Web Services

Benoît Garbinato

distributed object programming lab

Outline

- ☐ Definition and origin of web services
- ☐ General-purpose web services
- ☐ RESTful web services
- ☐ Using JAX-WS and JAX-RS

dop lab

Web Service – Definitions

a service offered by an electronic device to another electronic device, communicating with each other via the world wide web

wikipedia
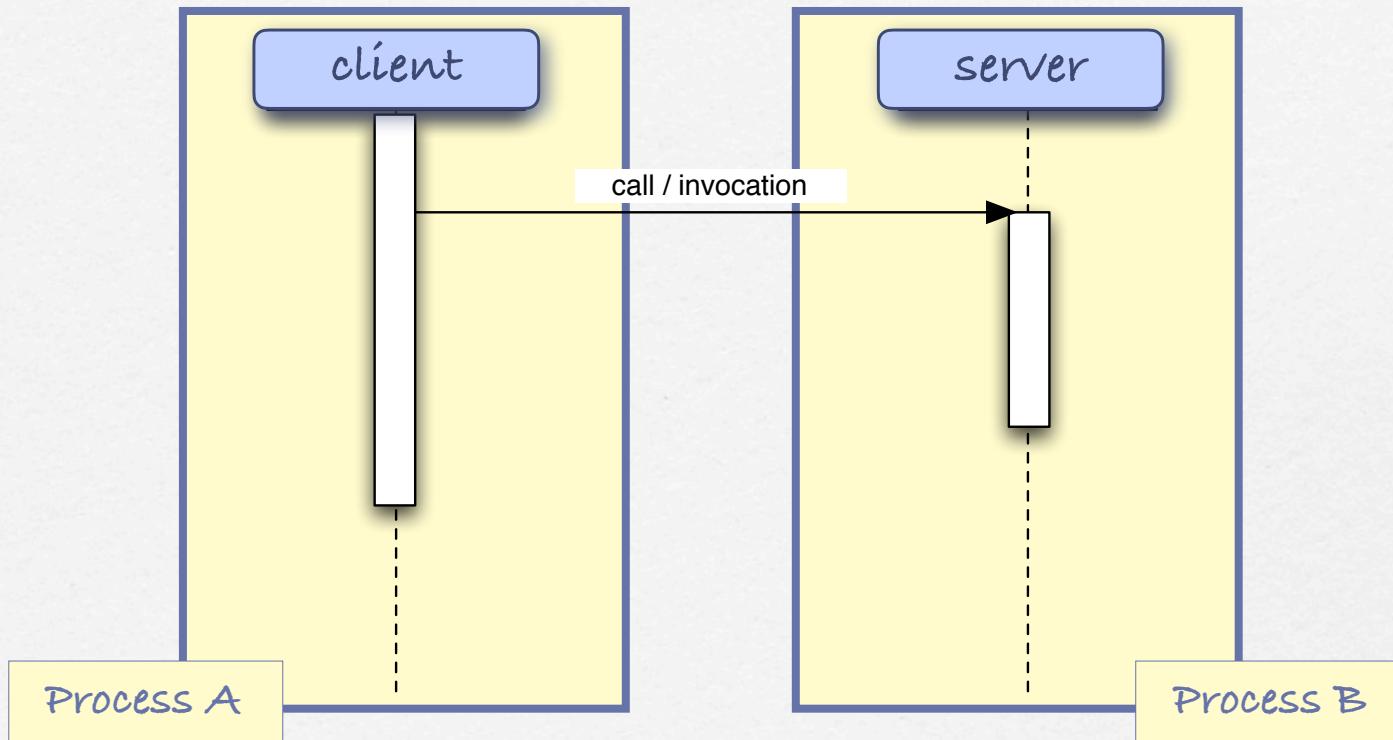
rmi* on http

*rmi ⇌ remote method invocation

software system designed to support interoperable machine-to-machine interaction over a network

world wide web consortium (w3c)

