39     }

41     public void validate() {
42         if(getNumar()==0)
43             addFieldError("numar", "Introduceti numerul");
44         if((numar>10) || (numar<1))
45             addFieldError("numar", "Trebuie sa fie cuprins intre 1 si 10");
46     }

48     @Override
49     public String execute(){
50         int deGhicit;
51         Integer objIncerari=(Integer)session.get("incercari");
52         if(objIncerari==null){
53             Random random=new Random();
54             deGhicit=random.nextInt(10)+1;
55             session.put("guess", Integer.valueOf(deGhicit));
56         }
57         else{
58             deGhicit=((Integer)session.get("guess")).intValue();

```

```

59         incercari=objIncerari.intValue();
60     }
61     incercari++;
62     session.put("incercari",Integer.valueOf(incercari));
63     if(numar==deGhicit){
64         session.remove("guess");
65         session.remove("incercari");
66         return SUCCESS;
67     }
68     else{
69         if(incercari==maxIncerari){
70             session.remove("guess");
71             session.remove("incercari");
72             return ERROR;
73         }
74         else{
75             return INPUT;
76         }
77     }
78 }
79 }

```

Componenta *view* conține fișierele

- *index.jsp*

```

1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <%@ taglib prefix="s" uri="/struts-tags"%>
3 <html>
4     <body>
5         <h1> Ghici&#355;i num&#259;rul &#238;n 3
6             &#238;ncerc&#259;ri</h1>
7         <p/>
8         <s:if test="#session.incercari!=null">
9             &#206;ncercarea <s:property value="#session.incercari"/>
10        </s:if>
11        <s:else>
12            &#206;ncercarea 1
13        </s:else>
14        <p>
15            <s:form action="guess">
16                <s:textfield name="numar" label="Alegerea mea" />
17                <br/>
18                <s:submit value="Verifica numarul"/>
19            </s:form>
20        </body>
21    </html>

```

- *success.jsp*

```

1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <%@ taglib prefix="s" uri="/struts-tags"%>
3 <html>
4     <body>
5         <h1> Ghici&#355;i num&#259;rul &#238;n
6             <s:property value="%{maxIncerari-1}" />&#238;ncerc&#259;ri
7         </h1>
8         <p/>
9         Num&#259;rul a fost ghicit &#238;n

```

```

10      <s:property value="{incercari-1}" /> &#238;ncerc&#259;ri !
11      <br/>
12      <a href="http://localhost:8080/guess">
13          &#206;nc&#259; o dat&#259;
14      </a>
15  </body>
16 </html>

```

- *error.jsp*

```

1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <%@ taglib prefix="s" uri="/struts-tags"%>
3 <html>
4   <body>
5     <h1> Ghici&#355;i num&#259;rul &#238;n
6       <s:property value="{maxIncercari-1}" />&#238;ncerc&#259;ri
7     </h1>
8     <p/>
9     Ati dep&#259;&#351;it num&#259;rul de &#238;ncerc&#259;ri !
10    </p>
11    Num&#259;rul nu a fost ghicit!
12    <br/>
13    <a href="http://localhost:8080/guess">
14        &#206;nc&#259; o dat&#259;
15    </a>
16  </body>
17 </html>

```

Clasa unui interceptor implementează interfața

`com.opensymphony.xwork2.interceptor.Interceptor.`

Interfața declară metodele

- `public void init()`
- `public void destroy()`
- `public String intercept(ActionInvocation actionInvocation) throws Exception`

Introducem un interceptor pentru generarea numărului ales aleator.

```

2 package exemplu;
3 import java.util.Map;
4 import java.util.Random;
5 import com.opensymphony.xwork2.Action;
6 import com.opensymphony.xwork2.ActionInvocation;
7 import com.opensymphony.xwork2.interceptor.Interceptor;
8
9 public class RandomNumberInterceptor implements Interceptor{
10
11     @Override

```

```

12 public void destroy(){
13     // TODO Auto-generated method stub
14 }
15
16 @Override
17 public void init(){
18     // TODO Auto-generated method stub
19 }
20
21 @Override
22 public String intercept(ActionInvocation actionInvocation)
23     throws Exception{
24     // Se preia sesiunea
25     Map session=actionInvocation.getInvocationContext().getSession();
26     // Fixeaza numarul aleator la inceputul aplicatiei
27     if(!session.containsKey("guess")||!(session.get("guess") != null)){
28         Random random = new Random();
29         int deGhicit = random.nextInt(10)+1;
30         session.put("guess",deGhicit);
31     }
32     // Invocarea celorlalte sarcini
33     return actionInvocation.invoke();
34 }
35 }

```

Metoda `invoke` a clasei `ActionInvocation` apelează următorul pas din procesarea interceptorilor iar în final se returnează un cod de retur.

Modificările suferite de cele trei componente sunt
Componenta *control*

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC
3     "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4     "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7     <package name="default" extends="struts-default">
8         <interceptors>
9             <interceptor name="randomInterceptor"
10                 class="exemplu.RandomNumberInterceptor"/>
11             <interceptor-stack name="myStack">
12                 <interceptor-ref name="defaultStack"/>
13                 <interceptor-ref name="randomInterceptor"/>
14             </interceptor-stack>
15         </interceptors>
16         <action name="guess" class="exemplu.GuessNumber">
17             <interceptor-ref name="myStack"/>
18             <result name="input">/jsp/index.jsp</result>
19             <result>/jsp/success.jsp</result>
20             <result name="error">/jsp/error.jsp</result>
21         </action>
22     </package>
23 </struts>

```

Componenta *model*

```

1 package exemplu;
2 import java.util.Map;

```

```

3 import org.apache.struts2.interceptor.SessionAware;
4 import com.opensymphony.xwork2.ActionSupport;

6 public class GuessNumber extends ActionSupport
7     implements SessionAware{
8     private int numar;
9     private int incercari=1;
10    private Map session;
11    private int maxIncerari=4;

13    public int getNumar() {
14        return numar;
15    }
16    public void setNumar(int numar) {
17        this.numar = numar;
18    }

20    public int getIncerari() {
21        return incercari;
22    }
23    public void setIncerari(int incercari) {
24        this.incerari = incercari;
25    }

27    public void setMaxIncerari(int maxIncerari) {
28        this.maxIncerari = maxIncerari;
29    }
30    public int getMaxIncerari(){
31        return maxIncerari;
32    }

34    @Override
35    public void setSession(Map session) {
36        this.session = session;
37    }

39    public void validate() {
40        if(getNumar()==0) {
41            addFieldError("numar", "Introduceti numerul");
42        }
43        if((numar>10) || (numar<1))
44            addFieldError("numar", "Trebuie sa fie cuprins intre 1 si 10");
45    }

47    @Override
48    public String execute(){
49        int deGhicit=((Integer)session.get("guess")).intValue();
50        Integer objIncerari=(Integer)session.get("incercari");
51        if(objIncerari!=null){
52            incercari=objIncerari.intValue();
53        }
54        System.out.println(incercari+" "+numar+" "+deGhicit);
55        incercari++;
56        session.put("incercari",Integer.valueOf(incercari));
57        if(numar==deGhicit){
58            session.remove("guess");
59            session.remove("incercari");
60            return SUCCESS;
61        }

```

```

62     else{
63         if(incercari==maxIncercari){
64             session.remove("guess");
65             session.remove("incercari");
66             return ERROR;
67         }
68         else{
69             return INPUT;
70         }
71     }
72 }
73 }

```

Componenta *view* nu suferă modificări.

2.2.5 Aplicații *Struts2* prin modelul programat

Modelul programat se obține prin utilizarea adnotărilor. O acțiune este denumită printr-un identificator, de exemplu *xyz*. Clasa corespunzătoare va fi *actions.XyzAction*. Această clasă conține metoda

```
public String execute(),
```

care este invocată de *Struts2* pentru îndeplinirea cererii clientului.

String-ul returnat fixează componenta *view* în adnotarea `org.apache.struts2.convention.annotation.Result`. Câmpurile adnotării sunt `name`, `location`, `type`, `params`. În cazul mai multor adnotari `Result` acestea se includ în adnotarea `org.apache.struts2.convention.annotation.Results` potrivit șablonului

```

import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.Results;

@Results({
    @Result(. . .),
    @Result(. . .),
    . . .
})

```

Aceste resurse se găsesc în arhiva `struts2-convention-plugin-*.jar`.

Reluăm aplicația din secțiunea 2.2.3.

Cmmdc.jsp:

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3   <head>
4     <title>
5       Calculul celui mai mare divizor comun a două numere naturale
6     </title>
7   </head>
8   <body>
9     <h3> Introduceți

```

```

10 <s:form action="cmmdc">
11   <s:textfield label="Primul numar" name="m"/>
12   <s:textfield label="Al doilea numar" name="n" />
13   <s:submit value="Calculeaza"/>
14 </s:form>
15 </body>
16 </html>

```

cu acțiunea corespunzătoare

```

1 package actions;
2 import com.opensymphony.xwork2.ActionContext;
3 import java.util.Map;
4 import org.apache.struts2.convention.annotation.Result;

6 @Result(name="success", location="/jsp/ResultCmmdc.jsp")

8 public class CmmdcAction{
9   public String execute(){
10    long c=cmmdc(m,n);
11    Map attr=ActionContext.getContext().getSession();
12    attr.put("cmmdc",Long.valueOf(c).toString());
13    return "success";
14  }

16  public long cmmdc(long m, long n){. . .}

18  private long m;
19  public long getM() {
20    return m;
21  }
22  public void setM(long m) {
23    this.m = m;
24  }

26  private long n;
27  public long getN() {
28    return n;
29  }
30  public void setN(long n) {
31    this.n = n;
32  }
33  private String message;
34  public void setMessage(String message){
35    this.message = message;
36  }
37  public String getMessage() {
38    return message;
39  }
40 }

```

În cazul exemplului 2.2.2 se lansează acțiunea

/mystruts2-anapp/listajud

cu

```

1 package actions;
2 import com.opensymphony.xwork2.ActionContext;

```

```

3 import java.util.*;
4 import java.io.*;

6 import org.apache.struts2.convention.annotation.Result;
7 @Result(name="success", location="/jsp/AppJud.jsp")

9 public class ListajudAction{
10     private HashMap<String, RefJudet> refJudete=
11         new HashMap<String, RefJudet>();

13     public String execute(){
14         return "success";
15     }

17     public List<RefJudet> getJudeteList() throws IOException{
18         List<RefJudet> list=new ArrayList<RefJudet>();
19         InputStream fis=this.getClass().getResourceAsStream("judete.txt");
20         InputStreamReader isr=new InputStreamReader(fis);
21         BufferedReader br=new BufferedReader(isr);
22         String s="", jud, capit, abbrev;
23         do{
24             s=br.readLine();
25             if(s!=null){
26                 String[] st=s.split(" ");
27                 jud=st[0];
28                 capit=st[1];
29                 abbrev=st[2];
30                 RefJudet bean=new RefJudet();
31                 bean.setJud(jud);
32                 bean.setCapit(capit);
33                 bean.setAbrev(abbrev);
34                 list.add(bean);
35                 refJudete.put(jud, bean);
36             }
37         } while(s!=null);
38         Map attr=ActionContext.getContext().getSession();
39         attr.put("refJudete", refJudete);
40         return list;
41     }
42 }
43 }

```

AppJud.jsp are codul

```

1 <%@ taglib prefix="s" uri="/struts-tags" %>
2 <html>
3     <head>
4         <title>Referinte Judet</title>
5     </head>
6     <body>
7         <h1> Referinte despre judete </h1>
8         <p/>
9         <s:form action="judbean">
10             <s:select name="selectat" label="Judete"
11                 list="judeteList" listKey="%{jud}" listValue="%{jud}" />
12             <s:submit />
13         </s:form>
14     </body>
15 </html>

```


căreie îi corespunde acțiunea dată de clasa

```

1 package actions;
2 import java.util.*;
3 import com.opensymphony.xwork2.ActionContext;
4 import org.apache.struts2.convention.annotation.Result;

6 @Result(name="success", location="/jsp/ResultJud.jsp")

8 public class JudbeanAction{
9     private String jud=null;
10    private String capit=null;
11    private String abbrev=null;
12    private String selectat;

14    public String execute() throws Exception {
15        Map attr=ActionContext.getContext().getSession();
16        HashMap<String,RefJudet> refJudete =
17            (HashMap<String,RefJudet>)attr.get("refJudete");
18        RefJudet rj=refJudete.get(selectat);
19        jud=rj.getJud();
20        capit=rj.getCapit();
21        abbrev=rj.getAbrev();
22        return "success";
23    }

25    public String getJud(){
26        return jud;
27    }

29    public String getCapit(){
30        return capit;
31    }

33    public String getAbrev(){
34        return abbrev;
35    }

37    public void setSelectat(String selectat){
38        this.selectat=selectat;
39    }

41    public String getSelectat(){
42        return selectat;
43    }
44 }

```

Adnotările

- org.apache.struts2.convention.annotation.InterceptorRef
- org.apache.struts2.convention.annotation.InterceptorRefs

declară interceptorii necesari unei acțiuni.

În cazul exemplului tratat în secțiunea dedicată interceptorilor, clasa interceptorului *actions.RandomNumber Interceptor* rămâne nemodificată iar interceptorul se declară în fișierul **struts.xml**

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4   "http://struts.apache.org/dtds/struts-2.0.dtd">
5 <struts>
6   <package name="anint" extends="struts-default">
7     <interceptors>
8       <interceptor name="randomInterceptor"
9         class="actions.RandomNumberInterceptor" />
10    </interceptors>
11  </package>
12 </struts>

```

Clasa acțiunii *actions.GuessNumber* devine

```

1 package actions;
2 import java.util.Map;
3 import java.util.Random;
4 import org.apache.struts2.interceptor.SessionAware;
5 import com.opensymphony.xwork2.ActionSupport;
6 import org.apache.struts2.convention.annotation.Result;
7 import org.apache.struts2.convention.annotation.Results;
8 import org.apache.struts2.convention.annotation.Action;
9 import org.apache.struts2.convention.annotation.InterceptorRef;
10 import org.apache.struts2.convention.annotation.InterceptorRefs;
11
12 @Results({
13   @Result(name="error", location="/jsp/error.jsp"),
14   @Result(name="success", location="/jsp/success.jsp"),
15   @Result(name="input", location="/jsp/index.jsp"),
16 })
17
18 @org.apache.struts2.convention.annotation.ParentPackage(value = "anint")
19 @Action(value="guess",
20   interceptorRefs={@InterceptorRef("randomInterceptor"),
21     @InterceptorRef("defaultStack")})
22
23 public class GuessNumber extends ActionSupport implements SessionAware{
24   . . .
25 }

```

2.2.6 Struts 2 prin maven

Aplicațiile dezvoltate corespund celor dezvoltate anterior, calculul celui mai mare divizor comun (*Cmmdc.java*) și regăsirea datelor unui județ (*JudBean.java*, *ListaJudeteAction.java*, *RefJudet.java*).

Modelul descriptiv

Dezvoltarea aplicației constă din:

1. Generarea aplicației (Windows)

```

set GroupID=action
set ArtifactID=mystruts2
set Version=1.0
set ArchetypeVersion=*
mvn archetype:generate -B
  -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -Dversion=%Version%
  -DarchetypeGroupId=org.apache.struts
  -DarchetypeArtifactId=struts2-archetype-convention
  -DarchetypeVersion=%ArchetypeVersion%
  -DremoteRepositories=http://struts.apache.org

```

unde * se înlocuiește cu versiunea *struts2-archetype-convention* folosită.

2. Se adaptează structura de cataloage și fișiere la

```

mystruts2
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> action
|   |   |   |   |--> cmmdc
|   |   |   |   |   Cmmdc.java
|   |   |   |   |--> appjud
|   |   |   |   |   JudBean.java
|   |   |   |   |   ListaJudeteAction
|   |   |   |   |   RefJudet.java
|   |   |--> resources
|   |   |   judete.txt
|   |   |   struts.xml
|   |--> webapp
|   |   |--> html
|   |   |   AlegeApp.html
|   |   |--> jsp
|   |   |   Cmmdc.jsp
|   |   |   AppJud.jsp
|   |   |   ResultCmmdc.jsp
|   |   |   ResultJud.jsp
|   |   |--> WEB-INF
|   |   |   web.xml
|   |   |   index.html
|   pom.xml

```

În clasa *ListaJudeteAction.java* fișierul *judete.txt* se încarcă prin

```
InputStream fis=this.getClass().getResourceAsStream("../judete.txt");
```

3. Prelucrarea constă din

- (a) `mvn clean package`
- (b) Fișierul `war` care rezultă se desfășoară în serverul Web.

Modelul programat

Indicăm doar diferențele față de varianta anterioară:

1. În fișierul `pom.xml` se adaugă:

```
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-convention-plugin</artifactId>
  <version>${struts2.version}</version>
</dependency>
```

2. Se adaptează structura de cataloage și fișiere la

```
mystruts2annotation
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> action
|   |   |   |   |--> cmmdc
|   |   |   |   |   Cmmdc.java
|   |   |   |   |--> appjud
|   |   |   |   |   JudBean.java
|   |   |   |   |   ListaJudeteAction
|   |   |   |   |   RefJudet.java
|   |   |   |--> resources
|   |   |   |   judete.txt
|   |   |--> webapp
|   |   |   |--> html
|   |   |   |   AlegeApp.html
|   |   |   |--> jsp
|   |   |   |   Cmmdc.jsp
|   |   |   |   AppJud.jsp
|   |   |   |   ResultCmmdc.jsp
|   |   |   |   ResultJud.jsp
|   |   |   |--> WEB-INF
|   |   |   |   web.xml
|   |   |   |   index.html
|   pom.xml
```

3. În paralel se adaptează pachetele claselor Java și referințele din fișierele `html`.

Probleme

- Struts în nor (GAE, Heroku, OpenShift)

2.3 Java Server Faces

Java Server Faces (JSF) este un cadru de dezvoltare (framework) a aplicațiilor Web, asemănător cu *Struts*.

Amintim următoarele două implementări JSF:

- JSF - RI (Reference Implementation), dezvoltat de firma Oracle-Sun Microsystems; JSF este inclusă în extensia *Java Enterprise Edition - JEE*.

Distribuția JSF de sine stătătoare constă dintr-un fișier `javax.faces-*.jar`, care trebuie copiat în catalogul `WEB-INF\lib` al aplicației.

- MyFaces, dezvoltat în cadrul fundației *apache*.

2.3.1 Structura unei aplicații JSF

- Partea de *view* din JSF poate fi dezvoltată utilizând:

- *Facelets* - caz în care vorbim de pagini *Facelets*, reprezentate de fișiere cu extensia `xhtml`. Resursele necesare tehnologiei *Facelets* trebuie descărcate suplimentar.

Resursele necesare sunt:

- * `javax.faces-*.jar`
- * `javax.servlet.jsp.jstl.jar`
- * `javax.servlet.jsp.jstl-api.jar`

- *JSP*

Resursele necesare sunt:

- * `javax.faces-*.jar`
- * `taglib-standard-impl-*.jar`, `taglib-standard-spec-*.jar` aflate în `TOMCAT_HOME\webapps\examples\WEB-INF\lib`. Acestea din urmă definesc *Java Standard Tag Library* (JSTL).

- Biblioteci de componente grafice:

- * *Primefaces*, dezvoltat de *Çağatay Çivici*, utilizarea căreia s-a răspândit în ultima vreme.

Resursele necesare sunt:

- `javax.faces-*.jar`

- taglibs-standard-impl-*.jar,
- taglibs-standard-spec-*.jar
- primefaces-*.jar
- * *Richfaces*

În toate cazuri se utilizează biblioteca de marcaje specifice JSF.

Facelets se impune datorită incompatibilităților dintre JSF și JSP în ciclul de activități pe care le desfășoară pentru rezolvarea unei apelări.

- Partea de *model* este alcătuită din componente Java (bean) care se încarcă cu datele furnizate din paginile JSP/Facelets, asigură funcționalitatea specifică aplicației și constituie sursa de date pentru paginile de afișare a rezultatelor. Clasele Java trebuie să implementeze interfața `java.io.Serializable`.

JSF instanțiază componentele Java și injectează datele în aceste componente.

- *Controller*-ul este dat de fișierul de configurare `faces-config.xml` în care se fixează
 - legătura dintre paginile de furnizare a datelor și cele care afișează rezultatul prelucrării, prin elementele `<navigation-rule>`. Astfel
 - * elementul `<from-view-id>` conține referința la pagina Facelets / JSP furnizoare de date;
 - * elementul `<navigation-case>` declară o posibilitate de ieșire. Corpul acestui element conține:
 - `<from-outcome>` element în care se declară stringul care direcționează ieșirea.
 - `<to-view-id>` conține referința la pagina Facelets / JSP de afișare a rezultatelor obținute în urma acțiunii corespunzătoare stringului din `<from-outcome>`.

Toate referințele se raportează la contextul aplicației.

Elementul `<navigation-rule>` poate fi evitat prin precizarea în instrucțiunea `return`, din codul acțiunii, a fișierului care tratează prelucrarea care urmează, de exemplu

```
return "/cmmdcOutput.jsp";
```

- componentele Java utilizate, prin elemente `<managed-bean>` având componența:

- * elementul `<managed-bean-name>` declară numele simbolic al unei componente Java (bean).
- * elementul `<managed-bean-class>` specifică clasa componente. Referința se raportează la catalogul `WEB-INF\classes`.
- * elementul `<managed-bean-scope>` specifică durata de existență a componente Java: `page`, `request`, `session`, `application`.

Într-un element `<managed-bean>` se pot fixa valorile unor câmpuri ale componente Java prin intermediul marcajelor

```
<managed-property>
  <property-name> . . . </property-name>
  <value> . . . </value>
</managed-property>
```

Elementul `<managed-bean>` poate fi evitat prin utilizarea adnotărilor aplicate componente Java.

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
```

```
@ManagedBean(name="nume_componenta")
@SessionScoped | @ApplicationScoped
@ManagedProperty(name="nume_camp", value="valoare_camp")
```

Dacă în adnotarea `@ManagedBean` lipsește atributul `name` și dacă identificatorul componente Java (clasei) este *XYZ* (adică fișierul clasei este *XYZ.java*) atunci în fișierele *view* referință la câmpurile și metodele clasei se face prin sintaxa `#{xyz.câmp}`, respectiv `#{xyz.metodă}`.

Fișierul *web.xml* este independent de aplicație și în principiu are codul

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <web-app version="2.5"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6   http://java.sun.com/xml/ns/javaee/web-app-2.5.xsd">
7
8   . . .
9   <servlet>
10    <servlet-name>Faces Servlet</servlet-name>
11    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
12    <load-on-startup>1</load-on-startup>
13  </servlet>
```

```

15 <servlet-mapping>
16   <servlet-name>Faces Servlet</servlet-name>
17   <url-pattern>*.faces sau *.jsf</url-pattern>
18 </servlet-mapping>
19 . . .
20 </web-app>

```

Desfășurarea unei aplicații JSF este

```

|--> webapps
|   |--> app_jsf_context
|   |   |--> WEB-INF
|   |   |   |--> lib
|   |   |   |   |   javax.faces.*.jar
|   |   |   |   |   alte_arhive_jar
|   |   |   |--> classes
|   |   |   |   |--> pachetul_aplicatiei
|   |   |   |   |   *.class (componentele Java)
|   |   |   |   |   web.xml
|   |   |   |   |   faces-config.xml
|   |   |   |   |   index.html
|   |   |   |   |   pagini_JSP sau pagini_Facelets

```

Prin *index.html* se lansează aplicația JSF.

Un ciclu de execuție JSF constă din

1. Afișarea componentei *view*;
2. Preluarea datelor necesare satisfacerii cererii;
3. Validarea datelor;
4. Actualizarea componentei *model* cu datele preluate;
5. Invocarea metodelor componentei *model* care rezolvă cererea clientului;
6. Afișarea răspunsului.

2.3.2 Marcaje JSF

Marcajele JSF sunt definite în două biblioteci. Ele se declară în paginile JSP prin

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

```

Nu este nevoie de specificarea lor în fișierul de configurare *web.xml*.

Amintim marcajele:

- **f:view** Vizualizează componentele grafice.
- **h:form** Pentru marcarea formularului.
- **h:outputText** Afișează un text.

Atribute ale marcajului:

Atribut	Fel	Descriere
value	obligatoriu	Textul ce se afișează. Poate fi și referința la câmpul unei componente Java.

- **h:inputText** Declară un câmp de introducere date de tip string.

Atribute ale marcajului:

Atribut	Fel	Descriere
value	obligatoriu	Câmpul componentei Java alimentat.
required	opțional	true / false , pentru validare.
id	opțional	Numele simbolic al câmpului.
validator	opțional	Referința la metoda de validare.

- **h:panelGrid** Componentele incluse sunt aranjate într-un tablou.

Atribute ale marcajului:

Atribut	Fel	Descriere
columns	obligatoriu	Numărul coloanelor.

- **h:selectOneMenu** Declară un control grafic de tip *select*.

Atribute ale marcajului:

Atribut	Fel	Descriere
value	obligatoriu	Câmpul componentei Java alimentat.
required	opțional	true / false , pentru validare.
id	opțional	Numele simbolic al câmpului.
validator	opțional	Referința la metoda de validare.

Opțiunile se definesc în elementul `<f:selectItem>` ale cărui atribute sunt

Atribut	Fel	Descriere
itemLabel	obligatoriu	Textul explicativ afișat.
itemValue	obligatoriu	Valoarea atașată opțiunii.

Exemplu:

```

<h:selectOneMenu required="true" value="#{conv.tip}">
  <f:selectItem itemLabel="Celsius -> Fahrenheit" itemValue="C2F" />
  <f:selectItem itemLabel="Fahrenheit -> Celsius" itemValue="F2C" />
</h:selectOneMenu>

```

- **h:commandButton** Declară un buton de comandă.

Atribute ale marcajului:

Atribut	Fel	Descriere
action	obligatoriu	Numele simbolic al acțiunii, declarat într-un marcaj <code><from-outcome></code> din <code>faces-config.xml</code> .
value	obligatoriu	Textul butonului.

- **h:messages** Afișează mesajele de eroare rezultate în urma validărilor.
- **h:message** Afișează mesajele de eroare rezultate în urma validărilor.

Atribute ale marcajului:

Atribut	Fel	Descriere
for	obligatoriu	Numele simbolic al câmpului supus verificării.

2.3.3 Aplicații JSF cu pagini Facelets

În acest caz fișierul *web.xml* se completează este

```

<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>

```

Componenta *view* a unei aplicații este alcătuită din pagini *Facelets*.

Pentru început, plecăm de la codul cu marcaje JSF (*cmmdcInput.xhtml*) - corespunzătoare unui formular de date pentru calculul celui mai mare divizor comun a două numere naturale

```

1 <html>
2   <h:form>
3     <h:panelGrid columns="2">
4       <h:outputText value="Primul num&#259;r" />
5       <h:inputText value="#{cb.sm}" />
6       <h:outputText value="Al doilea num&#259;r" />
7       <h:inputText value="#{cb.sn}" />
8       <h:commandButton action="#{cb.compute}" value="Calculeaz&#259;" />
9     </h:panelGrid>
10   </h:form>
11 </html>

```

Referința la resursele unei componente Java se indică cuprinse prin `#{ }` sau `${ }`. În acest cod, *cb* desemnează o componentă Java pentru calculul celui mai mare divizor comun a două numere naturale. *sm,sn* sunt câmpuri iar *compute* este o metodă - codul complet al clasei este dat mai jos.

O pagină *Facelets* poate corespunde unui șablon, o resursă *Facelets* reutilizabilă. În șablon se definesc zone a căror conținut se poate inițializa și care ulterior se pot modifica. O regiune se definește prin

```
<ui:insert name="numeZona">continut</ui:insert>
```

Zona poate fi inițializată cu conținutul unui alt fișier *Facelets* sau *html*, prin

```
<ui:include src="fisier.[x]html"/>
```

Într-o pagină *Facelets* dezvoltată pe șablon, referința la acesta se obține prin

```
<ui:composition template="sablon.xhtml">
    . . .
</ui:composition>
```

Conținutul aflat înafara elementului `<ui:composition>` este ignorat. Acest efect este anulat dacă în schimb se utilizează

```
<ui:decorate template="sablon.xhtml">
    . . .
</ui:decorate>
```

Modificarea unei zone se programează prin

```
<ui:define name="numeZona">
    noul continut
</ui:define>
```

Introducerea / definirea unui conținut nou *Facelets* se obține cu

```
<ui:component>
    . . .
</ui:component>
```

Conținutul aflat înafara acestui element este ignorat. Acest efect este anulat dacă în schimb se utilizează

```
<ui:fragment>
```

```
. . .
```

```
</ui:fragment>
```

Prin elementul

```
<ui:param name="nume_parametru" value="#{componentaJava.camp}"/>
```

se pot transmite parametrii către un document `xhtml`. Utilizarea acestei variante oferă o generalitate mărită documentului `xhtml`

Fie șablonul *template.xhtml*

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:ui="http://java.sun.com/jsf/facelets"
6       xmlns:f="http://java.sun.com/jsf/core"
7       xmlns:h="http://java.sun.com/jsf/html">
8   <ui:insert name="title">
9     Title
10  </ui:insert>
11  <br/>
12  <table width="100%">
13    <tr>
14      <td width="20%">
15        <ui:insert name="sidemenu">
16          Side Menu
17        </ui:insert>
18      </td>
19      <td width="80%">
20        <ui:insert name="body">
21          Body
22        </ui:insert>
23      </td>
24    </tr>
25  </table>
26  <br/>
27  <ui:insert name="subsol">
28    <ui:include src="footerTemplate.html" />
29  </ui:insert>
30 </html>
```

unde *footerTemplate.html* are codul

```

1 <html>
2   <body>
3     <i>Facelets technology</i>
4   </body>
5 </html>
```

Între *titlu* și *subsol* s-a introdus un tabel cu două coloane pentru un meniu (*sidemenu*) și pentru zona *body*.

Exemplul 2.3.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Pe acest șablon se dezvoltă paginile *Facelets* de introducere a datelor (*cm-mdcInput.xhtml*) și de afișare a rezultatelor (*cm-mdcOutput.xhtml*):

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html">
7
8   <ui:composition template="template.xhtml">
9     <ui:define name="title">
10      <h1>Cel mai mare divizor comun</h1>
11    </ui:define>
12    <ui:define name="body">
13      <h:form>
14        <h:panelGrid columns="2">
15          <h:outputText value="Primul num&#259;r" />
16          <h:inputText value="#{cb.sm}" />
17          <h:outputText value="Al doilea num&#259;r" />
18          <h:inputText value="#{cb.sn}" />
19          <h:commandButton action="#{cb.compute}" value="Calculeaz&#259;" />
20        </h:panelGrid>
21      </h:form>
22    </ui:define>
23  </ui:composition>
24 </html>

```

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html">
7
8   <ui:composition template="template.xhtml">
9     <ui:define name="title">
10      <h1>CMMDC Rezultat</h1>
11    </ui:define>
12    <ui:define name="body">
13      <h:outputText value="Cmmdc=" />
14      <h:outputText value="#{cb.sresult}" />
15    </ui:define>
16  </ui:composition>
17 </html>

```

Componenta *model* este dată de clasă

```

1 package cmmdc;
2
3 public class CmmdcBean implements java.io.Serializable{
4   private String sm;
5   private String sn;
6   private String sresult;
7 }

```

```

8  public CmmdcBean(){
10
11  public String getSm(){
12      return sm;
13  }
14
15  public void setSm(String sm){
16      this.sm=sm;
17  }
18
19  public String getSn(){
20      return sn;
21  }
22
23  public void setSn(String sn){
24      this.sn=sn;
25  }
26
27  public String getSresult(){
28      return sresult;
29  }
30  public String compute(){
31      long m=Long.parseLong(sm);
32      long n=Long.parseLong(sn);
33      long c=cmmdc(m,n);
34      sresult=Long.valueOf(c).toString();
35      return "OK";
36  }
37
38  private long cmmdc(long m,long n){. . .}

```

Componenta *controller* este dată de fișierul `faces-config.xml`

```

1  <?xml version='1.0' encoding='UTF-8'?>
2  <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5      http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig-2.2.xsd"
6      version="2.2">
7
8      <navigation-rule>
9          <description/>
10         <from-view-id>/cmmdcInput.xhtml</from-view-id>
11         <navigation-case>
12             <description/>
13             <from-outcome>OK</from-outcome>
14             <to-view-id>/cmmdcOutput.xhtml</to-view-id>
15         </navigation-case>
16     </navigation-rule>
17
18     <managed-bean>
19         <description/>
20         <managed-bean-name>cb</managed-bean-name>
21         <managed-bean-class>cmmdc.CmmdcBean</managed-bean-class>
22         <managed-bean-scope>session</managed-bean-scope>
23     </managed-bean>
24 </faces-config>

```

Aplicația se lansează prin (*index.html*)

```

1 <html>
2   <body>
3     <center>
4       <h1> Aplicație JSF </h1>
5       <a href="/myjsfFacelets/cmmdcInput.faces">
6         Aplicație JSF
7       </a>
8     </center>
9   </body>
10 </html>

```

Extensia *faces* semnalează serverului Web că se execută o aplicație JSF.

Extindem aplicația pentru a exemplifica utilizarea unui element `<ui:component>`.

În acest sens se definesc clasele *MenuItem*

```

1 public class MenuItem implements java.io.Serializable{
2   private String url;
3   private String label;
4
5   public void setUrl(String url){
6     this.url=url;
7   }
8   public String getUrl(){
9     return url;
10  }
11
12  public void setLabel(String label){
13    this.label=label;
14  }
15  public String getLabel(){
16    return label;
17  }
18
19  public MenuItem(){
20  }
21  public MenuItem(String url,String label){
22    this.url=url;
23    this.label=label;
24  }
25 }

```

și *MenuBean*

```

1 import java.util.Collection;
2 import java.util.ArrayList;
3 public class MenuBean implements java.io.Serializable{
4   private Collection<MenuItem> menus;
5
6   public Collection<MenuItem> getMenus(){
7     return menus;
8   }
9   public void setMenus(Collection<MenuItem> menus){
10    this.menus=menus;
11  }
12
13  public MenuBean(){
14    menus=new ArrayList<MenuItem>();
15    menus.add(new MenuItem("/cmmdcInput.faces","CMMDC automat"));
16  }
17 }

```

```

16     menus.add(new MenuItem("cmmdcInput1.faces","CMMDC manual"));
17     menus.add(new MenuItem("appjudInput.faces","App judete"));
18 }
19 }

```

O instanță a clasei *MenuBean* este creat de JSF prin

```

<managed-bean>
  <managed-bean-name>menuBean</managed-bean-name>
  <managed-bean-class>MenuBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

```

Acum suntem în măsură să definim componenta *view*: *sideMenu.xhtml* pentru elementul *sidemenu* definit în șablon.

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:ui="http://java.sun.com/jsf/facelets"
6       xmlns:f="http://java.sun.com/jsf/core"
7       xmlns:h="http://java.sun.com/jsf/html"
8       xmlns:c="http://java.sun.com/jstl/core">
9   <ui:component>
10     <!--
11     <c:forEach var="vmenu" items="#{menuBean.menus}">
12       -->
13     <c:forEach var="vmenu" items="#{menu}">
14       <a href="#{vmenu.url}">#{vmenu.label}</a><br/>
15     </c:forEach>
16   </ui:component>
17 </html>

```

Linia comentată corespunde versiunii fără elementul `<ui:param>` a fișierului *cmmdcOutput.xhtml*, care se modifică în

```

1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:ui="http://java.sun.com/jsf/facelets"
6       xmlns:h="http://java.sun.com/jsf/html">
7
8   <ui:composition template="template.xhtml">
9     <ui:define name="title">
10       <h1>CMMDC Rezultat</h1>
11     </ui:define>
12     <ui:define name="sidemenu">
13       <ui:include src="sideMenu.xhtml">
14         <ui:param name="menu" value="#{menuBean.menus}" />
15       </ui:include>
16     </ui:define>
17     <ui:define name="body">
18       <h:outputText value="Cmmdc" />
19       <h:outputText value="#{cb.sresult}" />
20     </ui:define>
21   </ui:composition>
22 </html>

```


Valoarea *menu* a atributului **name** din elementul `<ui:param` este introdus în rândul 14 din fișierul *cmmdcOutput.xhtml* și este folosit în rândul 13 din *sideMenu.xhtml*. Fiecare element al colecției *menuBean.menus* generează o linie în *sideMenu*.

Valoarea *vmenu* a atributului **var** din elementul `<c:forEach` din liniile 11 și 13 din fișierul *sideMenu.xhtml* conține înregistrarea curentă din colecția care se parcurge.

2.3.4 Aplicații JSF cu pagini JSP

Extensia **faces** a unei ancore este semnalul prin care serverul Web apelează JSF și coincide cu declarația din elementul `<url-pattern>` din fișierul *web.xml*.

Reluăm aplicația dezvoltată mai sus cu pagini JSP pentru partea de *view*

Exemplul 2.3.2 *Calculul celui mai mare divizor comun a două numere naturale cu verificarea datelor.*

Completăm codul clasei *cmmdc.CmmdcBean* cu codul de verificare a datelor care se injectează *sm*, *sn*

```

1 package cmmdc;
2 import javax.faces.application.FacesMessage;
3 import javax.faces.component.UIComponent;
4 import javax.faces.context.FacesContext;
5 import javax.faces.validator.ValidatorException;
6
7 public class CmmdcBean implements java.io.Serializable{
8     . . .
9     public void validateString(FacesContext context, UIComponent component,
10         Object value) throws ValidatorException{
11         if ((context==null)|| (component==null)) {
12             throw new NullPointerException();
13         }
14         if (value.toString().trim().equals("")){
15             throw new ValidatorException(new FacesMessage(
16                 "Nu ati completat campul"));
17         }
18         else{
19             String s=value.toString();
20             try{
21                 Long.parseLong(s);
22             }
23             catch(NumberFormatException e){
24                 throw new ValidatorException(new FacesMessage("Nu este numar"));
25             }
26         }
27     }
28 }

```

Partea de *view* este dată de fișierele JSP

- *apps.jsp*

```

1 <html>
2 <body>
3   <center>
4     <h1> Aplica&#355;ii JSF </h1>
5     <table border=2>
6       <tr>
7         <td>
8           <a href="/myjsfJSP/cmmdc.jsp">
9             Cmmdc cu preluare automat&#259; a datelor
10          </a>
11        </td>
12      </tr>
13    </table>
14  </center>
15 </body>
16 </html>

```

- *cmmdcInput.jsp*

```

1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4   <body>
5     <h1> Calculul Cmmdc </h1>
6     <f:view>
7       <p>
8         <h:form >
9           <h:panelGrid columns="3">
10            <h:outputText value="Primul numar este" />
11            <h:inputText id="m" value="#{cb.sm}" required="true"
12              validator="#{cb.validateString}" />
13            <h:message for="m" />
14            <h:outputText value="Al doilea numar este" />
15            <h:inputText id="n" value="#{cb.sn}" required="true"
16              validator="#{cb.validateString}" />
17            <h:message for="n" />
18            <h:commandButton id="submit" value="#{cb.compute}"
19              action="#{cb.compute}" />
20          </h:panelGrid>
21        </h:form>
22      </f:view>.
23    </body>
24  </html>

```

- *cmmdcOutput.jsp*

```

1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4   <body>
5     <f:view>
6       <h:outputText value="Cmmdc = #{cb.sresult}" />
7     </f:view>
8   </body>
9 </html>

```

Partea *controller* (`config-faces.xml`) are codul

```

1 <?xml version='1.0' encoding='UTF-8'?>
3 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6     http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
7   version="2.2">
9   <navigation-rule>
10     <description/>
11     <from-view-id>/cmmdcInput.jsp</from-view-id>
12     <navigation-case>
13       <description/>
14       <from-outcome>OK</from-outcome>
15       <to-view-id>/cmmdcOutput.jsp</to-view-id>
16     </navigation-case>
17   </navigation-rule>
19   <managed-bean>
20     <description/>
21     <managed-bean-name>cb</managed-bean-name>
22     <managed-bean-class>cmmdc.CmmdcBean</managed-bean-class>
23     <managed-bean-scope>session</managed-bean-scope>
24   </managed-bean>
25 </faces-config>

```

În exemplul dat, cadrul de lucru JSF injectează valorile celor două numere în componenta Java *cb*. Se pot prelua *manual* datele direct din formular prin intermediul variabilei de tip `ExternalContext`, prin care avem acces la variabile de tip `HttpServletRequest`, `HttpSession`.

