



Tehnologii Java

Curs -

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică

Universitatea "Al. I. Cuza" Iași



Arhitecturi orientate pe servicii

Cuprins

- Introducere
- Imagine de ansamblu: SOA
- Servicii Web
- Servicii Grid
- BPEL

Introducere

Ce este un serviciu software

Serviciu = "formă de muncă prestată în folosul sau în interesul cuiva"

Componentă software care expune o anumită funcționalitate în cadrul unui sistem distribuit.

- **Cui oferă ?**
- **Cum oferă ?**

Arhitecturi orientate pe servicii

OASIS (the Organization for the Advancement of Structured Information Standards):

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

Principiile SOA



- **Incapsulare**
- **Contract**
- **Autonomie**
- **Interoperabilitate**
- **Standardizare:** descoperire, comunicare, invocare
- **Reutilizare**
- **Compunere**

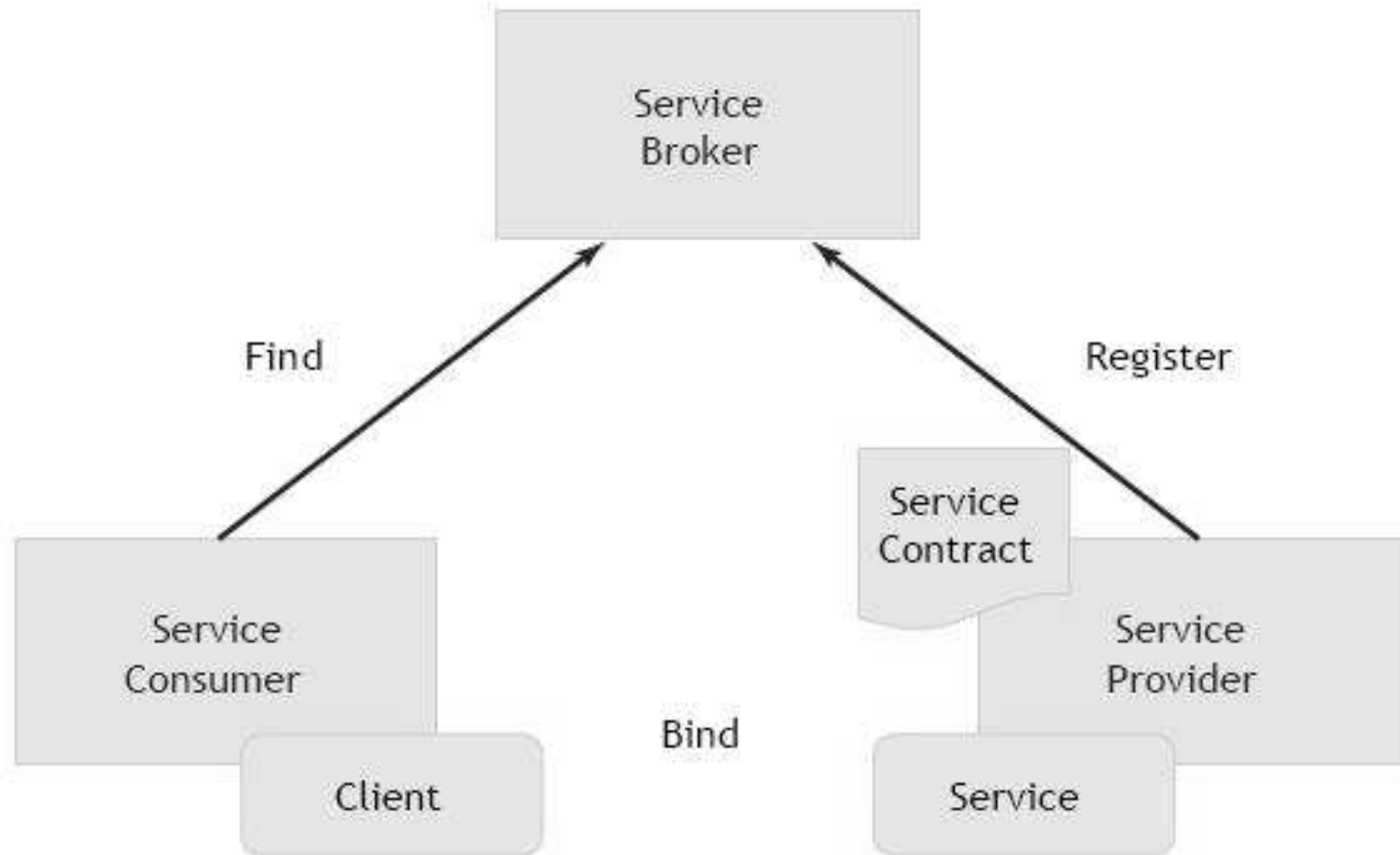


Interoperabilitate

Interoperabilitate = capacitatea unor aplicații de pe platforme diferite de a interacționa într-un mod standard. Platformă se referă la **sistem de operare** (Windows, Linux, Mac OS, etc) cât și la nivel de **programare** (Java, .NET, etc)

- Application-to-Application (A2A)
Enterprise Application Integration (EAI)
- Business-to-Business (B2B)
Business Integration (JBI)

Imagine de ansamblu SOA



Sisteme distribuite

- **CORBA:** Common Object Request Broker Architecture (Object Management Group)
- **Java RMI:** Java Remote Method Invocation (Sun)
- **DCE:** Distributed Computing Environment (Open Group)
- **DCOM:** Distributed Component Object Model (Microsoft)
- **WCF:** Windows Communication Foundation (Microsoft)
- **Web Services**

Procoale

	RMI	Corba	DCE	WS
Mecanism invocare	Java RMI	Corba RMI	RPC	JAX-RPC, .NET,...
Format date	Serializare	CDR	NDR	XML
Format mesaje	Flux	GIOP	PDU	SOAP
Protocol transfer	JRMP	IIOp	RPC CO	HTTP
Interfețe	Java	CORBA IDL	DCE IDL	WSDL
Descoperire	Java Registry	COS Naming	CDS	UDDI

JRMP = Java Remote Method Protocol. ORB = Object Request Broker. CDR = Common Data Representation. GIOP = General Inter-ORB Protocol. IIOp= Internet Inter-ORB Protocol. IDL = Interface Definition Language. COS = CORBA Object Services. RPC = Remote Procedure Call. NDR = Network Data Representation. PDU = Protocol Data Units. RPC CO = RPC Connect-Oriented protocol. IDL = Interface Definition Language. CDS = Cell Directory Service.

Avantaje - Dezavantaje

- **CORBA, Java RMI, DCE, DCOM, WCF, etc.**
 - + Folosite în industrie, maturitate, securitate, performanță
 - – Fie proprietare, fie dedicate unui limbaj, fie dificil de utilizat
- **Servicii Web**
 - + Interoperabilitate, bazate pe protocoale standard, XML, HTTP
 - – Performanță

'The Grid'

Ce este grid-ul



In 1998, Ian Foster si Carl Kesselman publicau cartea "The Grid-Blueprint for a New Computing Infrastructure":

Un grid computational este o infrastructura hardware si software care ofera acces la capabilitati de calcul de mare performanta intr-un mod sigur, consistent si ieftin

Gridul reprezinta coordonarea partajarii resurselor si a rezolvarii problemelor intr-o organizatie virtuala, dinamica, multi-institutionala



Proiecte

Globus: *Grid-ul este o infrastructura care permite folosirea intr-o maniera unitara, colaborativa a calculatoarelor, retelelor, bazelor de date si a altor intrumente stiintifice aflate in proprietatea si gestionate de diverse organizatii*

Gridbus: *Grid-ul este o forma de sistem paralel si distribuit care permite partajarea, selectare si agregarea unor resurse autonome, dispersate geografic, intr-un mod dinamic care poate fi specificat la executie in functie de disponibilitatea, performantele, costurile utilizarii lor sau de calitatea serviciilor cerute de utilizator.*

"World Wide Grid"

Viziunea: "vom asista probabil la o raspindire a unor 'utilitati software' care, intocmai ca si electricitatea sau telefonul, vor deservi locuintele personale si companiile pe tot cuprinsul tarii".