dop
l a b

RMI | Fundamental Idea
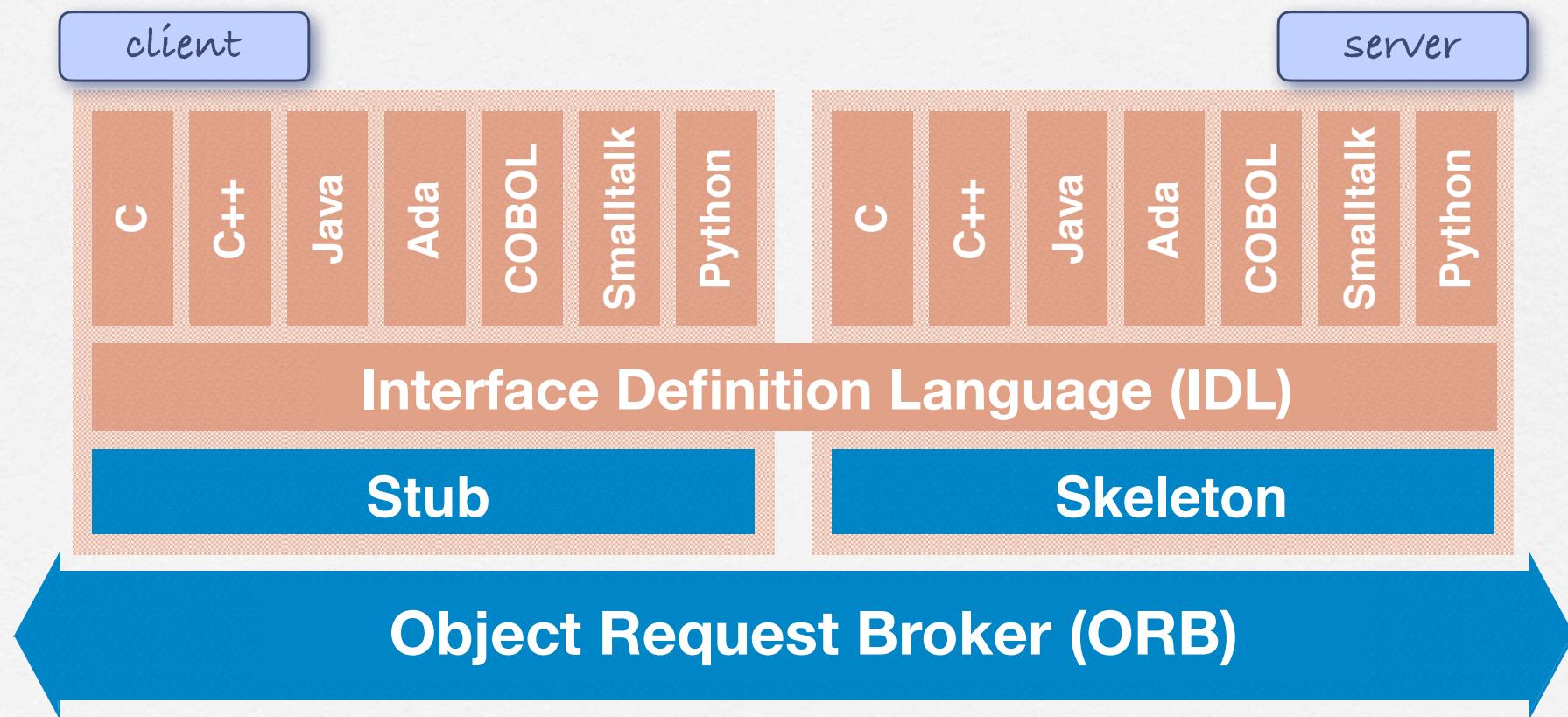
A remote method (procedure) is transparently invoked (called) across the network, as if it was local

dop lab

RMI | Some history

1979   Bill Joy introduces the "Berkeley Enhancements", mainly interprocess communication (IPC) facilities. The modern network Unix is born (BSD).

mid 80's   Sun Microsystems uses BSD Unix as operating system for their workstations. They extends it with RPC, on top of which they build NFS and NIS (later on NIS+).