```

ExternalContext context =
    FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request=(HttpServletRequest)context.getRequest();
HttpSession session=(HttpSession)context.getSession(true);

```

în care caz, în fișierul *cmmdcInput.jsp* trebuie denumită formularul de preluare prin

```
<h:form id="myform" >
```

Fișierul *CmmdcBean.java* devine

```

1 package cmmdc1;
2 import javax.servlet.http.HttpServletRequest;
3 import javax.faces.context.FacesContext;
4 import javax.faces.context.ExternalContext;
6 public class CmmdcBean implements java.io.Serializable{
7   private String sresult;

```

```

9  public CmmdcBean(){}
11 private long cmmdc(long m,long n){. . .}
13 public String getSresult(){
14     return sresult;
15 }
17 public String compute(){
18     ExternalContext context =
19     FacesContext.getCurrentInstance().getExternalContext();
20     HttpServletRequest request=(HttpServletRequest)context.getRequest();
21     String sm = request.getParameter("myform:sm");
22     String sn = request.getParameter("myform:sn");
23     long m=Long.parseLong(sm);
24     long n=Long.parseLong(sn);
25     long c=cmmdc(m,n);
26     sresult=Long.valueOf(c).toString();
27     return "OK";
28 }
29 }

```

Separarea preocupărilor

Componenta model a aplicației de calcul a celui mai mare divizor comun a două numere, *CmmdcBean* conține cod specific problemei dar și cod care vizează JSF. Arhitectura pe care o vom prezenta separă cele două aspecte:

- Partea specifică, adică calculul celui mai mare divizor comun se face într-o clasă POJO (*Plain Old Java Object*) fără nici o legătură JSF, *CmmdcBean*

```

1  package cmmdc.model;
2  public class CmmdcBean implements java.io.Serializable{
3      private long m=1;
4      private long n=1;
5      private long result;
6
7      public CmmdcBean(){}
8
9      public long getM(){
10         return m;
11     }
12     public void setM(long m){
13         this.m=m;
14     }
15
16     public long getN(){
17         return n;
18     }
19     public void setN(long n){
20         this.n=n;
21     }
22
23     public long getResult(){

```

```

24     return result;
25 }

27 public void compute(){
28     result=cmmdc(m,n);
29 }

31 private long cmmdc(long m,long n){...}
32 }

```

- Aplicația JSF interacționează cu clasa POJO doar prin intermediul unei clase *controller*, *CmmdcController*

```

1 package cmmdc.controller;
2 import javax.faces.component.UIInput;
3 import javax.faces.component.UIOutput;
4 import javax.faces.application.FacesMessage;
5 import javax.faces.context.FacesContext;
6 import cmmdc.model.CmmdcBean;

8 public class CmmdcController implements java.io.Serializable{
9     private CmmdcBean cmmdcBean;
10    private UIInput primulNumar;
11    private UIInput alDoileaNumar;
12    //private UIOutput rezultat;

14    public CmmdcBean getCmmdcBean(){
15        return cmmdcBean;
16    }
17    public void setCmmdcBean(CmmdcBean cmmdcBean){
18        this.cmmdcBean=cmmdcBean;
19    }

21    public UIInput getPrimulNumar(){
22        return primulNumar;
23    }
24    public void setPrimulNumar(UIInput primulNumar){
25        this.primulNumar=primulNumar;
26    }

28    public UIInput getAlDoileaNumar(){
29        return alDoileaNumar;
30    }
31    public void setAlDoileaNumar(UIInput alDoileaNumar){
32        this.alDoileaNumar=alDoileaNumar;
33    }

35    /*
36    public UIOutput getRezultat(){
37        return rezultat;
38    }
39    public void setRezultat(UIOutput rezultat){
40        this.rezultat=rezultat;
41    }
42    */

44    public String cmmdc(){
45        FacesContext facesContext = FacesContext.getCurrentInstance();

```

```

46     try{
47         cmmdcBean.compute();
48         //rezultat.setRendered(true);
49         facesContext.addMessage(null, new FacesMessage(
50             FacesMessage.SEVERITY.INFO, "OK", null));
51     }
52     catch(Exception e)    {
53         //rezultat.setRendered(false);
54         facesContext.addMessage(null, new FacesMessage(
55             FacesMessage.SEVERITY.ERROR, e.getMessage(), null));
56         System.out.println(e.getMessage());
57     }
58     return "OK";
59 }
60 }

```

Varianta comentată corespunde situației în care afișarea rezultatului se face în pagină JSF de preluare a datelor.

Componentele *view* și *controller* depind de locul unde are loc afișarea rezultatului:

- Afișarea se face în alt fișier JSP decât cel în care se preiau datele.

– Componenta *view* este formată din fișierele *cmmdcInput.jsp*

```

1  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3  <html>
4  <body>
5  <h1> Calculul Cmmdc </h1>
6  <f:view>
7  <p>
8  <h:form >
9  <h:panelGrid columns="3">
10 <h:outputText value="Primul numar este" />
11 <h:inputText id="m" required="true"
12     value="#{cmmdcController.cmmdcBean.m}"
13     binding="#{cmmdcController.primulNumar}" />
14 <h:message for="m" />
15 <h:outputText value="Al doilea numar este" />
16 <h:inputText id="n" required="true"
17     value="#{cmmdcController.cmmdcBean.n}"
18     binding="#{cmmdcController.alDoileaNumar}" />
19 <h:message for="n" />
20 <h:commandButton id="submit" value="Calculeaza"
21     action="#{cmmdcController.cmmdc}" />
22 </h:panelGrid>
23 </h:form>
24 </f:view>.
25 </body>
26 </html>

```

și *cmmdcOutput.jsp*

```

1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4   <body>
5     <f:view>
6       <h:outputText value=
7         "Cmmdc = #{cmmdcController.cmmdcBean.result}"/>
8     </f:view>
9   </body>
10 </html>

```

– Componenta *controller* - `faces-config.xml`

```

1 <?xml version='1.0' encoding='UTF-8'?>

3 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6     http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2.2.xsd"
7   version="2.2">

9   <navigation-rule>
10     <description/>
11     <from-view-id>/cmmdcInput.jsp</from-view-id>
12     <navigation-case>
13       <description/>
14       <from-outcome>OK</from-outcome>
15       <to-view-id>/cmmdcOutput.jsp</to-view-id>
16     </navigation-case>
17   </navigation-rule>

19   <managed-bean>
20     <description/>
21     <managed-bean-name>cmmdcController</managed-bean-name>
22     <managed-bean-class>
23       cmmdc.controller.CmmdcController
24     </managed-bean-class>
25     <managed-bean-scope>request</managed-bean-scope>
26     <managed-property>
27       <property-name>cmmdcBean</property-name>
28       <value>#{cmmdcBean}</value>
29     </managed-property>
30   </managed-bean>

32   <managed-bean>
33     <description/>
34     <managed-bean-name>cmmdcBean</managed-bean-name>
35     <managed-bean-class>cmmdc.model.CmmdcBean</managed-bean-class>
36     <managed-bean-scope>none</managed-bean-scope>
37   </managed-bean>
38 </faces-config>

```

- Afişarea se face în acelaşi fişier JSP în care se preiau datele.

– Componenta *view* este formată din fişierul `cmmdc.jsp`

```

1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

```

```

2 <?@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4   <body>
5     <h1> Calculul Cmmdc </h1>
6     <f:view>
7       <p>
8         <h:form >
9           <h:panelGrid columns="3">
10            <h:outputText value="Primul numar este" />
11            <h:inputText id="m" required="true"
12              value="#{cmmdcController.cmmdcBean.m}"
13              binding="#{cmmdcController.primulNumar}" />
14            <h:message for="m" />
15            <h:outputText value="Al doilea numar este" />
16            <h:inputText id="n" required="true"
17              value="#{cmmdcController.cmmdcBean.n}"
18              binding="#{cmmdcController.alDoileaNumar}" />
19            <h:message for="n" />
20            <h:commandButton id="submit" value="Calculeaza"
21              action="#{cmmdcController.cmmdc}" />
22          </h:panelGrid>
23        </h:form>
24        <h:outputFormat binding="#{cmmdcController.rezultat}"
25          rendered="false">
26          <h:outputText
27            value="Cmmdc = #{cmmdcController.cmmdcBean.result}" />
28        </h:outputFormat>
29      </f:view>.
30    </body>
31 </html>

```

– Componenta *controller* - faces-config.xml

```

1 <?xml version='1.0' encoding='UTF-8'?>
2
3 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6     http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig-2.2.xsd"
7   version="2.2">
8
9   <managed-bean>
10     <description />
11     <managed-bean-name>cmmdcController</managed-bean-name>
12     <managed-bean-class>
13       cmmdc.controller.CmmdcController
14     </managed-bean-class>
15     <managed-bean-scope>request</managed-bean-scope>
16     <managed-property>
17       <property-name>cmmdcBean</property-name>
18       <value>#{cmmdcBean}</value>
19     </managed-property>
20   </managed-bean>
21
22   <managed-bean>
23     <description />
24     <managed-bean-name>cmmdcBean</managed-bean-name>
25     <managed-bean-class>cmmdc.model.CmmdcBean</managed-bean-class>
26     <managed-bean-scope>none</managed-bean-scope>

```



```

27 | </managed-bean>
28 | </faces-config>

```

Exemplul 2.3.3 *Un fișier text conține informațiile {nume județ, capitala județului, abrevierea}, separate printr-un spațiu. Se cere construirea unei aplicații Web care pentru un județ indicat, afișează informațiile corespunzătoare din fișierul menționat.*

Inițializarea aplicației constă dintr-o componenta Java *CountyBean* care reține informațiile din fișier într-o variabilă de tip `HashMap<String, RefJudet>`, unde *RefJudet.java* este

```

1 package appjud;
2 public class RefJudet implements java.io.Serializable{
3     private String jud;
4     private String capit;
5     private String abbrev;
6
7     public RefJudet(){
8
9     public void setJud(String jud){
10         this.jud=jud;
11     }
12
13     public String getJud(){
14         return jud;
15     }
16
17     public void setCapit(String capit){
18         this.capit=capit;
19     }
20
21     public String getCapit(){
22         return capit;
23     }
24
25     public void setAbbrev(String abbrev){
26         this.abbrev=abbrev;
27     }
28
29     public String getAbbrev(){
30         return abbrev;
31     }
32 }

```

Această variabilă de tip `HashMap` este reținută de *sesiunea* aplicației. Codul componentei Java *CountyBean* este

```

1 package appjud;
2 import javax.faces.model.SelectItem;
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.Collection;
6 import java.io.InputStream;

```

```

7 import java.io.InputStreamReader;
8 import java.io.BufferedReader;
9 import javax.servlet.http.HttpSession;
10 import javax.faces.context.ExternalContext;
11 import javax.faces.context.FacesContext;

13 public class CountyBean implements java.io.Serializable{
14     private ArrayList<SelectItem> judete=null;

16     private HashMap<String,RefJudet> refJudete=new HashMap<String,RefJudet>();

18     public CountyBean(){
19         judete=new ArrayList<SelectItem>(50);
20         try{
21             InputStream fis=this.getClass().getResourceAsStream("judete.txt");
22             InputStreamReader isr=new InputStreamReader(fis);
23             BufferedReader br=new BufferedReader(isr);
24             String s="",jud,capit,abbrev;
25             do{
26                 s=br.readLine();
27                 if(s!=null){
28                     String[] st=s.split(" ");
29                     jud=st[0];
30                     capit=st[1];
31                     abbrev=st[2];
32                     judete.add(new SelectItem(jud,jud));

34                     RefJudet rj=new RefJudet();
35                     rj.setJud(jud);
36                     rj.setCapit(capit);
37                     rj.setAbbrev(abbrev);
38                     refJudete.put(jud,rj);
39                 }
40             }
41             while(s!=null);
42             ExternalContext context=
43                 FacesContext.getCurrentInstance().getExternalContext();
44             HttpSession session=(HttpSession)context.getSession(true);
45             session.setAttribute("refJudete",refJudete);
46             br.close();
47             isr.close();
48             fis.close();
49         }
50         catch(Exception e){
51             System.out.println("CountyBeanException : "+e.getMessage());
52         }
53     }

55     public Collection<SelectItem> getJudete(){
56         return judete;
57     }
58 }

```

Clasa **SelectItem** din API-ul JSF oferă o modalitate eficientă de completare a unei componente grafice *ComboBox* dintr-un document jsp.

Componenta Java care satisface cererea clientului este *JudBean.java*

```

1 package appjud;

```

```

2 import java.util.HashMap;
3 import javax.servlet.http.HttpSession;
4 import javax.faces.context.ExternalContext;
5 import javax.faces.context.FacesContext;

7 public class JudBean implements java.io.Serializable{
8     private String jud=null;
9     private String capit=null;
10    private String abbrev=null;

12    public JudBean(){ }

14    public String execute(){
15        ExternalContext context =
16            FacesContext.getCurrentInstance().getExternalContext();
17        HttpSession session=(HttpSession)context.getSession(true);
18        HashMap<String, RefJudet> refJudete =
19            (HashMap<String, RefJudet>)session.getAttribute("refJudete");
20        RefJudet rj=refJudete.get(jud);
21        capit=rj.getCapit();
22        abbrev=rj.getAbrev();
23        return "OK";
24    }

26    public void setJud(String jud){
27        this.jud=jud;
28    }

30    public String getJud(){
31        return jud;
32    }

34    public String getCapit(){
35        return capit;
36    }

38    public String getAbrev(){
39        return abbrev;
40    }
41 }

```

Fişierele JSP ataşate aplicaţiei sunt

- *appjudInput.jsp*

```

1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

4 <html>
5     <head>
6         <title> appjud </title>
7     </head>
8     <body>
9         <f:view>
10            <h:outputText value="Informatii despre judete" />
11            <br>
12            <h:form>
13                <h:outputText value="Referinte despre judetul : " />
14                <h:selectOneMenu value="#{judBean.jud}" required="true">

```

```

15         <f:selectItems value="#{countyBean.judete}" />
16     </h:selectOneMenu>
17     <p>
18         <h:commandButton id="submit" value=" Afiseaza "
19             action="#{judBean.execute}" />
20     </h:form>
21 </f:view>
22 </body>
23 </html>

```

● *appjudOutput.jsp*

```

1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4     <body>
5         <f:view>
6             <h:outputText value=" Referintele despre judetul " />
7             <h:outputText value="#{judBean.jud}" />
8             <p>
9                 <h:outputText value=" Capitala : " />
10                <h:outputText value="#{judBean.capit}" />
11            <p>
12                <h:outputText value=" Abrevierea : " />
13                <h:outputText value="#{judBean.abrev}" />
14            </f:view>
15        </body>
16    </html>

```

Fișierul *config-faces.xml* se completează cu

```

<navigation-rule>
    <description/>
    <from-view-id>/appjudInput.jsp</from-view-id>
    <navigation-case>
        <description/>
        <from-outcome>OK</from-outcome>
        <to-view-id>/appjudOutput.jsp</to-view-id>
    </navigation-case>
</navigation-rule>

<managed-bean>
    <description/>
    <managed-bean-name>countyBean</managed-bean-name>
    <managed-bean-class>appjud.CountyBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<managed-bean>
    <description/>
    <managed-bean-name>judBean</managed-bean-name>
    <managed-bean-class>appjud.JudBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

```

2.3.5 Componente grafice *Primefaces*

Biblioteca de componente grafice este declarată, alături de marcajele JSF, prin

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
```

Un fișier cu componente grafice *primefaces* va avea extensia **xhtml** și va avea structura

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
  <h:head>
    . . .
  </h:head>
  <h:body>
    . . .
  </h:body>
</html>
```

Componente grafice *primefaces*:

- **p:panel** Container de componente grafice.

Atribute ale marcajului:

Atribut	Fel	Descriere
id	opțional	Numele simbolic al elementului.
header	opțional	Textul din antet.
footer	opțional	Text subsol.

- **p:panelGrid** Tablou.

Atribute ale marcajului:

Atribut	Fel	Descriere
id	opțional	Numele simbolic al elementului.
columns	opțional	Numărul coloanelor.
cellpadding	opțional	Distanța dintre coloane în pixeli.

- **p:separator** Definește o linie de separație.

- **p:commandButton** Buton de comandă.

Atribute ale marcajului:

Atribut	Fel	Descriere
value	opțional	textul butonului.
actionListener	obligatoriu	Numele acțiunii.
update	opțional	Lista componentelor care vor fi modificate.

- **p:keyboard** Tastatură virtuală de editare.

Atribute ale marcajului:

Atribut	Fel	Descriere
id	opțional	Numele simbolic al elementului.
value	obligatoriu	Câmpul componentei Java alimentat.

- **p:ajax** Activitate desfășurată de *primefaces*.

Atribute ale marcajului:

Atribut	Fel	Descriere
event="change"	opțional	Natura acțiunii <i>primefaces</i> .
listener	opțional	Numele acțiunii care urmărește activitatea.
update	opțional	Lista componentelor care vor fi modificate.

- **p:selectOneMenu** Câmp de alegere a unui opțiuni. Mulțimea opțiunilor este dată în elementul `<f:selectItems value="sursa"/>`

Atribute ale marcajului:

Atribut	Fel	Descriere
id	opțional	Numele simbolic al elementului.
value	opțional	Câmpul componentei Java alimentat.
update	opțional	Lista componentelor care vor fi modificate.

Exemplul 2.3.4

Componenta *view* pentru aplicația de calcul a celui mai mare divizor comun.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:h="http://java.sun.com/jsf/html"
6   xmlns:f="http://java.sun.com/jsf/core"
7   xmlns:p="http://primefaces.org/ui">
8 <h:head>
9 </h:head>
10 <h:body>
11   <h1>Calcul cmmdc</h1>
12   <h:form>
13     <p:panel id="mypanel" header="CMMDC">
14       <p:separator/>
15       <p:panelGrid columns="2" cellpadding="5">
16         <h:outputText id="out1" value="Primul numar:"/>
17         <p:keyboard id="key1" value="#{cb.sm}">
18           <p:ajax event="change" listener="#{cb.validateSm}"
19             update="out0 key1" />
20         </p:keyboard>
21         <h:outputText id="out2" value="Al doilea numar:"/>
22         <p:keyboard id="key2" value="#{cb.sn}">
23           <p:ajax event="change" listener="#{cb.validateSn}"
24             update="out0 key2" />
25         </p:keyboard>
26         <p:commandButton value="Calculeaza!" actionListener="#{cb.compute}"
27           update="out0"/>
28         <h:outputText id="out0" value="#{cb.sresult}"/>
29       </p:panelGrid>
30     </p:panel>
31   </h:form>
32 </h:body>
33 </html>

```

Exemplul 2.3.5

Componenta view pentru exemplul cu afișarea datelor unui județ.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:h="http://java.sun.com/jsf/html"
6   xmlns:f="http://java.sun.com/jsf/core"
7   xmlns:p="http://primefaces.org/ui">
8 <h:head>
9 </h:head>
10 <h:body>
11   <h1>Referințe despre județe</h1>
12   <h:form>
13     <p:panel header="Județe">
14       <p:panelGrid columns="2" cellpadding="5">
15         <h:outputText id="out1" value="Județul:"/>
16         <p:selectOneMenu value="#{judBean.jud}" id="x2" update="out2">
17           <f:selectItems value="#{countyBean.județe}" />
18         </p:selectOneMenu>

```

```

19         <p:commandButton value="Cauta !" actionListener="#{judBean.execute}"
20             update="out3 out4 out5">
21     </p:commandButton>
22     <h:outputText id="out6" value=" "></h:outputText>
23     <h:outputText id="out7" value="Judetul"></h:outputText>
24     <h:outputText id="out3" value="#{judBean.jud}"></h:outputText>
25     <h:outputText id="out8" value="Capitala"></h:outputText>
26     <h:outputText id="out4" value="#{judBean.capit}"></h:outputText>
27     <h:outputText id="out9" value="Abrevierea"></h:outputText>
28     <h:outputText id="out5" value="#{judBean.abrev}"></h:outputText>
29 </p:panelGrid>
30 </p:panel>
31 </h:form>
32 </h:body>
33 </html>

```

Încărcarea unui fișier - Upload

Exemplul 2.3.6 *Aplicație de încărcare a unui fișier text (cu extensia txt) de dimensiune cel mult 1024 de octeți. Fișierul încărcat va fi afișat.*

Soluția bazată pe resursele JSF propriu-zise, utilizând adnotări are codul

```

1 import javax.faces.application.FacesMessage;
2 import javax.faces.bean.ManagedBean;
3 import javax.faces.bean.ViewScoped;
4 import javax.faces.bean.RequestScoped;
5 import javax.faces.component.UIComponent;
6 import javax.faces.context.FacesContext;
7 import javax.faces.context.ExternalContext;
8 import javax.faces.validator.ValidatorException;
9 import javax.servlet.http.Part;
10 import javax.servlet.http.HttpServletRequest;
11 import java.io.IOException;
12 import java.util.ArrayList;
13 import java.util.List;
14 import java.util.Scanner;

16 import java.io.File;
17 import java.io.FileOutputStream;
18 import java.io.OutputStreamWriter;
19 import java.io.BufferedWriter;

21 @ManagedBean(name="uploadPage")
22 @RequestScoped
23 public class UploadPage {
24     private Part uploadFile;
25     private String fileContent;

27     public void validateFile(FacesContext ctx, UIComponent comp,
28         Object value) {
29         List<FacesMessage> msgs = new ArrayList<FacesMessage>();
30         Part file = (Part)value;
31         if (file.getSize() > 1024) {
32             msgs.add(new FacesMessage("file too big"));

```



```

33     }
34     if (!"text/plain".equals(file.getContentType())) {
35         msgs.add(new FacesMessage("not a text file"));
36     }
37     if (!msgs.isEmpty()) {
38         throw new ValidatorException(msgs);
39     }
40 }

42 public String uploadFile() throws Exception{
43     String fs=System.getProperty("file.separator");
44     ExternalContext context =
45         FacesContext.getCurrentInstance().getExternalContext();
46     HttpServletRequest request=(HttpServletRequest)context.getRequest();
47     String fileRef=
48         request.getSession().getServletContext().getRealPath("/") + fs;

50     String myFileName=uploadFile.getSubmittedFileName();
51     System.out.println(myFileName);
52     try {
53         Scanner scanner=new Scanner(uploadFile.getInputStream());
54         StringBuffer sb=new StringBuffer();
55         while(scanner.hasNextLine())
56             sb.append(scanner.nextLine()+'\n');
57         fileContent=sb.toString();
58     }
59     catch (IOException e) {
60         FacesMessage msg =
61             new FacesMessage(FacesMessage.SEVERITY.ERROR,
62                 "error uploading file",
63                 null);
64         FacesContext.getCurrentInstance().addMessage(null, msg);
65     }
66     File f=new File(fileRef+fs+"WEB-INF"+fs+"upload"+fs+myFileName);
67     FileOutputStream fos=new FileOutputStream(f);
68     OutputStreamWriter osw=new OutputStreamWriter(fos);
69     BufferedWriter bw=new BufferedWriter(osw);
70     bw.write(fileContent,0,fileContent.length());
71     bw.close();
72     osw.close();
73     fos.close();
74     return "uploadOutput";
75 }

77 public Part getUploadFile() {
78     return uploadFile;
79 }

81 public void setUploadFile(Part uploadFile) {
82     this.uploadFile = uploadFile;
83 }

85 public String getFileContent() {
86     return fileContent;
87 }
88 }

```

Componente *view* este alcătuită din fişierele

- *upload.xhtml*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:f="http://xmlns.jcp.org/jsf/core"
6       xmlns:h="http://xmlns.jcp.org/jsf/html">
7 <body>
8   <f:view>
9     <h:form id="form" enctype="multipart/form-data">
10       <h:messages/>
11       <h:panelGrid columns="2">
12         <h:outputText value="File:" />
13         <h:inputFile id="file" value="#{uploadPage.uploadFile}"
14           validator="#{uploadPage.validateFile}" />
15       </h:panelGrid>
16       <h:commandButton value="Upload File"
17         action="#{uploadPage.uploadFile}" />
18     </h:form>
19   </f:view>
20 </body>
21 </html>

```

- *uploadOutput.xhtml*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:f="http://xmlns.jcp.org/jsf/core"
6       xmlns:h="http://xmlns.jcp.org/jsf/html">
7 <body>
8   <f:view>
9     <h:panelGrid id="content" columns="1">
10       <h:outputText value="Content:" />
11       <h:inputTextarea readonly="true" value="#{uploadPage.fileContent}"
12         rows="10" cols="100" />
13     </h:panelGrid>
14   </f:view>
15 </body>
16 </html>

```

2.3.6 JSF dezvoltat prin *maven*

Aplicațiile dezvoltate corespund celor dezvoltate anterior, calculul celui mai mare divizor comun (*Cmmdc.java*) și regăsirea datelor unui județ (*JudBean.java*, *ListaJudeteAction.java*, *RefJudet.java*).

Dezvoltarea aplicației constă din:

1. Generarea aplicației

```
set GroupID=myjsfFacelets
```

```

set ArtifactID=myjsf
set Version=1.0
mvn -B archetype:generate
    -DgroupId=%GroupID%
    -DartifactId=%ArtifactID%
    -Dversion=%Version%
    -DarchetypeArtifactId=maven-archetype-webapp

```

2. Se adaptează structura de cataloage și fișiere la

```

myjsfFacelets
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> myjsf
|   |   |   |   |--> cmmdc
|   |   |   |   |   CmmdBean.java
|   |   |   |   |--> cmmdc1
|   |   |   |   |   CmmdBean.java
|   |   |   |   |--> appjud
|   |   |   |   |   JudBean.java
|   |   |   |   |   CountyBean.java
|   |   |   |   |   RefJudet.java
|   |   |--> resources
|   |   |   judete.txt
|   |   |--> webapp
|   |   |   |--> WEB-INF
|   |   |   |   web.xml
|   |   |   |   faces-config.xml
|   |   |   footerTemplate.html
|   |   |   appjudInput.xhtml
|   |   |   appjudOutput.xhtml
|   |   |   apps.xhtml
|   |   |   cmmdcInput.xhtml
|   |   |   cmmdcInput1.xhtml
|   |   |   cmmdcOutput.xhtml
|   |   |   cmmdcOutput1.xhtml
|   |   |   sideMenu.xhtml
|   |   |   template.xhtml
|   |   |   index.html
|   pom.xml

```

În clasa *CountyBean.java* fișierul *judete.txt* se încarcă prin

```
InputStream fis=this.getClass().getResourceAsStream("../judete.txt");
```

3. Fișierul pom.xml se completează cu

```

<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.faces</artifactId>
    <version>*./*.</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>

```

```
<version>2.5</version>
</dependency>
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>javax.servlet.jsp.jstl</artifactId>
  <version>1.2.1</version>
</dependency>
```

unde *.*.* se înlocuiește cu versiunea JSF folosită.

4. Prelucrarea constă din

- (a) `mvn clean package`
- (b) Fișierul `war` care rezultă se desfășoară în serverul Web.

Observația 2.3.1

Dacă se înlocuiesc resursele `jar` de JSF de la *Oracle* cu cele din *apache myfaces* atunci programele funcționează corect fără nici o modificare.

Observația 2.3.2

Utilizând resursele JSF de la *apache myfaces* aplicația JSF este compatibilă cu *google appengine*.

Observația 2.3.3

Aplicațiile asamblate fără nici o resursă funcționează în *apache-tomee-**.

Întrebări recapitulative

1. Ce tehnologie stă la baza unei aplicații Struts2?
2. Pe ce se bazează realizarea interfeței grafice a unei aplicații Struts2?
3. Cum se elimină fișierul de configurare `struts.xml`?
4. Ce tehnologie stă la baza unei aplicații JSF2?
5. Pe ce se bazează realizarea interfeței grafice a unei aplicații JSF2?
6. Cum se elimină fișierul de configurare `faces-config.xml`?

Probleme

1. Eliminarea mesajelor de eroare.
2. WebSocket în JSF.
3. Ajax în JSF.

Capitolul 3

Google Web Toolkit (GWT)

Utilizând GWT, programatorul dezvoltă aplicația în Java și html iar GWT o transformă în JavaScript. Astfel se evită programarea în JavaScript. Într-o aplicație GWT se dezvoltă atât partea de client cât și partea de server. Partea de server este bazată pe tehnologia servlet.

Produsul GWT este distribuit gratuit de Google.

Instalarea produsului constă din dezarhivarea fișierului descărcat din Internet.

O aplicație GWT se dezvoltă

- în linie de comandă, cu **ant**, utilizând fișierul **build.xml** generat în cadrul fiecărei aplicații;
- în *Eclipse*, prin folosirea unei extensii adecvate (plug-in).

3.1 Dezvoltarea unei aplicații GWT prin *ant*

Din punct de vedere al structurii aplicației GWT, acesta poate fi

- simplă, fără apel de procedură la distanță. În acest caz, rezolvarea cererii este programată în clase aflate în catalogul **client**.
- cu apel de procedură la distanță.

O aplicație GWT se inițiază prin generarea unei structuri de cataloage și fișiere. Dacă se dorește realizarea unei aplicații cu punctul de intrare dată de clasa *context.MyApp* și care să se afle într-un catalog *catapp*, atunci generarea se obține prin comanda

```
webAppCreator -out catapp context.MyApp
```

lansată într-o fereastră DOS. *Contextul* poate reprezenta un șir de cataloage. Rezultatul este reprezentat în Fig. 3.1 și corespunde unei aplicații de întâmpinare.

```
catapp
|--> src
|   |--> context
|   |   |--> client
|   |   |   |   MyApp.java
|   |   |   |   GreetingService.java
|   |   |   |   GreetingServiceAsync.java
|   |   |--> server
|   |   |   |   GreetingServiceImpl.java
|   |   |--> shared
|   |   |   |   FieldVerifier.java
|   |   |   |   MyApp.gwt.xml
|--> test
|--> war
|   |--> WEB-INF
|   |   |   web.xml
|   |   |   MyApp.css
|   |   |   MyApp.html
|   |   |   favicon.ico
|   |   .classpath
|   |   .project
|   |   MyApp.launch
|   |   README.txt
|   |   build.xml
```

Figure 3.1: Inițializarea unei aplicații GWT.

Această structură reprezintă un proiect GWT. Spunem că aplicația este generată de perechea *(catapp, context.MyApp)*. Proiect generat este punctul de plecare pentru construirea oricărei alte aplicații, a cărei dezvoltare constă în modificarea, rescrierea fișierelor create și completarea cu altele noi. O asemenea aplicație GWT se mai numește și modul (*module*) GWT.

Fișierul *MyApp.gwt.xml* este un fișier de configurare unde trebuie declarate modulele externe utilizate.

Versiunea inițială este

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3   When updating your version of GWT, you should also update
4   this DTD reference, so that your app can take advantage of
5   the latest GWT module capabilities.
6 -->
7 <!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.5.1//EN"
8   "http://google-web-toolkit.googlecode.com/svn/tags/2.5.1/distro-source/
9   core/src/gwt-module.dtd">
10 <module rename-to='myapp'>
11   <!-- Inherit the core Web Toolkit stuff. -->
```



```

12 <inherits name='com.google.gwt.user.User' />
14 <!-- Inherit the default GWT style sheet. You can change      -->
15 <!-- the theme of your GWT application by uncommenting          -->
16 <!-- any one of the following lines.                             -->
17 <inherits name='com.google.gwt.user.theme.clean.Clean' />
18 <!-- <inherits name='com.google.gwt.user.theme.standard.Standard' /> -->
19 <!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome' /> -->
20 <!-- <inherits name='com.google.gwt.user.theme.dark.Dark' />   -->
22 <!-- Other module inherits                                     -->
24 <!-- Specify the app entry point class.                          -->
25 <entry-point class='unitbv.cs.td.client.MyApp' />
27 <!-- Specify the paths for translatable code                    -->
28 <source path='client' />
29 <source path='shared' />
31 </module>

```

O aplicație GWT poate fi executată în

- modul de dezvoltare. Rularea în acest mod se lansează prin

ant devmode

Verificarea aplicației se face prin intermediul navigatorului implicit.

- modul Web (de producție) - caz în care se generează arhiva **war** a aplicației. Se va executa

ant war

Cu notațiile utilizate mai sus, va rezulta fișierul *MyApp.war*. După desfășurarea aplicației într-un server Web, container de servlet, se va apela `http://host:port/MyApp/MyApp.html`.

3.2 Aplicație GWT fără apel de procedură la distanță

După generarea proiectului, aplicația GWT se dezvoltă parcurgând pașii (se presupune din nou că numele aplicației este *MyApp* aflat în catalogul *catapp*):

1. *Proiectarea interfeței grafice* vizează elementele care se definesc în fișierul *MyApp.html*, punctul de intrare în aplicație. Varianta inițială a fișierului este

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2 <!-- The HTML 4.01 Transitional DOCTYPE declaration-->
3 <!-- above set at the top of the file will set -->
4 <!-- the browser's rendering engine into -->
5 <!-- "Quirks Mode". Replacing this declaration -->
6 <!-- with a "Standards Mode" doctype is supported, -->
7 <!-- but may lead to some differences in layout. -->

9 <html>
10 <head>
11 <meta http-equiv="content-type"
12 <content="text/html; charset=UTF-8">

14 <!-- -->
15 <!-- Consider inlining CSS to reduce the number
16 <!-- of requested files -->
17 <!-- -->
18 <link type="text/css" rel="stylesheet" href="MyApp.css">

20 <!-- -->
21 <!-- Any title is fine -->
22 <!-- -->
23 <title>Web Application Starter Project</title>

25 <!-- -->
26 <!-- This script loads your compiled module. -->
27 <!-- If you add any GWT meta tags, they must -->
28 <!-- be added before this line. -->
29 <!-- -->
30 <script type="text/javascript" language="javascript"
31 <src="myapp/myapp.nocache.js"></script>
32 </head>

34 <!-- -->
35 <!-- The body can have arbitrary html, or -->
36 <!-- you can leave the body empty if you want -->
37 <!-- to create a completely dynamic UI. -->
38 <!-- -->
39 <body>

41 <!-- OPTIONAL: include this if you want history support -->
42 <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
43 <style="position: absolute; width: 0; height: 0; border: 0">
44 </iframe>

46 <!-- RECOMMENDED if your web app will not
47 <!-- function without JavaScript enabled -->
48 <noscript>
49 <div style="width: 22em; position: absolute; left: 50%;
50 <margin-left: -11em; color: red; background-color: white;
51 <border: 1px solid red; padding: 4px; font-family: sans-serif">
52 <Your web browser must have JavaScript enabled
53 <in order for this application to display correctly.
54 </div>
55 </noscript>

57 <h1>Web Application Starter Project</h1>

```

```

59  <table align="center">
60    <tr>
61      <td colspan="2" style="font-weight:bold;">
62        Please enter your name:</td>
63    </tr>
64    <tr>
65      <td id="nameFieldContainer"></td>
66      <td id="sendButtonContainer"></td>
67    </tr>
68  </table>
69 </body>
70 </html>

```

Liniile 56–68 sunt cele care trebuie adaptate aplicației dezvoltate.

Un *widget*¹ (element sau control) grafic va fi redat de navigator într-o fantă (slot) definită, uzual, printr-un container `div`

```
<div id="slot"></div>
```

2. *Construirea interfeței grafice* constă în definirea obiectelor Java care umplu fantele declarate mai sus. Acest lucru se programează în clasa *MyApp.java*, care implementează interfața `EntryPoint`, interfață ce declară doar metoda

```
public void onModuleLoad().
```

Implementarea acestei metode reprezintă tocmai construcția interfeței grafice. Interfața de programare GWT (API) conține o familie de clase *widget*. O instanță a unui *widget* se asociază fantei prin

```
RootPanel.get("slot").add(widget);
```

Dintre clasele *widget* amintim:

- **Label**

Constructori:

- `Label()`
- `Label(String text)`

Metode:

- `public void setText(String text)`

¹*widget=gadget* virtual, *gadget*=dispozitiv amuzant, fără însemnătate practică.

- **TextBox**

Constructori:

- `TextBox()`

Metode:

- `public String getText()`

- **Button**

Constructori:

- `Button(String text)`

Metode:

- `public HandlerRegistration addClickHandler(ClickHandler clickHandler)`

- **Containere de *widget***

`VerticalPanel` `VerticalSplitPanel`

`HorizontalPanel` `HorizontalSplitPanel`

`FlowPanel` `DockPanel`

Un *widget* se include într-un container cu metoda

`void add(Widget widget)`

3. *Generarea evenimentelor.* Activitățile / acțiunile care constituie obiectul aplicației GWT se lansează printr-un clic pe un buton. Fiecărui buton i se atribuie un obiect care implementează interfața `ClickHandler`. Activitățile amintite mai sus sunt definite în codul metodei `public void onClick(ClickEvent event)`.
4. *Programarea activităților corespunzătoare evenimentelor* atașate butoanelor, adică implementarea metodelor `onClick`.
5. *Fixarea elementelor de stil* ale elementelor grafice în fișierul *MyApp.css*. Atașarea la un *widget* a unui element de stil se obține cu metoda `public void addStyleName(String style)`.

Urmărim acești pași în

Exemplul 3.2.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Se generează proiectului GWT `unitbv.cs.td.Cmmdc`

- *Proiectarea interfeței grafice.* Considerăm interfeța grafică conținută într-un container de tip `VerticalPanel` *cmmdcPanel*

`CMMDC` ↔ `Label title`
`m=` ↔ `Label mLabel`
 ↔ `TextBox mTextBox` pentru introducerea primului număr
`n=` ↔ `nLabel`
 ↔ `TextBox nTextBox` pentru introducerea celui de al doilea număr
`Compute` ↔ `Button button`
 ↔ `Label cmmdcLabel` pentru afișarea rezultatului
 Containerul va fi redat în slotul `<div id="cmmdcPage" ></div>`.

- *Construirea interfeței grafice.* Codul care implementează interfața grafică imaginată mai sus este

```

public void onModuleLoad() {
    final Label title=new Label("CMMDC");
    final Label mLabel=new Label("m=");
    final Label nLabel=new Label("n=");
    final Label cmmdcLabel=new Label();
    final TextBox mTextBox=new TextBox();
    final TextBox nTextBox=new TextBox();
    final Button button = new Button("Compute");

    VerticalPanel cmmdcPanel = new VerticalPanel();
    cmmdcPanel.add(title);
    cmmdcPanel.add(mLabel);
    cmmdcPanel.add(mTextBox);
    cmmdcPanel.add(nLabel);
    cmmdcPanel.add(nTextBox);
    cmmdcPanel.add(button);
    cmmdcPanel.add(cmmdcLabel);

    RootPanel.get("cmmdcPage").add(cmmdcPanel);
}

```

- *Generarea evenimentelor.* Butonului i se asociază o instanță a clasei *MyClickHandler*, care conține acțiunile executate după clic pe buton.