Concret: "vom asista la crearea unui sistem care..."

- să coordoneze resurse care nu sunt supuse unui control centralizat,
- folosind protocoale si interfete standard, ne-proprietare, de uz general
- pentru a furniza **servicii** de calitate si ne-triviale

Servicii Web

Ce sunt serviciile Web ?

"A Web service is a software application that conforms to the Web Service Interoperability Organization's Basic Profile 1.0."

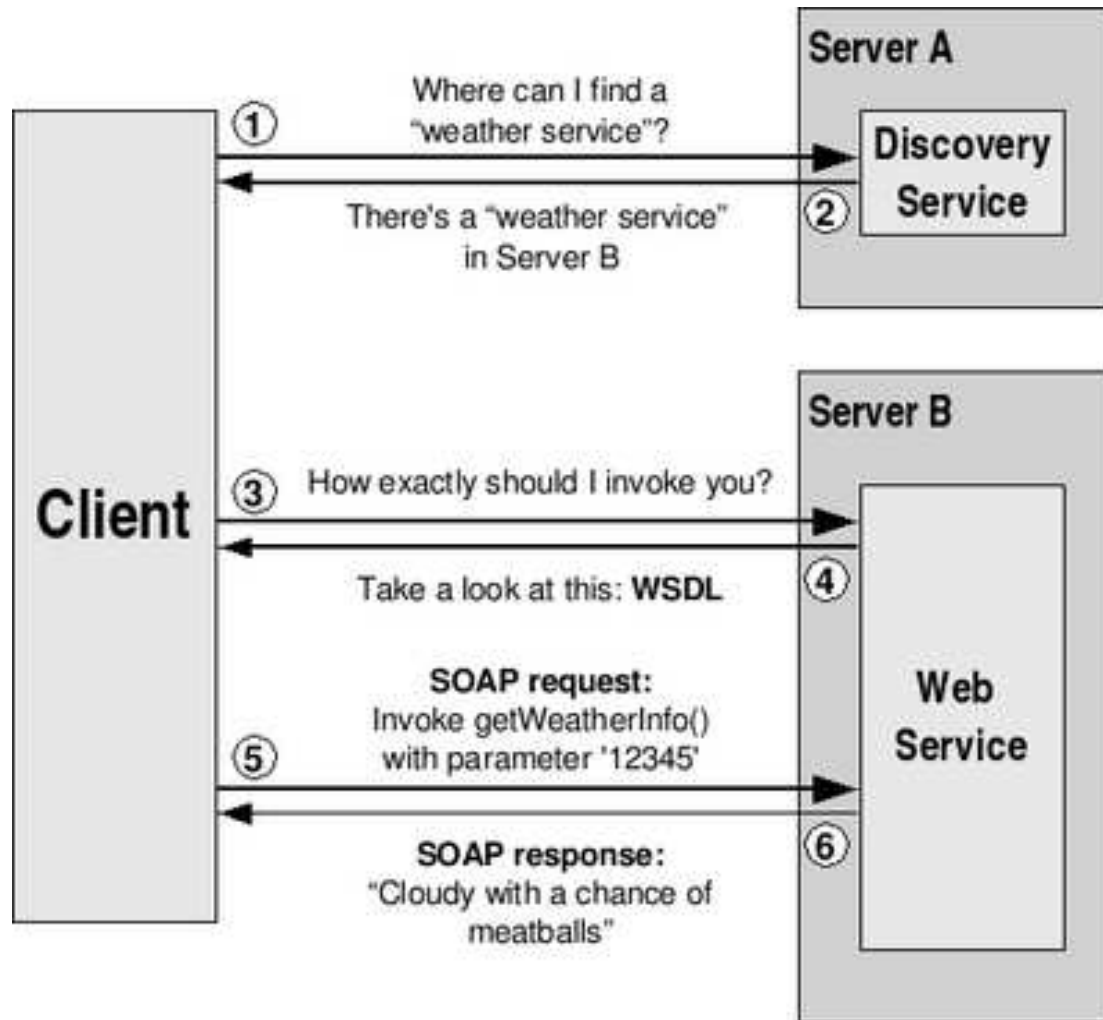
Implementare conformă principiilor SOA bazată pe protocoale standard:

- **HTTP**
- **SOAP**
- **WSDL**
- **UDDI**

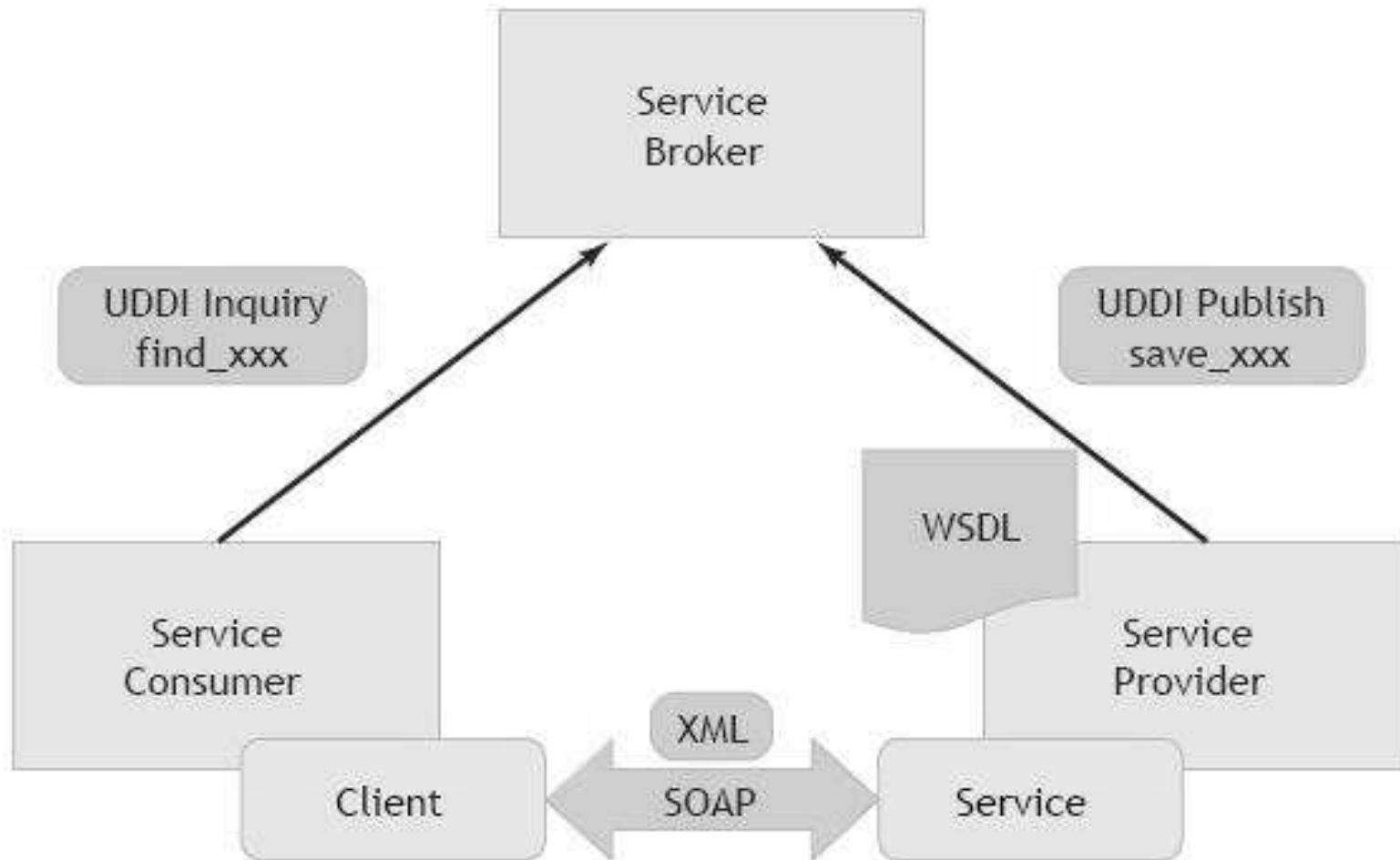
Cum funcționează WS ? (1)