1988   The Open Software Foundation (OSF) is formed to develop a portable open system platform, known as the Distributed Computing Environment (DCE). The latter proposes DCE RPC as basic communication mechanism.

mid 90's   The Object Management Group (OMG) follows the same approach to devise the Common Object Request Broker Architecture (CORBA) for object-based middleware. At the same time, Sun greatly simplifies & extends the RMI paradigm its Java & Jini platforms.

Today   Web Services are a widespread approach to invoke remote services on the web but they are really just a web-flavored version of the good old RPC/RMI paradigm, using HTTP & XML/JSON.

dop lab
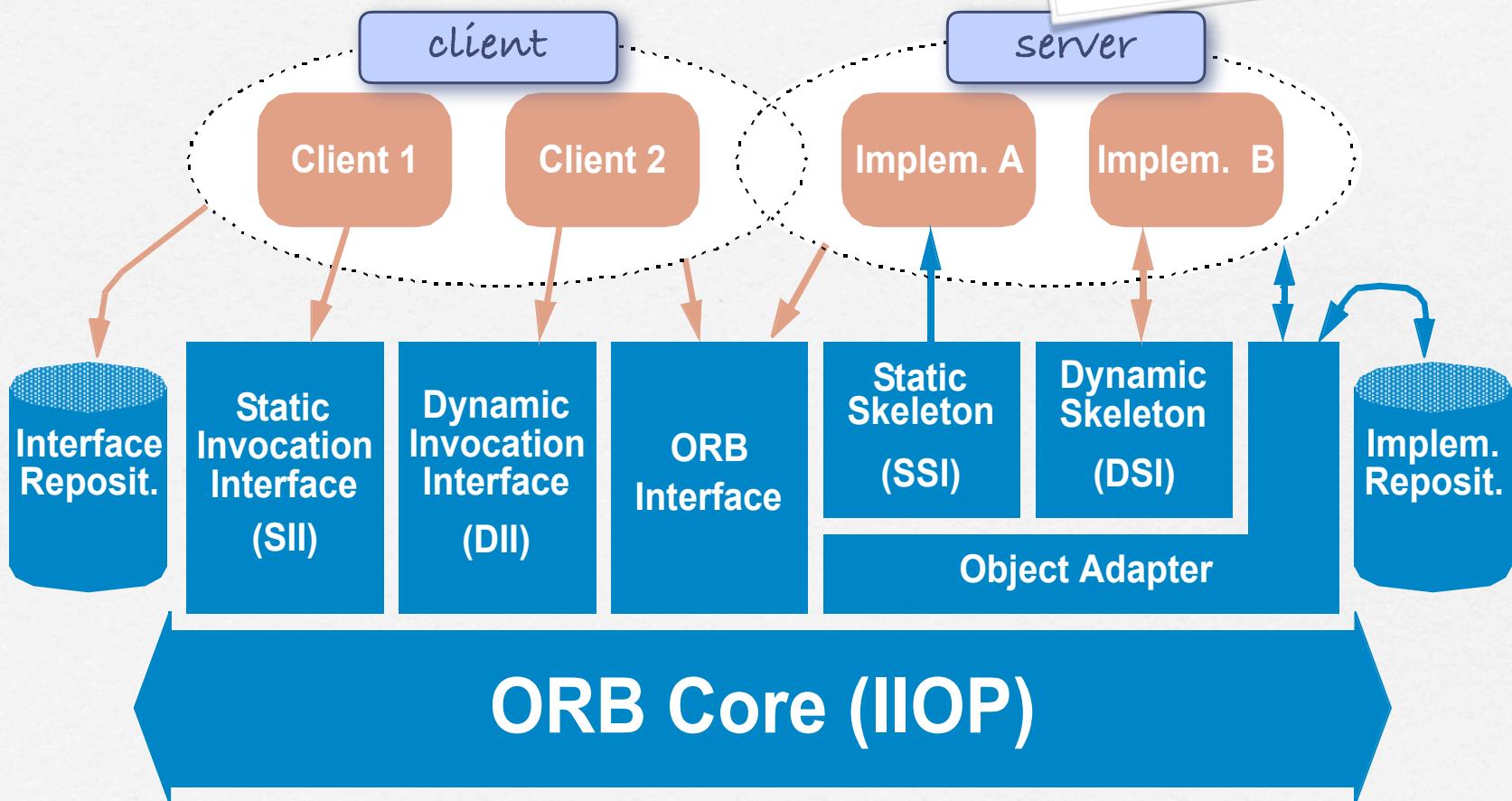
A glimpse at the CORBA promise

Common Object Request Broker Architecture

client

server

| C | C++ | Java | Ada | COBOL | Smalltalk | Python | | C | C++ | Java | Ada | COBOL | Smalltalk | Python |

Interface Definition Language (IDL)

Stub

Skeleton

Object Request Broker (ORB)