```

MyClickHandler clickHandler=new MyClickHandler(mTextBox,nTextBox,cmmdcLabel);
button.addClickHandler(clickHandler);

```

- *Programarea activităților corespunzătoare evenimentelor.* Acest pas core-spunde realizării clasei *MyClickHandler*. Pentru fiecare din cele două numere se verifică

1. dacă sunt furnizare;
2. dacă șirul de caractere introdus este număr.

În cazul în care o condiție de mai sus nu este îndeplinită atunci se afișează un mesaj de atenționare, iar în caz contrar se calculează cel mai mare divizor comun.

Codul clasei *MyClickHandler* este

```

1  class MyClickHandler implements ClickHandler{
2      TextBox mTextBox=null,nTextBox=null;
3      Label cmmdcLabel=null;

5      MyClickHandler(TextBox mTextBox,TextBox nTextBox,Label cmmdcLabel){
6          this.mTextBox=mTextBox;
7          this.nTextBox=nTextBox;
8          this.cmmdcLabel=cmmdcLabel;
9      }

11     private long cmmdc(long m,long n){. . .}

13     public void onClick(ClickEvent event){
14         String sm=mTextBox.getText();
15         String sn=nTextBox.getText();
16         long m=0,n=0;
17         if(sm.equals("")){
18             Window.alert("'m\'' nu este dat");
19             //GWT.log("'m\'' nu este dat",null);
20             cmmdcLabel.setText("?");
21             return;
22         }
23         if(sn.equals("")){
24             Window.alert("'n\'' nu este dat");
25             //GWT.log("'n\'' nu este dat",null);
26             cmmdcLabel.setText("?");
27             return;
28         }
29         try{
30             m=Long.parseLong(sm);
31         }
32         catch(NumberFormatException e){
33             Window.alert("'m\'' nu este numar");
34             cmmdcLabel.setText("?");
35             return;
36         }
37         try{
38             n=Long.parseLong(sn);
39         }
40         catch(NumberFormatException e){
41             Window.alert("'n\'' nu este numar");
42             cmmdcLabel.setText("?");
43             return;
44         }
45         long c=0;
46         if((m!=0)&&(n!=0)) c=cmmdc(m,n);
47         cmmdcLabel.setText("Cmmdc="+Long.valueOf(c).toString());
48     }
49 }

```

- *Fixarea elementelor de stil.* Fișierul *Cmmdc.css* conține

```

.pc-template-btn {
    display: block;
    font-size: 16pt;
}

.label-title{
    font-family: Georgia, "Times New Roman", Times, serif;
    font-weight: bold;
    font-size: 18pt
}

```

Codul complet al clasei *Cmmdc.java* este

```

1 package unitbv.cs.td.client;
2 import com.google.gwt.event.dom.client.ClickEvent;
3 import com.google.gwt.event.dom.client.ClickHandler;
4 import com.google.gwt.core.client.EntryPoint;
5 import com.google.gwt.user.client.ui.Button;
6 import com.google.gwt.user.client.ui.RootPanel;
7 import com.google.gwt.user.client.ui.VerticalPanel;
8 import com.google.gwt.user.client.ui.TextBox;
9 import com.google.gwt.user.client.ui.Label;
10 import com.google.gwt.user.client.Window;

12 public class Cmmdc implements EntryPoint {
13     public void onModuleLoad() {
14         final Label title=new Label("CMMDC");
15         title.addStyleName("label-title");
16         final Label mLabel=new Label("m=");
17         final Label nLabel=new Label("n=");
18         final Label cmmdcLabel=new Label();
19         final TextBox mTextBox=new TextBox();
20         final TextBox nTextBox=new TextBox();
21         final Button button = new Button("Compute");
22         button.addStyleName("pc-template-btn");
23         VerticalPanel cmmdcPanel = new VerticalPanel();
24         cmmdcPanel.setWidth("100%");
25         cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
26         cmmdcPanel.add(title);
27         cmmdcPanel.add(mLabel);
28         cmmdcPanel.add(mTextBox);
29         cmmdcPanel.add(nLabel);
30         cmmdcPanel.add(nTextBox);
31         MyClickHandler clickHandler=new MyClickHandler(mTextBox,nTextBox,
32             cmmdcLabel);
33         button.addClickHandler(clickHandler);
34         cmmdcPanel.add(button);
35         cmmdcPanel.add(cmmdcLabel);
36         RootPanel.get("cmmdcPage").add(cmmdcPanel);
37     }
38 }

```

3.3 Aplicație GWT cu apel de procedură la distanță

GWT asigură posibilitatea legăturii dintre o aplicație dezvoltată în acest mediu și un servlet. GWT oferă două variante de dezvoltare:

- Client al unei aplicații server exterioare aplicației GWT, caz în care vorbim de client http;

Serverul și clientul GWT trebuie să ruleze în același server Web (*Same Origin Policy*).

- Client și aplicația server aparțin aceleiași aplicații GWT, caz în care vorbim de apel de procedură la distanță GWT.

Schimbările de date au loc potrivit tehnologiei AJAX. În terminologia Google se vorbește de servicii implementate printr-un server apelat de o componentă client GWT. Serverul are o construcție specifică GWT.

Clientul este responsabil de trimiterea cererilor și de recepția răspunsurilor, iar partea de server de rezolvarea cererilor. Astfel se justifică terminologia de apel de procedură la distanță GWT. Atât clientul cât și serverul utilizează resurse GWT.

Client HTTP

La programarea unui client HTTP sarcina programatorului constă în

1. Lansarea cererii către server, în cazul de față un servlet;
2. Recepționarea și prelucrarea răspunsului.

Clasele implicate în comunicația cu servlet-ul aparțin pachetului `com.google.gwt.http.client`.

Recepționarea și prelucrarea răspunsului se programează implementând interfața `RequestCallback`, cu metodele

- `public void onError(Request request, Throwable exception)`
- `public void onResponseReceived(Request request, Response response)`

Un șablon simplu este


```
class MyRequestCallback implements RequestCallback{
    private static final int STATUS_CODE_OK=200;

    public void onError(Request request,Throwable exception){
        // tratarea erorii
    }

    public void onResponseReceived(Request request,Response response){
        int sc=response.getStatusCode();
        if(sc==STATUS_CODE_OK){
            // prelucrarea raspunsului
        }
        else{
            // tratarea mesajului receptionat de la server
        }
    }
}
```

Șablonul de lansare a unei cereri GET către servlet este

```
public void doGet(String url){
    RequestBuilder rb=new RequestBuilder(RequestBuilder.GET,url);
    try{
        Request response=rb.sendRequest(null, RequestCallback);
    }
    catch(RequestException e){
        // tratarea exceptiei
    }
}
```

Șablonul de lansare a unei cereri POST către servlet este

```
public void doPost(String url,String postData){
    RequestBuilder rb=new RequestBuilder(RequestBuilder.POST,url);
    rb.setHeader("Content-Type","application/x-www-form-urlencoded");
    try{
        Request response=rb.sendRequest(postData, RequestCallback);
    }
    catch(RequestException e){
        // tratarea exceptiei
    }
}
```

```
}

```

Pregătirea datelor în vederea expedierii se poate programa prin

```
StringBuffer sb=new StringBuffer();
String encodedParam=URL.encodeComponent(paramName)+"="+
    URL.encodeComponent(paramValue);
sb.append(encodedParam);
sb.append("&");
. . .
String postData=sb.toString();
```

O aplicație client http se poate verifica doar în modul Web.

Exemplul 3.3.1 *Client http dezvoltat pentru servlet-ul CmmdcServlet. Interfața grafică este identică cu cea din aplicația anterioară, de calcul a celui mai mare divizor comun a două numere naturale (Cmmdc.java).*

Metodele `onModuleLoad` ale claselor *HttpClient* și *Cmmdc* coincid. Reamintim că servlet-ul *CmmdcServlet* preia 3 parametri: cele două numere (m, n) ale căror cel mai mare divizor comun se calculează și un parametru *tip* care precizează natura răspunsului ("text/html" sau "text/plain").

Apelarea servlet-ului este făcută în metoda `onClick()` a clasei ce implementează interfața `ClickHandler`, în cazul de față *MyClickHandler*.

Codul clientului este

```
1 package unitbv.cs.td.client;
2 import com.google.gwt.core.client.EntryPoint;
3 import com.google.gwt.user.client.ui.*;
4 import com.google.gwt.http.client.*;
5 import com.google.gwt.user.client.Window;
6 import com.google.gwt.event.dom.client.*;

8 public class HttpClient implements EntryPoint {
9     public void onModuleLoad() {
10         Label title=new Label("CMMDC");
11         title.addStyleName("label-title");
12         Label mLabel=new Label("m=");
13         Label nLabel=new Label("n=");
14         Label cmmdcLabel=new Label();
15         TextBox mTextBox=new TextBox();
16         TextBox nTextBox=new TextBox();
17         Button button = new Button("Compute");
18         button.addStyleName("pc-template-btn");
19         VerticalPanel cmmdcPanel = new VerticalPanel();
20         cmmdcPanel.setWidth("100%");
21         cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
22         cmmdcPanel.add(title);
23         cmmdcPanel.add(mLabel);
```

```

24     cmmdcPanel.add(mTextBox);
25     cmmdcPanel.add(nLabel);
26     cmmdcPanel.add(nTextBox);
27     MyClickHandler clickHandler=
28         new MyClickHandler(mTextBox,nTextBox,cmmdcLabel);
29     button.addClickHandler(clickHandler);
30     cmmdcPanel.add(button);
31     cmmdcPanel.add(cmmdcLabel);
32     RootPanel.get("cmmdcPage").add(cmmdcPanel);
33 }
34 }

36 class MyClickHandler implements ClickHandler{
37     TextBox mTextBox=null,nTextBox=null;
38     Label cmmdcLabel=null;

40     MyClickHandler(TextBox mTextBox,TextBox nTextBox,Label cmmdcLabel){
41         this.mTextBox=mTextBox;
42         this.nTextBox=nTextBox;
43         this.cmmdcLabel=cmmdcLabel;
44     }

46     public void doGet(String url){
47         RequestBuilder rb=new RequestBuilder(RequestBuilder.GET,url);
48         MyRequestCallback rc=new MyRequestCallback(cmmdcLabel);
49         try{
50             Request response=rb.sendRequest(null,rc);
51         }
52         catch(RequestException e){
53             cmmdcLabel.setText("RequestException : "+e.getMessage());
54         }
55     }

57     public void onClick(ClickEvent event) {
58         String url="/appcmmdc/cmmdc";
59         String sm=mTextBox.getText();
60         String sn=nTextBox.getText();
61         long m=0,n=0;
62         if(sm.equals("")){
63             Window.alert("'m\' nu este dat");
64             cmmdcLabel.setText("?");
65             return;
66         }
67         if(sn.equals("")){
68             Window.alert("'n\' nu este dat");
69             cmmdcLabel.setText("?");
70             return;
71         }
72         try{
73             m=Long.parseLong(sm);
74         }
75         catch(NumberFormatException e){
76             Window.alert("'m\' nu este numar");
77             cmmdcLabel.setText("?");
78             return;
79         }
80         try{
81             n=Long.parseLong(sn);
82         }

```

```

83     catch (NumberFormatException e){
84         Window.alert("\'n\' nu este numar");
85         cmmdcLabel.setText("?");
86         return;
87     }
88     if ((m!=0)&&(n!=0)){
89         String urlExtins=url+"?m="+sm+"&n="+sn+"&tip=text/plain";
90         doGet(urlExtins);
91     }
92 }
93 }

95 class MyRequestCallback implements RequestCallback{
96     Label label;
97     private static final int STATUS_CODE_OK=200;

99     MyRequestCallback(Label label){
100         this.label=label;
101     }

103     public void onError(Request request, Throwable e){
104         label.setText("Connection error : "+e.getMessage());
105     }

107     public void onResponseReceived(Request request, Response response){
108         int sc=response.getStatusCode();
109         if (sc==STATUS_CODE_OK){
110             label.setText(response.getText());
111         }
112         else{
113             label.setText("STATUS CODE : "+sc);
114         }
115     }
116 }

```

Apel de procedură la distanță GWT

Construcția clientului și a serverului este supusă unor restricții: metodele serviciului sunt definite într-o interfață care este implementată de server în clase POJO.

Legătura dintre client și server se face prin intermediul fișierului `web.xml` prin adăugarea elementelor

```

<servlet>
    <servlet-name>nume-servlet</servlet-name>
    <servlet-class>...server.ServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>nume-servlet</servlet-name>
    <url-pattern>/myapp/urlPattern</url-pattern>
</servlet-mapping>

```

Datele din elementul `xml <url-pattern>` cuprind

- Denumirea unui catalog *myapp*. Catalogul apare numai în arhiva war iar numele coincide cu numele aplicației (clasei client) scris cu litere mici.
- Parametrul *urlPattern* va fi utilizat în clasa client la apelarea metodelor oferite de server.

Dezvoltarea clientului

Partea de client este alcătuită din:

1. Interfața serviciului.

Metodele care pot fi invocate de client se declară într-o interfață

```
package ...client;
import com.google.gwt.user.client.rpc.RemoteService;

public interface InterfataService extends RemoteService{
    public tip metoda1(tip var,.. . .);
    public tip metoda2(tip var,.. . .);
    . . .
}
```

2. Interfața asincronă.

Interfața asincronă are rolul de a semnală clientului recepționarea răspunsului dat de server. Numele acestei interfețe se formează din numele interfeței serviciului plus sufixul **Async**. Cele două interfețe conțin aceleași metode dar au semnături diferite.

```
package ...client;
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface InterfataServiceAsync{
    public void metoda1(tip var,.. . .,AsyncCallback<tip> callback);
    public void metoda2(tip var,.. . .,AsyncCallback<tip> callback);
    . . .
}
```

tip corespunde clasei obiectului returnat de metodă.

Dacă o metodă returnează o valoare de tip predefinit atunci *tip* va fi clasa acoperitoare.

Dacă metoda reîntoarce `void` atunci *tip*=`Void`.

3. Interfață asincronă este implementată de client și conține prelucrările referitoare la recepția răspunsului furnizat de server. Interfața `AsyncCallback` conține două metode

- `public void onSuccess(Object result)`
- `public void onFailure(Throwable caught)`

Programatorul definește acțiunile care se fac în cazul în care apelul metodei la distanță s-a terminat cu succes, respectiv cu insucces.

Structura clasei este

```
package ...client;

import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.*;

public class ClientCallback implements AsyncCallback {
    public void onFailure(Throwable caught) {
        GWT.log("Error ", caught);
        caught.printStackTrace();
        Window.alert(caught.toString());
    }

    public void onSuccess(Object result) {
        // cod client
        . . .
    }
}
```

4. Clientul propriu-zis este o aplicație GWT

```

package ...client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.user.client.rpc.ServiceDefTarget;

public class Client implements EntryPoint {

    public void apelMetoda1(. . .){
        InterfataServiceAsync myService=
            (InterfataServiceAsync)GWT.create(InterfataService.class);
        ServiceDefTarget target=(ServiceDefTarget)myService;
        String relativeURL=GWT.getModuleBaseURL()+"urlPattern";
        target.setServiceEntryPoint(relativeURL);
        myService.metoda1(variabile actuale pentru metoda1,
            new ClientCallback());
    }

    public void onModuleLoad(){
        . . .
        final Button button=new Button(. . .);
        . . .

        button.addClickHandler(new ClickHandler(){
            public void onClick(ClickEvent event){
                . . .
                apelMetoda1(lista variabile);
            }
        });
        . . .
        RootPanel.get("slot").add(button);
    }
}

```

unde

- *urlPattern* este cel precizat în fișierul *web.xml*, fără numele catalogului.

- *slot* corespunde containerului `div` în care se include butonul.

Dezvoltarea serverului

Programul server implementează metodele interfeței declarată în pachetul client.

```
package ...server;
import client.*;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class ServiceImpl extends RemoteServiceServlet
    implements InterfataService{
    public tip metoda1(tip var,.. .){. . .}
    public tip metoda2(tip var,.. .){. . .}
    . . .
}
```

Exemplul 3.3.2 *Transformăm programul GWT de calcul a celui mai mare divizor comun a două numere naturale într-o aplicație cu apel la distanță GWT.*

Interfața aplicației este

```
1 package unitbv.cs.td.client;
2 import com.google.gwt.user.client.rpc.RemoteService;
3
4 public interface CmmdcService extends RemoteService{
5     public long cmmdc(long m,long n);
6 }
```

iar interfața asincronă

```
1 package unitbv.cs.td.client;
2 import com.google.gwt.user.client.rpc.AsyncCallback;
3
4 public interface CmmdcServiceAsync{
5     public void cmmdc(long m,long n,AsyncCallback<Long> callback);
6 }
```

Implementarea interfeței asincrone este

```
1 package unitbv.cs.td.client;
2
3 import com.google.gwt.user.client.rpc.AsyncCallback;
4 import com.google.gwt.core.client.GWT;
5 import com.google.gwt.user.client.Window;
6 import com.google.gwt.user.client.ui.RootPanel;
7 import com.google.gwt.user.client.ui.VerticalPanel;
8 import com.google.gwt.user.client.ui.HTML;
```



```

10 public class CmmdcCallback implements AsyncCallback {
11     public void onFailure(Throwable caught) {
12         GWT.log("Error ", caught);
13         caught.printStackTrace();
14         Window.alert(caught.toString());
15     }
16     public void onSuccess(Object result) {
17         // Varianta 1
18         HTML html =
19             new HTML("<h1 style='border:1px gray solid;color:red;'>" +
20                 "Cmmdc = " + result.toString() + "</h1>");
21         VerticalPanel vp = new VerticalPanel();
22         vp.add(html);
23         RootPanel.get().add(vp);

24         // Varianta 2
25         // long rez=((Long)result).longValue();
26         // Window.alert("Cmmdc="+rez);
27     }
28 }
29 }

```

cu clientul

```

1 import com.google.gwt.core.client.EntryPoint;
2 import com.google.gwt.user.client.ui.Button;
3 import com.google.gwt.user.client.ui.RootPanel;
4 import com.google.gwt.user.client.ui.VerticalPanel;
5 import com.google.gwt.user.client.ui.TextBox;
6 import com.google.gwt.user.client.ui.Label;
7 import com.google.gwt.user.client.Window;
8 import com.google.gwt.core.client.GWT;
9 import com.google.gwt.user.client.rpc.ServiceDefTarget;
10 import com.google.gwt.event.dom.client.ClickEvent;
11 import com.google.gwt.event.dom.client.ClickHandler;

12
13 public class CmmdcClient implements EntryPoint {
14     public void getResult(long m, long n){
15         CmmdcServiceAsync cmmdcService=
16             (CmmdcServiceAsync)GWT.create(CmmdcService.class);
17         ServiceDefTarget sdt=(ServiceDefTarget)cmmdcService;
18         String endpoint=GWT.getModuleBaseURL()+"cmmdcrpc";
19         sdt.setServiceEntryPoint(endpoint);
20         cmmdcService.cmmdc(m,n,new CmmdcCallback());
21     }

22
23     public void onModuleLoad() {
24         Label title=new Label("CMMDC");
25         title.addStyleName("label-title");
26         Label mLabel=new Label("m=");
27         Label nLabel=new Label("n=");
28         final Label cmmdcLabel=new Label();
29         final TextBox mTextBox=new TextBox();
30         final TextBox nTextBox=new TextBox();
31         Button button = new Button("Compute");
32         button.addStyleName("pc-template-btn");
33         VerticalPanel cmmdcPanel = new VerticalPanel();
34         cmmdcPanel.setWidth("100%");
35         cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
36         cmmdcPanel.add(title);

```

```

37 cmmdcPanel.add(mLabel);
38 cmmdcPanel.add(mTextBox);
39 cmmdcPanel.add(nLabel);
40 cmmdcPanel.add(nTextBox);
41 button.addClickHandler(new ClickHandler(){
42     public void onClick(ClickEvent event){
43         String sm=mTextBox.getText();
44         String sn=nTextBox.getText();
45         long m=0,n=0;
46         if(sm.equals("")){
47             Window.alert("'m\'," nu este dat");
48             cmmdcLabel.setText("?");
49             return;
50         }
51         if(sn.equals("")){
52             Window.alert("'n\'," nu este dat");
53             cmmdcLabel.setText("?");
54             return;
55         }
56         try{
57             m=Long.parseLong(sm);
58         }
59         catch(NumberFormatException e){
60             Window.alert("'m\'," nu este numar");
61             cmmdcLabel.setText("?");
62             return;
63         }
64         try{
65             n=Long.parseLong(sn);
66         }
67         catch(NumberFormatException e){
68             Window.alert("'n\'," nu este numar");
69             cmmdcLabel.setText("?");
70             return;
71         }
72         long c=0;
73         if((m!=0)&&(n!=0))
74             getResult(m,n);
75     }
76 });
77 cmmdcPanel.add(button);
78 cmmdcPanel.add(cmmdcLabel);
79 RootPanel.get("cmmdcPage").add(cmmdcPanel);
80 }
81 }

```

Serverul, adică implementarea serviciului este

```

1 package unitbv.cs.td.server;
2 import com.google.gwt.user.server.rpc.RemoteServiceServlet;
3 import unitbv.cs.td.client.CmmdcService;

5 public class CmmdcServiceImpl extends RemoteServiceServlet
6     implements CmmdcService{
7     public long cmmdc(long m,long n){. . .}
8 }

```

web.xml se completează cu

```
<servlet>
```

```

    <servlet-name>cmmdcrpc</servlet-name>
    <servlet-class>unitbv.cs.td.server.CmmdcServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>cmmdcrpc</servlet-name>
    <url-pattern>/cmmdcclient/cmmdcrpc</url-pattern>
</servlet-mapping>

```

Test GWT

Generând modulul GWT cu opțiunea suplimentară *-junit*

```
webAppCreator -out catapp -junit cale_către\junit*.jar context.MyApp
```

în catalogul *catapp* se crează suplimentar structura

```

catapp
. . .
|--> test
|   |--> context
|       |   |--> client
|       |       |   MyAppTest.java
|       |       |   MyAppJUnit.gwt.xml
. . .

```

Obiectivele **ant**: **test.dev** și **test.prod** asigură efectuarea testelor prin intermediul lui *junit*.

În cazul unei aplicații cu apel de procedură la distanță GWT sarcina programatorului constă din (cu adaptarea corespunzătoare a denumirii clasei *MyApp*):

- Editarea clasei *MyAppTest* pe baza șablonului prezent;
- Modificarea / adaptarea corespunzătoare a fișierului de configurare *MyAppJUnit.gwt.xml*.

Pentru aplicația anterioară codul clasei *unitbv.cs.td.client.CmmdcClientJUnit* este

```

1 package unitbv.cs.td.client;

3 import unitbv.cs.td.shared.FieldVerifier;
4 import com.google.gwt.core.client.GWT;
5 import com.google.gwt.junit.client.GWTTestCase;
6 import com.google.gwt.user.client.rpc.AsyncCallback;
7 import com.google.gwt.user.client.rpc.ServiceDefTarget;

9 public class CmmdcClientTest extends GWTTestCase {

11     public String getModuleName() {
12         return "unitbv.cs.td.CmmdcClientJUnit";

```

```

13 }
14
15 public void testCmmdcService() {
16     // Crearea unui obiect de apelare.
17     CmmdcServiceAsync cmmdcService = GWT.create(CmmdcService.class);
18     ServiceDefTarget target = (ServiceDefTarget) cmmdcService;
19     target.setServiceEntryPoint(GWT.getModuleBaseURL() +
20         "cmmdcclient/cmmdcrpc");
21     delayTestFinish(10000);
22
23     // Apel server.
24     cmmdcService.cmmdc(56,48, new AsyncCallback<Long>() {
25         public void onFailure(Throwable caught) {
26             fail("Request failure: " + caught.getMessage());
27         }
28
29         public void onSuccess(Long result) {
30             assertTrue(result.longValue()==8);
31             finishTest();
32         }
33     });
34 }
35 }

```

Fișierul de configurare *CmmdcClientJUnit.gwt.xml* are codul

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <module>
3     <!-- Inherit our applications main module. -->
4     <inherits name='unitbv.cs.td.CmmdcClient' />
5
6     <!-- Specify the path to any remote services. -->
7     <servlet path="/cmmdcclient/cmmdcrpc"
8         class="unitbv.cs.td.server.CmmdcServiceImpl" />
9
10 </module>

```

Exemplul 3.3.3 *Aplicație de consultare a unei baze de date. Se consideră o bază de date AgendaEMail alcătuită dintr-un singur tabel adrese (nume varchar(20), email varchar(30)). Se cere realizarea unei aplicații de consultare a agendei de adrese e-mail.*

Codul interfeței

```

1 package unitbv.cs.td.client;
2 import java.util.List;
3 import com.google.gwt.user.client.rpc.RemoteService;
4
5 public interface AdreseService extends RemoteService{
6     public List<String> getEmail(String nume);
7     public String getNume(String email);
8 }

```

Codul interfeței asincrone

```

1 package unitbv.cs.td.client;
2 import com.google.gwt.user.client.rpc.AsyncCallback;
3 import java.util.List;

5 public interface AdreseServiceAsync{
6     public void getEmail(String nume,AsyncCallback<List<String>> callback);
7     public void getNume(String email,AsyncCallback<String> callback);
8 }

```

Implementarea interfeței asincrone este (*AdreseCallback.java*)

```

1 package unitbv.cs.td.client;

3 import com.google.gwt.user.client.rpc.AsyncCallback;
4 import com.google.gwt.core.client.GWT;
5 import com.google.gwt.user.client.Window;
6 import com.google.gwt.user.client.ui.*;
7 import java.util.*;

9 public class AdreseCallback implements AsyncCallback {
10     public void onFailure(Throwable caught) {
11         GWT.log(" Error ", caught);
12         caught.printStackTrace();
13         Window.alert(caught.toString());
14     }

16     public void onSuccess(Object result) {
17         final Label labelRez=new Label();
18         Grid grid=new Grid(1,2);
19         if(result instanceof String){
20             String nume=(String)result;
21             final Label labelNume=new Label((String)result);
22             labelRez.setText("Numele cautat : ");
23             grid.setWidget(0,0,labelRez);
24             grid.setWidget(0,1,labelNume);
25             RootPanel.get("rez").add(grid);
26         }
27         if(result instanceof List){
28             final ListBox listAdrese=new ListBox();
29             List<String> list=(List<String>)result;
30             for(int i=0;i<list.size();i++){
31                 listAdrese.addItem((String)list.get(i));
32             }
33             labelRez.setText("Adresele cautate : ");
34             grid.setWidget(0,0,labelRez);
35             grid.setWidget(0,1,listAdrese);
36             RootPanel.get("rez").add(grid);
37         }
38     }
39 }

```

Codul programului client este

```

1 package unitbv.cs.td.client;

3 import com.google.gwt.core.client.EntryPoint;
4 import com.google.gwt.core.client.GWT;
5 import com.google.gwt.user.client.ui.*;
6 import com.google.gwt.user.client.rpc.ServiceDefTarget;

```

```

7 import com.google.gwt.event.dom.client.ClickEvent;
8 import com.google.gwt.event.dom.client.ClickHandler;
9 import java.util.*;

11 public class AgendaEMail implements EntryPoint {
12     public void findByNume(String nume){
13         AdreseServiceAsync adreseService=
14             (AdreseServiceAsync)GWT.create(AdreseService.class);
15         ServiceDefTarget target=(ServiceDefTarget) adreseService;
16         String relativeURL=GWT.getModuleBaseURL()+" adrese";
17         target.setServiceEntryPoint(relativeURL);
18         adreseService.getEmail(nume, new AdreseCallback());
19     }

21     public void findByEMail(String email){
22         AdreseServiceAsync adreseService=
23             (AdreseServiceAsync)GWT.create(AdreseService.class);
24         ServiceDefTarget target=(ServiceDefTarget) adreseService;
25         String relativeURL=GWT.getModuleBaseURL()+" adrese";
26         target.setServiceEntryPoint(relativeURL);
27         adreseService.getNum(e-mail, new AdreseCallback());
28     }

30     public void onModuleLoad(){
31         final Button button=new Button("Cauta");
32         final Label labelCriteriu=new Label("Criteriul de cautare");
33         final Label labelEntitate=new Label("Entitatea cautata");
34         final ListBox listBoxCriteriu=new ListBox();
35         final TextBox textBoxEntitate=new TextBox();

37         VerticalPanel adresePanel=new VerticalPanel();
38         Label title=new Label("Agenda de adrese E-MAIL");
39         title.addStyleName("label-title");
40         Grid grid=new Grid(2,2);
41         grid.setWidget(0,0,labelCriteriu);
42         listBoxCriteriu.addItem("nume");
43         listBoxCriteriu.addItem("email");
44         listBoxCriteriu.setVisibleItemCount(1);
45         grid.setWidget(0,1,listBoxCriteriu);
46         grid.setWidget(1,0,labelEntitate);
47         grid.setWidget(1,1,textBoxEntitate);
48         button.addClickHandler(new ClickHandler(){
49             public void onClick(ClickEvent event){
50                 String s=textBoxEntitate.getText();
51                 if(!"".equals(s)){
52                     if(listBoxCriteriu.getSelectedIndex()==0)
53                         findByNume(s);
54                     else
55                         findByEMail(s);
56                 }
57             }
58         });
59         adresePanel.add(title);
60         adresePanel.add(grid);
61         adresePanel.add(button);
62         RootPanel.get(" adresePanel").add(adresePanel);
63     }
64 }

```

Partea specifică aplicației din pagina Web (*AgendaTelefonica.html*) este

```
<h1>AgendaEMail</h1>
<p>
<div id="adresePanel"> </div>
</p>
<p>
<div id="rez"> </div>
</p>
```

cu fixarea containerelor în care GWT include *widget*-urile grafice. Implementarea interfeței, adică serverul, are codul

```
1 package unitbv.cs.td.server;
2 import unitbv.cs.td.client.*;
3 import java.util.*;
4 import java.sql.*;
5 import com.google.gwt.user.server.rpc.RemoteServiceServlet;
6
7 public class AdreseServiceImpl extends RemoteServiceServlet
8     implements AdreseService{
9     public static final String DRIVER=
10         "org.apache.derby.jdbc.ClientDriver";
11     public static final String PROTOCOL=
12         "jdbc:derby://localhost:1527/AgendaEMail";
13
14     public List<String> getEmail(String nume){
15         List<String> adrese;
16         try {
17             Class.forName(DRIVER).newInstance();
18             Connection con = DriverManager.getConnection(PROTOCOL);
19             Statement s = con.createStatement();
20             ResultSet rs =
21                 s.executeQuery("SELECT email FROM adrese where nume='"+nume+"'");
22             adrese=new ArrayList<String>();
23             while(rs.next()){
24                 adrese.add(rs.getString("email"));
25             }
26         }
27         catch (Exception e) {
28             e.printStackTrace();
29             adrese=null;
30         }
31         finally {
32             try {
33                 DriverManager.getConnection("jdbc:derby::shutdown=true");
34             }
35             catch (SQLException ignore) {}
36         }
37         return adrese;
38     }
39
40     public String getNume(String email) {
41         String nume="";
42         try {
43             Connection con = DriverManager.getConnection(PROTOCOL);
44             Statement s = con.createStatement();
45             ResultSet rs =
46                 s.executeQuery("SELECT nume FROM adrese where email='"+email+"'");
```

```

47         if(rs.next())
48             nume=rs.getString("nume");
49     }
50     catch (Exception e) {
51         e.printStackTrace();
52         nume=null;
53     }
54     finally {
55         try {
56             DriverManager.getConnection("jdbc:derby::shutdown=true");
57         }
58         catch (SQLException ignore) {}
59     }
60     return nume;
61 }
62 }

```

web.xml se completează cu

```

<servlet>
  <servlet-name>adrese</servlet-name>
  <servlet-class>unitbv.cs.td.server.AdreseServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>adrese</servlet-name>
  <url-pattern>/agendaemail/adrese</url-pattern>
</servlet-mapping>

```

În plus, driverul bazei de date trebuie copiat în catalogul `war\WEB-INF\lib`.

3.4 Crearea unui widget client

În paralel cu oferta de clase *widget* puse la dispoziție de GWT există posibilitatea de a crea *widget*-e proprii sau client.

O variantă de creare este prezentată în continuare.

O arhitectură posibilă a unei asemenea aplicații este dată în Fig. 3.2. Desfășurarea bibliotecii de *widget*-e este delimitată prin linii orizontale. *Widget*-ele aparțin pachetului *mywidgets...client*. *MyWidget.css* conține clasele *css* utilizate de *widget*-uri. Acest fapt este specificat în fișierul de configurare *Widgets.gwt.xml*, având codul

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 1.6.4//EN"
3   "http://google-web-toolkit.googlecode.com/svn/tags/1.6.4/
4   distro-source/core/src/gwt-module.dtd">
5 <module rename-to='mywidgets'>
6   <!-- Inherit the core Web Toolkit stuff. -->
7   <inherits name='com.google.gwt.user.User' />

9   <stylesheet src='MyWidget.css' />
10 </module>

```



```

catapp
|--> src
-----
|    |--> mywidgets
|    |    |. . .
|    |    |--> client
|    |    |    | WidgetClass1.java
|    |    |    |    |. . .
|    |    |--> public
|    |    |    | MyWidget.css
|    |    |    | Widgets.gwt.xml
|    |    |
|    |
|    |--> context
|    |    |--> client
|    |    |    | MyApp.java
|    |    |    |. . .
|    |--> war
|    |    |. . .
|    .classpath
|    .project
|    MyApp.launch
|    README.txt
|    build.xml

```

Figure 3.2: Aplicație GWT cu bibliotecă de *widget*-e.

Utilizarea bibliotecii *mywidgets* se declară în fișierul de configurare al aplicației *MyApp.gwt.xml* prin

```
<inherits name='mywidgets...Widgets' />
```

Caracterul ...corespunde lanțului din definiția pachetului, mai puțin *client*. Clasa unui *widget* client extinde clasa

```
com.google.gwt.user.client.ui.Composite.
```

Clasa *Composite* trebuie privită ca o clasă acoperitoare unui obiect de tip *Widget*. Elementele care formează interfața grafică a *widget*-ului client se includ într-un *container* reprezentabil grafic (de exemplu *VerticalPanel*). Acest *container* este încărcat de aplicația GWT care utilizează un *widget* client. În codul *widget*-ului client acest lucru este exprimat prin **protected** `void initWidget(Widget container)`. Această metodă poate fi apelată o singură dată.

Șablonul codului unui *widget* client este

```
package *.client;
import com.google.gwt.user.client.ui.Composite;
. . .
public class ClasaWidget extends Composite {
    public ClasaWidget() {
        . . .
        initWidget(container);
    }
    . . .
}
```

Exemplul 3.4.1 *Se crează o bibliotecă alcătuită din două clase widget:*

- *HelloNameWidget* - generează un cadru cu un câmp pentru nume și un buton. După clic pe buton apare mesajul "Hello nume".
- *CmmdcWidget* – aplicația anterioară transformată în widget.

Cele două *widget*-e au codurile, respectiv

```
1 package mywidgets.client;
2 import com.google.gwt.user.client.ui.*;
3 import com.google.gwt.event.dom.client.*;
4
5 public class HelloNameWidget extends Composite{
6     public HelloNameWidget() {
```

```

7   Label title=new Label(" Hello Name Widget");
8   title.addStyleName("label-title");
9   Button button = new Button(" Apasa-ma");
10  Label nameLabel=new Label(" Introduceti numele");
11  final Label sLabel=new Label();
12  final TextBox nameTextBox=new TextBox();

14  button.addStyleName("button");
15  VerticalPanel vPanel = new VerticalPanel();
16  vPanel.addStyleName("vpanel");
17  vPanel.add( title );
18  vPanel.setWidth("100%");
19  vPanel.setHorizontalAlignment( VerticalPanel.ALIGN_CENTER);
20  vPanel.add( nameLabel);
21  vPanel.add( nameTextBox);
22  vPanel.add( button);
23  vPanel.add( sLabel);
24  button.addClickHandler(new ClickHandler(){
25      public void onClick(ClickEvent event){
26          String name=nameTextBox.getText();
27          sLabel.setText(" Hello "+name+" !");
28      }
29  });
30  initWidget( vPanel);
31 }
32 }

```

și

```

1  package mywidgets.client;
2  import com.google.gwt.user.client.ui.*;
3  import com.google.gwt.user.client.Window;
4  import com.google.gwt.event.dom.client.*;

6  public class CmmdcWidget extends Composite {
7      public CmmdcWidget() {
8          Label title=new Label("Cmmdc Widget");
9          title.addStyleName("label-title");
10         Label mLabel=new Label("m=");
11         Label nLabel=new Label("n=");
12         Label cmmdcLabel=new Label();
13         TextBox mTextBox=new TextBox();
14         TextBox nTextBox=new TextBox();
15         Button button = new Button(" Calculeaza");
16         button.addStyleName("button");

18         VerticalPanel cmmdcPanel = new VerticalPanel();
19         cmmdcPanel.setWidth("100%");
20         cmmdcPanel.addStyleName("vpanel");
21         cmmdcPanel.setHorizontalAlignment( VerticalPanel.ALIGN_CENTER);
22         cmmdcPanel.add( title );
23         cmmdcPanel.add( mLabel);
24         cmmdcPanel.add( mTextBox);
25         cmmdcPanel.add( nLabel);
26         cmmdcPanel.add( nTextBox);
27         CmmdcWidgetClickHandler clickHandler=
28             new CmmdcWidgetClickHandler( mTextBox, nTextBox, cmmdcLabel);
29         button.addClickHandler( clickHandler);
30         cmmdcPanel.add( button);
31         cmmdcPanel.add( cmmdcLabel);

```

```

33     initWidget(cmmdcPanel);
34 }
35 }
37 class CmmdcWidgetClickHandler implements ClickHandler { . . . }

```

Codul aplicației GWT care folosește *widget*-ele client definite anterior este

```

1 package unitbv.cs.td.client;
2 import com.google.gwt.core.client.EntryPoint;
3 import com.google.gwt.user.client.ui.*;
4 import mywidgets.client.*;
5
6 public class MyApp implements EntryPoint {
7
8     public void onModuleLoad() {
9         HorizontalPanel hPanel=new HorizontalPanel();
10        hPanel.setSpacing(50);
11        CmmdcWidget cw=new CmmdcWidget();
12        HelloNameWidget hnw=new HelloNameWidget();
13        hPanel.add(hnw);
14        hPanel.add(cw);
15        RootPanel.get("ginterface").add(hPanel);
16    }
17 }

```

3.5 Încărcarea unui fișier - GWT Upload

GWT oferă o soluție prefabricată problemei încărcării unui fișier al clientului pe calculatorul server.

Aplicația GWT corespunde clientului care alege fișierul de încărcat și transmite datele unui server receptor, reprezentat, în cazul exemplului dezvoltat, de un servlet. Clasele *widget* ajutătoare sunt `FormPanel` și `FileUpload`. Într-un container `FormPanel` pot fi incluse *widgete* de tip `TextBox`, `PasswordTextBox`, `RadioButton`, `CheckBox`, `TextArea`, `FileUpload`, `Hidden`.

Dintre metodele care intervin la încărcarea unui fișier amintim:

- `public void setAction(String url)`
- `public void setMethod(FormPanel.METHOD_POST)`
În alt context se poate folosi `FormPanel.METHOD_GET`.
- `public void setEncoding(FormPanel.ENCODING_MULTIPART)`
În alt context se poate folosi `FormPanel.ENCODING_URLENCODED`.
- `public void submit()`

- `HandlerRegistration addSubmitHandler(FormPanel.SubmitHandler handler)`

Interfața `SubmitHandler`

- `HandlerRegistration addSubmitCompleteHandler(FormPanel.SubmitCompleteHandler handler)`

Fiecare din interfețele `SubmitHandler` și `SubmitCompleteHandler` declară metoda `void onSubmit(FormPanel.SubmitEvent event)` în care se programează activitățile înaintea expedierii formularului și respectiv, după recepția răspunsului furnizat de servlet.

Widgetul UploadFile afișează o fereastră de dialog prin care se selectează fișierul ce urmează a fi încărcat.

Codul clasei client este

```

1 package unitbv.cs.td.client;

3 import com.google.gwt.core.client.*;
4 import com.google.gwt.user.client.Window;
5 import com.google.gwt.user.client.ui.*;
6 import com.google.gwt.event.dom.client.ClickEvent;
7 import com.google.gwt.event.dom.client.ClickHandler;

9 public class Upload implements EntryPoint{
10     public void onModuleLoad(){
11         final FormPanel form = new FormPanel();
12         form.setAction("http://localhost:8080/upload/upload");
13         form.setEncoding(FormPanel.ENCODING_MULTIPART);
14         form.setMethod(FormPanel.METHOD_POST);

16         Label title=new Label("File upload");
17         title.addStyleName("label-title");

19         VerticalPanel panel = new VerticalPanel();
20         form.setWidget(panel);

22         panel.add(title);

24         // Crearea unui widget FileUpload
25         FileUpload upload = new FileUpload();
26         upload.setName("uploadFormElement");
27         panel.add(upload);

29         // Adaugarea unui buton "submit"
30         panel.add(new Button("Submit", new ClickHandler() {
31             public void onClick(ClickEvent event) {
32                 form.submit();
33             }
34         }));

36         // Activitati premergatoare expedierii formularului.
37         form.addSubmitHandler(new FormPanel.SubmitHandler() {
38             public void onSubmit(FormPanel.SubmitEvent event) {
39

```