Cum funcționează WS ? (2)



Cum funcționează WS ? (3)



Protocolul HTTP

- Cel mai utilizat protocol Internet
- Metoda principală de transmitere a informației pe Web
- Protocol de tip *cerere-răspuns* între clienți și servere
- Comunicarea client-server se realizează prin intermediul unei conexiuni TCP la un anumit port (80).

Avantajul folosirii HTTP în WS: Portul TCP 80 este în permanență deschis

SOAP

- *Simple Object Access Protocol*
- Format pentru trimiterea de mesaje bazat pe XML
- Protocol de comunicare între aplicații (via Internet)
- Independent de platformă sau limbaj
- Simplu și extensibil: `envelope`, `header`, `body`, `fault`
- Permite evitarea firewall-urilor
- Standard W3C

SOAP: exemplu

Mesaj cerere

```
<soap:Envelope xmlns:m="http://demo/ ...">
  <soap:Body>
    <m:sayHelloRequest>
      <name>duke</name>
    </m:sayHelloRequest>
  </soap:Body>
</soap:Envelope>
```

Mesaj răspuns

```
<soap:Envelope xmlns:m="http://demo/ ...">
  <soap:Body>
    <m:sayHelloResponse>
      <return>Hello duke !</return>
    </m:sayHelloResponse>
  </soap:Body>
</soap:Envelope>
```

WSDL



- *Web Services Description Language*
- Limbaj bazat pe XML
- Permite specificarea modului de accesare a serviciilor Web
- Permite descrierea serviciilor Web
- Structura: `portType`, `message`, `types`, `binding`
- In curs de standardizare W3C



WSDL: exemplu

```
<message name="sayHelloRequest">  
  <part name="parameters" type="xs:string"/>  
</message>
```

```
<message name="sayHelloResponse">  
  <part name="parameters" type="xs:string"/>  
</message>
```

```
<portType name="Hello">  
  <operation name="sayHello">  
    <input message="sayHelloRequest"/>  
    <output message="sayHelloResponse"/>  
  </operation>  
</portType>
```

UDDI

- *Universal Description, Discovery, and Integration*
- Sistem de regiștri bazat pe XML independent de platformă
- Facilitarea comunicării între serviciile oferite de firme:
White Pages, Yellow Pages, Green Pages
- Implementează conceptul de *service broker*
- Este interogată de mesaje SOAP ce solicită un serviciu Web
- Oferă acces către documentul descriptiv WSDL al serviciului Web
- Acceptare redusă



Crearea serviciilor Web în Java



Tipuri de servicii Web

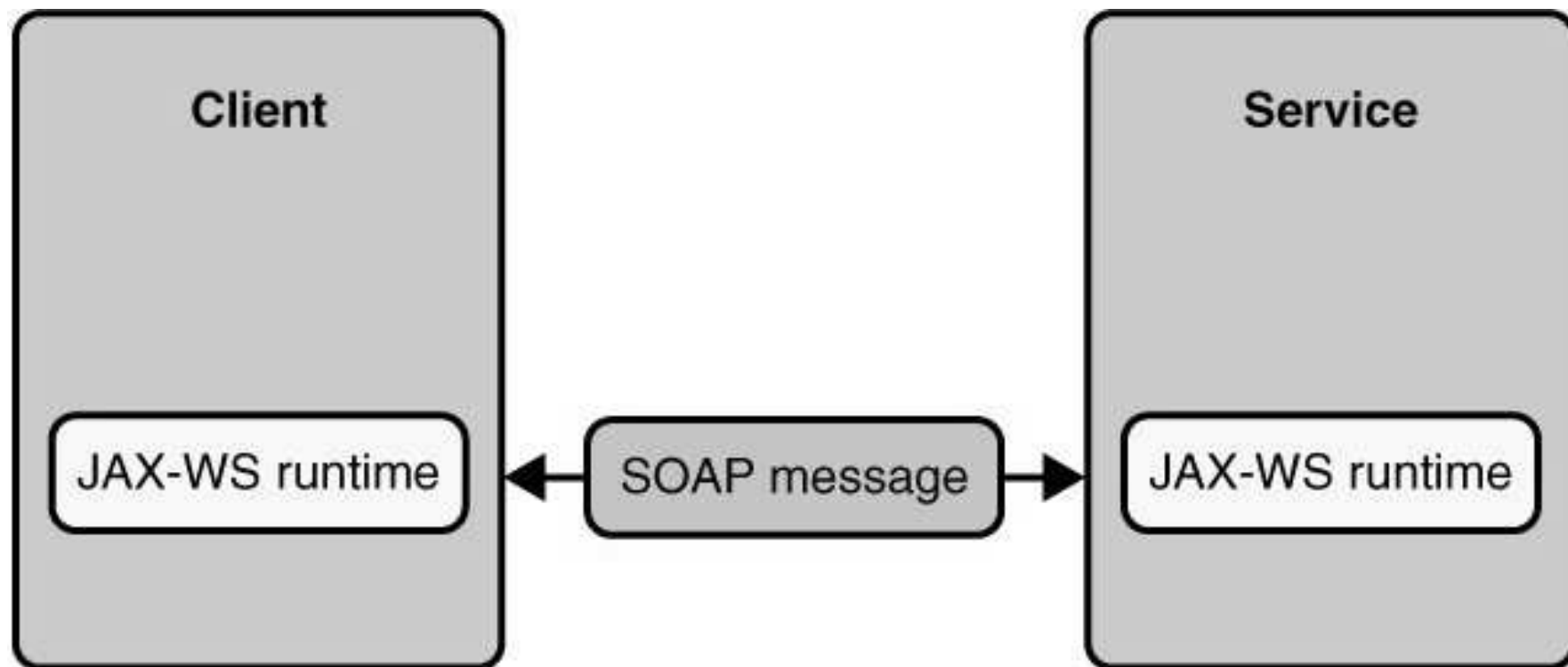
- **Clasice ("big")** bazate pe SOAP, WSDL, etc.
JAX-WS - The Java API for XML Web Services
QoS
- **RESTful** - Representational State Transfer (REST)
JAX-RS - Project Jersey
simplitate



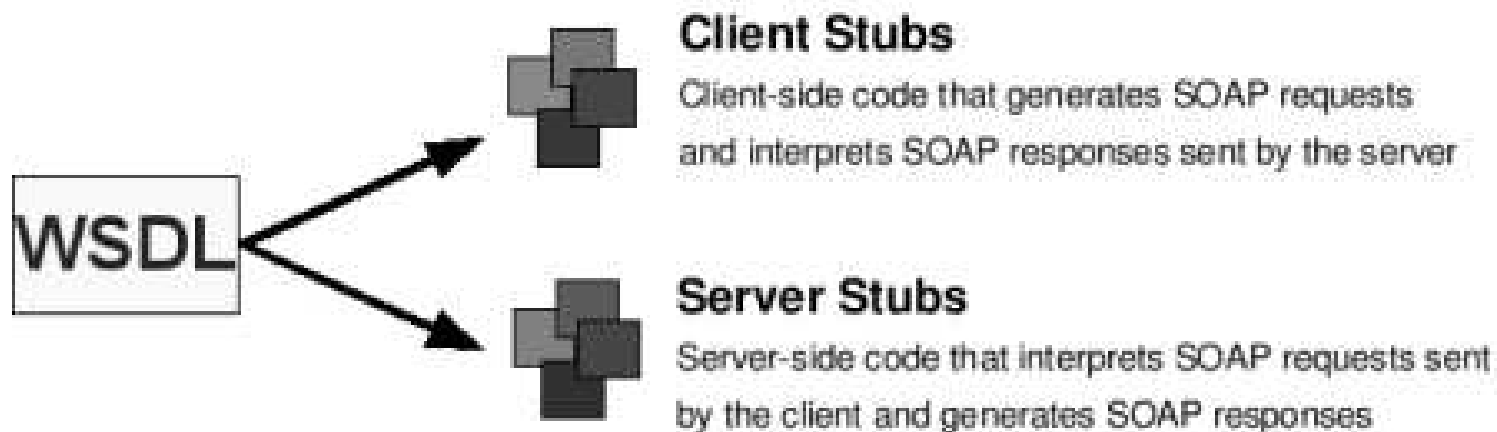
Crearea unui serviciu Web cu JAX-WS



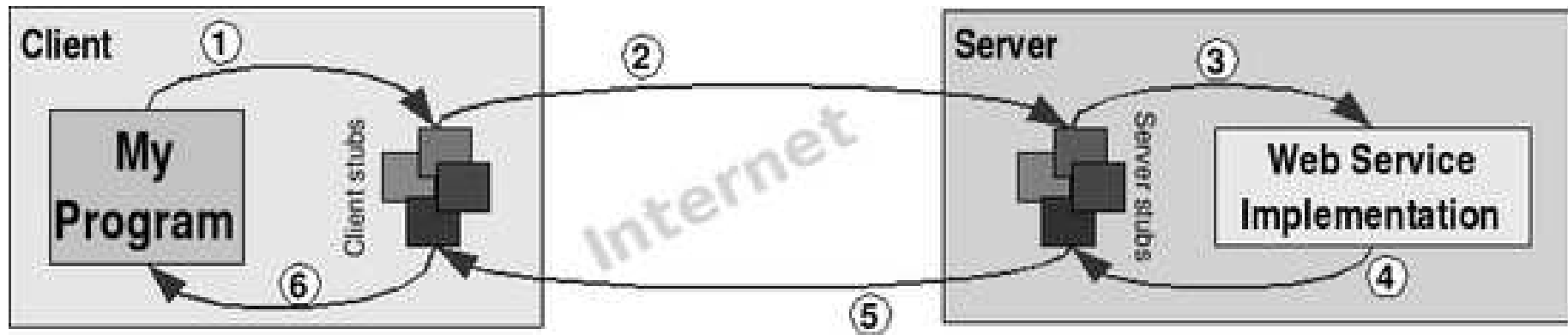
Imagine de ansamblu



Componentele *stub*



Remote Proxy



Crearea serviciului

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

@WebService(serviceName="Greeting")
public class Hello {

    @WebMethod(operationName="sayHi")
    public String operation(@WebParam(name = "name") String param) {
        return "Hi " + param + " !";
    }

    @WebMethod
    public String sayHello(String name) {
        return "Hello " + name + " !";
    }
}
```

Testarea serviciului Web



`http://localhost:8080/HelloApp/Greeting?Tester`



Crearea unui client (1)

Generarea de artefacte

```
@WebServiceClient(name = "Greeting", targetNamespace = "http://demo/",
    wsdlLocation = "http://localhost:8080/HelloApp/Greeting?WSDL")
public class Greeting extends Service {
    // cod generat de JAX-WS
    @WebEndpoint(name = "HelloPort")
    public Hello getHelloPort() {
        return ...;
    }
}
```

Utilizarea serviciului

```
public class HelloClient {
    public static void main(String[] args) throws Exception {
        Greeting service = new Greeting();
        Hello helloPort = helloService.getHelloPort();
        System.out.println(helloPort.sayHello("duke"));
    }
}
```

Crearea unui client (2)

```
@WebServlet(name="HelloServlet", urlPatterns={"/HelloServlet"})
public class HelloServlet extends HttpServlet {
    //-----
    @WebServiceRef(wsdlLocation =
        "WEB-INF/wsdl/localhost_8080/HelloApp/Greeting.wsdl")
    private Greeting service;
    //-----

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) {
        ...
        out.println("<p>" + sayHello("world") + "</p>");
        ...
    }
    private String sayHello(java.lang.String arg0) {
        helloservice.endpoint.Hello port = service.getHelloPort();
        return port.sayHello(arg0);
    }
}
```

Message Handlers

Sunt **interceptori** dedicați mesajelor prin care comunică serviciile web.

- *Protocol Handlers* - depinde de protocol
`SOAPHandler<SOAPMessageContext>`
pot accesa orice parte a mesajului: header + payload
- *Logical Handlers* - nu depind de protocol
`LogicalHandler<LogicalMessageContext>`
pot accesa doar continutul: payload
- `handleMessage`, `handleFault`

Scrierea unui *SOAPHandler* (1)

```
public class SOAPLoggingHandler implements
    SOAPHandler<SOAPMessageContext> {
    ...
    public boolean handleMessage(SOAPMessageContext smc) {
        logToSystemOut(smc);
        return true;
    }

    public boolean handleFault(SOAPMessageContext smc) {
        logToSystemOut(smc);
        return true;
    }

    public void close(MessageContext messageContext) {
    }
    ...
}
```

Scrierea unui *SOAPHandler* (2)



```
private void logToSystemOut(SOAPMessageContext smc) {
    Boolean outboundProperty = (Boolean)
        smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

    if (outboundProperty.booleanValue()) {
        out.println("\nOutbound message:");
    } else {
        out.println("\nInbound message:");
    }

    SOAPMessage message = smc.getMessage();
    try {
        message.writeTo(out);
        out.println("");
    } catch (Exception e) {
        out.println("Exception in handler: " + e);
    }
}
```



Scrierea unui *LogicalHandler* (1)

```
public class LogicalLoggingHandler implements
    LogicalHandler<LogicalMessageContext> {
    ...

    public boolean handleMessage
        (LogicalMessageContext context) {
        return processMessage(context);
    }

    public boolean handleFault
        (LogicalMessageContext context) {
        return processMessage(context);
    }

    public void close(MessageContext context) {
        // Clean up Resources
    }
}
```

Scrierea unui *LogicalHandler* (2)