```

42         File file=new File(pathTomcat+fs+"webapps"+
43             contextPath+fs+"upload"+fs+fileName);
44         byte[] b=new byte[1024];
45         FileOutputStream fos=new FileOutputStream(file);
46         int s=0;
47         do{
48             s=in.read(b,0,1024);
49             if(s!=-1)
50                 fos.write(b,0,s);
51         }
52         while(s!=-1);
53         fos.close();
54         in.close();
55         out.println("The file "+fileName+" has been uploaded !");
56         out.close();
57     }
58 }
59 }
60 catch(Exception e){
61     System.out.println("Exception: "+e.getMessage());
62 }
63 }
64 }

```

Pentru utilizarea aplicației client în modul Web, după compilare, conținutul catalogului *war* se copiază în catalogul `... \webapps` din *apache-tomcat*. Dintr-un navigator, clientul se apelează prin `http://host:8080/GwtUpload/GwtUpload.html`.

3.6 GWT prin *Google AppEngine*

Integrarea unei aplicații client GWT în platforma *Google AppEngine* (GAE) de Cloud Computing, necesită executarea următoarelor operații:

1. Se construiește aplicația GWT;
2. Se compilează aplicația GWT în vederea utilizării în modul Web;
3. Catalogul *war* se completează cu fișierul *appengine-web.xml* plasându-l în catalogul *WEB-INF*;
4. Din catalogul aplicației se lansează simulatorul GAE.

Exemplul 3.6.1 *Integrarea aplicației GWT de calcul a celui mai mare divizor comun în platforma Google de Cloud Computing.*

În urma operațiilor de mai sus rezultă structura

```
gwtcmmdc
|   . . .
|--> war
|   |--> cmmdc
|   |   | . . .
|   |--> WEB-INF
|   |   |--> classes
|   |   |   | . . .
|   |   |--> lib
|   |   |   | . . .
|   |   |   | appengine-web.xml
|   |   |   | web.xml
|   |   Cmmdc.html
|   |   Cmmdc.css
```

După lansarea simulatorului GAE, aplicația se apelează dintr-un navigator prin
`http://localhost:8080`.

Întrebări recapitulative

1. Ce funcționalități asigură interfața de programare AJAX?
2. Enumerați tipurile de aplicații GWT amintite.
3. Enumerați tipurile de aplicații apel de procedură la distanță GWT amintite.
4. Care este modalitatea utilizată pentru dezvoltarea unei aplicații GWT?

Partea III

SERVICII WEB

Capitolul 4

Servicii prin apel de procedură la distanță

Un serviciu Web este o aplicație client-server cu serverul găzduit de un server Web, apelabil prin aplicația client și realizat potrivit unei interfețe de programare specifice. Protocolul de comunicație este **http**.

Sunt cunoscute următoarele tipuri de servicii Web:

- Servicii bazate pe modelul Remote Procedure Call (RPC) - Apel de Procedură de la Distanță.

Protocolul de reprezentare a cererii și a răspunsului (de serializare / deserializare) variază. Din acest punct de vedere sunt cunoscute:

- Servicii *xml-rpc* (www.xmlrpc.org). Cererea și răspunsul sunt transmise prin cod **xml** cuprins în corpul mesajului **http**.
- Servicii *json-rpc* (www.json-rpc.org) bazat pe reprezentarea JSON.
- Servicii *hessian*. Se utilizează un protocol pentru serializare / deserializare bazat pe reprezentarea binară a datelor. Protocolul a fost dezvoltat de firma *Caucho Technologies* (2007).
- Servicii bazate pe interfața de programare *Java API for XML Web Services - JAX-WS*. Interfața de programare JAX-WS este varianta cea mai recentă pentru serviciile cunoscute sub numele de servicii *soap-rpc*.

Pentru fiecare caz semnalat mai sus sunt realizate implementări în mai multe limbaje / platforme de programare.

- Servicii REST.

REpresentational State Transfer (REST) este un model de arhitectură de aplicație distribuită¹.

REST specifică modul cum o resursă - entitate care conține informație specifică - este definită și cum poate fi adresată.

Identificarea unei resurse se face printr-un URI (*Universal Resource Identifier*).

Interfața standard de programare a unui serviciu REST este *Java API for XML Restful Services - JAX-RS*.

Transferul resursei către un client și prelucrarea resursei se face prin operații indicate de antetele mesajului `http GET`, `POST`, `PUT`, `DELETE`, etc.

4.1 Descrierea unui serviciu JAX-WS

Spre deosebire de aplicațiile bazate pe apelul de procedură la distanță (RMI, CORBA,) unde prezentarea ofertei se face printr-o interfața Java, într-un serviciu JAX-WS, JAX-RS descrierea sau specificarea acestuia se realizează prin sublimbaje `xml`:

- *Web Service Description Language - WSDL* pentru servicii JAX-WS.

Descrierea datelor din mesajele vehiculate se face prin *XML Schema*, de asemenea un sublimbaj `xml`.

- *Web Application Description Language - WADL* pentru servicii JAX-RS.

Mesajele dintre client și server folosesc protocolul de reprezentare *Simple Object Access Protocol - SOAP* care este independent de platforma de calcul și de limbajul de programare. SOAP este tot un sublimbaj `xml`, standard *World Wide Web Consortium - W3C*. În instrumentele actuale de dezvoltare, SOAP este transparent programatorului, dar Java ofera suport de programare prin pachetul `javax.xml.soap`.

Produsul *soapUI* oferă posibilitatea vizualizării mesajelor SOAP ale serviciilor Web pornind de la descrierea acestora prin WSDL, respectiv WADL.

Considerăm familiarizarea cu XML-Schema și WSDL importantă.

¹REST a fost introdus de Roy Fielding, în teza sa de doctorat din 2000. Roy Fielding este autorul principal al specificațiilor protocolului `http`.

4.1.1 XML Schema

O schemă este un model pentru descrierea structurii informației. În contextul `xml`, o schemă descrie un model pentru o familie de documente `xml`.

Primul obiectiv al unei scheme este de a permite validarea automată a structurii unui document `xml`.

Familiarizarea cu elementele de conținut și sintaza *XML Schema* se va începe cu un exemplu:

Documentul `struct.xml`

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <mk:structura xmlns:mk="http://www.distr2.edu"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.distr2.edu struct-schema.xsd">
5
6   <mk:disciplina fel="obligatoriu">
7     <mk:nume> Analiza numerica </mk:nume>
8     <mk:fond-de-timp>
9       <mk:curs>2</mk:curs>
10      <mk:seminar>1</mk:seminar>
11      <mk:laborator>1</mk:laborator>
12    </mk:fond-de-timp>
13  </mk:disciplina>
14  <mk:disciplina fel="obligatoriu">
15    <mk:nume> Programare distribuita </mk:nume>
16    <mk:fond-de-timp>
17      <mk:curs>2</mk:curs>
18      <mk:seminar>0</mk:seminar>
19      <mk:laborator>2</mk:laborator>
20    </mk:fond-de-timp>
21  </mk:disciplina>
22  <mk:disciplina fel="obligatoriu">
23    <mk:nume> Soft matematic </mk:nume>
24    <mk:fond-de-timp>
25      <mk:curs>2</mk:curs>
26      <mk:seminar>0</mk:seminar>
27      <mk:laborator>1</mk:laborator>
28    </mk:fond-de-timp>
29  </mk:disciplina>
30 </mk:structura>

```

de exemplu, are schema `struct-schema.xsd`

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3   xmlns:mk="http://www.distr2.edu"
4   targetNamespace="http://www.distr2.edu">
5
6   <xsd:element name="structura">
7     <xsd:complexType>
8       <xsd:choice minOccurs="0" maxOccurs="unbounded">
9         <xsd:element ref="mk:disciplina"/>
10      </xsd:choice>
11    </xsd:complexType>
12  </xsd:element>

```

```

14 <xsd:element name="disciplina">
15   <xsd:complexType mixed="true">
16     <xsd:sequence>
17       <xsd:element ref="mk:nume" />
18       <xsd:element ref="mk:fond-de-timp" />
19     </xsd:sequence>
20     <xsd:attribute name="fel" type="xsd:string" use="required" />
21   </xsd:complexType>
22 </xsd:element>

24 <xsd:element name="nume" type="xsd:string" />

26 <xsd:element name="fond-de-timp">
27   <xsd:complexType mixed="true">
28     <xsd:sequence>
29       <xsd:element ref="mk:curs" />
30       <xsd:element ref="mk:seminar" />
31       <xsd:element ref="mk:laborator" />
32     </xsd:sequence>
33   </xsd:complexType>
34 </xsd:element>

36 <xsd:element name="curs" type="xsd:string" />

38 <xsd:element name="seminar" type="xsd:string" />

40 <xsd:element name="laborator" type="xsd:string" />
41 </xsd:schema>

```

Alte marcaje utilizabile sunt:

- `<xsd:simpleType name=". . ." >`
`<xsd:restriction base="xsd:string" >`
`<xsd:maxLength value="n" />`
`<xsd:pattern value="[0-9]" />`
`</xsd:restriction>`
`</xsd:simpleType>`
- `<xsd:complexType name="aaa" >`
`<xsd:sequence>`
`<xsd:element name="bbb" type="ccc"`
`minOccurs="0" maxOccurs="unbounded" />`
`</xsd:sequence>`
`</xsd:complexType>`

Tipuri simple: string, byte, integer, decimal, boolean, time, date, etc.

Grupuri. Se pot defini grupuri de elemente și de atribute

```

<xsd:group name="tipElementDisciplina" >
  <xsd:sequence>
    <xsd:element name="nume" type="tipNume" />
    <xsd:element name="fond-de-timp" type="tipFondTimp" />
  </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="atribDisciplina" >
  <xsd:attribute name="fel" type="xsd:string" use="required" />
  <xsd:attribute name="finalizare" type="xsd:string" />

```

```

</xsd:attributeGroup>

<xsd:complexType name="tipdisciplina" >
  <xsd:sequence>
    <xsd:group ref="tipElementDisciplina" />
    <xsd:attributeGroup ref="attribDisciplina" />
  </xsd:sequence>
</xsd:complexType>

```

Comentariile

- destinate a fi citite de oameni se introduc în marcajul `xsd:documentation`;
- destinate a fi procesate se introduc în marcajul `xsd:appinfo`.

Ambele marcaje trebuie înglobate în `xsd:annotation`.

```

<xsd:annotation>
  <xsd:documentation xml:lang="en">
    text
  </xsd:documentation>
  <xsd:appinfo source=". . ." >
    <bind xmlns=". . ." >
      <class name=". . ." />
    </bind>
  </xsd:appinfo>
</xsd:annotation>

```

Schemele se pot compune prin

```
<xsd:include schemaLocation="fisier.xsd" />
```

Pentru a evita conflictul între numele atribuite diverselor elemente se introduce:

- Spațiul de nume - namespace - definit ca un șir de caractere sub forma unui URI (Universal Resource Identifier);
- Numele calificat - *QName* qualified name - alcătuit dintr-un nume local asociat cu namespace.

Procesarea documentelor `xml` face apel la *numele calificat* al unei entități. Un spațiu de nume se definește prin

```

<xsd:schema targetNamespace="numeleSpatiuluiDeNume"
  xmlns:prefix="numeleSpatiuluiDeNume"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

```

Prin folosirea prefixului se specifică spațiul de nume al fiecărei entități utilizate.

4.1.2 WSDL

WSDL (*Web Service Description Language*) este un limbaj xml pentru descrierea serviciilor Web.

Presupunem existența următoarelor entități

```
|| Server Web la adresa http://localhost:8080
||     Serviciu Web /CmmdcWS
||         Operatie
||             Nume: cmmdc
||         Operatie
||             Nume: . . .
```

Serviciul se va apela prin *http://localhost:8080/CmmdcWS*. Această referință se numește *punct final - endpoint*.

Pentru a evita conflictul între numele atribuite diverselor elemente se utilizează denumiri calificate. iar structura de mai sus devine

```
|| Server Web la adresa http://localhost:8080
||     Serviciu Web /CmmdcWS
||         Operatie
||             Nume local: cmmdc
||             Namespace: http://cs.unitbv.ro/ws
||         Operatie
||             Nume local: . . .
```

Pentru acest exemplu, operația este definită de o metodă având doi parametri și returnează un rezultat:

```
|| Operatie
||     Nume local: cmmdc
||     Namespace: http://cs.unitbv.ro/ws
||     Parametrii:
||         m: string
||         n: string
||     Returneaza:
||         string
```

Tipurile (string, int, long, etc) utilizate sunt precizate în spațiul de nume <http://www.w3.org/2001/XMLSchema>.

Definirea operației devine

```
|| Operatie
||     Nume local: cmmdc
||     Namespace: http://cs.unitbv.ro/ws
||     Parametrii:
||         m: string din http://www.w3.org/2001/XMLSchema
||         n: string din http://www.w3.org/2001/XMLSchema
||     Returneaza:
||         string din http://www.w3.org/2001/XMLSchema
```

În terminologia serviciilor Web RPC se utilizează

- mesaj de intrare - input message - pentru datele de apelare a unei metode;

- parte - part - pentru un parametru al mesajului de intrare;
- mesaj de ieșire - output message - pentru datele returnate de operație.

Asfel vom avea

```

|| Operatie
||   Nume local: cmmdc
||   Namespace: http://cs.unitbv.ro/ws
||   Input message:
||     Part 1:
||       Name: m
||       Type: string din http://www.w3.org/2001/XMLSchema
||     Part 2:
||       Name: n
||       Type: string din http://www.w3.org/2001/XMLSchema
||   Output message:
||     Part:
||       Name: return
||       Type: string din http://www.w3.org/2001/XMLSchema

```

Codul unui mesaj de intrare poate fi

```

<pre:cmmdc xmlns:pre="http://cs.unitbv.ro/ws">
  <m>56</m>
  <n>48</n>
</pre:cmmdc>

```

iar mesajul de ieșire poate fi

```

<pre:cmmdc xmlns:pre="http://cs.unitbv.ro/ws">
  <return>8</return>
</pre:cmmdc>

```

XMLSchema permite definirea de tipuri complexe:

```

<xsd:schema targetNamespace="http://cs.unitbv.ro/ws"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cmmdcRequest">
    <xsd:complexType>
      <xsd:element name="m" type="xsd:string"/>
      <xsd:element name="n" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Definirea operației va constă din

1. schema

```

<xsd:schema targetNamespace="http://cs.unitbv.ro/ws"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cmmdcRequest">
    <xsd:complexType>
      <xsd:element name="m" type="xsd:string"/>
      <xsd:element name="n" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

2. operația propriu-zisă

```

|| Operatie
||   Nume local: cmmdc
||   Namespace:  http://cs.unitbv.ro/ws
||   Input message:
||     Part 1:
||       Name: cerereCmmdc
||       Type: cmmdcRequest din http://cs.unitbv.ro/ws
||   Output message:
||     Part 1:
||       Name: return
||       Type: string din http://www.w3.org/2001/XMLSchema

```

iar mesajul de intrare va fi

```

<pre:cmmdcRequest xmlns:pre="http://cs.unitbv.ro/ws">
  <m>56</m>
  <n>48</n>
</pre:cmmdcRequest>

```

Extindem incluzând și răspunsul oferit de operație

1. schema

```

<xsd:schema targetNamespace="http://cs.unitbv.ro/ws"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cmmdcRequest">
    <xsd:complexType>
      <xsd:element name="m" type="xsd:string"/>
      <xsd:element name="n" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="responseCmmdc" type="xsd:string"/>
</xsd:schema>

```

2. operația propriu-zisă

```

|| Operatie
||   Nume local: cmmdc
||   Namespace:  http://cs.unitbv.ro/ws
||   Input message:
||     Part 1:
||       Name: cerereCmmdc
||       Type: cmmdcRequest din http://cs.unitbv.ro/ws
||   Output message:
||     Part 1:
||       Name: raspunsCmmdc
||       Type: responseCmmdc din http://cs.unitbv.ro/ws

```

Răspunsul va fi

```

<pre:cmmdcResponse xmlns:pre="http://cs.unitbv.ro/ws">
  8
</pre:cmmdcResponse>

```

Acest mod de definire a unui serviciu Web poartă numele de *stilul document*². Acest stil este impus de WS-I (Web Services Interoperability organization).

Determinarea operației apelate se face exclusiv pe baza tipului din mesajul de intrare. Nu pot exista două operații cu datele de intrare definite prin același nume calificat.

Asamblând, se obține

```

|| Server Web la adresa http://localhost:8080
||   Schema
||       . . .
||   Operatie
||       Nume local: cmmdc
||       Namespace: http://cs.unitbv.ro/ws
||       Input message:
||           Part 1:
||               Name: cerereCmmdc
||               Type: cmmdcRequest din http://cs.unitbv.ro/ws
||       Output message:
||           Part 1:
||               Name: raspunsCmmdc
||               Type: responseCmmdc din http://cs.unitbv.ro/ws

```

PortType. Operațiile sunt grupate în colecții numite *PortType*. Un PortType este denumită printr-un nume calificat (nume local și namespace).

```

|| Server Web la adresa http://localhost:8080
||   Schema
||       . . .
||   PortType
||       Nume local: functiiWS
||       Namespace: http://cs.unitbv.ro/ws
||
||       Operatie
||           Nume local: cmmdc
||           Namespace: http://cs.unitbv.ro/ws
||           Input message:
||               Part 1:
||                   Name: cerereCmmdc
||                   Type: cmmdcRequest din http://cs.unitbv.ro/ws
||           Output message:
||               Part 1:
||                   Name: raspunsCmmdc
||                   Type: responseCmmdc din http://cs.unitbv.ro/ws
||
||       Operatie
||           . . .
||
||   PortType
||       . . .

```

² Un alt mod de definire, nerecomandat în prezent, este *stilul RPC*.

Binding. Prin *binding* se specifică protocolul utilizat pentru prelucrarea și transmiterea mesajelor. Cel mai utilizat protocol pentru prelucrarea mesajelor - aproape complet transparent programatorului - este *Simple Object Access Protocol* SOAP, iar pentru transportul mesajelor este *Hyper Text Transport Protocol* HTTP.

```
|| Name      : binding1
|| Port type : functiiWS
|| Format    : SOAP
|| Transport : HTTP
```

Port. Serviciul se poate instala / desfășura pe mai multe calculatoare. Fiecare asemenea calculator devine un port. Implementarea propriu-zisă a operațiilor poate fi diferită - chiar în limbaje de programare diferite.

Astfel, obținem următoarea descriere a serviciului

```
|| Serviciu Web
|| Schema
||     . . .
||
|| PortType
||     Nume local: functiiWS
||     Namespace: http://cs.unitbv.ro/ws
||
||     Operatie
||         . . .
||
|| PortType
||     . . .
||
|| Binding
||     Name      : binding1
||     Port type : functiiWS
||     Format     : SOAP
||     Transport : HTTP
||     . . .
||
|| Port
||     Name      : port1
||     Binding   : binding1
||     Endpoint:
||     . . .
```

Toate noțiunile introduse pentru descrierea serviciului Web vor face parte dintr-un același spațiu de nume *targetNamespace*

```
|| Serviciu Web
|| Target namespace: http://cs.unitbv.ro/ws
|| Schema
||     . . .
||
|| PortType
```

```

||      Nume local: functiiWS
||      Namespace: http://cs.unitbv.ro/ws
||
||      Operatie
||      . . .
||
||      PortType
||      . . .
||
||      Binding
||      Name      : binding1
||      Port type : functiiWS
||      Format     : SOAP
||      Transpoat : HTTP
||      . . .
||
||      Port
||      Name      : port1
||      Binding   : binding1
||      Endpoint:
||      . . .

```

Aceasta descriere a unui serviciu Web corespunde standardului *WSDL*.
Structura unui document *wsdl* cuprinde

```

<definitions>
  <types>
    Definirea tipurilor de date utilizate
  </types>
  <message>
    Definirea mesajelor utilizate. Un mesaj corespunde
    parametrilor sau rezultatelor functiilor ce compun
    serviciul.
  </message>
  <portType>
    Declara functiile serviciului.
    Un port defineste un punct de conexiune cu serviciul Web.
  </portType>
  <binding>
    Declara protocoalele utilizate de serviciul web.
  </binding>
</definitions>

```

Pentru un serviciu Web, documentul *wsdl* generat depinde de produsul informatic care sustine serviciul Web. Funcție de acest produs, sunt introduse în documentul *wsdl* și alte elemente.

Exemplul 4.1.1 *Fișierul wsdl pentru serviciul Web de calcul a celui mai mare divizor comun a două numere naturale generat de Metro.*

```

1 <!--
2   Published by JAX-WS RI (http://jax-ws.java.net).
3   RI's version is Metro/2.3.1 (UNKNOWN_BRANCH=false;
4   2015-01-15T16:53:43+0100) JAXWS-RI/2.2.10
5   JAXWS-API/2.2.11 JAXB-RI/2.2.10-b140802.1033
6   JAXB-API/2.2.12-b140109.1041 svn-revision#unknown.
7 -->
8 <!--
9   Generated by JAX-WS RI (http://jax-ws.java.net).
10  RI's version is Metro/2.3.1 (UNKNOWN_BRANCH=false;
11  2015-01-15T16:53:43+0100) JAXWS-RI/2.2.10
12  JAXWS-API/2.2.11 JAXB-RI/2.2.10-b140802.1033
13  JAXB-API/2.2.12-b140109.1041 svn-revision#unknown.
14 -->
15 <definitions xmlns:wsu="http://docs.oasis-open.org/wss/
16   2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
17   xmlns:wsp="http://www.w3.org/ns/ws-policy"
18   xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
19   xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
20   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
21   xmlns:tns="http://server.cmmdc/"
22   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
23   xmlns="http://schemas.xmlsoap.org/wsdl/"
24   targetNamespace="http://server.cmmdc/"
25   name="CmmdcWSService">
26   <types>
27     <xsd:schema>
28       <xsd:import namespace="http://server.cmmdc/"
29         schemaLocation=
30           "http://localhost:8080/jaxws-cmmdc/cmmdcws?xsd=1"/>
31     </xsd:schema>
32   </types>
33   <message name="cmmdc">
34     <part name="parameters" element="tns:cmmdc"/>
35   </message>
36   <message name="cmmdcResponse">
37     <part name="parameters" element="tns:cmmdcResponse"/>
38   </message>
39   <portType name="CmmdcWS">
40     <operation name="cmmdc">
41       <input wsam:Action=
42         "http://server.cmmdc/CmmdcWS/cmmdcRequest"
43         message="tns:cmmdc"/>
44       <output wsam:Action=
45         "http://server.cmmdc/CmmdcWS/cmmdcResponse"
46         message="tns:cmmdcResponse"/>
47     </operation>
48   </portType>
49   <binding name="CmmdcWSPortBinding" type="tns:CmmdcWS">
50     <soap:binding
51       transport="http://schemas.xmlsoap.org/soap/http"
52       style="document"/>
53     <operation name="cmmdc">
54       <soap:operation soapAction="" />
55     <input>

```

```
56      <soap:body use="literal" />
57    </input>
58    <output>
59      <soap:body use="literal" />
60    </output>
61  </operation>
62 </binding>
63 <service name="CmmdcWSService">
64   <port name="CmmdcWSPort"
65     binding="tns:CmmdcWSPortBinding">
66     <soap:address
67       location="http://localhost:8080/jaxws-cmmdc/cmmdcws" />
68   </port>
69 </service>
70 </definitions>
```

4.1.3 Mesaje SOAP

Un mesaj SOAP este un document XML constând din

- o învelitoare (*envelope*) care poate conține
- un număr arbitrar de antete (*header*);
- un corp (*body*);
- un număr variabil de obiecte atașate (*attachments*) MIME (Multipurpose Internet Mail Exchange).

Astfel un mesaj SOAP apare sub forma documentului XML

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

Facilitățile Java de manipulare a mesajelor SOAP sunt conținute în modulul `java.xml.ws`, pachetul `javax.xml.soap`, din distribuția JDK.

Crearea unui mesaj SOAP

```
MessageFactory mf=MessageFactory.newInstance();  
SOAPMessage soapMsg=mf.createMessage();
```

Mesajul creat are definită structura de bază a mesajului SOAP: înveliștoarea, un antet și corp. Aceste elemente pot fi accesate prin

```
SOAPPart part=soapMsg.getSOAPPart();  
SOAPEnvelope envelope=part.getEnvelope();  
SOAPHeader header=envelope.getHeader();  
SOAPBody body=envelope.getBody();
```

Completarea corpului unui mesaj SOAP

Oricărui element îi este asociat un obiect `javax.xml.soap.Name`. Din acest obiect se pot afla

- `String getLocalName()`
- `String getPrefix()`
- `String getURI()`
- `String getQualifiedName()`

Numele calificat are structura

```
<prefix:NumeleLocal xmlns:prefix="uri">
```

Elementul cu numele local "e1" se generează prin

```
Name n1=envelope.createName("e1");
```

În *body* un element se poate include, pe baza numelui creat prin

```
SOAPElement e1=body.addBodyElement(n1);
```

În general, un element se include în elementul părinte, care poate fi chiar și *body*, prin metoda clasei `SOAPElement`

```
SOAPElement addChildElement(Name name)
```

Completăm elementul *e1* cu un *text*: "primul", prin

```
e1.addTextNode("primul");
```


Exemplul 4.1.2 *Mesajul SOAP*

```

1 <SOAP-ENV:Envelope
2   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3 <SOAP-ENV:Header/>
4 <SOAP-ENV:Body>
5   <e1>
6     primul
7     <e11>
8       al treilea
9     </e11>
10  </e1>
11  <e2>
12    al doilea
13  </e2>
14 </SOAP-ENV:Body>
15 </SOAP-ENV:Envelope>

```

se obține cu programul

```

1 import javax.xml.soap.MessageFactory;
2 import javax.xml.soap.SOAPMessage;
3 import javax.xml.soap.SOAPPart;
4 import javax.xml.soap.SOAPEnvelope;
5 import javax.xml.soap.SOAPBody;
6 import javax.xml.soap.SOAPElement;
7 import javax.xml.soap.Name;
8 import java.io.FileOutputStream;
9
10 public class MsgSOAPSender{
11   public static void main(String [] args){
12     Name name=null;
13     try{
14       MessageFactory mf=MessageFactory.newInstance();
15       SOAPMessage soapMsg=mf.createMessage();
16       SOAPPart part=soapMsg.getSOAPPart();
17       SOAPEnvelope envelope=part.getEnvelope();
18       SOAPBody body=envelope.getBody();
19       Name n1=envelope.createName("e1");
20       SOAPElement e1=body.addBodyElement(n1);
21       e1.addTextNode("primul");
22       Name n2=envelope.createName("e2");
23       SOAPElement e2=body.addBodyElement(n2);
24       e2.addTextNode("al doilea");
25       Name n11 =envelope.createName("e11");
26       SOAPElement e11=e1.addChildElement(n11);
27       e11.addTextNode("al treilea");
28       FileOutputStream f=new FileOutputStream("MySOAPMessage.xml");
29       soapMsg.writeTo(f);
30     }
31     catch(Exception e){
32       System.out.println("Exception : "+e.getMessage());
33     }
34   }
35 }

```

Un mesaj SOAP se poate salva într-un fișier text cu

```
FileOutputStream f=new FileOutputStream(. . .);
soapMsg.writeTo(f);
```

Conținutul fișierului este

```
SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/><SOAP-ENV:Body><e1>primul<e11>al treilea</e11></e1>
<e2>al doilea</e2></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Preluarea elementelor din corpul unui mesaj SOAP

```
SOAPBody body=. . .
SOAPBodyElement element=null;
Iterator iterator=body.getChildElements();
while(iterator.hasNext()){
    element=(SOAPBodyElement)iterator.next();
    String name=element.getElementName();
    if(name.getLocalName().equals("numeCamp")){
        String s=element.getValue();
        . . .
    }
}
```

```
1 import javax.xml.soap.MessageFactory;
2 import javax.xml.soap.SOAPBody;
3 import javax.xml.soap.SOAPBodyElement;
4 import javax.xml.soap.SOAPMessage;
5 import javax.xml.soap.SOAPEnvelope;
6 import javax.xml.soap.SOAPPart;
7 import javax.xml.soap.SOAPElement;
8 import javax.xml.soap.Name;
9 import java.util.Iterator;
10 import java.io.FileInputStream;
11 import org.w3c.dom.Node;
12 import org.w3c.dom.Text;

14 public class MsgSOAPReceiver{
15     public static void analyze(SOAPElement rootElement){
16         Iterator iterator=rootElement.getChildElements();
17         while(iterator.hasNext()){
18             Object obj=iterator.next();
19             if(!(obj instanceof Text)){
20                 SOAPElement element = (SOAPElement)obj;
21                 short nodeType=element.getNodeType();
22                 System.out.println(nodeType);
23                 Name name=element.getElementName();
24                 System.out.println("name : " + name.getLocalName());
25                 System.out.println("value : " + element.getValue());
26                 analyze(element);
27             }
28         }
29     }
30 }
```

```
29 }
31 public static void main(String[] args) {
32     try {
33         FileInputStream fis=new FileInputStream("MySOAPMessage.xml");
34         MessageFactory mf = MessageFactory.newInstance();
35         SOAPMessage soapMsg = mf.createMessage(null, fis);
36         SOAPPart part=soapMsg.getSOAPPart();
37         SOAPEnvelope envelope=part.getEnvelope();
38         SOAPBody body=envelope.getBody();
39         analyze(body);
40     }
41     catch (Exception ex) {
42         ex.printStackTrace();
43     }
44 }
45 }
```

Observația 4.1.1

Produsul *Oracle-Open Message Queue* oferă posibilitatea transformării unui mesaj SOAP în mesaj JMS și invers.

Transformarea unui mesaj SOAP în mesaj JMS

```
SOAPMessage soapMsg=. . .
Message msg=
    MessageTransformer.SOAPMessageIntoJMSMessage(soapMsg,session);
```

Transformarea unui mesaj JMS în mesaj SOAP

```
MessageFactory mf=. . .
Message msg=. . .
SOAPMessage soapMsg =
    MessageTransformer.SOAPMessageFromJMSMessage(msg,mf);
```

4.2 Servicii JAX-WS

JSR (Java Specification Request) 109 definește o interfață de programare (API) pentru realizarea serviciilor Web bazate pe RPC : *Java API for XML Web Services (JAX-WS)*. Un asemenea serviciu Web se poate implementa prin:

- **servlet:**
Serviciul este implementat ca o clasă Java care rulează într-un container Web, fiind integrat într-un servlet. Integrarea este complet transparentă programatorului.
- **sesiune EJB (Enterprise Java Bean) fără stare (stateless session):**
Serviciul rulează într-un container EJB.

4.2.1 Serviciu Web ca servlet în *apache-tomcat* prin *ant*

Cadrul de lucru pe care îl vom utiliza poate fi implementarea de referință *jaxws-ri-** sau *Metro*, dezvoltate de *Oracle*.

Se oferă suport pentru dezvoltarea serviciului Web pe serverele:

- *apache-tomcat*;
- *jetty*;
- *glassfish*.

Dacă serviciul se desfășoară în serverul Web *apache-tomcat* sau *jetty* atunci fișierele *jar* aparținând suportul JAX-WS utilizat vor fi depuse în catalogul `WEB-INF\lib` al serviciului. Serviciul se arhivează cu extensia *war*.

Dezvoltarea aplicației server

Clasa serverului este o clasă POJO cu adnotări specifice. Vom dezvolta serviciul Web pentru calculul celui mai mare divizor comun a două numere naturale.

Definirea serviciului, a operațiilor pe care le oferă serviciul și a parametrilor de intrare pentru fiecare operație se face utilizând *adnotările* `@WebService`, respectiv `@WebMethod` și `@WebParam`.

Structura clasei server va fi

```
1 package cmmdc.server;
2 import javax.jws.WebMethod;
3 import javax.jws.WebParam;
4 import javax.jws.WebService;
```

```

6 @WebService()
7 public class CmmdcWS {
9     @WebMethod
10     public long cmmdc(@WebParam(name = "m") long m,
11                       @WebParam(name = "n") long n) { . . . }
12 }

```

Programatorul va completa codul metodei *cmmdc*.
Vom considera structura

```

myapp
|--> src
|   |--> mypackage
|   |   |--> server
|   |   |   *.java
|--> war
|   |--> WEB-INF
|   |   |--> classes
|   |   |--> lib
|   |   |   *.jar
|   |   |   web.xml
|   |   |   sun-jaxws.xml

```

Compilarea se face prin intermediul clasei `com.sun.tools.ws.ant.Apt`.
(apt - Annotation Proccessing Tool) aflat în resursele *jaxws-ri/metro*. Codul corespunzător din `build.xml` este

```

<taskdef name="apt" classname="com.sun.tools.ws.ant.Apt">
  <classpath refid="myclasspath"/>
</taskdef>

<target name="build-server" depends="init">
  <apt
    fork="true"
    debug="true"
    destdir="war/WEB-INF/classes"
    sourcedestdir="war/WEB-INF/classes"
    sourcepath="src">
    <classpath>
      <path refid="myclasspath"/>
    </classpath>
    <source dir="src">
      <include name="**/server/*.java"/>
    </source>
  </apt>
</target>

```

Pe baza claselor din `source=src\mypackage\server` se vor genera în `sourcedestdir` o serie de clase iar rezultatul compilării se depun în catalogul indicat de `destdir`.

În cazul exemplului considerat aceste clase sunt *cmmdc.server.jaxws.Cmmdc.java* și *cmmdc.server.jaxws.CmmdcResponse.java*.

Numele pachetului este alcătuit din numele pachetului clasei serviciului plus `.jaxws`. Numele metodei generează numele clasei corespunzătoare mesajului de intrare (*input message*) iar pentru numele clasei corespunzătoare mesajului de ieșire (*output message*) se adaugă sufixul **Response**.

Se utilizează două fișiere de configurare

- *sun-jaxws.xml*

```

1 ?xml version="1.0" encoding="UTF-8"?>
3 <endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'
4   version='2.0'>
5   <endpoint
6     name='jaxws-cmmdc'
7     implementation='cmmdc.server.CmmdcWS'
8     url-pattern='/cmmdcws' />
9 </endpoints>

```

- *web.xml*

```

1 ?xml version="1.0" encoding="UTF-8"?>
3 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
4   <listener>
5     <listener-class>
6       com.sun.xml.ws.transport.http.servlet.WSServletContextListener
7     </listener-class>
8   </listener>
9   <servlet>
10    <servlet-name>cmmdcws</servlet-name>
11    <servlet-class>
12      com.sun.xml.ws.transport.http.servlet.WSServlet
13    </servlet-class>
14    <load-on-startup>1</load-on-startup>
15  </servlet>
16  <servlet-mapping>
17    <servlet-name>cmmdcws</servlet-name>
18    <url-pattern>/cmmdcws</url-pattern>
19  </servlet-mapping>
20  <session-config>
21    <session-timeout>60</session-timeout>
22  </session-config>
23 </web-app>

```

Se rețin următoarele corelații ale denumirilor:

- *url-pattern* fixat în *web.xml* este redeclarat în fișierele *sun-jaxws.xml*.
- Numele serviciului declarat în *sun-jaxws.xml* trebuie să fie numele arhivei **war**.

Pentru exemplul nostru, *wsdl*-ul serviciului va fi disponibil la

http://host:8080/jaxws-cmmdc/cmmdcws?wsdl

Dezvoltarea aplicației client. Realizarea aplicației client presupune ca serviciul să fie activ pe serverul Web.

Dezvoltarea părții (aplicației) client începe cu generarea unor clase care mijlocesc apelarea serviciului. Generarea se face utilizând utilitarul `wsimport` din `jdk` pe baza accesării fișierului `wsdl` asociat serviciului.

Codul obiectivului din `build.xml` este

```
<target name="generate-client" depends="init">
  <exec executable="${env.JAVA_HOME}/bin/wsimport">
    <arg value="-d"/>
    <arg value="${build.dir}"/>
    <arg value="-keep"/>
    <arg value="-p"/>
    <arg value="${app.name}.client"/>
    <arg value="${wsdl.uri}"/>
  </exec>
</target>
```

Opțiunea `-d` specifică locația unde se depun fișierele generate, opțiunea `-p` indică pachetul din care fac parte clasele generate.

Dintre aceste clase, în codul clientului propriu-zis se folosește clasa *CmmdcWSService*. Numele clasei s-a obținut adăugând sufixul *Service* la numele clasei server. Această clasă conține metoda *getCmmdcWSPort()* ce returnează un reprezentant al serviciului pe calculatorul clientului.

Astfel referința la serviciu se obține prin

```
CmmdcWSService service=new CmmdcWSService();
CmmdcWS port=service.getCmmdcWSPort();
```

Prin variabila *port* putem apela orice operație a serviciului.
Codul clientului sincron este

```
1 package cmmdc.client;
2 import java.util.Scanner;

4 public class CmmdcClient {
5     public static void main(String[] args) {
6         try {
7             CmmdcWS port=new CmmdcWSService().getCmmdcWSPort();
8             Scanner scanner=new Scanner(System.in);
9             System.out.println("m=");
10            long m=scanner.nextLong();
11            System.out.println("n=");
12            long n=scanner.nextLong();
13            long result=port.cmmdc(m,n);
14            System.out.println("Cmmdc="+result);
```

```

15 |     }
16 |     catch (Exception e) {
17 |         System.out.printf("AnException : "+e.getMessage());
18 |     }
19 | }
20 | }

```

Client Web al serviciului Web

Clientul Web este reprezentat de pagina JSP (*index.jsp*)

```

1 <html>
2   <body>
3     <form method="post">
4       <table>
5         <tr>
6           <td>Primul numar este </td>
7           <td><input type="text" name="m" size=5 value="1"> </td>
8         </tr>
9         <tr>
10          <td>Al doilea numar este </td>
11          <td><input type="text" name="n" size=5 value="1"> </td>
12        </tr>
13        <tr>
14          <td><input type="submit" value=" Calculeaza "></td>
15        </tr>
16      </table>
17    </form>
18    <%
19    try {
20      cmmdc.client.CmmdcWSService service=
21        new cmmdc.client.CmmdcWSService();
22      cmmdc.client.CmmdcWS port=service.getCmmdcWSPort();
23      String sm = request.getParameter("m");
24      String sn = request.getParameter("n");
25      long m=((sm==null)||("".equals(sm)))?1:Long.parseLong(sm);
26      long n=((sn==null)||("".equals(sn)))?1:Long.parseLong(sn);
27      long rez=port.cmmdc(m,n);
28      String result=Long.valueOf(rez).toString();
29      out.println("Result = "+result);
30    }
31    catch (Exception e) {
32      out.println("Exception : "+e.getMessage());
33    }
34    %>
35    <hr/>
36  </body>
37 </html>

```

Structura aplicației Web care se desfășoară în serverul Web este

```

client-web
|-->WEB-INF
|   |--> classes
|   |   |   *.class
|   |--> lib

```



```
|      |      web.xml
|      |      index.jsp
```

Fișierele **class** sunt cele generate de **wsimport** la dezvoltarea aplicației. Fișierul *web.xml* este

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5     http://java.sun.com/xml/ns/j2ee/web-app-2.4.xsd">
6   <session-config>
7     <session-timeout>
8       30
9     </session-timeout>
10  </session-config>
11  <welcome-file-list>
12    <welcome-file>index.jsp</welcome-file>
13  </welcome-file-list>
14 </web-app>
```

Valoarea *jspclient* din *sun-web.xml* reprezintă numele de apel al aplicației client și coincide cu numele arhivei **war**.

Clienți *asincroni*

Metro oferă posibilitatea dezvoltării de clienți *asincroni*. Caracterul asincron se referă la faptul că programul client nu se blochează în așteptarea răspunsului oferit de serviciul Web.

De această dată utilitarul **wsimport** se folosește indirect, mai precis prin intermediul clasei **com.sun.tools.ws.ant.WsImport** din *Metro*.

Generarea metodelor utilizate de client în acest scop se obține incluzând proprietatea

binding="etc/custom-client.xml"

în sarcina **wsimport**, iar conținutul fișierului *custom-client.xml* este

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <bindings
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
5   wsdlLocation="http://localhost:8080/jaxws-cmmdc/cmmdcws?wsdl"
6   xmlns="http://java.sun.com/xml/ns/jaxws">
7   <bindings node="wsdl:definitions">
8     <enableAsyncMapping>true</enableAsyncMapping>
9   </bindings>
10 </bindings>
```

Codul corespunzător din **build.xml** este

```

<taskdef name="wsimport" classname="com.sun.tools.ws.ant.WsImport">
  <classpath refid="myclasspath"/>
</taskdef>

<target name="generate-client" depends="init">
  <wsimport
    debug="true"
    verbose="${verbose}"
    keep="true"
    destdir="${build.dir}"
    package="${app.name}.client"
    binding="etc/custom-client.xml"
    xendorsed="true"
    wsdl="${wsdl.uri}">
    <arg line="-extension"/>
  </wsimport>
</target>

```

Pe calculatorul clientului, în clasa reprezentând serviciul (*CmmdcWS* - pentru exemplul tratat), pentru fiecare metodă a serviciului se generează una din metodele

- `public Response<MetodaResponse> metodaAsync(lista parametrilor formali)`
- `public Future<?> metodaAsync(lista parametrilor formali, AsyncHandler<MetodaResponse> asyncHandler)`

Pentru exemplul tratat *metoda=cmmdc*.