```
private boolean processMessage
(LogicalMessageContext context) {
    Boolean outboundProperty = (Boolean)
        context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

    if (outboundProperty) {
        out.println("\nOutbound message:");
    } else {
        out.println("\nInbound message:");
    }
    LogicalMessage lm = context.getMessage();
    Source payload = lm.getPayload();

    // Process Payload Source
    printSource(payload);
    return true;
}
```

Utilizarea interceptorilor

Definirea unui lanț de filtrare: MyMessageHandler.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-name>logHandler</handler-name>
      <handler-class>demo.MyLogMessageHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

Adnotarea clasei ce descrie serviciul

```
@WebService
@HandlerChain(file = "MyMessageHandler.xml")
@WebService(serviceName="Greeting")
public class Hello {
    ...
}
```

Comunicare sincronă/asincronă

- **Sincronă:** clientul apelează un serviciu Web și așteaptă răspunsul
- **Asincronă:** clientul apelează un serviciu Web dar nu așteaptă răspunsul
 - **polling model** - clientul cere rezultatul atunci când are nevoie de el
`javax.xml.ws.Response extends Future<T>`
 - **callback model** - clientul înregistrează un handler care va fi notificat la apariția răspunsului
`avax.xml.ws.AsyncHandler`

Servicii Web *RESTful*

REST

- **REpresentational State Transfer**
- 2000, Roy Fielding
- Stil arhitectural de concepere a sistemelor software distribuite
- WWW = Exemplu de sistem care se conformează principiilor REST
- Concepte:
 - client-server, cerere-răspuns
 - identificator-resursă-reprezentare
 - operații CRUD pe resurse

Caracteristicile REST

- Client-Server
- Stateless
- Cache
- Uniform interface
- Named resources
- Interconnected resource representations
- Layered components

Servicii Web RESTful

Serviciu Web RESTfull = implementare HTTP a unui serviciu Web ce se conformează cu principiile REST. Uzual, reprezintă **o mulțime de resurse**. Este definit prin:

- un identificator URI pentru serviciu:
`http://example.com/resources`
- tipul MIME oferit de serviciu: Text, JSON, XML, YAML, etc
- mulțimea de operații oferite: HTTP POST, GET, PUT, DELETE

Exemplu

```
@Path("/hello/{username}")
public class HelloResource {

    @Context
    private UriInfo context;

    @GET
    @Produces("text/plain")
    public String getText(@PathParam("username") String username) {
        return "Hello " + username;
    }

    @PUT
    @Consumes("text/plain")
    public void putText(String content) { }
}
```

Crearea si testarea serviciului

```
http://localhost:8080/HelloApp/resources/hello/world  
--> Hello world
```

```
http://localhost:8080/HelloApp/resources/hello/duke  
--> Hello duke
```

"Restful WS from entity classes"

```
@Path("/users")
public class Users {
    private static EntityManagerFactory factory;
    private static String userName = null;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String userList() {
        factory = Persistence.createEntityManagerFactory("MyPU");
        EntityManager em = factory.createEntityManager();
        Query q = em.createQuery("SELECT u FROM User u");
        List<User> userList = q.getResultList();
        StringBuilder response = new StringBuilder();
        for (User user : userList) {
            response.append(user.getName() + " ");
        }
        return response.toString();
    }
}
```

Servicii Grid

WS Resource Framework



Incepand cu 2003, domeniile "calcul pe grid" si "servicii web" au inceput un proces de convergenta, scopul fiind accesarea grid-ului ca un serviciu web (Grid Service).

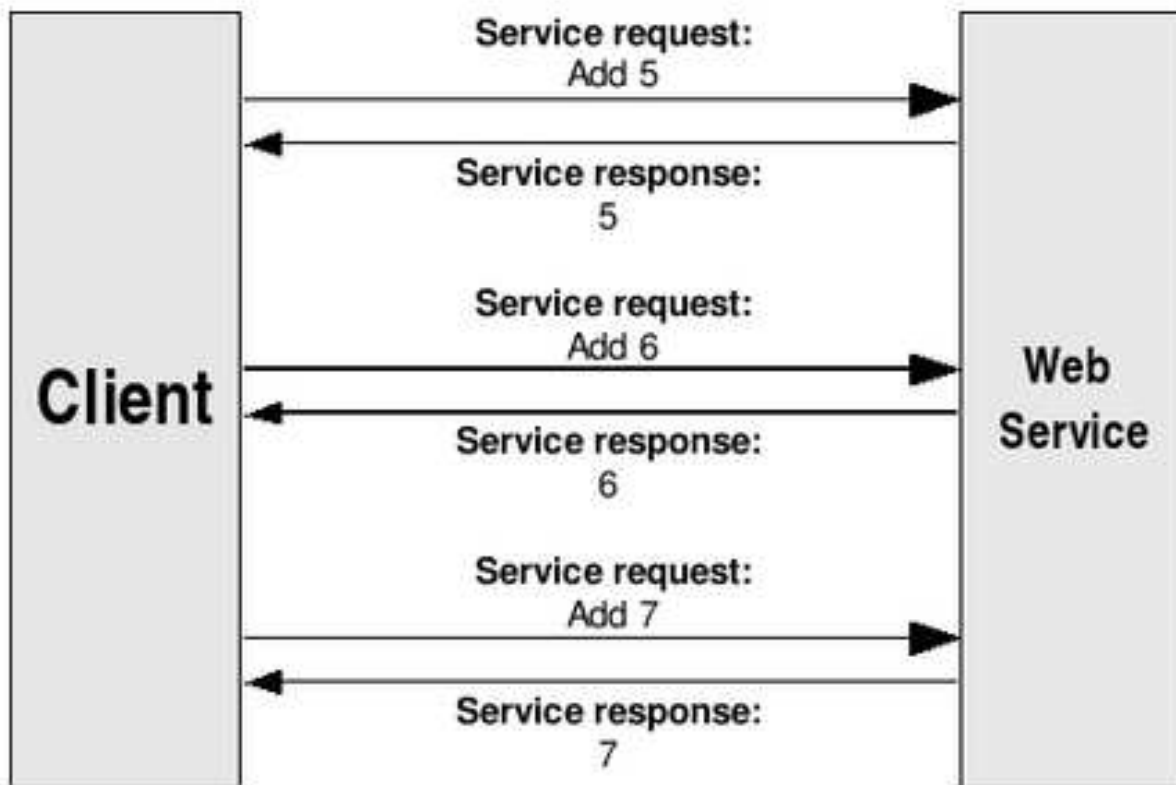
**Open Grid Services Architecture (OGSA) →
Web Service Resource Framework (WSRF)**

isi propune sa defineasca o serie de protocoale si interfete care sa ofere un cadru de lucru precis pentru toate sistemele si aplicatiile ce utilizeaza grid-ul.



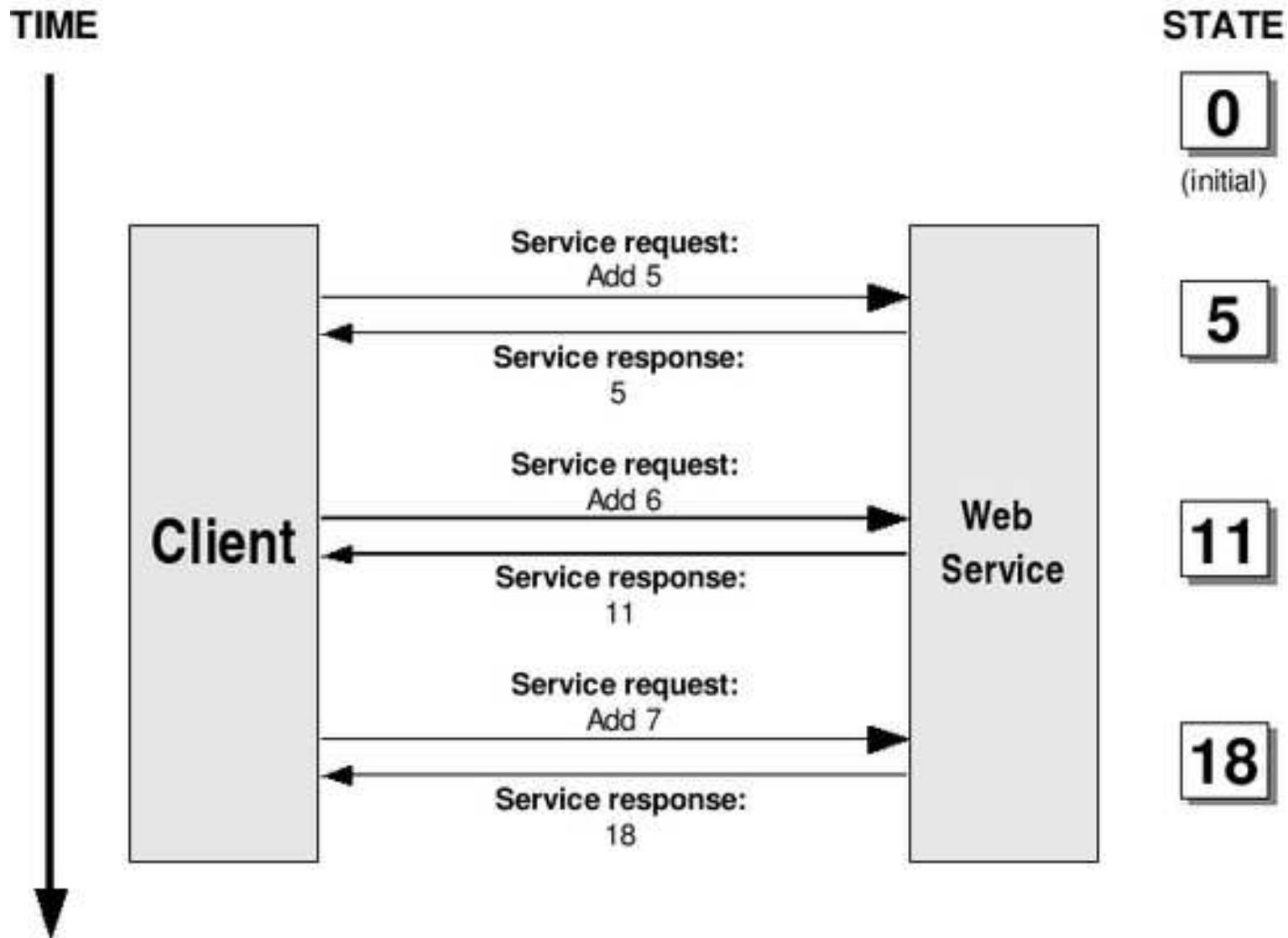
Comunicarea *stateless*

TIME

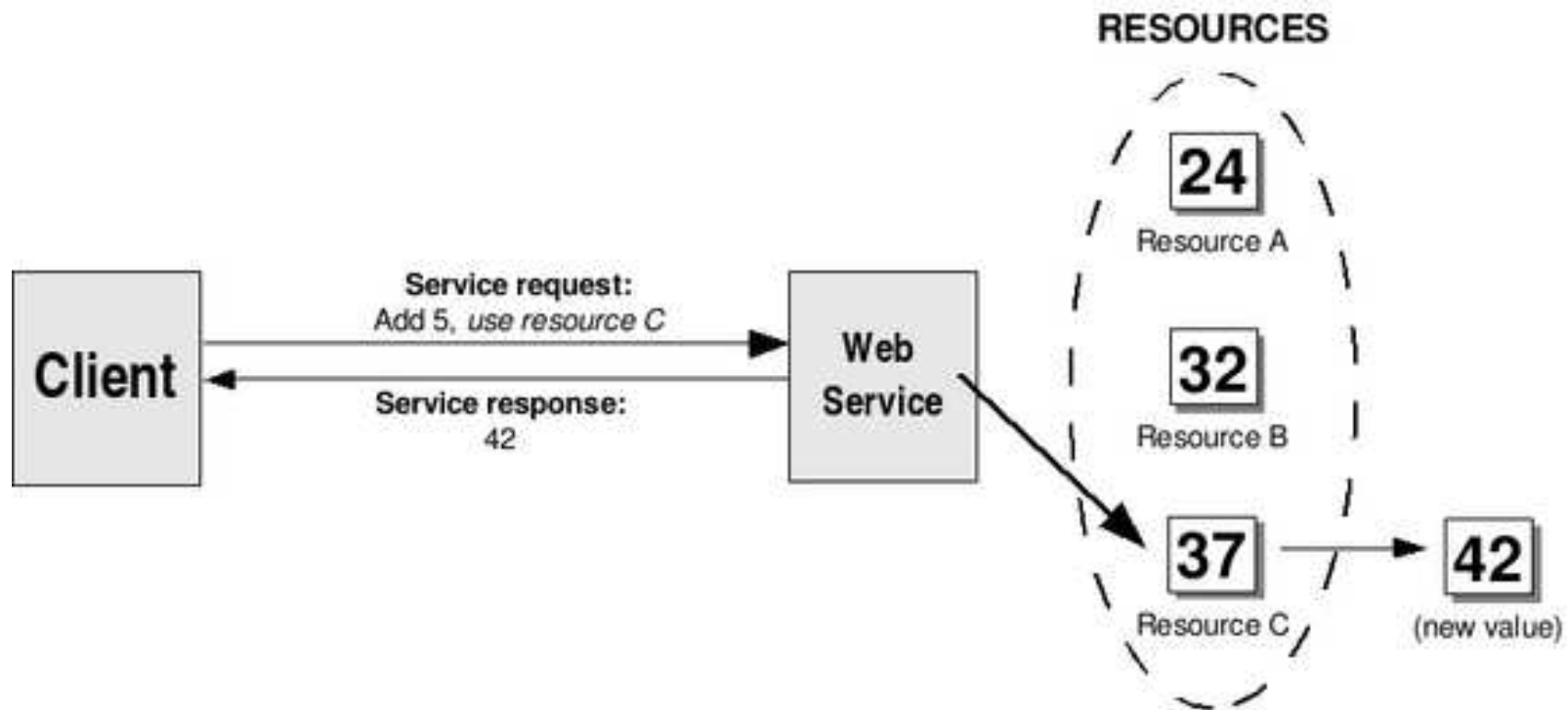


No state information is kept!

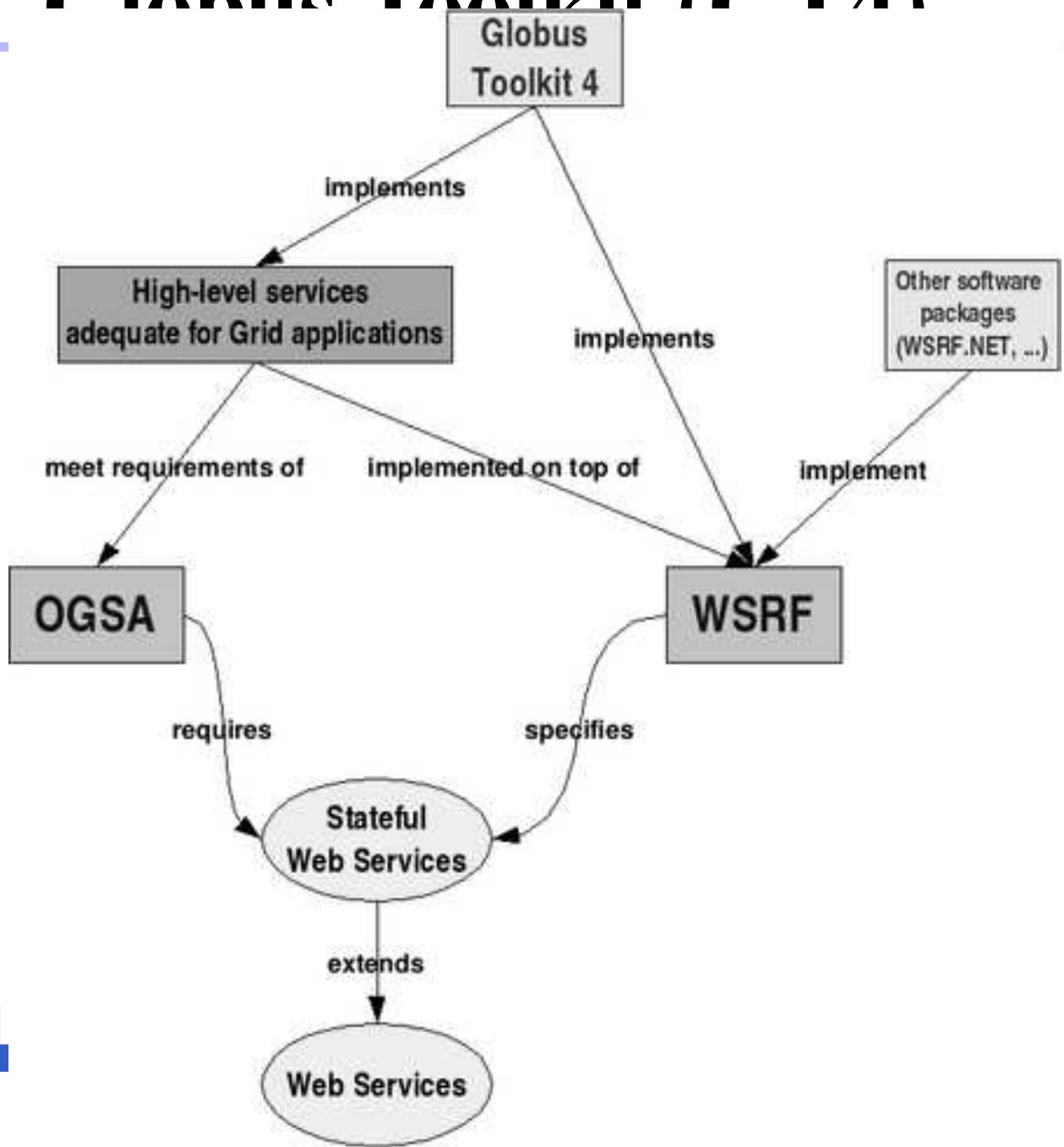
Comunicarea *stateful*



Abordarea orientată pe *resurse*



Globus Toolkit (GT4)





BPEL (Business Process Execution Language)



Arhitectura SOA

- Un standard pentru expunerea și accesarea aplicațiilor sub formă de servicii
→ **servicii web**
- Infrastructură pentru comunicare și gestiunea serviciilor
→ **ESB** (Enterprise Service Bus)
- Limbaj specializat pentru compunerea funcționalităților simple în unele complexe care să modeleze procese economice
→ **BPEL**

Programarea în ansamblu

- **Programarea în ansamblu (programming in the large)**

Stil de programare care descrie la un nivel înalt logica tranzițiilor stărilor unui sistem. Aspectele surprinse se referă la: trimiterea/primirea de mesaje, compensarea tranzacțiilor eșuate, etc.

- **Programarea în detaliu (programming in the small)**

Stilul clasic de programare care se ocupă cu descrierea explicită a comportamentului sistemului.

Ce este BPEL ?

- limbaj de programare bazat pe XML, în curs de standardizare (2003) la OASIS
- permite definirea de procese abstracte sau executabile
- permite compunerea serviciilor web
- se bazează pe aceleași protocoale ca și serviciile web (WSDL, SOAP, etc.)
- susținere puternică din partea industriei (Oracle, IBM, Microsoft, BEA, etc.)
 - designere BPEL (dezvoltare)