Cele două metode oferă posibilitatea construirii a câte unui program client specific, denumite, respectiv modelul *polling* și modelul *callback*.

Au intervenit interfețele

- `public interface Response<T> extends java.util.concurrent.Future<T>`

Un obiect de acest tip conține răspunsul serviciului. Răspunsul se obține cu metoda `T get()`, moștenită de la `Future`.

Metoda boolean `isDone()` returnează `true` dacă s-a primit răspuns.

- `public interface AsyncHandler<T>`

Interfața declară metoda `void handleResponse(Response<T> res)` responsabilă de prelucrarea răspunsului.

Exemplul 4.2.1 Client construit pe modelul *polling*.

```

1 package cmmdc.client;
2 import javax.xml.ws.Response;
3 import java.util.Scanner;

5 public class CmmdcAsyncClient {
6     public static void main(String[] args) {
7         long delta=500;
8         try {
9             CmmdcWS port=new CmmdcWSService().getCmmdcWSPort();
10            Scanner scanner=new Scanner(System.in);
11            System.out.println("m=");
12            long m=scanner.nextLong();
13            System.out.println("n=");
14            long n=scanner.nextLong();

16            Response<CmmdcResponse> response=port.cmmdcAsync(m,n);
17            while(!response.isDone()){
18                System.out.println("Wait "+delta+" ms");
19                Thread.sleep(delta);
20            }
21            CmmdcResponse output=response.get();
22            long result=output.getReturn();
23            System.out.println("Cmmdc="+result);
24        }
25        catch (Exception e) {
26            System.out.printf("AnException : "+e.getMessage());
27        }
28    }
29 }

```

Exemplul 4.2.2 Client construit pe modelul callback.

```

1 package cmmdc.client;
2 import java.util.concurrent.ExecutionException;
3 import java.util.concurrent.Future;
4 import javax.xml.ws.Response;
5 import javax.xml.ws.AsyncHandler;
6 import java.util.Scanner;

8 public class CmmdcAsyncClient {
9     public static void main(String[] args) {
10         long delta=500;
11         try {
12             CmmdcWS port=new CmmdcWSService().getCmmdcWSPort();
13             Scanner scanner=new Scanner(System.in);
14             System.out.println("m=");
15             long m=scanner.nextLong();
16             System.out.println("n=");
17             long n=scanner.nextLong();

19             CmmdcAsyncHandler asyncHandler=new CmmdcAsyncHandler();
20             Future<?> response=port.cmmdcAsync(m,n,asyncHandler);
21             while(!response.isDone()){
22                 System.out.println("Wait "+delta+" ms");
23                 Thread.sleep(delta);
24             }

```

```

25     CmmdcResponse output=asyncHandler.getResponse();
26     long result=output.getReturn();
27     System.out.println("Cmmdc="+result);
28 }
29 catch (Exception e) {
30     System.out.printf("AnException : "+e.getMessage());
31 }
32 }
33 }

35 class CmmdcAsyncHandler implements AsyncHandler<CmmdcResponse>{
36     private CmmdcResponse output;

38     public void handleResponse(Response<CmmdcResponse> response){
39         try{
40             output=response.get();
41         }
42         catch(ExecutionException e){
43             System.out.println("ExecutionException : "+e.getMessage());
44             e.printStackTrace();
45         }
46         catch(InterruptedException e){
47             System.out.println("InterruptedException : "+e.getMessage());
48             e.printStackTrace();
49         }
50     }

52     CmmdcResponse getResponse(){
53         return output;
54     }
55 }

```

4.2.2 Componentă EJB sesiune stateless ca serviciu Web

Șablonul pentru crearea unei componente EJB de tip *stateless session* ca serviciu Web de tip JAX-WS este

```

import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService
@Stateless
public class Componenta{
    @WebMethod
    public tip metoda(@WebParam(name="numeVarFormala")tip numeVarFormala,...){...}
    ...
}

```

Exemplul 4.2.3 Cel mai mare divizor comun a două numere naturale

Componentei EJB are codul

```

1 package cmmdcws;
2 import javax.jws.WebService;
3 import javax.jws.WebMethod;
4 import javax.jws.WebParam;
5 import javax.ejb.Stateless;

7 @WebService
8 @Stateless
9 public class CmmdcEJB {

11     @WebMethod
12     public long cmmdc(@WebParam(name = "m") long m,
13                      @WebParam(name = "n") long n){. . .}
14 }

```

Clasa compilată obișnuit se arhivează cu extensia `jar` și se desfășoară în serverul de aplicație (*glassfish*). Diferența față de varianta de serviciu ca servlet constă în structura arhivei care se desfășoară în serverul Web.

```

cmmdc-ejb
|--> cmmdcws
|   |   CmmdcEJB.class

```

Clientul este unul obișnuit pentru serviciul Web de tip JAX-WS. În cazul de față codul clientului este

```

1 package client;
2 import cmmdcws.*;
3 import java.util.Scanner;

5 public class CmmdcClient {
6     public static void main(String[] args) {
7         try {
8             CmmdcEJB port=new CmmdcEJBService().getCmmdcEJBPort();
9             Scanner scanner=new Scanner(System.in);
10            System.out.println("m=");
11            long m=scanner.nextLong();
12            System.out.println("n=");
13            long n=scanner.nextLong();
14            long result=port.cmmdc(m,n);
15            System.out.println("Cmmdc="+result);
16        }
17        catch (Exception e) {
18            System.out.printf("AnException : "+e.getMessage());
19        }
20    }
21 }

```

Reamintim că dezvoltarea clientului se face cu serviciul Web activ, fiind necesară generarea unor clase - reprezentante ale serviciului - pe calculatorul clientului. La generarea claselor de către `wsimport`, referința URL a serviciului Web trebuie să corespundă serverului Web care conține serviciul în timpul utilizării.

4.2.3 Servicii *jaxws* dezvoltate prin *maven*

Dezvoltarea aplicației server

Vom dezvolta serviciul pentru calculul celui mai mare divizor comun cu

- contextul *jaxws-cmmdc*
- numele de apel *cmmdcws*

Acești parametri sunt specificați în fișierul `sun-jaxws.xml`.

Dezvoltarea aplicației constă din:

1. Generarea aplicației

- Sistemul de operare Windows

```
set GroupID=cmmdc.server
set ArtifactID=jaxws-cmmdc
set Version=1.0
mvn archetype:generate -B
  -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -DarchetypeArtifactId=maven-archetype-webapp
  -Dversion=%Version%
```

- Sistemul de operare Linux

```
GroupID=cmmdc.server
ArtifactID=jaxws-cmmdc
Version=1.0
mvn archetype:generate -B
  -DgroupId=$GroupID
  -DartifactId=$ArtifactID
  -DarchetypeArtifactId=maven-archetype-webapp
  -Dversion=$Version
```

2. Se adaptează structura de cataloage și fișiere la

```
jaxws-cmmdc
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> cmmdc
|   |   |   |   |--> server
|   |   |   |   |   CmmdcWS.java
|   |   |--> resources
|   |   |--> webapp
|   |   |   |--> WEB-INF
|   |   |   |   web.xml
|   |   |   |   sun-jaxws.xml
|   pom.xml
```

3. Fișierul `pom.xml` se completează cu

```

<dependencies>
  . . .
  <dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>webservices-rt</artifactId>
    <version>2.3.1</version>
  </dependency>
  . . .
</dependencies>
<repositories>
  <repository>
    <id>maven2-repository.java.net</id>
    <name>Java.net Repository for Maven 2</name>
    <url>http://download.java.net/maven/2/</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>maven2-repository.java.net</id>
    <name>Java.net Repository for Maven 2</name>
    <url>http://download.java.net/maven/2/</url>
    <layout>default</layout>
  </pluginRepository>
</pluginRepositories>

```

4. Prelucrarea constă din

- (a) `mvn clean package`
- (b) Fișierul `war` care rezultă se desfășoară în *glassfish*.

Dezvoltarea aplicației client

Dezvoltarea aplicației constă din:

1. Generarea aplicației

- Sistemul de operare Windows

```

set GroupID=cmmmc.client
set ArtifactID=jaxws-cmmmc-client
set Version=1.0
mvn archetype:generate -B
  -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -DarchetypeArtifactId=maven-archetype-quickstart
  -Dversion=%Version%

```

- Sistemul de operare Linux

```

GroupID=cmmmc.client
ArtifactID=jaxws-cmmmc-client
Version=1.0
mvn archetype:generate -B
  -DgroupId=$GroupID
  -DartifactId=$ArtifactID
  -DarchetypeArtifactId=maven-archetype-quickstart
  -Dversion=$Version

```

2. Se adaptează structura de cataloage și fișiere la

```

jaxws-cmmdc-client
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> cmmdc
|   |   |   |   |--> client
|   |   |   |   |   CmmdcClient.java
|   |   |   |   |   pom.xml

```

În clasa *CmmdcClient.java* se introduce

```
import cmmdc.server.*;
```

3. Fișierul pom.xml se completează cu

```

<dependencies>
  . . .
  <dependency>
    <groupId>org.glassfish.metro</groupId>
    <artifactId>webservicess-rt</artifactId>
    <version>2.3</version>
  </dependency>
  . . .
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.jvnet.jax-ws-commons</groupId>
      <artifactId>jaxws-maven-plugin</artifactId>
      <version>2.3.1</version>
      <configuration>
        <!-- Needed with JAXP 1.5 -->
        <vmArgs>
          <vmArg>-Djavax.xml.accessExternalSchema=all</vmArg>
        </vmArgs>
      </configuration>
      <executions>
        <execution>
          <goals>
            <goal>wsimport</goal>
          </goals>
          <configuration>
            <wsdlUrls>
              <wsdlUrl>
                http://localhost:8080/jaxws-cmmdc/cmmdcws?wsdl
              </wsdlUrl>
            </wsdlUrls>
            <packageName>cmmdc.client</packageName>
          </configuration>
          <phase>generate-sources</phase>
        </execution>
      </executions>
    </plugin>
  </plugins>

```


4. Prelucrarea constă din

- (a) `mvn clean compile`
- (b) `mvn exec:java -Dexec.mainClass="cmmdc.client.CmmdcClient"`

Întrebări recapitulative

1. Ce fel de aplicații definește interfața de programare JAX-WS?
2. Care este rolul fișierului `wsdl`?

Capitolul 5

Servicii JAX-RS

5.1 Representational State Transfer

REpresentational State Transfer (REST) este un model de arhitectură de aplicație distribuită REST specifică modul cum o resursă - entitate care conține informație specifică - este definită și cum poate fi adresată.

Identificarea unei resurse se face printr-un URI (*Universal Resource Identifier*).

Transferul resursei către un client sau prelucrarea resursei se face utilizând o interfață care conține o mulțime de operații `http: GET, POST, PUT, DELETE`.

Resursele sunt fără stare iar modelul de aplicație este cel de client-server.

Pentru sistemele care satisfac aceste restricții se utilizează terminologia *RESTful*.

Principalul exemplu de sistem RESTful este World Wide Web (WWW) cu protocolul Hyper Text Transfer Protocol (HTTP).

Serviciile Web bazate pe REST se bazează pe accesul la resurse - definite prin identificatori și nu pe apelarea unor metode, ca în modelul Remote Procedure Call (RPC).

Standardul *JAX-RS Java API for RESTful Web Services* a ajuns la versiunea 2.1 (Versiunea 1 este definită de JSR 311, versiunea 2 este definită de JSR 339 iar versiunea 2.1 este definită de JSR 370).

Interfața de programare RESTful se referă atât la partea de server cât și la partea de client Java.

Într-un serviciu RESTful se pun în evidență următoarele entități:

- *Clasă resursă* - *Resource class*. Resursa Web este reprezentată de o clasă Java cu adnotări JAX-RS. *Clasă resursă rădăcină* - *Root resource class*.

Clasă cu adnotarea `@Path`. Resursele adiacente se definesc relativ la această clasă (resursă).

- *Metoda de identificare a cererii - Request method designator.* Adnotarea `@GET` / `@POST` / `@PUT` / `@DELETE` este folosită pentru identificarea cererii HTTP în vederea desemnării metodei de generare / prelucrare a resursei.
- *Metodă de generare / prelucrare a resursei - Resource method.*
- *Localizator a resurselor adiacente - Sub-resource locator.* Metodă pentru localizarea a resurselor adiacente, adică a resurselor care se specifică relativ la resursa rădăcină.
- *Metoda de generare / prelucrare a unei resurse adiacente - Sub-resource method.*
- *Provider* o implementare a interfeței JAX-RS.

O implementare de referință (*Reference Implementation* - RI) este oferită de pachetul `jersey-*.*.*` realizat de *Oracle*. În prezent, în Java, există mai multe implementări pentru servicii REST.

5.2 Server *Jersey* I

5.2.1 Generarea resurselor

Instalarea. Se va descărca o arhivă care conține resursele necesare sub formă de fișiere jar.

Serverul Web care se va utiliza va fi *apache-tomcat*. În vederea desfășurării, serviciul se arhivează cu extensia `war`. Numele arhivei va desemna numele serviciului. Structura care se arhivează va fi

```
catalogul_serviciului_RESTful
|--> WEB-INF
|   |--> classes
|   |   |--> resources
|   |   |   |   *.class
|   |--> lib
|   |   *.jar
|   web.xml
|   index.html sau index.jsp
```

Printre fişierele **.class* se găsesc resursele serviciului. *index.html* sau *index.jsp* oferă posibilitatea apelării serviciului şi de obicei reprezintă un client Web. Alternativ un serviciu JAX-RS se poate apela dintr-un program client. Fişierul *web.xml* este

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6   <servlet>
7     <servlet-name>Jersey Web Application</servlet-name>
8     <servlet-class>
9       org.glassfish.jersey.servlet.ServletContainer
10    </servlet-class>
11    <init-param>
12      <param-name>
13        jersey.config.server.provider.packages
14      </param-name>
15      <param-value>resources</param-value>
16    </init-param>
17    <load-on-startup>1</load-on-startup>
18  </servlet>
19  <servlet-mapping>
20    <servlet-name>Jersey Web Application</servlet-name>
21    <url-pattern>/resources/*</url-pattern>
22  </servlet-mapping>
23 </web-app>

```

Resursele jar necesare sunt cuprinse în distribuţia *jersey*.

Exemplul 5.2.1 *Serviciul RESTful Hello World care constă în furnizarea textului "Hello World".*

Codul sursă al resursei este:

```

1 package resources;

3 import javax.ws.rs.Produces;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.GET;

7 @Path("helloworld")
8 public class HelloWorldResource {

10   public HelloWorldResource() {}

12   @GET
13   @Produces("text/plain")
14   public String getText() {
15     return ("Hello World");
16   }

18   @Path("html")
19   @GET
20   @Produces("text/html")
21   public String getAsHtml() {

```

```

22     return "<html><head></head><body bgcolor=\"#bbeebe\"><center>
23         <p>Hello World</p></center></body></html>";
24 }
25
26 @Path("xml")
27 @GET
28 @Produces("application/xml")
29 public String getAsXml() {
30     return "<response>Hello World</response>";
31 }
32 }

```

Adnotarea `@Path("helloworld")` face ca localizarea resursei să fie

`http://host:port/NumeServiciuRESTful/resources/helloworld`

Metoda `getText()` este o metodă de generare a resursei și are adnotările

- `@GET` - răspunde la o cerere GET;
- `@Produces(String mime-type)` - rezultatul produs are tipul MIME (Multipurpose Internet Mail Exchange) "text/plain".

Suplimentar sunt definite două resurse adiacente. În urma adnotării `@Path("html")`, localizarea resursei este

`http://host:port/NumeServiciuRESTful/resources/helloworld/html`

La o solicitare GET, resursa (clasa) furnizează clientului o pagină `html`.

Analog se tratează resursa adiacentă cu adnotarea `@Path("xml")`.

Desfășurând serviciul într-un server Web (cu numele de apel *HelloWorld*) și apelând `http://localhost:8080/HelloWorld/resources/application.wadl` se obține descrierea WADL a serviciului:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <application xmlns="http://wadl.dev.java.net/2009/02">
3   <doc xmlns:jersey="http://jersey.java.net/"
4     jersey:generatedBy="Jersey: 2.9 2014-05-22 05:12:10"/>
5   <doc xmlns:jersey="http://jersey.java.net/"
6     jersey:hint="This is simplified WADL with user
7     and core resources only. To get full WADL with
8     extended resources use the query parameter detail. Link:
9     http://localhost:8080/HelloWorld/resources/application.wadl?detail=true"/>
10  <grammars/>
11  <resources base="http://localhost:8080/HelloWorld/resources/">
12    <resource path="helloworld">
13      <method id="getText" name="GET">
14        <response>
15          <representation mediaType="text/plain"/>
16        </response>
17      </method>
18      <resource path="html">
19        <method id="getAsHtml" name="GET">

```

```

20         <response>
21             <representation mediaType="text/html" />
22         </response>
23     </method>
24 </resource>
25 <resource path="xml">
26     <method id="getAsXml" name="GET">
27         <response>
28             <representation mediaType="application/xml" />
29         </response>
30     </method>
31 </resource>
32 </resource>
33 </resources>
34 </application>

```

Apelarea serviciului prin intermediul unui navigator se poate face prin intermediul fișierului *index.html* - client Web

```

1 <html>
2 <body>
3 <table border="2">
4 <tr>
5 <td>
6 <a href="/HelloWorld/resources/helloworld">
7 Rezultat "text/plain"</a>
8 </td>
9 </tr>
10 <tr>
11 <td>
12 <a href="/HelloWorld/resources/helloworld/html">
13 Rezultat "text/html"</a>
14 </td>
15 </tr>
16 <tr>
17 <td>
18 <a href="/HelloWorld/resources/helloworld/xml">
19 Rezultat "application/xml"</a>
20 </td>
21 </tr>
22 </table>
23 </body>
24 </html>

```

În acest caz, apelarea este `http://host:port/HelloWorld`.

Dacă catalogul resurselor (*resources* în cazul exemplului anterior) se dorește ascuns, atunci se poate înlocui cu o referință virtuală, de exemplu *webresources*.

În acest caz codul de apelare va fi

`http://localhost:8080/HelloWorld/webresources/helloworld`.

Aceast efect se obține introducând clasa

```

1 package resources;
2 import java.util.Set;
3 import javax.ws.rs.ApplicationPath;
4 import javax.ws.rs.core.Application;

```

```

6 @ApplicationPath("webresources")
7 public class MyApplication extends Application {
8 }

```

Clasa `javax.ws.rs.core.Application` conține metodele

- `public Set<Class<?>> getClasses()`
- `public Set<Class<?>> getSingletons()`

Prezența arhivei `jersey-container-servlet-*.jar` permite eliminarea fișierului `web.xml`.

În exemplul anterior, codul `html` a fost generat de o metodă. Dacă se oferă un fișier `html` existent, atunci acesta se furnizează prin

```

import javax.servlet.ServletContext;
...
@Context ServletContext sc;
@Path("html")
@Produces("text/html")
@GET
public InputStream doGet() {
    return sc.getResourceAsStream("/Index.html");
}

```

Caracterul `/` din argumentul metodei `getResourceAsStream` precizează rădăcina aplicației, mai precis catalogul aplicației din `webapps`.

Printr-o adnotare `@Context` se pot defini variabile de tip `UriInfo`, `Request`, `HttpHeaders`, `ServletContext`, etc.

Exemplu de furnizare a unei imagini *jpg*, *png*, aflat în catalogul *resources* este dată de metoda

```

1 package resources;
2 import java.net.URL;
3 import javax.ws.rs.Produces;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.GET;
6 import javax.activation.DataSource;
7 import javax.activation.FileDataSource;

9 @Path("image")
10 public class ImageResource {

12     public ImageResource() {}

14     @GET
15     @Produces("image/jpeg")
16     public DataSource getImageRep() {
17         URL jpgURL = this.getClass().getResource("forest.jpg");
18         return new FileDataSource(jpgURL.getFile());
19     }
20 }

```


Pe partea de client, imaginea se obține cu metoda

```
import java.awt.Image;
import java.awt.Toolkit;
...
public static Image httpGetImage(String urlStr) throws IOException {
    URL url=new URL(urlStr);
    return Toolkit.getDefaultToolkit().getImage(url);
}
```

unde *urlStr* localizează resursa.

5.2.2 Preluarea parametrilor

Există mai multe metode de programare a preluării argumentelor provenind din formulare html sau din programe client și generarea resursei răspuns:

- Prin context;
- Prin adnotarea `@QueryParam` — `@FormParam`.
- Prin adnotarea `@PathParam`

Parametrii necesari pentru generarea unei resurse pot fi transmiși sub forma unei liste plasat în urma adresei URL a serviciului RESTful. Argumentele sunt separate prin virgulă iar separatorul dintre URL și lista argumentelor este `/`. În serviciul RESTful va apare adnotarea `@PathParam`.

- Prin componente Java.

Preluarea parametrilor prin context

Preluarea unui parametru implică utilizarea mai multor clase ale interfeței JAX-RS.

- Interfața `javax.ws.rs.core.UriInfo`

Metode

- `MultivaluedMap<java.lang.String,java.lang.String>`
`getQueryParameters()`

Returnează lista parametrilor corespunzători cererii.

- Interfața `javax.ws.rs.core.MultivaluedMap<K,V>`
`extends java.util.Map<K,java.util.List<V>>`

Metode

– `V getFirst(K key)`

Returnează valoarea corespunzătoare cheii *key*.

- Clasa `javax.ws.rs.core.Response`

Metode

– `public static Response.ResponseBuilder ok(Object entity, String mimetype)`

Crează un obiect care conține reprezentarea răspunsului.

- Clasa `javax.ws.rs.core.Response.ResponseBuilder`

Metode

– `public abstract Response build()`

Crează obiectul `Response` în urma executării metodei `ok`.

Exemplul 5.2.2 *Serviciu RESTful care furnizează mesajul "Hello nume", unde "nume" este un parametru al clientului. Un al doilea parametru "tip" fixează natura răspunsului `text/plain` sau `text/html`.*

```

1 package resources;
2 import javax.ws.rs.core.MultivaluedMap;
3 import javax.ws.rs.core.Response;
4 import javax.ws.rs.core.Context;
5 import javax.ws.rs.core.UriInfo;
6 import javax.ws.rs.Path;
7 import javax.ws.rs.GET;
8 import java.net.URLDecoder;

10 @Path("hello")
11 public class HelloResource {

13     public HelloResource() {}

15     @Context UriInfo uriInfo;
16     @GET
17     public Response doGet(){
18         MultivaluedMap<String,String> params=uriInfo.getQueryParameters();
19         String nume=params.getFirst("nume");
20         String tip=null;
21         try{
22             tip=URLDecoder.decode(params.getFirst("tip"),"UTF-8");
23         }
24         catch(Exception e){
25             System.out.println(e.getMessage());
26         }
27         System.out.println("Param : "+nume+" "+tip);
28         Response r=null;
29         switch(tip){

```

```

30     case "text/plain":
31         r=Response.ok(getPlainRep(ume),"text/plain").build();
32         break;
33     case "text/html":
34         r=Response.ok(getHtmlRep(ume),"text/html").build();
35         break;
36     }
37     return r;
38 }

40 public String getPlainRep(String ume) {
41     return "Hello "+ume+" !";
42 }

44 public String getHtmlRep(String ume) {
45     return "<html><head></head><body bgcolor=\"#aeeaa\"><center>
46         <h1>Hello "+ume+" ! </h1></center></body></html>";
47 }
48 }

```

Preluarea argumentelor prin adnotarea @QueryParam | @FormParam

Utilizarea adnotărilor @QueryParam, @FormParam permite preluarea parametrilor unui formular transmiși prin metoda GET, respectiv POST. În acest caz serverul Web injectează parametrii metodelor în care este prezentă adnotarea.

Codul serviciului cu metoda GET este:

```

1 package resources;
2 import javax.ws.rs.Path;
3 import javax.ws.rs.GET;
4 import javax.ws.rs.QueryParam;
5 import javax.ws.rs.core.Response;
6 import java.net.URLDecoder;

8 @Path("hello")
9 public class HelloResource {

11     public HelloResource() {}

13     @GET
14     public Response processQuery(
15         @QueryParam("ume") String ume,
16         @QueryParam("tip") String tip) {
17         String tip0=null;
18         try{
19             tip0=URLDecoder.decode(tip,"UTF-8");
20         }
21         catch(Exception e){
22             System.out.println(e.getMessage());
23         }
24         Response r=null;
25         switch(tip0){
26             case "text/plain":
27                 r=Response.ok(getPlainRep(ume),"text/plain").build();

```

```

28         break;
29     case "text/html":
30         r=Response.ok(getHtmlRep( nume ), "text/html").build();
31         break;
32     }
33     return r;
34 }

36 public String getPlainRep(String nume) {
37     return "Hello "+nume+" !";
38 }

40 public String getHtmlRep(String nume) {
41     return "<html><head></head><body bgcolor=\"#aaceaa\"><center>
42         <h1>Hello "+nume+" ! </h1></center></body></html>";
43     }
44 }

```

Apelarea serviciului *RESTful* dintr-un navigator se obține cu

```

1 <html>
2 <body>
3 <center>
4 <h1> Serviciu Hello de tip RESTful </h1>
5 <form method="get"
6     action="/Hello/resources/hello">
7 <p> Introduceți numele </p>
8 <input type="text" name="nume" size="10" />
9 <p> Selectați tipul răspunsului </p>
10 <select name="tip">
11 <option value="text/plain"> Text/Plain </option>
12 <option value="text/html"> Text/Html </option>
13 </select>
14 <p>
15 <input type="submit" value="Apeleaza" />
16 </form>
17 </center>
18 </body>
19 </html>

```

Utilizând metoda POST în programul serviciului, adnotarea `@QueryParam` se înlocuiește cu adnotarea `@FormParam`.

```

1 package resources;
2 import javax.ws.rs.Path;
3 import javax.ws.rs.POST;
4 import javax.ws.rs.GET;
5 import javax.ws.rs.FormParam;
6 import javax.ws.rs.Consumes;
7 import javax.ws.rs.Produces;
8 import javax.ws.rs.core.Response;
9 import java.net.URLDecoder;
10 import java.io.InputStream;
11 import javax.ws.rs.core.Context;
12 import javax.servlet.ServletContext;

14 @Path("hello")
15 public class HelloResource {
16     @Context ServletContext sc;

```

```

18 public HelloResource() {}
19
20 @POST
21 @Consumes("application/x-www-form-urlencoded")
22 public Response processQuery(
23     @FormParam("nume") String nume,
24     @FormParam("tip") String tip) {
25     String tip0=null;//URLDecoder.decode(tip,"UTF-8");
26     try{
27         tip0=URLDecoder.decode(tip,"UTF-8");
28     }
29     catch(Exception e){
30         System.out.println(e.getMessage());
31     }
32     System.out.println(nume+" : "+tip0);
33     Response r=null;
34     switch(tip0){
35         case "text/plain":
36             r=Response.ok(getPlainRep(nume),"text/plain").build();
37             break;
38         case "text/html":
39             r=Response.ok(getHtmlRep(nume),"text/html").build();
40             break;
41     }
42     return r;
43 }
44
45 public String getPlainRep(String nume) {
46     return "Hello "+nume+" !";
47 }
48
49 public String getHtmlRep(String nume) {
50     return "<html><head></head><body bgcolor=\"#aeeaa\"><center><h1>Hello "+
51         nume+" ! </h1></center></body></html>";
52 }
53 }

```

Adnotarea @PathParam

În cazul utilizării metodei `http GET` o soluție mai simplă pentru fixarea unui număr mic de parametri este introducerea lor în adnotarea `@Path` sub forma

```
@Path("refCale/{param1},{param2},. . .")
```

Metoda care preia parametri utilizează adnotarea `@PathParam`:

```
public T getResursa(@PathParam("param1") String p1,
                   @PathParam("param2") String p2,. . .){. . .}
```

Stringul care fixează URL-ul de apelare va fi de forma

.../resources/refCale/valParam1,valParam2

Exemplul 5.2.3 *Serviciu RESTful pentru calculul celui mai mare divizor comun a două numere naturale.*

Codul serviciului este

```

1 package resources;
2 import javax.ws.rs.Path;
3 import javax.ws.rs.PathParam;
4 import javax.ws.rs.Produces;
5 import javax.ws.rs.GET;
6 import javax.ws.rs.core.MediaType;

8 @Path("cmmdc/{num1},{num2}")
9 public class CmmdcResource {

11     public CmmdcResource(){}

13     @GET
14     @Produces(MediaType.APPLICATION_XML)
15     public String getCmmdcAsXML(@PathParam("num1") String sm,
16                                @PathParam("num2") String sn){
17         System.out.println(sm+" "+sn);
18         long m=Long.parseLong(sm);
19         long n=Long.parseLong(sn);
20         long c=cmmdc(m,n);
21         String result=(new Long(c)).toString();
22         return "<?xml version='1.0' encoding='UTF-8'?'><rezultat>"+
23             result+"</rezultat>";
24     }

26     private long cmmdc(long m,long n) { . . . }
27 }

```

În acest caz un client Web se obține cu AJAX

```

1 <!doctype html>
2 <head>

4 <script type="text/javascript" >
5 <!--
6     function initRequest() {
7         if (window.XMLHttpRequest) {
8             return new XMLHttpRequest();
9         } else if (window.ActiveXObject){
10             return new ActiveXObject("Microsoft.XMLHTTP");
11         }
12     }

14     function compute() {
15         var mField=document.getElementById("m");
16         var nField=document.getElementById("n");
17         var url = "http://localhost:8080/cmmdcapp/resources/cmmdc/" +
18             escape(mField.value)+" "+escape(nField.value);
19         var req = initRequest();
20         req.onreadystatechange = function() {
21             if (req.readyState == 4) {

```

```

22         if (req.status == 200) {
23             parseMessages(req.responseXML);
24         } else {
25             alert(req.status+" : "+req.statusText);
26         }
27     }
28 };
29 req.open("get", url, true);
30 req.send(null);
31 }

32
33 function parseMessages(responseXML) {
34     var r = responseXML.getElementsByTagName("rezultat")[0];
35     var cmmdc=r.childNodes[0].nodeValue;
36     document.getElementById("rezultat").innerHTML="Cmmdc = "+cmmdc;
37 }
38 →
39 </script>

41 <title> Cmmdc AJAX</title>
42 </head>
43 <body bgcolor="#bbccbb">

45 <h1>Cmmdc with AJAX</h1>
46 <form>
47 <table>
48 <tr>
49 <td><label> Primul numar </label></td>
50 <td>
51 <input type="number" id="m" size="5"
52     required min="1">
53 </td>
54 </tr>
55 <tr>
56 <td><label> Al doilea numar </label></td>
57 <td>
58 <input type="number" id="n" size="5"
59     required min="1">
60 </td>
61 </tr>
62 <tr>
63 <td>
64 <p><input type="button" value="Calculeaza" onClick="compute()">
65 </td>
66 </tr>
67 </table>
68 </form>
69 <p>
70 Cel mai mare divizor comun al celor doua numere este
71 <p>
72 <div id="rezultat" />
73 </body>
74 </html>

```

5.2.3 Aplicație cu server asincron

Programarea satisfacerii cererii în mod asincron se realizează prin lansarea unui fir de execuție responsabil de rezolvarea cererii. În acest scop

- Elementul `<servlet>` din `web.xml` se completează cu

```
<async-supported>true</async-supported>
```

- Metoda cu execuția asincronă conține variabila formală

```
@Suspended AsyncResponse response
```

cu referințele

```
import javax.ws.rs.container.Suspended;
import javax.ws.rs.container.AsyncResponse;
```

- Șablonul de programare al firului de execuție este

```
new Thread()->{
    // Cod pentru formarea raspunsului 'response'
    response.resume(response);
}.start();
```

- Optional se poate fixa o limită în timp pentru rezolvarea cererii

```
import java.util.concurrent.TimeUnit;
...
int timeOut=...;
response.setTimeout(timeOut, TimeUnit.SECONDS);
response.setTimeoutHandler((asyncResp) -> {
    asyncResp.resume(Response.status(Response.Status.REQUEST_TIMEOUT).build());
});
```

Argumentul metodei `setTimeoutHandler` este un obiect care implementează interfața `javax.ws.rs.container.TimeoutHandler`. Această interfață declară metoda

```
void handleTimeout(AsyncResponse asyncResponse)
```

Pentru exemplul anterior codul serverului asincron este


```

1 package resources;
2 import javax.ws.rs.QueryParam;
3 import javax.ws.rs.core.Response;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.GET;
6 import javax.ws.rs.container.Suspended;
7 import javax.ws.rs.container.AsyncResponse;
8 import java.util.concurrent.TimeUnit;

10 @Path("async")
11 public class HelloResource {

12     @GET
13     public void processForm(@Suspended AsyncResponse response,
14                             @QueryParam("nume") String nume,
15                             @QueryParam("tip") String tip) {

16         new Thread()->{
17             Response r=null;
18             String s;
19             switch(tip){
20                 case "text/plain":
21                     s="Hello "+nume+"!";
22                     r=Response.ok(s,"text/plain").build();
23                     break;
24                 case "text/html":
25                     s="<html><head></head><body bgcolor=\"#aaceaa\"><center><h1>Hello "+
26                       nume+" ! </h1></center></body></html>";
27                     r=Response.ok(s,"text/html").build();
28                     break;
29             }
30             response.resume(r);
31         }).start();

32         response.setTimeout(5, TimeUnit.SECONDS);
33         //client will receive a HTTP 408 (timeout error) after 10 seconds
34         response.setTimeoutHandler((asyncResp) -> {
35             asyncResp.resume(Response.status(Response.Status.REQUEST_TIMEOUT).build());
36         });
37     }
38 }

```

În mediu JEE (de exemplu *glassfish*) codul de mai sus se simplifică în sensul că generarea firului de execuție în care se crează răspunsul nu mai este în sarcina programatorului.

Execuția asincronă a unui serviciu RESTful este indicată schematic în Fig. 5.1.

5.2.4 Jersey în JEE

Pachetul *jersey* este conținut în *glassfish*, *payara*. În consecință arhiva *war* destinată desfășurării unei aplicații nu trebuie să conțină resursele care țin de

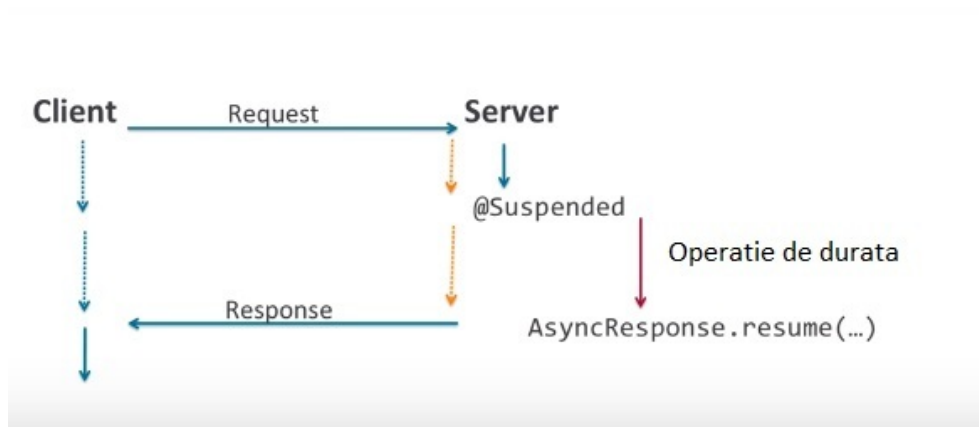


Figure 5.1: Execuția asincronă

jersey.

Aplicațiile dezvoltate anterior funcționează fără nici o modificare în *glassfish*.

Semnalăm arhitectura unei aplicații care utilizează o componentă EJB de tip *session stateless*, care este injectată în clasa serviciului RESTful. Injectarea se poate programa utilizând:

- adnotările `javax.annotation.ManagedBean` și `javax.inject.Inject`.

În acest caz, este nevoie de prezența unui fișier *beans.xml*, având codul

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://java.sun.com/xml/ns/javaee"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     http://java.sun.com/xml/ns/javaee/beans-1.0.xsd">
6 </beans>
  
```

- adnotarea `javax.ejb.EJB`.

Exemplul 5.2.4 *Calculul celui mai mare divizor comun a două numere naturale.*

Structura aplicației este

```

Cmmdc
|--> WEB-INF
|    |--> classes
|    |    |--> cmmdc
|    |    |    |
|    |    |    | App.class
|    |    |--> resources
  
```

```

|         |         |         | CmmdcQueryResources.class
|         |         beans.xml
|         |         web.xml
|         index.html

```

Codul clasei *CmmdcQueryResources* este

```

1 package resources;
2 import javax.ws.rs.QueryParam;
3 import javax.servlet.ServletContext;
4 import javax.ws.rs.core.Response;
5 import javax.ws.rs.Produces;
6 import javax.ws.rs.Path;
7 import javax.ws.rs.GET;
8 import java.net.URLDecoder;
9 import javax.annotation.ManagedBean;
10 import javax.inject.Inject;
11 import cmmdc.App;

13 @Path("cmmdcquery")
14 @ManagedBean
15 public class CmmdcQueryResource {
16     @Inject
17     App obj;

19     @GET
20     public Response processQuery(
21         @QueryParam("m") String sm,
22         @QueryParam("n") String sn,
23         @QueryParam("tip") String tip) {
24         String tip0=URLDecoder.decode(tip);
25         long m=Long.parseLong(sm);
26         long n=Long.parseLong(sn);
27         System.out.println(m+" : "+n+" : "+tip0);
28         long c=obj.cmmdc(m,n);
29         String message=(new Long(c)).toString();
30         Response r=null;
31         switch(tip0){
32             case "text/plain":
33                 r=Response.ok(getPlainRep(message),"text/plain").build();
34                 break;
35             case "text/html":
36                 r=Response.ok(getHtmlRep(message),"text/html").build();
37                 break;
38         }
39         return r;
40     }

42     public String getPlainRep(String msg) {
43         return msg;
44     }

46     public String getHtmlRep(String msg) {
47         return "<html><head></head><body bgcolor=\"#aaceaa\"><center>
48             <h1>Cmmdc : "+ msg+"</h1></center></body></html>";
49     }
50 }

```

Este recomandat ca serviciul RESTful să implementeze funcțiile CRUD

(Create, Read, Update, Delete) asociindu-le, respectiv cu metodele PUT, GET, POST, DELETE.

Exemplul 5.2.5 *Server asincron*

```

1 package resources;
2 import javax.ws.rs.GET;
3 import javax.ws.rs.Path;
4 import javax.ws.rs.QueryParam;
5 import javax.ws.rs.container.Suspended;
6 import javax.ws.rs.container.AsyncResponse;
7 import javax.ejb.Stateless;
8 import javax.ejb.Asynchronous;

10 @Stateless
11 @Path("/cmmdc")
12 public class AsynchronousCmmdcResource {

14     @GET
15     @Asynchronous
16     public void asyncRestMethod(@Suspended final AsyncResponse asyncResponse,
17         @QueryParam("m") String sm,
18         @QueryParam("n") String sn) {
19         long m=Long.parseLong(sm);
20         long n=Long.parseLong(sn);
21         System.out.println(m+" : "+n);
22         long r=cmmdc(m,n);
23         String result = new Long(r).toString();
24         asyncResponse.resume(result);
25     }

27     private long cmmdc(long m,long n) { . . . }

29 }

```

Elementul <web-app> din fișierul web.xml trebuie să fie

```

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1">

```

5.3 Client *Jersey*

Resursele interfeței de programare (API) *Jersey Client*, conținute în pachetul `javax.ws.rs.client`. Această variantă asigură o prelucrare mai simplă și unitară a rezultatelor furnizate de un serviciu RESTful.

Recepționarea răspunsului se poate programa

- sincron - clientul se blochează până la sosirea răspunsului;
- asincron

Client sincron

Șablonul de prelucrare constă din:

1. Crearea unei instanțe a clasei `javax.ws.rs.client.Client`

```
Client client = ClientBuilder.newClient();
```

2. Fixarea adresei serviciului RESTful în interfața `javax.ws.rs.client.WebTarget`.