 - servere BPEL (execuție)

Compunerea serviciilor

- **Orchestrare:** există un proces central cu rol de coordonare explicită a serviciilor; acestea nu sunt conșiente că sunt implicate într-un serviciu compus.
- **Coreografie:** nu există un coordonator central; fiecare serviciu web știe când să se execute și cu cine interacționează - efort colaborativ bazat pe sincronizare și schimbarea de mesaje.

BPEL este conceput pe paradigma orchestrării.

Proces economic

Proces economic (business process) = colecție de invocări coordonate ale unor servicii și activități adiacente responsabile cu crearea unui răspuns, definite pentru una sau mai multe organizații.

Exemplu: procesul de desfășurare a delegațiilor membrilor unei companii:

- extragerea datelor despre angajat
- determinarea unei rute până la destinație
- apelarea serviciilor transportatorilor pentru determinarea prețului optim

Tipuri de procese

- **Abstracte:** permit specificarea mesajelor publice care pot fi schimbate între servicii; nu includ detalii interne legate de fluxul procesului și nu pot fi executate.
- **Executabile:** permit specificarea tuturor detaliilor procesului și pot fi executate de un server BPEL (orchestration engine); conceptual, există două abordări:
 - **structuri de control** (Microsoft - XLANG)
 - **grafuri orientate** (IBM - WSFL)

Execuția unui proces

1. Procesul primește o cerere de la un client
2. Procesul efectuează o serie de invocări ale serviciilor implicate

Un proces BPEL este format din mai multe **activități**.

- primitive
- structurate

Execuția activităților se face:

- secvențial
- paralel

3. Procesul răspunde clientului

Activități primitive

- `<invoke>`
- `<receive>`
- `<reply>`
- `<assign>`
- `<throw>`
- `<wait>`
- `<terminate>`

Activități structurate

Permit combinarea activităților simple și definirea procesului într-o manieră algoritmică.

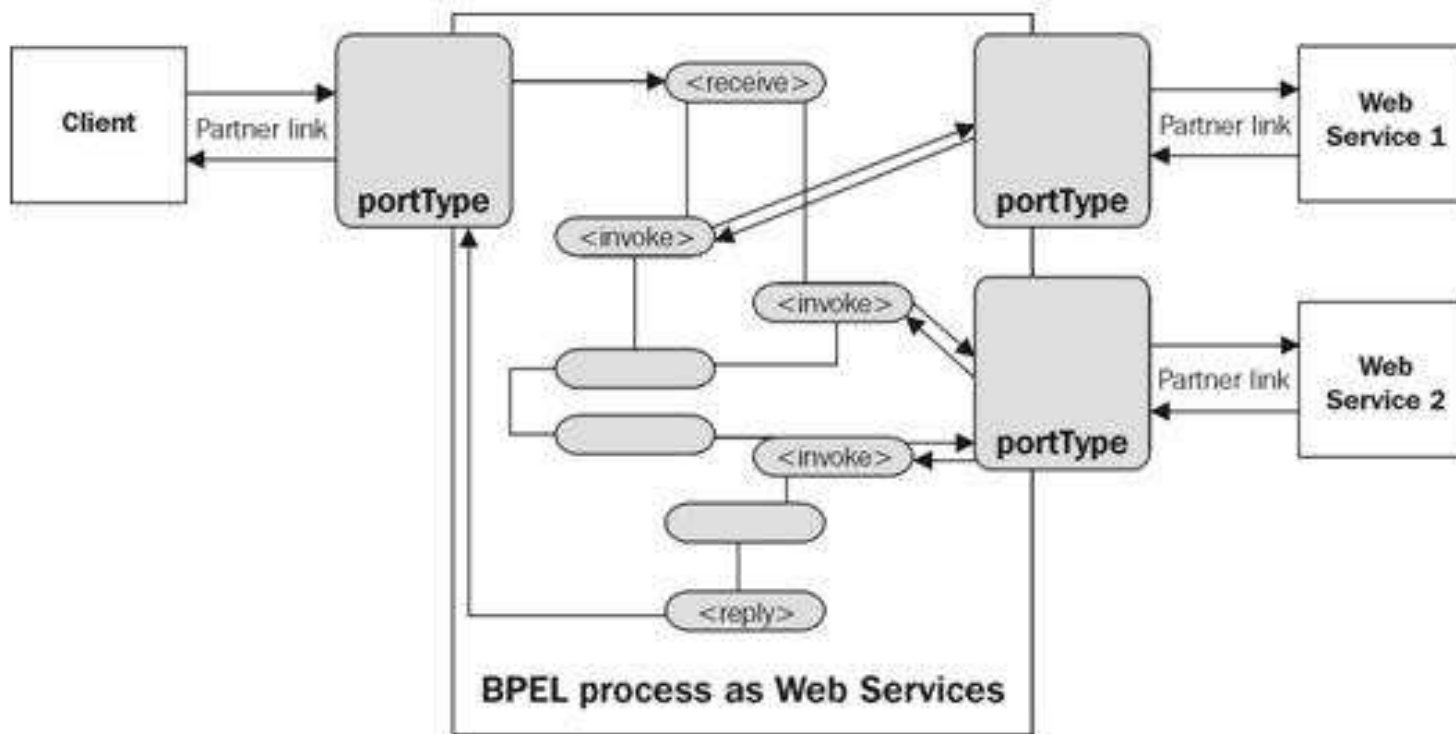
- `<sequence>` - invocarea secvențială (în ordine) a unui set de activități
- `<flow>` - invocarea în paralel a unui set de activități
- `<switch>` - execuție condiționată
- `<while>` - execuție repetitivă
- `<pick>` - selectarea unei variante dintr-o mulțime de alternative

Partner Link

Partner Link = legătură cu orice participant la proces; necesare pentru a modela comunicarea asincronă sau multiplă cu același serviciu. Partenerii pot fi:

- clientul care a realizat cererea
- serviciile invocate de proces
- servicii care invocă procesul (în operații de callback)
- servicii mixte (care sunt invocate sau invocă)

Reprezentare schematică





Exemplu de implementare BPEL



Declararea unui proces