```
WebTarget webTarget=client.target(String url_resursa);  
WebTarget webTarget=client.target(URI uri_resursa);
```

3. Utilizarea metodelor clasei `WebTarget`:

- `WebTarget path(String path)`
Adresa obținută prin juxtapunere cu `\path`.
- `Invocation.Builder request()`
`Invocation.Builder request(String... acceptedResponseTypes)`
`Invocation.Builder request(MediaType... acceptedResponseTypes)`
Construiește obiectul care execută apelarea resursei.
- `URI getUri()`
Furnizează adresa resursei.
- `WebTarget queryParams(String name, Object... values)`
Adaugă parametrii cererii.

4. Apelarea resurselor prin metodele interfeței `javax.ws.rs.client.Invocation.Builder`:

- `<T> T get(Class<T> responseType)`
- `<T> T post(Class<T> responseType)`
- `<T> T post(Entity<?> entity, Class<T> responseType)`
- `AsyncInvoker async()`

`javax.ws.rs.client.SyncInvoker` este suprainterfață pentru `Invocation.Builder`.

5. Alternativ, interfața `javax.ws.rs.client.AsyncInvoker` oferă metoda

- `Future<Response> get()`

- `<T> Future<T> get(InvocationCallback<T> callback)`

Interfața `javax.ws.rs.client.InvocationCallback<RESPONSE>` declară metodele

- `void completed(RESPONSE response)`
Destinat prelucrării răspunsului.
- `void failed(Throwable throwable)`
Este apelat în cazul unei erori.

Pentru Exemplul 5.2.1 codul sursă al clasei client în varianta sincronă este

```

1 package hw;
2 import javax.ws.rs.client.Client;
3 import javax.ws.rs.client.ClientBuilder;
4 import javax.ws.rs.client.WebTarget;
5 import javax.ws.rs.client.Invocation;

7 public class JerseyClient {
8     public static void main(String args[]) {
9         Client client = ClientBuilder.newClient();
10        String rootURL =
11            "http://localhost:8080/HelloWorld/resources/helloworld";
12        String response="";
13        WebTarget webTarget=null;
14        Invocation.Builder invocationBuilder=null;

16        webTarget=client.target(rootURL);
17        invocationBuilder=webTarget.request();
18        response=invocationBuilder.get(String.class);
19        System.out.println("PLAIN TEXT");
20        System.out.println(response+"\n");

22        invocationBuilder=webTarget
23            .path("html")
24            .request();
25        response=invocationBuilder.get(String.class);
26        System.out.println("HTML");
27        System.out.println(response+"\n");

29        invocationBuilder=webTarget
30            .path("xml")
31            .request();
32        response=invocationBuilder.get(String.class);
33        System.out.println("XML");
34        System.out.println(response);
35    }
36 }
```

O soluție mai elaborată pentru generarea adresei serviciului REST este

```

import java.net.URI;
import javax.ws.rs.core.UriBuilder;
...
String host="http://localhost";
```

```

int port=8080;
String context="HelloWorld";
String resources="resources";
String path="helloworld";

URI rootURL=UriBuilder.fromUri(host)
    .port(port)
    .path(context)
    .path(resources)
    .path(path)
    .build();

```

Client asincron

Varianta asincronă de client este

```

1 package hw;
2 import javax.ws.rs.client.Client;
3 import javax.ws.rs.client.ClientBuilder;
4 import javax.ws.rs.client.InvocationCallback;
5 import java.util.concurrent.CountDownLatch;
6 import java.util.concurrent.TimeUnit;

8 public class JerseyAsyncClient {
9     public static void main(String args[]) {
10         System.out.println("Asynchronous Execution\n");
11         String rootURL =
12             "http://localhost:8080/HelloWorld/resources/helloworld";
13         String [] urls={rootURL,rootURL+"/html",rootURL+"/xml"};
14         CountDownLatch cdl=new CountDownLatch(urls.length);
15         Client client=ClientBuilder.newClient();
16         for(String url:urls){
17             client.target(url)
18                 .request()
19                 .async()
20                 .get(new InvocationCallback<String>(){
21                     @Override
22                     public void completed(String msg){
23                         System.out.println("Received: "+" \n"+msg);
24                         cdl.countDown();
25                     }
26                     @Override
27                     public void failed(Throwable throwable){
28                         System.out.println(throwable.getMessage());
29                         cdl.countDown();
30                     }
31                 });
32         }
33         try {
34             cdl.await();
35             client.close();
36         }
37         catch (final InterruptedException e) {
38             e.printStackTrace();
39         }
40     }
41 }

```

Fiecare tip de resursă a fost generat ca o subresursă cu o cale specifică. Această abordare a făcut posibilă realizarea unui client Web și a unui client bazat pe clasa `java.net.URL`. Un client *jersey* poate prelua o resursă pe baza tipului MIME (*Multipurpose Internet Mail Extensions*). Modificăm codul clasei *HelloWorldResource* în

```

1 package resources;

3 import javax.ws.rs.Produces;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.GET;
6 import javax.ws.rs.core.MediaType;

8 @Path("helloworld")
9 public class HelloWorldResource {

11     public HelloWorldResource() {}

13     @GET
14     @Produces(MediaType.TEXT_PLAIN)
15     public String getText() {
16         return "Hello World";
17     }

19     @GET
20     @Produces(MediaType.TEXT_HTML)
21     public String getAsHtml() {
22         return "<html><head></head><body bgcolor=\"#bbeebb\"><center>
23             <p>Hello World</p></center></body></html>";
24     }

26     @GET
27     @Produces(MediaType.TEXT_XML)
28     public String getAsXml() {
29         return "<response>Hello World</response>";
30     }

32     @GET
33     @Produces(MediaType.APPLICATION_XML)
34     public String getAsAppXml() {
35         return "<response>Hello World</response>";
36     }

38     @GET
39     @Produces(MediaType.APPLICATION_JSON)
40     public String getAsJson() {
41         return "[" + "Hello World" + "]";
42     }
43 }

```

iar codul clasei client *JerseyClient* în

```

1 package hw;
2 import javax.ws.rs.client.Client;
3 import javax.ws.rs.client.ClientBuilder;
4 import javax.ws.rs.client.WebTarget;

```



```

5 import javax.ws.rs.client.Invocation;
6 import javax.ws.rs.core.MediaType;

8 public class JerseyClient {
9     public static void main(String args[]) {
10         String serviceURL="http://localhost:8080/HelloWorld/resources/helloworld";
11         Client client = ClientBuilder.newClient();
12         WebTarget webTarget=client.target(serviceURL);
13         String response;
14         System.out.println("MediaType.TEXT_PLAIN_TYPE");
15         response=webTarget.request(MediaType.TEXT_PLAIN_TYPE)
16             .get(String.class);
17         System.out.println(response);
18         System.out.println();
19         System.out.println("MediaType.TEXT_HTML_TYPE");
20         response=webTarget.request(MediaType.TEXT_HTML_TYPE)
21             .get(String.class);
22         System.out.println(response);
23         System.out.println();
24         System.out.println("MediaType.TEXT_XML_TYPE");
25         response=webTarget.request(MediaType.TEXT_XML_TYPE)
26             .get(String.class);
27         System.out.println(response);
28         System.out.println();
29         System.out.println("MediaType.APPLICATION_XML_TYPE");
30         response=webTarget.request(MediaType.APPLICATION_XML_TYPE)
31             .get(String.class);
32         System.out.println(response);
33         System.out.println();
34         System.out.println("MediaType.APPLICATION_JSON_TYPE");
35         response=webTarget.request(MediaType.APPLICATION_JSON_TYPE)
36             .get(String.class);
37         System.out.println(response);
38         System.out.println();
39     }
40 }

```

Codul clientului pentru Exemplul 5.2.2, cu transmisia datelor prin

- *Context* sau *QueryParam*

```

1 package hello;
2 import javax.ws.rs.client.Client;
3 import javax.ws.rs.client.ClientBuilder;
4 import javax.ws.rs.client.WebTarget;
5 import java.util.Scanner;

7 public class JerseyClient {
8     public static void main(String args[]) {
9         Client client = ClientBuilder.newClient();
10        WebTarget webTarget=
11            client.target("http://localhost:8080/Hello/resources/hello");
12        Scanner scanner=new Scanner(System.in);
13        System.out.println("Introduceti tipul raspunsului : ");
14        System.out.println("( plain / html )");
15        String tip=scanner.next();
16        System.out.println("Introduceti numele : ");
17        String nume=scanner.next();
18        String response=webTarget

```

```

19         .queryParam("nume", nume)
20         .queryParam("tip", "text/" + tip)
21         .request()
22         .get(String.class);
23     System.out.println(response);
24 }
25 }

```

• *FormParam*

```

1 package hello;
2 import javax.ws.rs.client.Client;
3 import javax.ws.rs.client.Entity;
4 import javax.ws.rs.client.ClientBuilder;
5 import javax.ws.rs.client.WebTarget;
6 import javax.ws.rs.core.Form;
7 import javax.ws.rs.core.Response;
8 import java.util.Scanner;

10 public class JerseyClient {
11     public static void main(String args[]) {
12         Client client = ClientBuilder.newClient();
13         WebTarget webTarget =
14             client.target("http://localhost:8080/Hello/resources/hello");
15         Scanner scanner = new Scanner(System.in);
16         System.out.println("Introduceti tipul raspunsului : ");
17         System.out.println("( plain | html )");
18         String tip = scanner.next();
19         System.out.println("Introduceti numele : ");
20         String nume = scanner.next();
21         Form f = new Form().param("nume", nume).param("tip", "text/" + tip);
22         Response response = webTarget
23             .request()
24             .post(Entity.form(f));
25         String r = response.readEntity(String.class);
26         System.out.println(r);
27     }
28 }

```

• *PathParam*

```

1 import com.sun.jersey.api.client.Client;
2 import com.sun.jersey.api.client.WebResource;
3 import javax.ws.rs.core.MediaType;
4 import java.util.Scanner;

6 public class JerseyClient {
7     public static void main(String args[]) {
8         Scanner scanner = new Scanner(System.in);
9         System.out.println("Primul numar : ");
10        long m = scanner.nextLong();
11        System.out.println("Al doilea numar : ");
12        long n = scanner.nextLong();
13        String sm = (new Long(m)).toString();
14        String sn = (new Long(n)).toString();
15        Client client = Client.create();
16        WebResource webResource = client.resource(

```

```

17         "http://localhost:8080/cmmdcapp/resources/cmmdc/"+sm+", "+sn );
18     String response=webResource
19         .accept(MediaType.APPLICATION_XML)
20         .get(String.class);
21     System.out.println();
22     System.out.println(MediaType.APPLICATION_XML);
23     System.out.println(response);
24 }
25 }

```

5.4 Server *Jersey* II

5.4.1 Date prin componentă Java

Datele către serviciul RESTful cât și rezultatele pot fi trimise respectiv recepționate printr-o componentă Java (bean). Datele care urmează a fi schimbate între client și server se includ în componente Java. La recepție programatorul are acces la o componentă Java.

Componenta Java va fi reprezentată sub forma unui document **xml** sau **json** ce va fi inclusă în corpul unui mesaj **http**.

Dacă componenta Java se reprezintă ca document **xml** atunci resursele din *jersey* sunt suficiente dar în cazul unui document **json** sunt necesare resurse suplimentare, funcție de tehnologia utilizată:

- *MOXy POJO based JSON binding*
- *jackson*
- *jettison*

Componenta Java

Clasa componentei Java (POJO) - *CmmdcBean.java*.

```

1 package resources;
2 //import javax.xml.bind.annotation.XmlRootElement;
3
4 //@XmlRootElement(name="date")
5 public class CmmdcBean{
6     private String sm;
7     private String sn;
8     private String result;
9
10    public void setSm(String sm){
11        this.sm=sm;
12    }
13    public String getSm(){

```

```

14 |     return sm;
15 | }
17 | public void setSn(String sn){
18 |     this.sn=sn;
19 | }
20 | public String getSn(){
21 |     return sn;
22 | }
24 | public void setResult(String result){
25 |     this.result=result;
26 | }
27 | public String getResult(){
28 |     return result;
29 | }
30 | }

```

Liniile comentate trebuie decommentate dacă comunicația se bazează pe mesaje xml.

JSON prin *MOXy POJO based JSON binding*

Resursele suplimentare pentru *MOXy POJO based JSON binding* se obțin prin *maven* (`mvn clean dependency:copy-dependencies`)

```

<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>org.eclipse.persistence.moxy</artifactId>
  <version>2.6.4</version>
</dependency>

jersey-entity-filtering-*.jar
jersey-media-moxy-*.jar
org.eclipse.persistence.asm-*.jar
org.eclipse.persistence.core-*.jar
org.eclipse.persistence.moxy-*.jar

```

În clientul *Jersey* componenta Java este acoperită de un obiect `javax.ws.rs.client.Entity`, iar tipul datelor este

- `MediaType.APPLICATION_JSON`,
- `MediaType.APPLICATION_XML`.
- Clasa `javax.ws.rs.client.Entity<T>`

Metode

- `public static <T> Entity<T> entity(T entity, MediaType media Type)`

Exemplul 5.4.1

- Clasa serviciului RESTful - *CmmdcResource.java*.

```

1 package resources;
2 import javax.ws.rs.Consumes;
3 import javax.ws.rs.POST;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.Produces;
6 import javax.ws.rs.core.MediaType;

8 @Path("cmmdc")
9 public class CmmdcResource {
10     public CmmdcResource() {}

12     @POST
13     @Produces(MediaType.APPLICATION_JSON)
14     @Consumes(MediaType.APPLICATION_JSON)
15     public CmmdcBean myJob(CmmdcBean obj){
16         String sm=obj.getSm();
17         String sn=obj.getSn();
18         long m=Long.parseLong(sm);
19         long n=Long.parseLong(sn);
20         long c=cmmdc(m,n);
21         String cmmdc=(new Long(c)).toString();
22         CmmdcBean cb=new CmmdcBean();
23         cb.setResult(cmmdc);
24         return cb;
25     }

27     public long cmmdc(long m,long n) { . . . }
28 }

```

În cazul comunicației prin mesaje xml se va folosi `MediaType.APPLICATION_XML`.

- Client de tip Jersey

```

1 package cmmdc;
2 import javax.ws.rs.client.Client;
3 import javax.ws.rs.client.ClientBuilder;
4 import javax.ws.rs.client.WebTarget;
5 import javax.ws.rs.client.Entity;
6 import javax.ws.rs.core.MediaType;
7 import java.util.Scanner;
8 import java.io.PrintWriter;

10 public class JerseyClient {
11     public static void main(String args[]) {
12         Scanner scanner=new Scanner(System.in);
13         System.out.println("Primul numar : ");
14         long m=scanner.nextLong();
15         System.out.println("Al doilea numar : ");
16         long n=scanner.nextLong();
17         String sm=(new Long(m)).toString();
18         String sn=(new Long(n)).toString();
19         CmmdcBean bean=new CmmdcBean();
20         bean.setSm(sm);
21         bean.setSn(sn);

```

```

23     Client client = ClientBuilder.newClient();
24     String sURI=
25         sURI="http://localhost:8080/JsonCmmdcMoxy/resources/cmmdc";

27     WebTarget target=client.target(sURI);
28     CmmdcBean obj=target.request()
29         .post(Entity.entity(bean, MediaType.APPLICATION_JSON),
30             CmmdcBean.class);
31     System.out.println(obj.getResult());
32 }
33 }

```

• Client Web (*index.html*)

```

1 <html>
2 <head>
3 <script type="text/javascript" >
4 <!--
5     function initRequest() {
6         if (window.XMLHttpRequest) {
7             return new XMLHttpRequest();
8         } else if (window.ActiveXObject){
9             return new ActiveXObject("Microsoft.XMLHTTP");
10        }
11    }

13    function compute() {
14        var mField=document.getElementById("m");
15        var nField=document.getElementById("n");
16        var url = "/JsonCmmdc/resources/cmmdc"
17        var msg=JSON.stringify({
18            "sm":mField.value,
19            "sn":nField.value
20        });
21        var req = initRequest();

23        req.onreadystatechange = function() {
24            if (req.readyState == 4) {
25                if (req.status == 200) {
26                    parseMessages(req.responseText);
27                } else {
28                    alert(req.status+" : "+req.statusText);
29                }
30            }
31        };
32        req.open("post", url, true);
33        req.setRequestHeader("Content-Type","application/json");
34        req.send(msg);
35    }

37    function parseMessages(responseText) {
38        var res=responseText;
39        var s=JSON.parse(res);
40        document.getElementById("rezultat").innerHTML=
41            "Cmmdc = "+s.result;
42    }
43    -->

```

```

44  </script>
45  <title> Cmmdc AJAX</title>
46  </head>
47  <body>
48    <h1>Cmmdc with AJAX</h1>
49    <p>
50      Primul numar :
51      <input type="text" id="m" value="1" size="15" >
52    <p>
53      Al doilea numar :
54      <input type="text" id="n" value="1" size="15" >
55    <p>
56      <input type="button" value="Calculeaza" onClick="compute()" >
57    <p>
58      Cel mai mare divizor comun a celor doua numere este
59    <p>
60      <div id="rezultat" />
61  </body>
62 </html>

```

JSON prin *Jackson*

Resursele suplimentare pentru *MOXy POJO based JSON binding* se obțin prin *maven* (mvn clean dependency:copy-dependencies)

```

<dependency>
  <groupId>org.glassfish.jersey.media</groupId>
  <artifactId>jersey-media-json-jackson</artifactId>
  <version>${jersey.version}</version>
</dependency>

```

JSON prin *Jettison*

Resursa suplimentarea este *jersey-media-json-jettison-*.jar*.

Componente XML

- Client de tip Jersey

```

1  package cmmdc;
2  import javax.ws.rs.client.Client;
3  import javax.ws.rs.client.ClientBuilder;
4  import javax.ws.rs.client.WebTarget;
5  import javax.ws.rs.core.Entity;
6  import javax.ws.rs.core.MediaType;
7  import java.util.Scanner;
8  import java.io.PrintWriter;

10 public class JerseyClient {
11   public static void main(String args[]) {
12     Scanner scanner=new Scanner(System.in);
13     System.out.println("Primul numar : ");
14     long m=scanner.nextLong();
15     System.out.println("Al doilea numar : ");

```

```

16     long n=scanner.nextLong();
17     String sm=(new Long(m)).toString();
18     String sn=(new Long(n)).toString();
19     CmmdcBean bean=new CmmdcBean();
20     bean.setSm(sm);
21     bean.setSn(sn);

23     Client client = ClientBuilder.newClient();
24     String sURI=
25         "http://localhost:8080/XMLCmmdc/resources/cmmdc";

27     WebTarget target=client.target(sURI);
28     CmmdcBean obj = target.request()
29         .post(Entity.entity(bean, MediaType.APPLICATION_XML),
30             CmmdcBean.class);
31     System.out.println(obj.getResult());
32 }
33 }

```

• Clientul Web

```

1 <html>
2 <head>
3 <script type="text/javascript" >
4 <!--
5     function initRequest() {
6         if (window.XMLHttpRequest) {
7             return new XMLHttpRequest();
8         } else if (window.ActiveXObject){
9             return new ActiveXObject("Microsoft.XMLHTTP");
10        }
11    }

13    function compute() {
14        var mField=document.getElementById("m").value;
15        var nField=document.getElementById("n").value;
16        var url = "/XMLCmmdc/resources/cmmdc"
17        var msg=
18            "<?xml version='1.0' encoding='UTF-8' standalone='yes'?'>" +
19            "<date><sm>" +mField+"</sm><sn>" +nField+"</sn></date>";
20        var req = initRequest();

22        req.onreadystatechange = function() {
23            if (req.readyState == 4) {
24                if (req.status == 200) {
25                    parseMessages(req.responseText);
26                } else {
27                    alert(req.status+" : "+req.statusText);
28                }
29            }
30        };
31        req.open("post", url, true);
32        req.setRequestHeader("Content-Type", "application/xml");
33        req.send(msg);
34    }

36    function parseMessages(responseText) {
37        var res=responseText;

```



```

38     document.getElementById("rezultat").innerHTML="Cmmdc = "+res;
39 }
40 —>
41 </script>
42 <title> Cmmdc AJAX</title>
43 </head>
44 <body>
45     <h1>RESTful Cmmdc with AJAX cu mesaj XML</h1>
46     <p>
47         Primul numar :
48         <input type="text" id="m" value="1" size="15" >
49     <p>
50         Al doilea numar :
51         <input type="text" id="n" value="1" size="15" >
52     <p>
53         <input type="button" value="Calculeaza" onClick="compute()" >
54     <p>
55         Cel mai mare divizor comun a celor doua numere este
56     <p>
57     <div id="rezultat" />
58 </body>
59 </html>

```

Aplicații de verificare a unui server REST

- *Advanced REST Client* disponibil in *Chrome Web Store*
- *restclient-ui-fat-*.jar*


```
java -jar restclient-ui-fat-*.jar
```
- *SoapUi-**

5.4.2 Dezvoltare prin *maven*

Dezvoltarea aplicației server

Dezvoltarea aplicației Web constă din:

1. Generarea aplicației

```

set GroupId=resources
set ArtifactId=. . .
set jersey-version=. . .
set Version=1.0
mvn -B archetype:generate
-DgroupId=%GroupId%
-DartifactId=%ArtifactId%
-Dversion=%Version%
-DarchetypeArtifactId=jersey-quickstart-webapp
-DarchetypeGroupId=org.glassfish.jersey.archetypes
-DinteractiveMode=false
-DarchetypeVersion=%jersey-version%

```

2. Se adaptează structura de cataloage și fișiere la

```

hello
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> HelloWorldResource.java
|   |   |--> resources
|   |   |--> webapp
|   |       |--> WEB-INF
|   |       |   web.xml
|   |       |   index.html
|   pom.xml

```

3. Prelucrarea constă din

- (a) `mvn clean package`
- (b) Fișierul `war` care rezultă se desfășoară în serverul Web.

Dezvoltarea aplicației client

Dezvoltarea aplicației constă din:

1. Generarea aplicației

```

set GroupId=hw
set ArtifactId=client
set jersey-version=2.22.1
set Version=1.0
set ArchetypeArtifactId=maven-archetype-quickstart
mvn -B archetype:generate
    -DarchetypeArtifactId=%ArchetypeArtifactId%
    -DgroupId=%GroupId%
    -DartifactId=%ArtifactId%
    -Dversion=%Version%
    -DinteractiveMode=false

```

2. Se adaptează structura de cataloage și fișiere la

```

hello-client
|--> src
|   |--> main
|   |   |--> java
|   |   |   |--> hw
|   |   |       JerseyClient.java
|   |   |       Client.java
|   pom.xml

```

3. Fișierul `pom.xml` se completează cu

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.5.1</version>
      <inherited>true</inherited>
      <configuration>
        <source>1.7</source>
        <target>1.7</target>
      </configuration>
    </plugin>
  </plugins>
</build>
<dependencies>
  . . .
  <dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-client</artifactId>
    <version>${jersey.version}</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.connectors</groupId>
    <artifactId>jersey-apache-connector</artifactId>
    <version>${jersey.version}</version>
  </dependency>
  . . .
</dependencies>
<properties>
  <jersey.version>2.22.1</jersey.version>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

```

4. Prelucrarea constă din

- (a) `mvn clean compile`
- (b) `mvn exec:java -Dexec.mainClass="hw.JerseyClient"`
respectiv
`mvn exec:java -Dexec.mainClass="hw.Client"`

Întrebări recapitulative

1. Enumerați modalitățile de transmitere a datelor într-o aplicație REST.
2. Precizați tipurile de clienți ale unui serviciu REST.

Probleme

1. Exemplu când este necesară prezența metodelor `public Set<Class<?>> getClasses() | getSingletons()` cu `javax.ws.rs.core.Application`.

2. Exemplu când este necesară prezența claselor de serializare / deserializare în aplicații cu date furnizate prin componente Java.
3. Exemplu când este necesară secvența de cod

```
ClientConfig config = new ClientConfig(*.class, . . .);  
Client client = ClientBuilder.newClient(config);
```

4. Rx client
5. Sse Server sent even

Partea IV

MODELUL OSGi

Capitolul 6

Modelul *OSGi*

OSGi - *Open Source Gateway initiative*, 1999, (semnificația numelui fiind astăzi depășită) a dezvoltat un model de cadru de lucru (platformă) privind:

- gestiunea ciclului de viață a unei aplicații (application life cycle management);
- registru de servicii;
- mediu de execuție;
- module.

Pe această bază au fost dezvoltate interfețe de programare (API), servicii, extensii OSGi (OSGi layers).

Cadrul de lucru conține un model specific de aplicație sub formă de componentă sau modul OSGi (bundle for deployment). O asemenea componentă poate pune la dispoziția altor componente funcționalități, comportându-se ca un serviciu (ofertant de servicii) sau poate executa o acțiune punctuală. O componentă OSGi se prezintă sub forma unei arhive **jar**.

În esență, scopul unui cadru de lucru OSGi este oferirea unui mediu pentru crearea și integrarea uniformă de unități (module, componente) de soft.

Un modul este dat de

- O interfață de programare (API), pentru rezolvarea unei probleme (opțional);
- O familie de clase realizate în vederea soluționării unei probleme;
- Mulțimea resurselor necesare claselor.

Un modul în tehnologia OSGi se va înregistra ca un serviciu, într-un *registru de servicii* specific cadrului de lucru.

Crearea unei asemenea servicii presupune definirea unei interfațe și a clasei (claselor) care o implementează, definind două componente OSGi.

O componentă OSGi se poate instala, lansa în execuție, opri, actualiza și dezinstala.

Cadrul de lucru conține un registru de servicii care permite unei componente OSGi să sesizeze existența, apariția sau dispariția unor servicii.

Interfața de programare OSGi (API) este alcătuit din

- Nucleul OSGi (*Core Specification*)
- Extensii OSGi (*Compendium Specification*):
LogService, *ConfigurationAdmin*, *HttpService*
- *Enterprise OSGi*:
RemoteServices, *JDBCService*, *JPAService*

Programarea unei aplicații (serviciu) OSGi se poate face în mod

- *imperativ* prin existența unei clase ce implementează interfața `org.osgi.framework.BundleActivator`.
- *declarativ* prin utilizarea unor resurse OSGi suplimentare.

6.1 Cadre de lucru OSGi

Există mai multe implementări a modelului OSGi:

- *apache-felix*
- *equinox*
- *Knopflerfish*
- *apache-karaf*

Fiecare cadru de lucru oferă un mediu OSGi.

OSGi prin *apache-felix*

Instalarea constă în dezarhivarea arhivei descărcate din Internet.

Utilizare. Din catalogul unde s-a instalat *apache-felix*, mediul se lansează prin

```
java -jar bin\felix.jar
```

În catalogul în care s-a instalat *apache-felix* se va crea un subcatalog *felix-cache* care este folosit de cadrul de lucru.

Comenzile OSGi sunt

Comanda	Funcționalitatea
lb	Afișează lista modulelor OSGi instalate.
exit <int>	Părăsește și închide cadrul de lucru.
install file:modulOSGi.jar	Instalează modulul OSGi
	La instalare unui modul i se atribuie în vederea identificării un număr natural <i>id</i> .
start id	Lansează modulul OSGi <i>id</i> .
start file: modulOSGi.jar	Instalează și lansează modulul OSGi
stop id	Oprește modulul OSGi <i>id</i> .
uninstall id	Dezinstalează modulul OSGi <i>id</i> .

Interfața de lucru, *Apache Felix Gogo*, implementează RFC (*Request for Comments*) 147 publicat de *Internet Engineering Task Force* (IETF).

Lansarea se poate realiza și dintr-un program Java:

```

1 import java.util.ArrayList;
2 import java.util.HashMap;
3 import java.util.List;
4 import java.util.Map;
5 import java.util.ServiceLoader;
6 import org.osgi.framework.Bundle;
7 import org.osgi.framework.BundleContext;
8 import org.osgi.framework.BundleException;
9 import org.osgi.framework.launch.Framework;
10 import org.osgi.framework.launch.FrameworkFactory;
11 public class Launcher{
12     public static void main(String args[]) throws BundleException {
13         FrameworkFactory frameworkFactory =
14             ServiceLoader.load(FrameworkFactory.class).iterator().next();
15         Map<String, String> config = new HashMap<>();
16         Framework framework = frameworkFactory.newFramework(config);
17         framework.start();
18         BundleContext context = framework.getBundleContext();
19         List<Bundle> bundles = new ArrayList<>();
20         bundles.add(context.installBundle(
21             "file:lib/org.apache.felix.gogo.command-0.14.0.jar"));
22         bundles.add(context.installBundle(
23             "file:lib/org.apache.felix.gogo.runtime-0.12.1.jar"));

```

```

24 bundles.add(context.installBundle(
25     "file:lib/org.apache.felix.gogo.shell-0.10.0.jar"));
26 for (Bundle bundle : bundles){
27     bundle.start();
28 }
29 }
30 }

```

OSGi prin *Equinox*

Instalare. Dintr-o distribuție *Eclipse*, se copiază în catalogul propriu fișierul *org.eclipse.osgi-*.jar*, schimbând numele în *equinox.jar*. Alternativ, fișierul menționat mai sus se poate descărca de la <http://download.eclipse.org/eclipse/equinox>.

Utilizare. *Equinox* a adoptat interfața *Apache Felix Gogo*. În vederea lansării se va crea structura

```

equinox
|--> configuration
|   |   config.ini
|   org.eclipse.equinox.console_*.jar
|   org.apache.felix.gogo.runtime_*.jar
|   org.apache.felix.gogo.shell_*.jar
|   org.apache.felix.gogo.command_*.jar
|   equinox.jar

```

unde *config.ini* are codul

```

1 osgi.bundles=file\:org.eclipse.equinox.console_*.jar@start,
2   file:\org.apache.felix.gogo.runtime_*.jar@start,
3   file:\org.apache.felix.gogo.shell_*.jar@start,
4   file:\org.apache.felix.gogo.command_*.jar@start

```

Lansarea se obține prin

```
java -jar equinox.jar -console
```

apărând prompt-ul *osgi>*.

Comenzile sunt cele utilizate în *apache-felix*.

În plus se pot utiliza și comenzile

Comanda	Funcționalitatea
ss	<i>short status</i> – Afixează lista modulelor OSGi instalate.
exit	Părăsește și închide cadrul de lucru.

Cadrul de lucru folosește subcatalogul *configuration*.

OSGi prin *knopflerfish*

Instalarea constă în dezarhivarea arhivei descărcate din Internet, reprezentat de un fișier `jar` executabil.

Utilizare. Din catalogul `...\knopflerfish_osgi-*.*.*/osgi` se lansează `java -jar framework.jar`. Toate comenzile de operare sunt lansate dintr-o interfață grafică.

6.2 Programare imperativă - Crearea unui modul OSGi

Un modul sau componentă OSGi (bundle) se poate afla în stările `UNINSTALLED`, `INSTALLED`, `RESOLVED`, `STARTING`, `STOPPING`, `ACTIVE`, constante definite de interfața `org.osgi.framework.Bundle`. Doar cadrul de lucru OSGi poate instanția o componentă OSGi.

O componentă OSGi este compusă din

1. O clasă ce implementează interfața `BundleActivator`.

Interfața `BundleActivator` declară metodele

- `public void start(BundleContext ctx)`
Activități executate la lansarea modulului OSGi.
- `public void stop(BundleContext ctx)`
Activități executate la oprirea modulului OSGi.

Dacă se definește un modul interfață atunci această clasă lipsește.

2. Un fișier text `manifest.mf` de proprietăți (nume: valoare) ale modulului OSGi, cu extensia `mf`. Acest fișier trebuie să se termine cu o linie vidă.

Proprietăți OSGi

Numele proprietății	Semnificația
Bundle-Name	Numele modului OSGi
Bundle-Version	Versiunea modului OSGi
Bundle-Activator	Clasa modului care implementează interfața BundleActivator
Bundle-SymbolicName	Numele simbolic atribuit modului OSGi
Bundle-Description	Descrierea modului OSGi
Manifest-Version	Versiunea tipului de <i>manifest</i>
Bundle-ManifestVersion	Versiunea <i>manifestului</i> atașat modului OSGi
Bundle-Vendor	Producătorul modului OSGi
Bundle-Classpath	Calea către resursele suplimentare utilizate
Import-Package	Lista pachetelor utilizate de modulul OSGi
Export-Package	Lista pachetelor ce pot fi utilizate de alte module OSGi

Structura arhivei `jar` a unei componente OSGi este

```

structura de cataloage corespunzatoare pachetului aplicatiei
| . . . *.class
META-INF
| manifest.mf

```

Programarea se bazează pe interfața **BundleContext**, care este implementat de fiecare cadru de lucru OSGi în parte.

Metode

- **ServiceRegistration** `registerService(String clazz, Object service, java.util.Dictionary properties)`

Înregistrează în cadrul de lucru serviciul definit de obiectul *service*, de tip *clazz* și cu proprietățile adiacente date de al treilea parametru. Rezultatul este folosit de cadrul de lucru OSGi.

- **ServiceReference[]** `getServiceReferences(String clazz, String filter) throws InvalidSyntaxException`

Returnează referințele de tip *clazz*.

- **Object** `getService(ServiceReference reference)`

Returnează obiectul corespunzător referinței.

6.3 Exemple

Exemplul 6.3.1 *Calculul celui mai mare divizor comun a două numere naturale - ca serviciu OSGi.*

Aplicația se compune din 3 module OSGi:

- interfață;

1. Clasa interfeței *cmmdc.ICmmdc.java*

```
1 package cmmdc;  
  
3 public interface ICmmdc{  
4     public long cmmdc(long m, long n);  
5 }
```

2. Fișierul *manifest.mf*

```
1 Bundle-ManifestVersion: 2  
2 Bundle-SymbolicName: icmmdc  
3 Bundle-Name: Interfata Cmmdc  
4 Bundle-Version: 1.0.0  
5 Export-Package: cmmdc; version="1.0.0"
```

- serviciu care implementează interfața;

1. Clasa *cmmdc.service.Activator* implementează interfața **BundleActivator**. În metoda **start**, se înregistrează o instanță a clasei ce implementează interfața ca serviciu a cadrului OSGi.

```
1 package cmmdc.service;  
2 import cmmdc.ICmmdc;  
3 import org.osgi.framework.BundleActivator;  
4 import org.osgi.framework.BundleContext;  
  
6 public class Activator implements BundleActivator{  
7     public void start(BundleContext context){  
8         context.registerService(ICmmdc.class.getName(),  
9             new CmmdcService(), null);  
10        System.out.println("Registering Cmmdc service.");  
11    }  
  
13    public void stop(BundleContext context) {}  
14 }
```

2. Clasa *cmmdc.service.CmmdcService* implementează interfața *ICmmdc*

```

1 package cmmdc.service;

3 public class CmmdcService implements cmmdc.ICmmdc{
4     public long cmmdc(long m, long n){. . .}
5 }

```

3. Fișierul *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: cmmdcservice
3 Bundle-Name: Cmmdc Service
4 Bundle-Version: 1.0.0
5 Bundle-Activator: cmmdc.service.Activator
6 Import-Package: org.osgi.framework, cmmdc

```

- client care utilizează interfața.

Prezentăm două variante de modul OSGi client.

1. Referința obiectului serviciu se obține prin intermediul obiectului *context:BundleContext*.

- (a) Clasa care implementează interfața **BundleActivator** este *cmmdc.client.Activator.java*. În metoda **start**, prin context se obține o referință la serviciul care implementează interfața *ICmmdc*, apoi se folosește funcționalitatea obținută - apelând metoda *cmmdc*.

```

1 package cmmdc.client;
2 import cmmdc.ICmmdc;
3 import java.util.Scanner;
4 import org.osgi.framework.BundleActivator;
5 import org.osgi.framework.BundleContext;
6 import org.osgi.framework.ServiceReference;

8 public class Activator implements BundleActivator{
9     public void start(BundleContext context){
10         try{
11             ServiceReference[] refs=
12                 context.getServiceReferences(ICmmdc.class.getName(),
13                     null);
14             if(refs!=null){
15                 ICmmdc obj=(ICmmdc)context.getService(refs[0]);
16                 Scanner scanner=new Scanner(System.in);
17                 System.out.println("Client Cmmdc 1");
18                 System.out.println("Primul numar : ");
19                 long m=scanner.nextLong();
20                 System.out.println("al doilea numar : ");
21                 long n=scanner.nextLong();
22                 long c=obj.cmmdc(m,n);
23                 System.out.println("Cmmdc = "+c);
24             }
25         }
26     }
27 }

```

```

26     catch(Exception e){
27         System.out.println(" Client Exception : "+e.getMessage());
28     }
29 }

31 public void stop(BundleContext context) {}
32 }

```

(b) Fișierul *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: clientcmmdc1
3 Bundle-Name: Cmmdc Client
4 Bundle-Version: 1.0.0
5 Bundle-Activator: cmmdc.client.Activator
6 Import-Package: org.osgi.framework, cmmdc

```

2. Referința obiectului serviciu se obține prin intermediul unui obiect de tip **ServiceTracker**, clasă introdusă pentru simplificarea lucrului cu componente OSGi.