```
<process name="insuranceSelectionProcess"
  targetNamespace="http://packtpub.com/bpel/example/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:ins="http://packtpub.com/bpel/insurance/"
  xmlns:com="http://packtpub.com/bpel/company/" >
  <partnerLinks>
    <partnerLink name="client"
      partnerLinkType="com:selectionLT"
      myRole="insuranceSelectionService"/>
    <partnerLink name="insuranceA"
      partnerLinkType="ins:insuranceLT"
      myRole="insuranceRequester"
      partnerRole="insuranceService"/>
    <partnerLink name="insuranceB"
      partnerLinkType="ins:insuranceLT"
      myRole="insuranceRequester"
      partnerRole="insuranceService"/>
  </partnerLinks>
```



Definirea variabilelor



...

```
<variables>
  <!-- input for BPEL process -->
  <variable name="InsuranceRequest"
            messageType="ins:InsuranceRequestMessage" />
  <!-- output from insurance A -->
  <variable name="InsuranceAResposne"
            messageType="ins:InsuranceResponseMessage" />
  <!-- output from insurance B -->
  <variable name="InsuranceBResposne"
            messageType="ins:InsuranceResponseMessage" />
  <!-- output from BPEL process -->
  <variable name="InsuranceSelectionResponse"
            messageType="ins:InsuranceResponseMessage" />
</variables>
```

...



Invocarea serviciilor

```
<sequence>
  <!-- Receive the initial request from client -->
  <receive partnerLink="client"
    portType="com:InsuranceSelectionPT"
    operation="SelectInsurance"
    variable="InsuranceRequest" createInstance="yes" />
  <!-- Make concurrent invocations to Insurance A and B -->
  <flow>
    <!-- Invoke Insurance A web service -->
    <invoke partnerLink="insuranceA"
      portType="ins:ComputeInsurancePremiumPT"
      operation="ComputeInsurancePremium"
      inputVariable="InsuranceRequest"
      outputVariable="InsuranceAResposne" />

    <!-- Invoke Insurance B web service -->
    <invoke partnerLink="insuranceB"
      ... />
  </flow>
```

Crearea răspunsului

```
<!-- Select the best offer and construct the response -->
<switch>
  <case condition="bpws:getVariableData('InsuranceAResposne',
    'confirmationData', '/confirmationData/Amount')
    <= bpws:getVariableData('InsuranceBResposne',
    'confirmationData', '/confirmationData/Amount')">
    <!-- Select Insurance A -->
    <assign>
      <copy>
        <from variable="InsuranceAResposne" />
        <to variable="InsuranceSelectionResponse" />
      </copy>
    </assign>
  </case>
  <otherwise>
    <!-- Select Insurance B -->
    ...
  </otherwise>
</switch>
```

Crearea răspunsului

```
<!-- Send a response to the client -->
<reply partnerLink="client"
      portType="com:InsuranceSelectionPT"
      operation="SelectInsurance"
      variable="InsuranceSelectionResponse" />
</sequence>
</process>
```

BPEL și Java

Avantaje BPEL vs. Java

- **Portabilitate:** Procesele BPEL pot fi executate pe servere bazate pe platforma Java sau .NET
- Suport **specific** pentru procese economice: de lungă durată, asincrone, slab-cuplate
- **Concurență**
- **Secvențializare**
- **Compensări:** similare tranzacțiilor (*Long-Running Transactions (LRT)*)
- Suport pentru comunicare **stateful**

BPEL + Java

Extinderea specificațiilor BPEL pentru compunerea și altor tipuri de resurse: EJB, JMS, RMI, etc.

- Integrarea limbajelor Java și BPEL (BPELJ)
Snippet = secvență de cod Java inclusă în definiția unui proces BPEL
- Descrierea resurselor cu WSDL (Web Services Invocation Framework)

Servere și instrumente BPEL

Java

- * Oracle BPEL Process Manager
- * IBM WebSphere Business Integration Server Foundation
- * BEA WebLogic Integration
- * OpenStorm Service Orchestrator
- * Vergil VCAB Server
- * Active Endpoints ActiveWebflow Server
- * ActiveBPEL engine
- * Fivesight Process eXecution Engine
- * Apache Agila

.NET

- * Microsoft BizTalk
- * OpenStorm Service Orchestrator