Constructor

- `public ServiceTracker(BundleContext context, String clazz, ServiceTrackerCustomizer customizer)`

Metode

- `public void open()`
 - `public Object getService()`
- (a) Clasa care implementează interfața **BundleActivator** este *cmmdc.client.Activator.java*. În metoda **start**, se crează o instanță de tip **ServiceTracker** corespunzătoare serviciului generat de interfața *ICmmdc*, care este pornită prin metoda **open** și în final se obține o referință a serviciului, adică o instanță a clasei care implementează interfața *ICmmdc*.

```

1 package cmmdc.client;
2 import cmmdc.ICmmdc;
3 import java.util.Scanner;
4 import org.osgi.framework.BundleActivator;
5 import org.osgi.framework.BundleContext;
6 import org.osgi.util.tracker.ServiceTracker;

8 public class Activator implements BundleActivator{
9     public void start(BundleContext context){
10         ServiceTracker serviceTracker=new ServiceTracker(context,
11             ICmmdc.class.getName(), null);
12         serviceTracker.open();
13         ICmmdc obj=(ICmmdc)serviceTracker.getService();
14         System.out.println(" Client Cmmdc 2");
15         Scanner scanner=new Scanner(System.in);

```

```

16      System.out.println("Primul numar : ");
17      long m=scanner.nextLong();
18      System.out.println("al doilea numar : ");
19      long n=scanner.nextLong();
20      long c=obj.cmmdc(m,n);
21      System.out.println("Cmmdc = "+c);
22  }

24  public void stop(BundleContext context) {}
25  }

```

(b) Fișierul *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: clientcmmdc2
3 Bundle-Name: Cmmdc Client
4 Bundle-Version: 1.0.0
5 Bundle-Activator: cmmdc.client.Activator
6 Import-Package: org.osgi.framework,org.osgi.util.tracker,cmmdc

```

Să presupunem că s-au generat cele module OSGi sub forma

interfața	<i>cmmdc.jar</i>
serviciul (implementarea interfeței)	<i>cmmdcservice.jar</i>
client	<i>client1cmmdc.jar</i>

Execuția în mediul *apache-felix* constă din

```

start file:d:/ . ./icmmdc.jar
start file:d:/ . ./cmmdcservice.jar
start file:d:/ . ./clientcmmdc1.jar

```

6.4 Dezvoltare OSGi prin *apache-maven*

Generarea unei componente OSGi prin maven se bazează pe *plugin*-ul *maven-bundle-plugin*, bazat pe utilitarul *bnd*.

Codurile claselor Java rămân nemodificate, iar fișierele *manifest.mf* vor fi generate de maven, dar conținutul lor va trebui specificat în fișierele *pom.xml*.

Exemplificăm cu aplicația de calcul a celui mai mare divizor comun, dezvoltată în capitolul *OSGi*.

Într-un catalog, de exemplu - *cmmdc*, se generează trei module *maven*

```

mvn archetype:generate -B
-DgroupId=%GroupID%
-DartifactId=%ArtifactID%
-DarchetypeArtifactId=maven-archetype-quickstart
-Dversion=%Version%

```

respectiv cu datele


```
set GroupID=cmmdc
set ArtifactID=interface
set Version=1.0.0
```

```
set GroupID=cmmdc.service
set ArtifactID=impl
set Version=1.0.0
```

```
set GroupID=cmmdc.client
set ArtifactID=client
set Version=1.0.0
```

În cataloagele generate clasa `App.java` se înlocuiește cu clasele aplicației, după cum urmează

Catalog	Clase
interface	ICmmdc.java
impl	CmmdcService.java Activator.java
client	Activator.java

iar fișiere `pom.xml` cu cele reproduse mai jos.

Fișierul `pom.xml` din catalogul *interface*

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>

7   <groupId>cmmdc</groupId>
8   <artifactId>icmmdc</artifactId>
9   <version>1.0.0</version>
10  <packaging>bundle</packaging>

12  <name>interface</name>
13  <url>http://maven.apache.org</url>

15  <properties>
16    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17  </properties>

19  <dependencies>
20    <dependency>
21      <groupId>junit</groupId>
22      <artifactId>junit</artifactId>
23      <version>3.8.1</version>
24      <scope>test</scope>
25    </dependency>
26    <dependency>
27      <groupId>org.osgi</groupId>
28      <artifactId>org.osgi.core</artifactId>
29      <version>4.3.0</version>
30    </dependency>
31  </dependencies>
32  <build>
33    <plugins>
```

```

34     <plugin>
35       <groupId>org.apache.felix</groupId>
36       <artifactId>maven-bundle-plugin</artifactId>
37       <extensions>true</extensions>
38       <configuration>
39         <instructions>
40           <Bundle-SymbolicName>
41             ${project.artifactId}
42           </Bundle-SymbolicName>
43           <Export-Package>
44             cmmdc
45           </Export-Package>
46         </instructions>
47       </configuration>
48     </plugin>
49   </plugins>
50 </build>
51 </project>

```

Fișierul pom.xml din catalogul *impl*

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>

6
7   <groupId>cmmdc</groupId>
8   <artifactId>cmmdcservice</artifactId>
9   <version>1.0.0</version>
10  <packaging>bundle</packaging>

11
12  <name>impl</name>
13  <url>http://maven.apache.org</url>

14
15  <properties>
16    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17    <path>.../cmmdc</path>
18  </properties>

19
20  <dependencies>
21    <dependency>
22      <groupId>junit</groupId>
23      <artifactId>junit</artifactId>
24      <version>3.8.1</version>
25      <scope>test</scope>
26    </dependency>
27    <dependency>
28      <groupId>org.osgi</groupId>
29      <artifactId>org.osgi.core</artifactId>
30      <version>4.3.0</version>
31    </dependency>
32    <dependency>
33      <groupId>icmmdc-1.0.0</groupId>
34      <artifactId>cmmdc.icmmdc</artifactId>
35      <version>1.0.0</version>
36      <scope>system</scope>
37      <systemPath>${path}/interface/target/icmmdc-1.0.0.jar</systemPath>
38    </dependency>

```

```

39 </dependencies>
40 <build>
41   <plugins>
42     <plugin>
43       <groupId>org.apache.felix</groupId>
44       <artifactId>maven-bundle-plugin</artifactId>
45       <extensions>true</extensions>
46       <configuration>
47         <instructions>
48           <Bundle-SymbolicName>
49             ${project.artifactId}
50           </Bundle-SymbolicName>
51           <Import-Package>
52             org.osgi.framework,cmmdc
53           </Import-Package>
54           <Bundle-Activator>cmmdc.service.Activator</Bundle-Activator>
55           <Export-Package/>
56         </instructions>
57       </configuration>
58     </plugin>
59   </plugins>
60 </build>
61 </project>

```

Fișierul `pom.xml` din catalogul *client*

```

1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>

7   <groupId>cmmdc</groupId>
8   <artifactId>clientcmmdc</artifactId>
9   <version>1.0.0</version>
10  <packaging>bundle</packaging>

12  <name>client</name>
13  <url>http://maven.apache.org</url>

15  <properties>
16    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17    <path>.../cmmdc</path>
18  </properties>

20  <dependencies>
21    <dependency>
22      <groupId>junit</groupId>
23      <artifactId>junit</artifactId>
24      <version>3.8.1</version>
25      <scope>test</scope>
26    </dependency>
27    <dependency>
28      <groupId>org.osgi</groupId>
29      <artifactId>org.osgi.core</artifactId>
30      <version>4.3.0</version>
31    </dependency>
32    <dependency>
33      <groupId>icmmdc-1.0.0</groupId>

```

```

34     <artifactId>cmmmc.icmmdc</artifactId>
35     <version>1.0.0</version>
36     <scope>system</scope>
37     <systemPath>${path}/interface/target/icmmdc-1.0.0.jar</systemPath>
38   </dependency>
39 </dependencies>
40 <build>
41   <plugins>
42     <plugin>
43       <groupId>org.apache.felix</groupId>
44       <artifactId>maven-bundle-plugin</artifactId>
45       <extensions>true</extensions>
46       <configuration>
47         <instructions>
48           <Bundle-SymbolicName>
49             ${project.artifactId}
50           </Bundle-SymbolicName>
51           <Import-Package>
52             org.osgi.framework,cmmmc
53           </Import-Package>
54           <Bundle-Activator>cmmmc.client.Activator</Bundle-Activator>
55           <Export-Package/>
56         </instructions>
57       </configuration>
58     </plugin>
59   </plugins>
60 </build>
61 </project>

```

În catalogul aplicației se introduce fișierul *pom.xml*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project>
3   <modelVersion>4.0.0</modelVersion>
4   <artifactId>cmmmc</artifactId>
5   <groupId>cmmmc</groupId>
6   <version>1.0.0</version>
7   <name>Simple cmmmc</name>
8   <packaging>pom</packaging>
9   <modules>
10     <module>interface</module>
11     <module>impl</module>
12     <module>client</module>
13   </modules>
14   <build>
15     <pluginManagement>
16       <plugins>
17         <plugin>
18           <groupId>org.apache.maven.plugins</groupId>
19           <artifactId>maven-compiler-plugin</artifactId>
20           <configuration>
21             <source>1.8</source>
22             <target>1.8</target>
23           </configuration>
24         </plugin>
25         <plugin>
26           <groupId>org.apache.felix</groupId>
27           <artifactId>maven-bundle-plugin</artifactId>
28           <version>3.2.0</version>

```

```

29         <extensions>true</extensions>
30     </plugin>
31 </plugins>
32 </pluginManagement>
33 </build>
34 </project>

```

și se lansează comanda maven `mvn clean package`, în urma căreia se crează componentele OSGi ale aplicației.

6.5 Programare declarativă

Modelul declarativ de realizare a unui modul OSGi nu mai necesită realizarea unei clase care să implementeze interfața `BundleActivator`. Prezența fișierului `manifest.mf` rămâne obligatorie. Există mai multe variante dezvoltate de programare declarativă:

- *Declarative Service*
- *Blueprint*
- *iPOJO*

6.5.1 Programare declarativă prin *Declarative Service*

Modelul de programare este funcțional pe platformele *apache-felix* și *equinox*, dar resursele suplimentare necesare a fi instalate sunt diferite.

Cadrul <i>OSGi</i>	Componenta <i>OSGi</i> (*.jar)
felix	org.osgi.compendium-4.*.*.jar
	org.apache.felix.scr-*.jar
equinox	org.eclipse.osgi.services-3.*.*.v*.jar
	org.eclipse.equinox.util_*.jar
	org.eclipse.equinox.ds-*.jar

Cu excepția interfețelor, programatorul trebuie să editeze un fișier de configurare `OSGI-INF\component.xml`, care se include în arhiva `jar` a componentei OSGi. Acest fișierul de configurare este menționat în fișierul `manifest.mf` prin linia

`Service-Component: OSGI-INF/component.xml`

Astfel arhiva modulului OSGi are structura

```

structura de cataloage corespunzatoare pachetului aplicatiei
| . . . *.class
META-INF
|   manifest.mf
OSGI-INF
|   component.xml

```

Un serviciu programat prin modelul declarativ se poate apela de către un client programat prin modelul imperativ, dar se poate programa și un modul OSGi client prin model declarativ.

Exemplul 6.5.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicația este alcătuită din cele trei componente:

- interfața

```

1.
1 package cmmdc;
3 public interface ICmmdc{
4     public long cmmdc(long m,long n);
5 }

```

2. *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: icmmdc
3 Bundle-Name: Interfata Cmmdc
4 Bundle-Vendor: unitbv
5 Bundle-Version: 1.0.0
6 Export-Package: cmmdc;version="1.0.0"

```

- clasa serviciului OSGi, care implementează interfața

```

1.
1 package cmmdc.service;
3 public class CmmdcService implements cmmdc.ICmmdc{
4     public long cmmdc(long m, long n){. . .}
5 }

```

2. *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: cmmdcservice
3 Bundle-Name: Cmmdc Service
4 Bundle-Version: 1.0.0
5 Import-Package: cmmdc
6 Service-Component: OSGI-INF/component.xml

```

3. *component.xml*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
3     name="DS Service Sample">
4     <implementation class="cmmdc.service.CmmdcService"/>
5     <service>
6         <provide interface="cmmdc.ICmmdc"/>
7     </service>
8 </scr:component>

```

Client declarativ

Pentru compilarea aplicației client este nevoie de `osgi_R4_compendium-*.jar`.

```

1.
1 package cmmdc.client;
2 import java.util.Scanner;
3 import cmmdc.ICmmdc;
4
5 public class CmmdcClient {
6     private ICmmdc service;
7
8     public void bindMyService(ICmmdc a) {
9         System.out.println("Service was set");
10        service = a;
11        Scanner scanner=new Scanner(System.in);
12        System.out.println("m=");
13        long m=scanner.nextLong();
14        System.out.println("n=");
15        long n=scanner.nextLong();
16        System.out.println("Cmmdc = "+service.cmmdc(m,n));
17    }
18
19    public void unbindMyService(ICmmdc a) {
20        if(service==a)
21            service = null;
22        System.out.println("Service was unset");
23    }
24 }

```

2. *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: clientds
3 Service-Component: OSGI-INF/component.xml
4 Private-Package: cmmdc.client
5 Bundle-Name: Declarative Service Cmmdc Client
6 Bundle-Description: DS Cmmdc Client

```

```

7| Bundle-Version: 1.0.0
8| Import-Package: cmmdc

```

3. *component.xml*

```

1| <?xml version="1.0" encoding="UTF-8"?>
2| <scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
3|     name="DS Client">
4|     <implementation class="cmmdc.client.CmmdcClient"/>
5|     <reference interface="cmmdc.ICmmdc"
6|         name="ICmmdc" cardinality="1..1" policy="static"
7|         bind="bindMyService"
8|         unbind="unbindMyService"/>
9| </scr:component>

```

Sunt declarate metodele care

- consumă serviciul (**bind=...**);
- disponibilizează serviciul injectat (**unbind=...**).

6.5.2 Programare declarativă prin *Blueprint*

Modelul de programare este funcțional pe platformele *apache-felix* și *equinox*. Implementarea de referință este dată în proiectul *apache-aries*.

Cadrul OSGi	Componenta OSGi (*.jar)
felix & equinox	org.apache.felix.configadmin-*.jar
	pax-logging-api-*.jar
	pax-logging-service-*.jar
	org.apache.aries.blueprint-*.jar
	org.apache.aries.proxy-*.jar

Toate părțile unei aplicații (interfețe, implementările lor, clienții) trebuie să facă parte din aceeași componentă OSGi. Un client este reprezentat de două clase:

- Înregistratorul (*listener*);

Mediul OSGi asigură serviciul, adică un obiect care implementează interfața aplicației. Trebuie programate două metode, una în care se utilizează serviciul (uzual se apelează un clientul propriu-zis) și a doua în care se disponibilizează serviciul. Cele două metode sunt precizate în fișierul de configurare OSGI-INF\blueprint\config.xml.

- Clientul propriu-zis.

Serviciile înregistrate pot fi apelate și de un client programat imperativ.

În afara fișierului `manifest.mf` mai este necesar fișierul de configurare `OSGI-INF\blueprint\config.xml`.

Astfel arhiva modulului OSGi are structura

```
structura de cataloage corespunzatoare pachetului aplicatiei
| . . . *.class
META-INF
|   manifest.mf
OSGI-INF
|--> blueprint
|       |       config.xml
```

Exemplul 6.5.2 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicația va conține:

- Clasele `cmmmc.ICmmdc` și `cmmmc.service.CmmdcService` amintite anterior.
- Clasa *listener* `cmmmc.client.RegistrationListener`

```
1 package cmmmc.client;
2 import java.util.Map;
3 import java.util.Scanner;
4 import cmmmc.ICmmdc;
5
6 public class RegistrationListener {
7     public void register(ICmmdc obj, Map properties) {
8         VisualCmmdcClient myApp=new VisualCmmdcClient(obj);
9     }
10
11     public void unregister(ICmmdc obj, Map properties) {
12         System.out.println("Cmmmdc service unregistered");
13     }
14 }
```

Clasa `VisualCmmdcClient` este un client cu interfața grafică. (Se va utiliza `setDefaultCloseOperation(javax.swing.WindowConstants.HIDE_ON_CLOSE)`.)

- *manifest.mf*

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: blueprint
3 Bundle-Name: Blueprint Cmmdc Client
4 Bundle-Description: Blueprint Cmmdc App
5 Bundle-Version: 1.0.0
6 Import-Package: org.osgi.framework, javax.swing,
7   org.osgi.service.blueprint; version=" [1.0.0, 2.0.0) "
8 Export-Package: cmmdc, blueprint;
9   uses:= "org.osgi.framework"; version=" 1.0.0.SNAPSHOT"

```

- *OSGI-INF\blueprint\config.xml*

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
3   <bean id="cmmdc"
4     class="cmmdc.service.CmmdcService" scope="prototype"/>
5
6   <service id="serviceClient" ref="cmmdc">
7     <interfaces>
8       <value>cmmdc.ICmmdc</value>
9     </interfaces>
10    <registration-listener
11      registration-method="register"
12      unregistration-method="unregister">
13      <bean class="cmmdc.client.RegistrationListener"/>
14    </registration-listener>
15  </service>
16 </blueprint>

```

6.5.3 Programare declarativă prin *apache-iPOJO*

Acest model este funcțional pe platforma OSGi *apache felix*. În etapa de dezvoltare conținutul fișierului **manifest.mf** este extins cu date dintr-un alt fișier *metadata.xml* - aspectul declarativ al modelului de programare. Datele acestui fișier definesc componenta OSGi care se înregistrează pe platforma OSGi dată de *apache-felix*.

Completarea fișierului **manifest.mf** se face automat, cu instrumente specifice *apache-ant* sau *apache-maven*.

Exemplul 6.5.3 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicația este alcătuită din cele trei componente:

- interfața

```

1 package cmmdc;
2
3 public interface ICmmdc{
4   public long cmmdc(long m, long n);
5 }

```

- clasa serviciului OSGi, care implementează interfața

```

1 package cmmdc.service;

3 public class CmmdcService implements cmmdc.ICmmdc{
4     public long cmmdc(long m, long n){. . .}
5 }

```

metadata.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ipojo
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="org.apache.felix.ipojo
5     http://felix.apache.org/ipojo/schemas/CURRENT/core.xsd"
6     xmlns="org.apache.felix.ipojo">

8     <component classname="cmmdc.service.CmmdcService" name="CmmdcProvider">
9         <provides />
10    </component>

12    <instance component="CmmdcProvider" name="CmmdcService" />
13 </ipojo>

```

- client OSGi

```

1 package cmmdc.client;
2 import cmmdc.ICmmdc;

4 public class CmmdcClient implements Runnable {

6     /**
7      * Intarziere intre doua apelari automate
8      */
9     private static final int DELAY = 10000;

12    /**
13     * Reprezentant al serviciului injectat de iPOJO
14     */
15    private ICmmdc m_cmmdc;

17    /**
18     * Indicator pentru depistarea starsitului
19     */
20    private boolean m_end;

22    /**
23     * Argumente pentru metodele invocate
24     * injectate de containerul OSGi
25     */
26    private String sm;
27    private String sn;

29    /**
30     * Metoda run (java.lang.Runnable)
31     */
32    public void run() {

```

```

33         while (!m_end) {
34             try {
35                 invokeCmmdcServices();
36                 Thread.sleep(DELAY);
37             } catch (InterruptedException ie) {
38                 /* se verifica conditia de sfarsit */
39             }
40         }
41     }

42     /**
43     * Invocarea serviciului
44     */
45     public void invokeCmmdcServices() {
46         int i=0;
47         long m=Long.parseLong(sm);
48         long n=Long.parseLong(sn);
49         System.out.println(m_cmmdc.cmmdc(m,n));
50         m_end=true;
51     }

52     /**
53     * Start
54     */
55     public void starting() {
56         Thread thread = new Thread(this);
57         m_end = false;
58         thread.start();
59     }

60     /**
61     * Stop
62     */
63     public void stopping() {
64         m_end = true;
65     }
66 }

```

metadata.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ipojo
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="org.apache.felix.ipojo
5     http://felix.apache.org/ipojo/schemas/CURRENT/core.xsd"
6     xmlns="org.apache.felix.ipojo">
7
8     <component classname="cmmdc.client.CmmdcClient">
9         <requires field="m_cmmdc" />
10        <callback transition="validate" method="starting" />
11        <callback transition="invalidate" method="stopping" />
12        <properties>
13            <property field="sm" name="sm" />
14            <property field="sn" name="sn" />
15        </properties>
16    </component>
17
18    <instance component="cmmdc.client.CmmdcClient">
19        <property name="sm" value="56" />

```

```
20 <property name="sn" value="42" />
21 </instance>
22 </ipojo>
```

Execuția constă din instalarea și lansarea componentelor

```
start file:. . ./org.apache.felix.ipojo-*.jar
start file:. . ./org.apache.felix.shell-*.jar
start file:. . ./org.apache.felix.ipojo.arch-*.jar

start file:. . ./icmmdc.jar
start file:. . ./cmmdcservice.jar
start file:. . ./clientcmmdc.jar
```

6.6 Serviciul OSGi de jurnalizare

Interfața de programare (API) OSGi conține pentru jurnalizare o serie de interfețe a căror implementare oferă servicii de jurnalizare.

În cazul mediului *apache-felix* serviciile de jurnalizare trebuie instalate, resursa fiind

org.apache.felix.log-.jar*

Interfața `org.osgi.service.log.LogService`

Clasa care implementează această interfață asigură jurnalizarea. Mesajele sunt reținute de mediul OSGi.

Sunt declarate nivelele `LogService.LOG_ERROR`, `LogService.LOG_WARNING`, `LogService.LOG_INFO`, `LogService.LOG_DEBUG`.

Metode

- `void log(int level, String message)`

Interfața `org.osgi.service.log.LogReaderService`

Are ca scop preluarea mesajelor de jurnalizare. Uzual aceste mesaje sunt prelucrate de un *ascultător*, un obiect care implementează interfața `LogListener`

Metode

- `void addLogListener(LogListener listener)`

Interfața `org.osgi.service.log.LogListener`

Metode

- `void logged(LogEntry entry)`

Instalarea tuturor serviciilor se obține cu o componenta OSGi având activatorul

```

1 import org.osgi.framework.BundleActivator;
2 import org.osgi.framework.BundleContext;
3 import org.osgi.framework.ServiceReference;
4 import org.osgi.service.log.LogService;
5 import org.osgi.service.log.LogReaderService;
6 import org.osgi.service.log.LogListener;
7 import org.osgi.service.log.LogEntry;

9 public class Activator implements BundleActivator{

11     public void start(BundleContext context) throws Exception {
12         ServiceReference ref =
13             context.getServiceReference(LogService.class.getName());
14         LogService logService=null;
15         if (ref != null){
16             logService = (LogService)context.getService(ref);
17         }
18         ref = context.getServiceReference(LogReaderService.class.getName());
19         if (ref != null){
20             LogReaderService reader = (LogReaderService) context.getService(ref);
21             reader.addLogListener(new LogWriter());
22         }
23         logService.log(LogService.LOG_INFO, "Pornirea serviciului de jurnalizare");
24     }

26     public void stop(BundleContext context) throws Exception{}

28     class LogWriter implements LogListener{
29         public void logged(LogEntry entry){
30             System.out.println(entry.getMessage());
31         }
32     }
33 }

```

împreună cu fișierul manifest

```

1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: LogService
3 Bundle-Name: LogService
4 Bundle-Version: 1.0.0
5 Bundle-Activator: Activator
6 Import-Package: org.osgi.framework,org.osgi.service.log

```

6.7 *Apache-karaf*

Apache-karaf este un mediu OSGi, care integrează o serie de funcționalități OSGi (blueprint, http, etc.). Folosim denumirea simplificată *Karaf*, pentru

mediul de lucru.

Instalarea produsului constă în dezarhivarea fișierului descărcat.
Lansarea mediului este

```
set JAVA_HOME=. . .
set KARAF_HOME=. . .
del %KARAF_HOME%\lock
del %KARAF_HOME%\instances\*
rmdir %KARAF_HOME%\instances
%KARAF_HOME%\bin\karaf.bat clean
```

În catalogul %KARAF_HOME% se generează catalogul *instances* iar în fereastra DOS va apare prompt-ul **karaf@root>**.

Oprirea se obține apăsând tastele CTRL+D.

Karaf posedă o consolă DOS dar și o consolă Web. Consola Web trebuie instalată

```
feature:install webconsole
```

Consola Web se apelează dintr-un navigator prin

```
http://localhost:8181/system/console
```

Pentru a schimba portul se crează în prealabil fișierul `etc\org.ops4j.pax.web.cfg` cu conținutul

```
org.osgi.service.http.port=8080
```

Instalarea componentelor OSGi se poate face

- copiindu-le în catalogul
%KARAF_HOME%\deploy
- în mod obișnuit, prin comanda `install file:...`

În acest caz fișierul `MANIFEST.mf` trebuie să conțină attributele

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: numeComponentaOSGi
```

Comenzi OSGi uzuale: `start n`, `stop n`, `install file:...`, `uninstall n`, `bundle:list`, `help`.

Karaf permite executarea componentelor OSGi programate imperativ și descriptiv prin

- *blueprint* dar pentru care trebuie instalate doar
 - `org.apache.aries.blueprint-*.jar`,
 - componenta OSGi a aplicației propriu-zise.
- *Declarative Service* caz în care trebuie declarat/instalat
`feature:install scr`

Întrebări recapitulative

1. Enumerați modurile de programare a unei aplicații *OSGi*.
2. Enumerați tehnicile de programare declarativă.
3. Enumerați elementele care trebuie adăugate pentru integrarea unei aplicații într-un mediu *OSGi*.

Probleme

- bnd in linie de comanda
- bnd in ant

Capitolul 7

OSGi distribuit

Integrarea unei aplicații Web într-o platformă *OSGi* necesită o abordare specifică. Integrată într-o platformă *OSGi*, aplicația Web nu mai este desfășurată nemijlocit în serverul Web, dar apelurile se vor adresa în continuare serverului Web. În consecință, este nevoie de o punte între platforma *OSGi* și serverul Web, care eventual să asigure funcționalități suplimentare.

7.1 Medii OSGi pentru aplicații distribuite

Apache-Karaf

Mediul OSGi *apache-karaf* utilizează serverul Web incorporat *Jetty*. Se va instala suportul pentru protocolul `http`

```
feature:install http
```

Apelarea aplicației servlet va fi `http://host:port/fișier.html`

Glassfish

Mediul *OSGi* are la bază platforma *apache-felix*. Glassfish oferă posibilități *OSGi* pentru serviciile

- *HttpService*
- *TransactionService*
- *JDBC Data Source Service*

- *JMS Resource Service*

Comenzile OSGi se apelează prin
`asadmin osgi comanda_OSGi`

Astfel pentru lansarea unui servlet se va executa comanda:
`asadmin osgi start file:. . ./arhiva.jar`

Apelarea aplicației servlet va fi `http://host:port/osgi/fișier.html`

Equinox Bridge Servlet

*Equinox Bridge Servlet*¹ este distribuit printr-o arhivă `war`, destinată a fi desfășurată într-un server Web precum *apache-tomcat* sau *jetty*. În felul acesta *Equinox Bridge Servlet* integrează platforma *OSGi equinox* într-un server Web container de servlet. *Equinox Bridge Servlet* lansează platforma *equinox*. Consola *OSGi* se obține apăsând tasta **Enter** în fereastra DOS atașată serverului Web.

Funcționarea corectă a servlet-ului *Equinox Bridge Servlet* se verifică prin apelul `http://localhost:8080/bridge/sp_test`. Bineînțeles, în acest caz s-a presupus că serverul Web în care s-a desfășurat *Equinox Bridge Servlet* este calculatorul local.

Platforma *equinox* instalată în serverul Web se poate instala, dezinstala, re-instala, porni și opri prin apelurile `http://host:port/bridge/numeApel`, unde *numeApel* are valorile, respectiv `sp_deploy`, `sp_undeploy`, `sp_redeploy`, `sp_start`, `sp_stop`.

7.2 Servlet ca modul OSGi

Prezentăm integrarea unei aplicații servlet într-o componentă *OSGi* bazată pe interfața *HttpService*. Cu foarte puține diferențe componenta *OSGi* se va putea utiliza pe platforme *OSGi* diferite *apache-karaf*, *glassfish*, *bridge*. Clasa servlet-ului rămâne nemodificată.

Integrarea servlet-ului prin HttpService

Interfața `org.osgi.service.http.HttpService` declară:
Metode

¹www.eclipse.org/equinox/server/http_in_container.php

- `void registerResources(String alias, String name, HttpContext context)` throws `NamespaceException`
- `void registerServlet(String alias, Servlet servlet, Dictionary init-params, HttpContext context)` throws `ServletException`, `NamespaceException`
- `void unregister(String alias)`

Structura componentei OSGi corespunzătoare unui servlet este Structura componentei OSGi este

```
|--> META-INF
|    |   MANIFEST.MF
|    ClasaServlet.class
|    Activator.class
|    fisier.html
```

Rămâne la latitudinea programatorului să includă sau nu pagina `html`.

Compilarea necesită accesul la unul din pachetele care implementează interfața `HttpService`:

- `org.eclipse.osgi.services_3.*.*.v*.jar`.
- `org.apache.felix.http.bundle-*.jar`

Prezentăm două moduri de programare în câte un exemplu.

Exemplul 7.2.1 Integrarea servlet-ului `HelloServlet`.

Aplicația servlet este alcătuită din clasa `HelloServlet.java` și o pagină `html` de apelare `helloname.html`.

Clasa `Activator` are codul

```
1 import org.osgi.framework.BundleActivator;
2 import org.osgi.framework.BundleContext;
3 import org.osgi.framework.ServiceReference;
4 import org.osgi.service.http.HttpService;
5
6 public class Activator implements BundleActivator{
7
8     public void start(BundleContext context) throws Exception{
9         ServiceReference sRef =
10             context.getServiceReference(HttpService.class.getName());
11         if (sRef != null){
12             HttpService service = (HttpService) context.getService(sRef);
13             service.registerServlet("/hello", new HelloServlet(), null, null);
14             service.registerResources("/helloname.html", "/index.html", null);
15         }
16     }
17
18     public void stop(BundleContext context) throws Exception{}
19 }
```

Fișierul `manifest.mf` al componentei *OSGi* este

```

1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: HelloServlet
4 Bundle-SymbolicName: HelloServlet
5 Bundle-Version: 1.0.0
6 Bundle-Activator: Activator
7 Import-Package: javax.servlet,
8     javax.servlet.http,
9     org.osgi.framework;version="1.3.0",
10    org.osgi.service.http;version="1.2.0"

```

Apelarea servlet-ului din pagina `html` este dependentă de platforma *OSGi* utilizată:

Cadrul <i>OSGi</i>	Apel (aplicație, servlet)
karaf	<code>http://host:8080/helloname.html/hello</code>
glassfish	<code>http://host:8080/osgi/helloname.html/osgi/hello</code>
Equinox Bridge Servlet	<code>http://host:port/bridge/helloname.html/bridge/hello</code>

bridge este numele de apel definit de *Equinox Bridge Servlet* în fișierul `web.xml`.

A doua variantă de programare se bazează pe utilizarea clasei `ServiceTracker` pentru înregistrarea servlet-ului.

Exemplul 7.2.2 Integrarea servlet-ului *CmmdcServlet*.

Aplicația servlet este alcătuită din clasa `CmmdcServlet.java` și o pagină `html` de apelare `cmmdc.html`.

Clasa *Activator* are codul

```

1 import org.osgi.framework.BundleActivator;
2 import org.osgi.framework.BundleContext;
3 import org.osgi.framework.ServiceReference;
4 import org.osgi.service.http.HttpService;
5 import org.osgi.util.tracker.ServiceTracker;

7 public class Activator implements BundleActivator{
8     private ServiceTracker httpServiceTracker;

10     public void start(BundleContext context) throws Exception{
11         httpServiceTracker = new HttpServiceTracker(context);
12         httpServiceTracker.open();
13     }

15     public void stop(BundleContext context) throws Exception {

```

```

16     httpServiceTracker.close();
17     httpServiceTracker = null;
18 }
19
20 private class HttpServiceTracker extends ServiceTracker{
21
22     public HttpServiceTracker(BundleContext context){
23         super(context, HttpService.class.getName(), null);
24     }
25
26     public Object addingService(ServiceReference reference){
27         HttpService httpService = (HttpService)context.getService(reference);
28         try {
29             httpService.registerResources("/cmmdc.html", "/cmmdc.html", null);
30             httpService.registerServlet("/cmmdc", new CmmdcServlet(), null, null);
31         }
32         catch (Exception e){
33             e.printStackTrace();
34         }
35         return httpService;
36     }
37
38     public void removedService(ServiceReference reference, Object service){
39         HttpService httpService = (HttpService) service;
40         httpService.unregister("/cmmdc.html");
41         httpService.unregister("/cmmdc");
42         super.removedService(reference, service);
43     }
44 }
45 }

```

Serverul Web va recunoaște apelul către fișierul *cmmdc.html* și către servlet prin numele de apel *cmmdc.html* și respectiv *cmmdc* - liniile de cod 29-30.

Fișierul *manifest.mf* al componentei *OSGi* este

```

1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: CmmdcServlet
4 Bundle-SymbolicName: CmmdcServlet
5 Bundle-Version: 1.0.0
6 Bundle-Activator: Activator
7 Bundle-Localization: plugin
8 Import-Package: javax.servlet, javax.servlet.http,
9     org.osgi.framework;version="1.3.0",
10    org.osgi.service.http;version="1.2.0",
11    org.osgi.util.tracker;version="1.3.1"

```

Varianta descriptivă în Bridge

Clasa care implementează interfața **BundleActivator** este înlocuită cu un fișier de configurare *plugin.xml*. Structura componentei *OSGi* este

```

|--> META-INF
|    |    MANIFEST.MF
|    ClasaServlet.class

```

```
|   fisier.html
|   plugin.xml
```

Pentru exemplul anterior fișierul `manifest.mf` este

```
1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: CmmdcServlet PlugIn
4 Bundle-SymbolicName: CmmdcServlet PlugIn; singleton:=true
5 Bundle-Version: 1.0.0
6 Bundle-Localization: plugin
7 Import-Package: javax.servlet , javax.servlet.http
8 Require-Bundle: org.eclipse.equinox.http.registry
```

iar conținutul fișierului `plugin.xml` este

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?eclipse version="3.0"?>
3 <plugin>
4   <extension-point id="servlets"
5     name="HttpService servlets"
6     schema="schema/servlets.exsd" />
7   <extension-point id="resources"
8     name="HttpService resources"
9     schema="schema/resources.exsd" />
10  <extension-point id="httpcontexts"
11    name="HttpService httpcontexts"
12    schema="schema/httpcontexts.exsd" />
13  <extension
14    id="myServlet"
15    point="org.eclipse.equinox.http.registry.servlets">
16    <servlet
17      alias="/plugin/cmmdc"
18      class="CmmdcServlet">
19    </servlet>
20  </extension>
21  <extension
22    id="myResource"
23    point="org.eclipse.equinox.http.registry.resources">
24    <resource
25      alias="/plugin/cmmdc.html"
26      base-name="/cmmdc.html"
27    />
28  </extension>
29 </plugin>
```

Se va proceda la fel ca în varianta anterioară cu observația că adresele de apelare, definite de atributul `alias`, vor fi

```
http://host:port/bridge/plugin/cmmdc.html
http://host:port/bridge/plugin/cmmdc
```

7.3 Servicii JAXRS cu jersey

Mediul OSGi utilizat este *apache-karaf* în care se instalează

```
feature:install http
```

Suportul JAXRS este asigurat de *jersey*.

Transmisia datelor cererii se face prin

- Annotarea `PathParam`
- Componente Java cu XML

În mediul OSGi trebuie instalate:

Resursa	Info
<code>javax.servlet-*.jar</code>	eclipse plugin
<code>jersey-all-*</code>	<code>com.eclipsesource.jaxrs</code>
<code>publisher-*.jar</code>	<code>com.eclipsesource.jaxrs</code>

Clasele Java ale aplicației RESTful se preiau fără modificare și li se atașează

- Clasa activatorului

```
package resources;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

public class Activator implements BundleActivator {
    private ServiceRegistration<MyResource> registration;

    public void start(BundleContext bundleContext) throws Exception {
        registration = bundleContext.registerService(MyResource.class, new MyResource(), null);
    }

    public void stop(BundleContext bundleContext) throws Exception {
        registration.unregister();
    }
}
```

- Fișierul `manifest.xml`

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: osgi.myapp
Bundle-SymbolicName: osgi.myapp
Bundle-Version: 1.0.0
Bundle-Activator: resources.Activator
Import-Package: javax.servlet,
    javax.servlet.http,
    org.osgi.framework;version="1.3.0",
    org.osgi.service.http;version="1.2.0",
    javax.ws.rs
```

Structura arhivei `jar` a componentei OSGi va fi

```
catalogul aplicatiei
|--> META-INF
|    |    MANIFEST.MF
|--> resources
|    |    Activator.class
|    |    *.class
```

Apelarea se face prin programe client folosind referința

`http://host:port/services/numeApel/...`

unde *numeApel* este valoarea declarată prin adnotarea `Path` a resursei cerute.

Probleme

- websocket prin OSGi
- whiteboard servlet
- Declarative Service servlet
- JAXRS Aplicatia imagine
- JAXRS Transmisia datelor prin celelalte tehnici
- JAXRS Modele descriptive
- JAXRS Desfășurare și apelare din glassfish, felix

Partea V

**JAVA MANAGEMENT
EXTENSIONS**

Capitolul 8

Java Management Extensions

Java Management Extensions (JMX) face posibilă ca un obiect Java să permită gestionarea metodelor și a anumitor câmpuri de către alte obiecte. Obiectele Java care își expun astfel resursele se numesc MBean - uri și formează temelia cadrului de lucru JMX.

Există mai multe tipuri de MBean-uri:

- Standard MBean;
- Dynamic MBean;
- Open MBean;
- Model MBean;
- MXBean;

Un obiect care gestionează resursele unui MBean se numește *agent* sau *server MBean*. Agentul dispune de mijloace care interacționează cu un MBean, permițându-i:

- accesul la valorile unui câmp și la modificarea lor;
- invocarea metodelor.

În general, un agent poate fi definit ca

- autorul unei acțiuni;
- factor care provoacă acțiuni;
- reprezentant al unei instituții însărcinat cu îndeplinirea unor acțiuni.

Terminologia *server MBean* se justifică prin faptul că poate fi invocat de un program client. În această ipostază, serverul MBean are rolul unui container de MBean-uri și de gestionare și execuție a apelurilor clienților.

Structura unei aplicații JMX conține două nivele:

- componentele MBean;
- agentul (serverul MBean).

8.1 Standard MBean

8.1.1 Crearea unui Standard MBean

O componentă MBean este alcătuită dintr-o interfață și o clasă care implementează interfața satisfăcând următoarele restricții:

1. interfața are numele clasei care o implementează având în plus sufixul `MBean`;
2. Interfața și clasa care o implementează aparțin aceluiași pachet;
3. constructorii și metodele expuse trebuie să fie publice.

În continuare câmpurile și metodele destinate expunerii se vor denumi atribut, respectiv operații. Fiecărui atribut `xxx` i se atașează cel puțin una din metodele

```
public void setXxx(tip xxx){  
    this.xxx=xxx;  
}
```

și / sau

```
public tip getXxx(){  
    return xxx;  
}
```

Un atribut se precizează doar prin aceste metode, fără definirea / declararea câmpului corespunzător. Câmpul se definește în clasa ce implementează interfața MBean-ului.

Astfel, un MBean este caracterizat de

- **atribute**
care pot fi consultate (citite), modificate (scrise) sau cu ambele opțiuni.
- **operații**
- **notificări**
cu evidența modificărilor suferite de atribute.

Exemplul 8.1.1

Construim un MBean cu

- **atributele**
 - **label**
ce poate fi numai citit;
 - **cursEuro**
care poate fi consultat și modificat;
- **operațiile**
 - **public String sayHello()**
afișează un mesaj;
 - **public long cmmdc(long m, long n);**
de calcul a celui mai mare divizor comun a două numere naturale.

Interfața *IntroMBean* este

```
1 package basic;  
  
3 public interface IntroMBean {  
4     // Operatii  
5     public String sayHello();  
6     public long cmmdc(long m, long n);  
  
8     // Attribute  
9     // read-only  
10    public String getLabel();  
11    // read-write  
12    public double getCursEuro();  
13    public void setCursEuro(double cursEuro);  
14 }
```

fiind implementat de clasa *Intro*

```

1 package basic;
3 public class Intro implements IntroMBean {
5     //Atribute
6     private final String label = "Fac. Matematica si Informatica";
7     private double cursEuro = 4.50;
9     public String getLabel() {
10         return label;
11     }
13     public double getCursEuro() {
14         return cursEuro;
15     }
17     public synchronized void setCursEuro(double cursEuro) {
18         this.cursEuro = cursEuro;
19     }
21     //Operatii
22     public String sayHello() {
23         String message="Hello World !";
24         System.out.println(message);
25         return message;
26     }
28     public long cmmdc(long m, long n){. . .}
29 }

```

În acest caz nu s-a implementat posibilitatea notificărilor atributului *cursEuro*.

8.1.2 Crearea unui MBeanServer

Un agent sau MBean server implementează interfața *MBeanServer*. Un asemenea obiect se obține printr-una din metodele statice

- `static MBeanServer ManagementFactory.getPlatformMBeanServer()`

Utilitarul *jconsole* permite conectarea la serverul MBean.

- `static MBeanServer MBeanServerFactory.createMBeanServer()`

În agent se înregistrează componente MBean. Un obiect MBean este caracterizat de un *nume*, un obiect de tip *ObjectName*. Înregistrarea și / sau crearea unei componente MBean necesită definirea în prealabil a acestui *nume*. Structura unei *nume* este

domeniu : numeAtribut=valAtribut ,numeAtribut=valAtribut ...

unde

- *domeniu*

este un nume simbolic (String). Dacă domeniul este stringul vid atunci se consideră valoarea implicită *DefaultDomain*.

- attribute uzuale:
 - *type=numele MBean-ului*
 - *index=număr de identificare a MBean-ului*

Cel puțin un atribut este obligatoriu.

Clasa `ObjectName`

Constructori

- `ObjectName(String nume)`
Parametrul *nume* are structura descrisă mai sus.
- `ObjectName(String domeniu, Hashtable<String,String> tabel)`
- `ObjectName(String domeniu, String numeAtribut, String valAtribut)`

Metode

- `static ObjectName getInstance(String nume)`

Înregistrarea și utilizarea MBean-ului face apel la metodele interfeței `MBeanServer`.

`ObjectInstance`

Un obiect de tip `ObjectInstance` este folosit pentru reprezentarea ansamblului alcătuit de un obiect `ObjectName` asociat unui MBean și numele clasei corespunzătoare.

Interfața `MBeanServer`

Metode

- `ObjectInstance registerMBean(Object obj, ObjectName nume)`
Înregistrează pe platformă, instanța *obj* a unui MBean având numele *nume*.

- `void unregister(ObjectName name)`
- `ObjectInstance createMBean(String numeClasă, ObjectName nume)`
 Crează și înregistrează un MBean de clasă *numeClasă* și de nume *nume*.
- `ObjectInstance createMBean(String numeClasă, ObjectName nume, Object[] param, String[] sign)`
 În plus, șirul *param* conține parametri constructorului, de tip, respectiv *sign*.
- `String getDefaultDomain()`
- `Object invoke(ObjectName mbeanName, String operationName, Object[] param, String[] paramTip)`
 Se invocă operația *operationName* a MBean-ului *mbeanName*. Parametri necesari operației împreună cu tipurile corespunzătoare sunt dați în variabilele *param* și respectiv *paramTip*.
- `Object getAttribute(ObjectName mbeanName, String atribut)`
 Returnează valoarea atributului *atribut* a mbean-ului *mbeanName*.
- `void setAttribute(ObjectName mbeanName, Attribute atribut)`
 Fixează valoarea atributului *atribut* a MBean-ului *mbeanName*. Reținem doar constructorul clasei `Attribute` prin
`Attribute(String nume, Object valoare)`
- `MBeanInfo getMBeanInfo(ObjectName mbeanName)`
 Returnează un obiect de tip `MBeanInfo` util inspectării resurselor unui MBean.

Există mai multe șabloane de programare a înregistrării unei componente MBean.

Exemplul 8.1.2 *Se crează două componente MBean de tip `Intro` care vor fi inspectate prin intermediul utilitarului `jconsole`¹ - distribuția `jdk`.*

Utilitarul `jconsole` permite apelarea operațiilor, modificarea atributelor și semnalează notificările.

¹În acest caz, este nevoie ca variabilele de tip clasă acoperitoare `Double`, `Long` să fie înlocuite prin tipuri predefinite.


```

1 package basic;
2 import java.lang.management.ManagementFactory;
3 import javax.management.ObjectName;
4 import javax.management.MBeanServer;

6 public class Main{
7     public static void main(String [] args){
8         String domeniu="";
9         if(args.length>0) domeniu=args[0];
10        try{
11            // Serverul platformei
12            MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

14            //Varianta 1
15            // Construirea ObjectName corespunzator MBean-ului
16            ObjectName mbeanObjectName =
17                new ObjectName(domeniu+":type=Intro,index=1");

19            // Crearea MBean-ului
20            Intro mbean = new Intro();

22            // Inregistrarea MBean-ului
23            mbs.registerMBean(mbean, mbeanObjectName);

25            //Varianta 2
26            mbeanObjectName=new ObjectName(domeniu+":type=Intro,index=2");
27            mbs.createMBean("basic.Intro",mbeanObjectName);

29            // Asteptare nedefinita
30            System.out.println("Waiting forever...");
31            while(true);
32        }
33        catch(Exception e){
34            System.err.println("Exception : "+e.getMessage());
35        }
36    }
37 }

```

Executarea aplicației revine la

1. Într-o fereastră DOS se lansează agentul *Main* prin
`java -Dcom.sun.manager.jmxremote basic.Main`
2. Într-o altă fereastră DOS se lansează
`jconsole`
 pornind utilitarul *jconsole*.
3. În fereastra de dialog *jconsole:Connect to Agent* se dă clic pe **Connect**.
4. În panoul *Tree* găsim domeniul dat. Prin clic pe domeniu apar MBean-urile de indice 1 și 2.
5. Prin clic pe unul din aceste MBean-uri, în panoul central avem acces la atributele și la operațiile lor.

Rezultatul unei operații apare într-o fereastră de informare (Fig. 8.1).

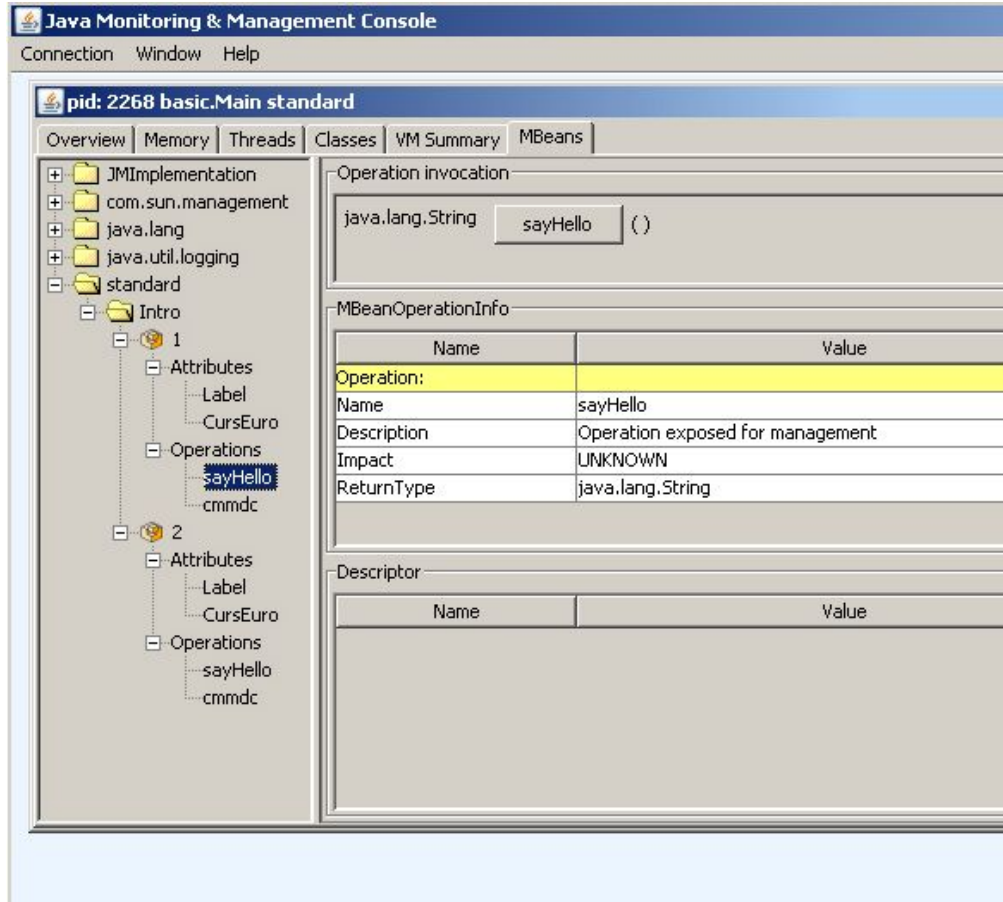


Figure 8.1: Rezultatele furnizate de *jcluster*.

8.1.3 Notificări

Pentru implementarea reținerii de către un agent MBean a notificărilor, clasa ce implementează interfața MBean-ului trebuie să extindă clasa `NotificationBroadcasterSupport`.

Modificările unui atribut se rețin într-un obiect de tip `Notification` și este transmis prin metoda

```
sendNotification(Notification notification)
```

a clasei `NotificationBroadcasterSupport`.

Suplimentar se definește metoda

```
public MBeanNotificationInfo[] getNotificationInfo()
```

în care se precizează

- tipul notificării - (constantă definită de clasa `AttributeChangeNotification`);

- clasa în care s-a generat notificarea

```
String name = AttributeChangeNotification.class.getName();
```

- o descriere a modificării.

Astfel, introducerea notificărilor presupune familiarizarea cu clasele

- **NotificationBroadcasterSupport**

Constructori

- `NotificationBroadcasterSupport()`

Metode

- `void sendNotification(Notification notificare)`

- **Notification**

Constructori

- `Notification(String type, Object source, long sequenceNumber)`
- `Notification(String type, Object source, long sequenceNumber, long timeStamp)`
- `Notification(String type, Object source, long sequenceNumber, long timeStamp, String mesaj)`
- `Notification(String type, Object source, long sequenceNumber, String mesaj)`

Metode

- `public void setUserData(Object userData)`
- `public Object getUserData()`

- **AttributeChangeNotification extends Notification**

Constructori

- `AttributeChangeNotification(Object source, long sequenceNumber, long timeStamp, String msg, String attributeName, String attributeType, Object oldValue, Object newValue)`

Câmpuri

- `static String ATTRIBUTE_CHANGE`
Semnalează schimbarea atributului

Exemplul 8.1.3 *Extinderea MBean-ului Intro cu notificarea modificărilor atributelor în jconsole.*

Implementarea notificărilor în clasa *Intro* presupune modificarea metodei *setCursEuro*. Totodată se adaugă metoda *getNotificationInfo*, ce furnizează informații referitoare la modificarea apărută. Codul clasei care implementează interfața devine

```

1 package agentn;
2 import javax.management.Notification;
3 import javax.management.AttributeChangeNotification;
4 import javax.management.NotificationBroadcasterSupport;
5 import javax.management.MBeanNotificationInfo;

7 public class IntroN extends NotificationBroadcasterSupport
8     implements IntroNMBean{
9     private long sequenceNumber=1;

11     //Attribute
12     private final String label = "Fac. Matematica si Informatica";
13     private double cursEuro = 4.50;

15     public String getLabel() { . . . }

17     public double getCursEuro() { . . . }

19     public synchronized void setCursEuro(double cursEuro) {
20         double oldCursEuro=this.cursEuro;
21         this.cursEuro = cursEuro;
22         //System.out.println("Curs de schimb euro : " + euro+" ron.");
23         Notification n=new AttributeChangeNotification(
24             this,
25             sequenceNumber++,
26             System.currentTimeMillis(),
27             "Schimbarea cursului Euro",
28             "cursEuro",
29             "double",
30             oldCursEuro,
31             cursEuro);

```

```

32     sendNotification(n);
33 }

35 public MBeanNotificationInfo [] getNotificationInfo () {
36     String [] types = new String [] {
37         AttributeChangeNotification.ATTRIBUTE_CHANGE
38     };
39     String name = AttributeChangeNotification.class.getName();
40     String description = "An attribute of this MBean has changed";
41     MBeanNotificationInfo info =
42         new MBeanNotificationInfo(types, name, description);
43     return new MBeanNotificationInfo [] {info};
44 }

46 // Operatii
47 public String sayHello () { . . . }

49 public long cmmdc(long m, long n) { . . . }
50 }

```

În *jconsole* pentru notificare, în prealabil este nevoie de subscriere.

8.1.4 Agent MBean

În exemplele anterioare resursele unui MBean au fost utilizate prin *jconsole*. Valorificarea resurselor unui MBean *Intro*, dintr-un MBeanServer (agent) se programează prin

Exemplul 8.1.4

```

1 package agent;
2 import java.io.IOException;
3 import javax.management.ObjectName;
4 import javax.management.MBeanServer;
5 import javax.management.MBeanServerFactory;
6 import javax.management.Attribute;
7 import java.util.Scanner;

9 public class Agent{
10     public static void main(String [] args) {
11         try {
12             // Crearea Agentului - MBeanServer
13             MBeanServer mbs = MBeanServerFactory.createMBeanServer();

15             // Crearea unui MBean
16             String domain = mbs.getDefaultDomain();
17             String className="agent.Intro";
18             String sObjectName=domain+":type="+className;
19             ObjectName mbeanObjectName = new ObjectName(sObjectName);
20             mbs.createMBean(className, mbeanObjectName);

22             // Utilizarea MBean-ului
23             // Apelarea operatiilor
24             String operatia="sayHello";

```

```

25      mbs.invoke(mbeanObjectName, operatia, null, null);

26
27      operatia="cmmdc";
28      System.out.println("Cmmdc al numerelor:");
29      Scanner scanner=new Scanner(System.in);
30      System.out.println("Primul numar:");
31      long m=scanner.nextLong();
32      System.out.println("Al doilea numar:");
33      long n=scanner.nextLong();
34      Object[] param={m,n};
35      String[] sign={"long", "long"};
36      Long r=(Long)mbs.invoke(mbeanObjectName, operatia, param, sign);
37      System.out.println("cmmdc="+r.toString());

38
39      // Utilizarea Atributelor
40      String label=(String)mbs.getAttribute(mbeanObjectName,"Label");
41      System.out.println("Valoarea atributului label : "+label);

42
43      System.out.println("Introduceti cursul euro");
44      double cursEuro=scanner.nextDouble();
45      Attribute curs=new Attribute("CursEuro", cursEuro);
46      mbs.setAttribute(mbeanObjectName, curs);
47      Double euro=(Double)mbs.getAttribute(mbeanObjectName,"CursEuro");
48      System.out.println("Valoarea atributului cursEuro : "+euro);
49  }
50  catch (Exception e){
51      System.out.println(e.getMessage());
52  }
53  }
54  }

```

8.1.5 Invocarea la distanță

Din punct de vedere al programării distribuite, cazul interesant este cel în care clasa ce implementează MBean-ul și clientul (agentul MBean) care o utilizează se află pe calculatoare diferite.

În acest caz:

- Este nevoie de o clasa *server* (agent) al cărui rol este
 - Instanțierea unui MBeanServer
 - Instanțierea unui server de conexiune, obiect de tip **MBeanServerConnection**. Bazat pe tehnologia RMI sau RMI-IIOP, acest obiect gestionează comunicatia dintre un client și serverul MBeanServer. Serverul de conexiune și serverul MBeanServer aparțin aceleiași clase.

În cazul utilizării tehnologiei RMI *rmiregistry* și serverul MBeanServer trebuie să ruleze pe același calculator.

Utilizând tehnologia RMI-IIOP, *orbd* și serverul MBeanServer pot rula pe calculatoare distincte.

– Lansarea în execuție a serverului de conexiune.

- Clientul dispune de interfața MBean-ului.

Interfața `MBeanServerConnection` este implementată de clasa `MBeanServer`. Șablonul de programare pentru instanțierea obiectului de tip `MBeanServerConnection` și lansarea sa în execuție poate fi:

```
String url="service:jmx:rmi:///jndi/rmi://host:port/numeServer"
// String url="service:jmx:iiop:///jndi/iiop://host:port/numeServer";

// Crearea unui server-conector
JMXServiceURL url = new JMXServiceURL(url);
JMXConnectorServer cs =
    JMXConnectorServerFactory.newJMXConnectorServer(url,null,MBeanServer);
//
// Pornirea server-conectorului
cs.start();
System.out.println("Press Enter to finish !");
System.in.read();
cs.stop();
```

Serverul de conexiune are un nume, *numeServer* care trebuie cunoscut de către client.

Lansarea în execuție a *serverului* este precedată de pornirea registrului *rmiregistry*, respectiv *orbd*.

Exemplul 8.1.5

```
1 package server;
2 import javax.management.MBeanServer;
3 import javax.management.MBeanServerFactory;
4 import javax.management.remote.JMXServiceURL;
5 import javax.management.remote.JMXConnectorServer;
6 import javax.management.remote.JMXConnectorServerFactory;
7
8 public class MBServer{
9     public static void main(String[] args) {
10         String host="localhost";
11         String port="1099";
12         //String port="1050";
13         if(args.length==0){
14             System.out.println("The name of the server is required");
15             System.exit(0);
16         }
17         if(args.length>=2)
18             host=args[1];
19         if(args.length>=3)
20             port=args[2];
21         try {
22             // Crearea MBeanServer
23             MBeanServer mbs = MBeanServerFactory.createMBeanServer();
```

```

25 // Crearea unui server-conector
26 String surl="service:jmx:rmi:///jndi/rmi://" +
27     host+": "+port+"/" +args[0];
28 // String surl="service:jmx:iiop:///jndi/iiop://" +
29 //     host+": "+port+"/" +args[0];
30 JMXServiceURL url=new JMXServiceURL(surl);
31 JMXConnectorServer cs=
32     JMXConnectorServerFactory.newJMXConnectorServer(url, null, mbs);

34 // Pornirea server-conectorului
35 cs.start();
36 System.out.println("Press Enter to finish !");
37 System.in.read();
38 cs.stop();
39 }
40 catch (Exception e){
41     System.out.println(e.getMessage());
42     e.printStackTrace();
43 }
44 }
45 }

```

Primul argument care trebuie furnizat programului anterior (*args[0]*) este numele serverului.

Clientul, la rândul lui, este nevoit să creeze un *conector* către *server* (agent), prin intermediul căruia obține un obiect ce implementează interfața *MBeanServerConnection*. Prin acest obiect, clientul va putea crea MBean-uri - în agent - și le va putea utiliza resursele.

Șablonul de programare pentru crearea obiectului *MBeanServerConnection* poate fi

```

JMXServiceURL url =
    new JMXServiceURL("service:jmx:rmi:///jndi/rmi://host:port/numeServer");
JMXConnector jmx = JMXConnectorFactory.connect(url, null);
// Obținerea obiectului de tip MBeanServerConnection
MBeanServerConnection cs = jmx.getMBeanServerConnection();

```

Resursele unui MBean se poate invoca prin

- metoda *invoke(mbeanObjectName, operationName, param, sign)* a interfeței *MBeanServerConnection*.
- crearea unui reprezentant local al MBean-ului prin metoda statică

```

mbeanClass proxy =
    (mbeanClass)MBeanServerInvocationHandler.newProxyInstance(
        mbeanServerConnection, mbeanObjectName, mbeanClass.class, true);

```

În final, MBean-ul se șterge din serverul de conexiune, prin *cs.unregisterMBean(mbeanObjectName)*.

Exemplul 8.1.6 *Client ce utilizează un MBean Intro, definit în Exemplul 8.1.1*

```

1 package client;
2 import java.util.Scanner;
3 import javax.management.MBeanServerConnection;
4 import javax.management.ObjectName;
5 import javax.management.Attribute;
6 import javax.management.MBeanServerInvocationHandler;
7 import javax.management.remote.JMXServiceURL;
8 import javax.management.remote.JMXConnector;
9 import javax.management.remote.JMXConnectorFactory;

11 public class Client {
12     public static void main(String[] args) {
13         String host="localhost";
14         String port="1099";
15         if(args.length<=1){
16             System.out.println("The server and domain names are required");
17             System.exit(0);
18         }
19         String serverName=args[0];
20         String domain=args[1];
21         if(args.length>=3) host=args[2];
22         if(args.length>=4) port=args[4];
23         Scanner scanner=new Scanner(System.in);
24         try {
25             //Crearea unui conector si a obiectului de tip MBeanServerConnection
26             String url="service:jmx:rmi:///jndi/rmi://" +
27                 host+":"+port+"/"+args[0];
28             // String url="service:jmx:iiop:///jndi/iiop://" +
29             //     host+":"+port+"/"+args[0];
30             JMXServiceURL url = new JMXServiceURL(url);
31             JMXConnector jmx = JMXConnectorFactory.connect(url, null);
32             MBeanServerConnection cs = jmx.getMBeanServerConnection();

33             // Domeniile agentului sunt
34             System.out.println("Domains:");
35             String domains[] = cs.getDomains();
36             for (int i = 0; i < domains.length; i++) {
37                 System.out.println("\tDomain[" + i + "] = " + domains[i]);
38             }

39             // iar domeniul implicit
40             System.out.println("DefaultDomain : " +cs.getDefaultDomain());
41             System.out.println("Domain : " +domain);

42             // Crearea unui MBean Intro
43             String className="basic.Intro";
44             String sObjectName=domain+":type="+className;
45             ObjectName mbeanObjectName = new ObjectName(sObjectName);
46             cs.createMBean(className, mbeanObjectName, null, null);

47             double cursEuro;
48             long m,n;

49             // Utilizarea MBean-ului
50             // Varianta 1 de invocare prin proxy

```

```

56      System.out.println("Varianta de invocare prin proxi");
57      IntroMBean proxy=
58          (IntroMBean)MBeanServerInvocationHandler.newProxyInstance(
59              cs,
60              mbeanObjectName,
61              client.IntroMBean.class,
62              true);

64      //Utilizarea operatiilor
65      // operatia "sayHello"
66      proxy.sayHello();

68      // operatia cmmdc
69      System.out.println("Cmmdc al numerelor:");
70      System.out.println("Primul numar:");
71      m=scanner.nextLong();
72      System.out.println("Al doilea numar:");
73      n=scanner.nextLong();
74      System.out.println("Cmmdc="+proxy.cmmdc(m,n));

76      // Utilizarea atributelor
77      System.out.println("Nume: "+proxy.getLabel());
78      System.out.println("Introduceti cursul euro");
79      cursEuro=scanner.nextDouble();
80      proxy.setCursEuro(cursEuro);
81      System.out.println("Euro : "+proxy.getCursEuro());

83      // Varianta 2 de invocare prin conexiune
84      System.out.println("Varianta de invocare prin conexiune");
85      // Apelarea operatiilor
86      String operatia="sayHello";
87      cs.invoke(mbeanObjectName,operatia,null,null);
88      operatia="cmmdc";
89      System.out.println("Cmmdc al numerelor:");
90      System.out.println("Primul numar:");
91      m=scanner.nextLong();
92      System.out.println("Al doilea numar:");
93      n=scanner.nextLong();
94      Object [] param={m,n};
95      String [] sign={"long","long"};
96      Long r=(Long)cs.invoke(mbeanObjectName,operatia,param,sign);
97      System.out.println("Cmmdc="+r.toString());

99      // Utilizarea Atributelor
100     String label=(String)cs.getAttribute(mbeanObjectName,"Label");
101     System.out.println("Valoarea atributului label : "+label);

103     System.out.println("Introduceti cursul euro");
104     cursEuro=scanner.nextDouble();
105     Attribute curs=new Attribute("CursEuro", cursEuro);
106     cs.setAttribute(mbeanObjectName,curs);
107     Double newEuro=(Double)cs.getAttribute(mbeanObjectName,"CursEuro");
108     System.out.println("Valoarea atributului euro : "+newEuro);
109     cs.unregisterMBean(mbeanObjectName);
110 }
111 catch(Exception e) {
112     System.out.println(e.getMessage());
113 }
114 }

```

115 }

Inspectarea și valorificarea resurselor unei componente MBean se poate face și prin intermediul metodei

```

1  private static void getMBeanResources(MBeanInfo info){
2  System.out.println("CLASA: \t" + info.getClassName());
3  System.out.println("DESCR: \t" + info.getDescription());
4  System.out.println("ATTRIBUTE");
5  MBeanAttributeInfo[] attrInfo = info.getAttributes();
6  if (attrInfo.length > 0) {
7      for (int i = 0; i < attrInfo.length; i++) {
8          System.out.println("\tNUME: \t" + attrInfo[i].getName());
9          System.out.println("\tDESCR: \t" + attrInfo[i].getDescription());
10         System.out.println("\tTIP: \t" + attrInfo[i].getType() +
11             " READ: " + attrInfo[i].isReadable() +
12             " WRITE: " + attrInfo[i].isWritable());
13     }
14 }
15 else
16     System.out.println("\tFara atribut !");
17 System.out.println("CONSTRUCTORI");
18 MBeanConstructorInfo[] constrInfo = info.getConstructors();
19 for (int i=0; i<constrInfo.length; i++) {
20     System.out.println("\tNUME: \t" + constrInfo[i].getName());
21     System.out.println("\tDESCR: \t" + constrInfo[i].getDescription());
22     System.out.println("\tPARAM: \t" +
23         constrInfo[i].getSignature().length + " parametri");
24 }
25 System.out.println("OPERATII");
26 MBeanOperationInfo[] opInfo = info.getOperations();
27 if (opInfo.length > 0) {
28     for (int i = 0; i < opInfo.length; i++) {
29         System.out.println("\tNUME: \t" + opInfo[i].getName());
30         System.out.println("\tDESCR: \t" + opInfo[i].getDescription());
31         System.out.println("\tPARAM: \t" +
32             opInfo[i].getSignature().length + " parametri");
33     }
34 }
35 else
36     System.out.println("\tFara operatii");
37 System.out.println("NOTIFICARI");
38 MBeanNotificationInfo[] notifInfo = info.getNotifications();
39 if (notifInfo.length > 0) {
40     for (int i = 0; i < notifInfo.length; i++) {
41         System.out.println("\tNUME: " + notifInfo[i].getName());
42         System.out.println("\tDESCR: " + notifInfo[i].getDescription());
43         String notifTypes[] = notifInfo[i].getNotifTypes();
44         for (int j = 0; j < notifTypes.length; j++) {
45             System.out.println("\tTIP: " + notifTypes[j]);
46         }
47     }
48 }
49 else
50     System.out.println("\tFara notificari");
51 }

```

Această metodă poate fi inserată în oricare din programele agent sau client.

Notificarea la distanță

Notificarea la distanță presupune utilizarea unui MBean posedând această facilități. Pe partea de client trebuie implementat interfața `NotificationListener` care declară metoda

```
public void handleNotification(Notification notification, Object
                               handback)
```

Atașarea și disponibilizarea clasei ce implementează interfața `NotificationListener` se obțin prin metodele interfeței `MBeanServerConnection`:

- `void addNotificationListener(ObjectName name, NotificationListener listener, NotificationFilter filter, Object handback) throws InstanceNotFoundException, IOException`
- `void removeNotificationListener(ObjectName name, ObjectName listener) throws InstanceNotFoundException, ListenerNotFoundException, IOException`

Exemplul 8.1.7 *Folosind exemplul 8.1.6 - dar cu MBean-ul creat pentru 8.1.3 se crează un client cu notificare, care sesizează modificarea valorii atributului `cursEuro` a MBean-ului `IntroN`.*

Implementarea interfeței `NotificationListener` este

```
1 package client;
2 import javax.management.Notification;
3 import javax.management.NotificationListener;
4 import javax.management.AttributeChangeNotification;
5
6 public class ClientListener implements NotificationListener {
7     public void handleNotification(Notification notification,
8         Object handback) {
9         System.out.println("\nReceived notification: " + notification);
10        AttributeChangeNotification myNotif=
11            (AttributeChangeNotification)notification;
12        System.out.println("Curs initial : " +
13            myNotif.getOldValue().toString());
14        System.out.println("Curs curent : " +
15            myNotif.getNewValue().toString());
16    }
17 }
```

Codul clasei client este

```
1 package client;
2 import javax.management.ObjectName;
3 import javax.management.MBeanInfo;
4 import javax.management.MBeanAttributeInfo;
5 import javax.management.MBeanConstructorInfo;
```

```

6 import javax.management.MBeanOperationInfo;
7 import javax.management.MBeanNotificationInfo;
8 import javax.management.MBeanServerConnection;
9 import javax.management.remote.JMXServiceURL;
10 import javax.management.remote.JMXConnector;
11 import javax.management.remote.JMXConnectorFactory;

13 public class ClientNotif {
14     public static void main(String[] args) {
15         String host="localhost";
16         String port="1099";
17         if(args.length<=1){
18             System.out.println("The server and domain names are required");
19             System.exit(0);
20         }
21         String serverName=args[0];
22         String domain=args[1];
23         if(args.length>=3) host=args[2];
24         if(args.length>=4) port=args[3];
25         ClientNotif obj=new ClientNotif();
26         try {
27             // Crearea unui conector si a obiectului
28             // de tip MBeanServerConnection
29             String url="service:jmx:rmi:///jndi/rmi://" +
30                 host+":"+port+"/"+args[0];
31             JMXServiceURL url = new JMXServiceURL(url);
32             JMXConnector jmx = JMXConnectorFactory.connect(url, null);
33             MBeanServerConnection cs = jmx.getMBeanServerConnection();

35             // Crearea obiectului ObjectName atasat MBean-ului IntroN
36             String className="basicn.IntroN";
37             String sObjectName=domain+":type="+className;
38             ObjectName mbeanObjectName = new ObjectName(sObjectName);

40             MBeanInfo info=cs.getMBeanInfo(mbeanObjectName);
41             getMBeanResources(info);

43             // Utilizarea notificarii
44             // Crearea unui ascultator
45             ClientListener listener = new ClientListener();
46             // Activarea notificatorului
47             cs.addNotificationListener(mbeanObjectName, listener, null, obj);

49             Thread.sleep(500);
50             // Disponibilizarea ascultatorului de notificari
51             System.out.println("Press Enter to finish !");
52             try{
53                 System.in.read();
54             }
55             catch(java.io.IOException e){}
56             cs.removeNotificationListener(mbeanObjectName, listener);
57             // Disponibilizarea obiectului MBeanObjectName
58             cs.unregisterMBean(mbeanObjectName);
59         }
60         catch(Exception e) {
61             System.out.println(e.getMessage());
62             e.printStackTrace();
63         }
64     }

```

```

66     private static void getMBeanResources(MBeanInfo info){. . .}
68 }

```

Clasa *Client* care crează MBean-ul trebuie lansată înaintea clasei *ClientNotif*. Aceste două clase pot rula pe calculatoare distincte.

Exemplul 8.1.8 *O aplicație servlet întreține un cont. Acțiunile ce pot fi întreprinse sunt: depunerea unei sume, extragerea unei sume în limita sol-dului și consultarea contului. Contul este implementat ca un MBean standard. Se cere urmărirea la distanță a modificărilor suferite de cont.*

Interfața contului (a MBean-ului) este

```

1 public interface ContMBean {
2     // Attribute
3     // read-write
4     public double getCont();
5     public void setCont(double cont);
6 }

```

implementat prin

```

1 import javax.management.Notification;
2 import javax.management.AttributeChangeNotification;
3 import javax.management.NotificationBroadcasterSupport;
4 import javax.management.MBeanNotificationInfo;

6 public class Cont extends NotificationBroadcasterSupport
7     implements ContMBean{
8     private long sequenceNumber=1;
9     private double cont;

11    public synchronized double getCont() {
12        return cont;
13    }

15    public synchronized void setCont(double cont) {
16        double oldCont=this.cont;
17        this.cont=cont;
18        Notification n=new AttributeChangeNotification(
19            this,
20            sequenceNumber++,
21            System.currentTimeMillis(),
22            "Schimbarea Cont",
23            "cont",
24            "double",
25            oldCont,
26            cont);
27        sendNotification(n);
28    }

30    public MBeanNotificationInfo[] getNotificationInfo() {
31        String[] types = new String[] {
32            AttributeChangeNotification.ATTRIBUTECHANGE

```

```

33     };
34     String name = AttributeChangeNotification.class.getName();
35     String description = "An attribute of this MBean has changed";
36     MBeanNotificationInfo info =
37         new MBeanNotificationInfo(types, name, description);
38     return new MBeanNotificationInfo[] {info};
39 }
40 }

```

Codul servlet-ului este

```

1  import java.io.IOException;
2  import javax.servlet.ServletException;
3  import javax.servlet.http.HttpServlet;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6  import javax.servlet.ServletOutputStream;
7  import javax.servlet.ServletConfig;
8  import javax.servlet.annotation.WebServlet;
9  import javax.servlet.annotation.WebInitParam;
10 import javax.management.ObjectName;
11 import javax.management.Attribute;
12 import javax.management.MBeanServerConnection;
13 import javax.management.remote.JMXServiceURL;
14 import javax.management.remote.JMXConnector;
15 import javax.management.remote.JMXConnectorFactory;

17 @WebServlet(urlPatterns = "/appdepozit",
18     initParams = {
19         @WebInitParam(name = "jmxServerHost", value = "localhost")
20     }
21 )
22 public class DepozitServlet extends HttpServlet {
23     MBeanServerConnection cs=null;
24     ObjectName mbeanObjectName=null;
25     String host;
26     String port="1099";
27     String server="server";

29     public void init(ServletConfig config) {
30         try {
31             super.init(config);
32             host=config.getInitParameter("jmxServerHost");
33             String url="service:jmx:rmi:///jndi/rmi://" + host + ":" + port + "/" + server;
34             //String url="service:jmx:iiop:///jndi/iiop://" + host + ":" + port + "/" + server;
35             JMXServiceURL url = new JMXServiceURL(url);
36             JMXConnector jmx = JMXConnectorFactory.connect(url, null);
37             cs = jmx.getMBeanServerConnection();
38             String domain = cs.getDefaultDomain();
39             String className="Cont";
40             String sObjectName=domain+":type="+className;
41             mbeanObjectName = new ObjectName(sObjectName);
42             cs.createMBean(className, mbeanObjectName, null, null);
43         }
44         catch(Exception e){
45             System.out.println(e.getMessage());
46             System.exit(0);
47         }
48     }

```

```

50 public void doGet(HttpServletRequest req, HttpServletResponse res)
51     throws ServletException, IOException{
52     ServletOutputStream out=res.getOutputStream();
53     String oper=req.getParameter("oper");
54     String message="";
55     double suma=0;
56     if(!oper.equals("con")){
57         String s=req.getParameter("suma");
58         suma=Double.parseDouble(s);
59     }
60     Double objValue=null;
61     try{
62         objValue=(Double)cs.getAttribute(mbeanObjectName,"Cont");
63     }
64     catch(Exception e){
65         message="JMX-Error : "+e.getMessage();
66     }
67     double value=objValue.doubleValue();
68     double x;
69     Attribute curs=null;
70     switch(oper){
71         case "dep":
72             x=value+suma;
73             curs=new Attribute("Cont", x);
74             try{
75                 cs.setAttribute(mbeanObjectName, curs);
76                 message="S-a depus suma";
77             }
78             catch(Exception e){
79                 message="JMX-Error : "+e.getMessage();
80             }
81             break;
82         case "ext":
83             if (value>=suma){
84                 x=value-suma;
85                 curs=new Attribute("Cont", x);
86                 try{
87                     cs.setAttribute(mbeanObjectName, curs);
88                     message="S-a extras suma";
89                 }
90                 catch(Exception e){
91                     message="JMX-Error : "+e.getMessage();
92                 }
93             }
94             else{
95                 message="Cererea nu poate fi indeplinita";
96             }
97             break;
98         case "con":
99             message="Suma din cont este "+value+" unit.";
100             break;
101     }
102     res.setContentType("text/html");
103     out.println("<html>");
104     out.println("<head><title>Depozit</title></head>");
105     out.println("<body>");
106     out.println("<h1>Operatiuni Cont</h1>");
107     out.println("<p>");

```



```

108     out.println( message);
109     out.println("</p>");
110     out.println("</body></html>");
111     out.close();
112 }

114 public void doPost(HttpServletRequest req, HttpServletResponse res)
115     throws ServletException, IOException{
116     doGet(req, res);
117 }
118 }

```

Pagina de apelare a servlet-ului fiind (*index.html*)

```

1 <html>
2   <head>
3     <title> Servlet-ul Hello </title>
4   </head>
5   <body bgcolor="#aaceaa">
6     <center>
7       <h1> Pagina de &#238;ntre&#355;inere a depozitului </h1>
8       <form method="post"
9         action="http://localhost:8080/appcont/appdepozit">
10        <p>Introduce&#355;i :
11        <p>Opera&#355;ia: <select name="oper" >
12          <option value="dep">Depunere
13          <option value="ext">Extragere
14          <option value="con">Consultare
15        </select>
16        <p>
17        <input type="text" name="suma" value="0">
18        <p>
19        <input type="submit" value="Executa">
20      </form>
21    </center>
22  </body>
23 </html>

```

Clientul care urmărește de la distanță contul are codul

```

1 import javax.management.ObjectName;
2 import javax.management.MBeanInfo;
3 import javax.management.MBeanAttributeInfo;
4 import javax.management.MBeanConstructorInfo;
5 import javax.management.MBeanOperationInfo;
6 import javax.management.MBeanNotificationInfo;
7 import javax.management.MBeanServerConnection;
8 import javax.management.Notification;
9 import javax.management.NotificationListener;
10 import javax.management.AttributeChangeNotification;
11 import javax.management.remote.JMXServiceURL;
12 import javax.management.remote.JMXConnector;
13 import javax.management.remote.JMXConnectorFactory;

15 public class ClientNotif {
16     public static void main(String[] args) {
17         String host="localhost";
18         String port="1099";
19         if(args.length==0){

```

```

20     System.out.println("The Server name is required");
21     System.exit(0);
22 }
23 if (args.length >= 2)    host=args[1];
24 if (args.length >= 3)    port=args[2];
25 ClientNotif obj=new ClientNotif();
26 try {
27     // Crearea unui conector si a obiectului de
28     // tip MBeanServerConnection
29     String surl="service:jmx:rmi:///jndi/rmi://" +
30     host+":"+port+"/"+args[0];
31     JMXServiceURL url = new JMXServiceURL(surl);
32     JMXConnector jmxc = JMXConnectorFactory.connect(url, null);
33     MBeanServerConnection cs = jmxc.getMBeanServerConnection();

35     String domain = cs.getDefaultDomain();
36     System.out.println("DefaultDomain : " +domain);
37     // Crearea obiectului ObjectName atasat MBean-ului Cont
38     String className="Cont";
39     String sObjectName=domain+":type="+className;
40     ObjectName mbeanObjectName = new ObjectName(sObjectName);

42     MBeanInfo info=cs.getMBeanInfo(mbeanObjectName);
43     getMBeanResources(info);

45     // Utilizarea notificarii
46     // Crearea unui ascultator
47     ClientListener listener = new ClientListener();
48     // Activarea notificatorului
49     cs.addNotificationListener(mbeanObjectName, listener, null, obj);

51     Thread.sleep(500);
52     // Disponibilizarea ascultatorului de notificari
53     System.out.println("Press Enter to finish !");
54     try{
55         System.in.read();
56     }
57     catch(java.io.IOException e){}
58     cs.removeNotificationListener(mbeanObjectName, listener);
59     // Disponibilizarea obiectului MBeanObjectName
60     cs.unregisterMBean(mbeanObjectName);
61 }
62 catch(Exception e) {
63     System.out.println(e.getMessage());
64     e.printStackTrace();
65 }
66 }

68 private static void getMBeanResources(MBeanInfo info){. . .}
69 }

71 class ClientListener implements NotificationListener {
72     public void handleNotification(Notification notification,
73     Object handback){
74         System.out.println("\nReceived notification: " + notification);
75         AttributeChangeNotification myNotif=
76         (AttributeChangeNotification)notification;
77         System.out.println("Sold initial : " +
78         myNotif.getOldValue().toString());

```

```
79 |     System.out.println("Sold curent : " +  
80 |         myNotif.getNewValue().toString());  
81 | }  
82 | }
```

Serverul MBean este cel prezentat în Exemplul 8.1.5.

Serverul MBean este plasat în servlet (în catalogul `WEB-INF/classes` al aplicației) iar clientul care urmărește de la distanță se poate afla oriunde.

După instalarea servlet-ului se lansează pe mașina acestuia server-ul MBean. Deoarece notificarea presupune existența MBean-ului, iar acesta se instanțează prin metoda `init` a servlet-ului este nevoie de apelarea servlet-ului, depunând 0 unități. După această operație se lansează în execuție programul *ClientNotif*.

Întrebări recapitulative

1. Ce posibilitate oferă Java Management Extensions (JMX) ?
2. Care este structura unui MBean standard ?
3. Precizați conținutul unui server MBean cu agenți la distanță.
4. Precizați termenul de notificare în legătură cu MBean.

Bibliografie

- [1] ATHANASIU I., COSTINESCU B., DRĂGOI O.A., POPOVICI F.I., 1998, *Limbaajul Java. O perspectivă pragmatică*. Ed. Computer Libris Agora, Cluj-Napoca.
- [2] BOIAN F.M., BOIAN R. F., 2004, *Tehnologii fundamentale Java pentru aplicații Web*. Ed. Albastră, Cluj-Napoca.
- [3] BOIAN F.M., 2011, *Servicii Web; Modele, Platforme, Aplicații*. Ed. Albastră, Cluj-Napoca.
- [4] BURAGA S.C., 2001, *Tehnologii Web*. Ed. Matrix Rom, București.
- [5] BURAGA S. (ed), 2007, *Programarea în Web 2.0.*, Ed. Polirom, Iași.
- [6] JURCĂ I., 2000, *Programarea rețelelor de calculatoare*. Ed. de Vest, Timișoara.
- [7] ALBOAIE L., BURAGA S., 2006, *Servicii Web*. Ed. Polirom, Iași.
- [8] TANASĂ Ș., ANDREI Ș., OLARU C., 2011, *Java de la 0 la extert*. Ed. Polirom, Iași.
- [9] TANASĂ Ș., OLARU C., 2005, *Dezvoltarea aplicațiilor Web folosind Java*. Ed. Polirom, București.
- [10] * * *, Java EE Tutorials - Oracle, www.oracle.comJavaJavaEE, 2017.
- [11] * * *, The Java Tutorials, Oracle, <https://docs.oracle.com/javase/tutorial/java/index.html>, 2017.