dop lab
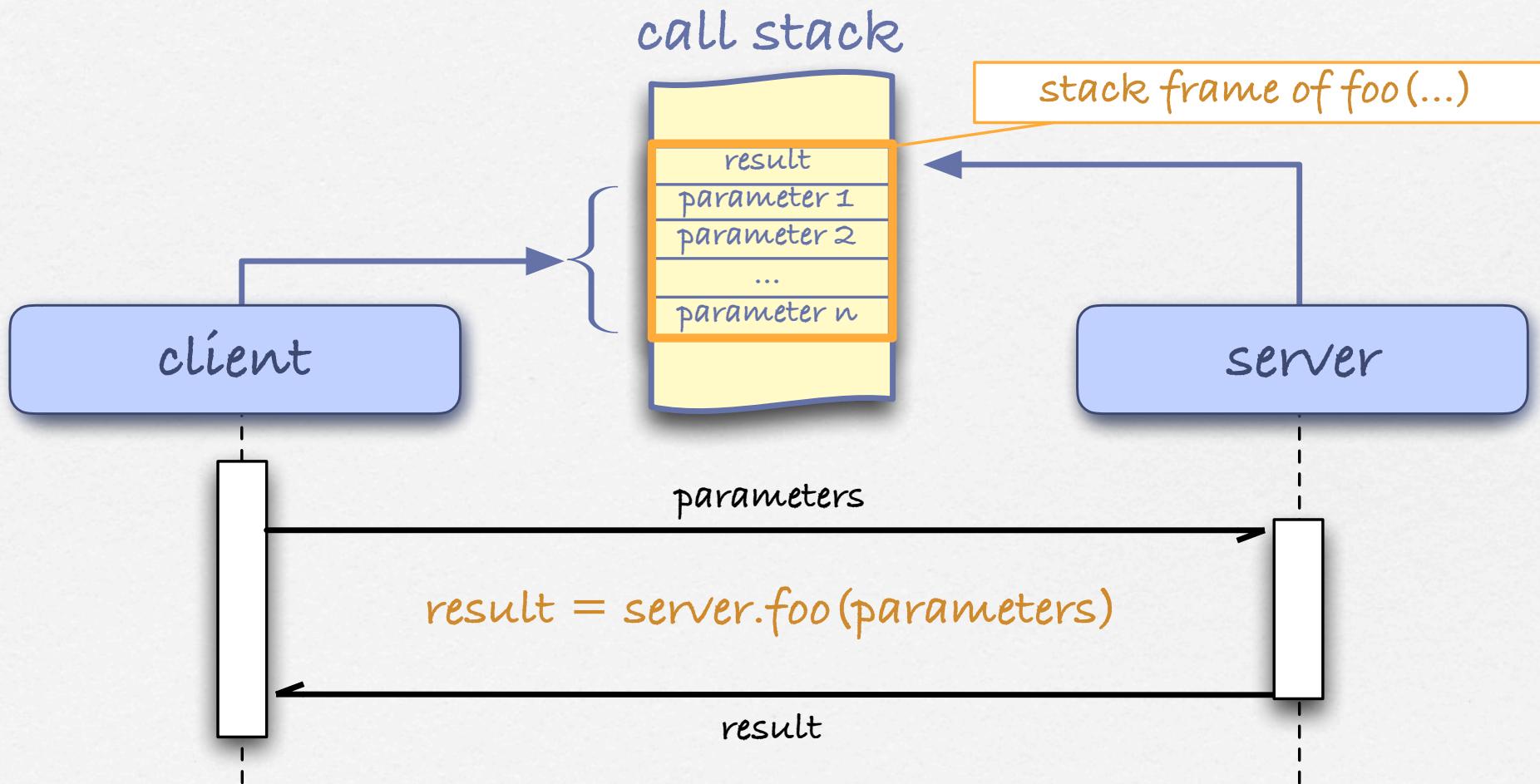
A glimpse at the CORBA reality
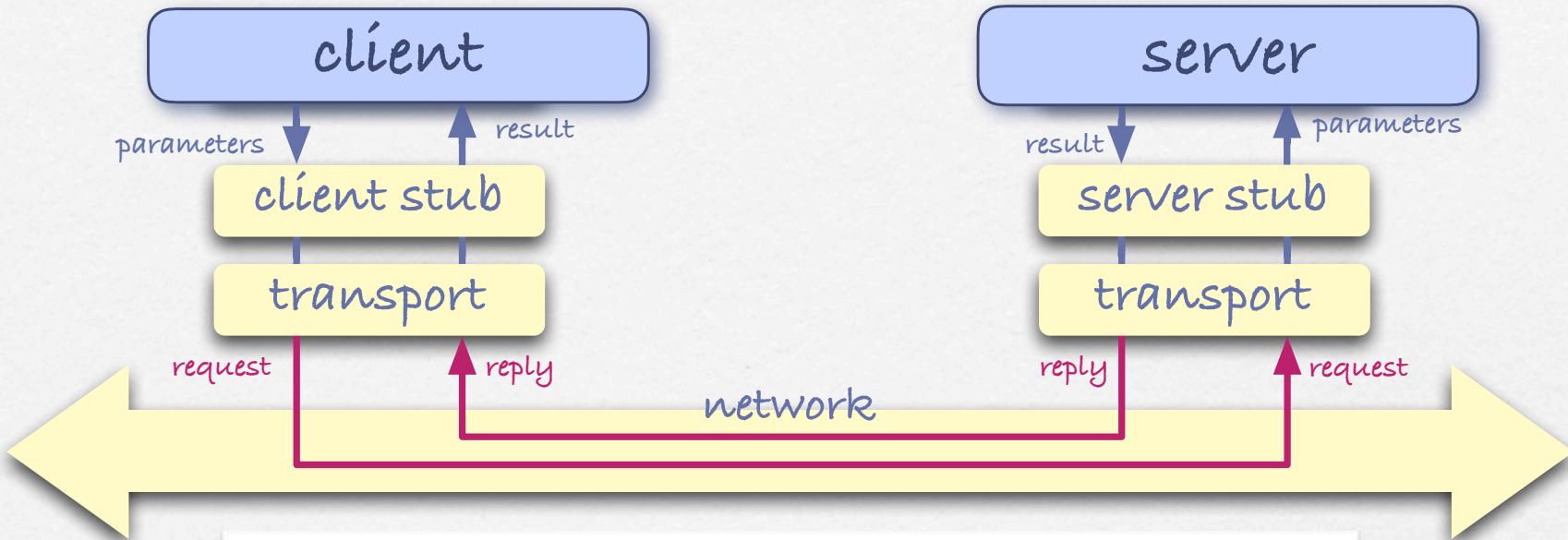
Common Object Request Broker Affliction

client   server

| Client 1 | Client 2 | Implem. A | Implem. B |

Interface Reposit.

Static Invocation Interface (SII)

Dynamic Invocation Interface (DII)

ORB Interface

Static Skeleton (SSI)

Dynamic Skeleton (DSI)

Object Adapter

Implem. Reposit.

ORB Core (IIOP)

IIOP ⇔ Internet Inter-ORB Protocol

dop l a b

Local Method Invocations

call stack

stack frame of foo(...)

result
parameter 1
parameter 2
...
parameter n

client

server

parameters

result = server.foo(parameters)

result

dop·.·
l a b

# Remote Method Invocation
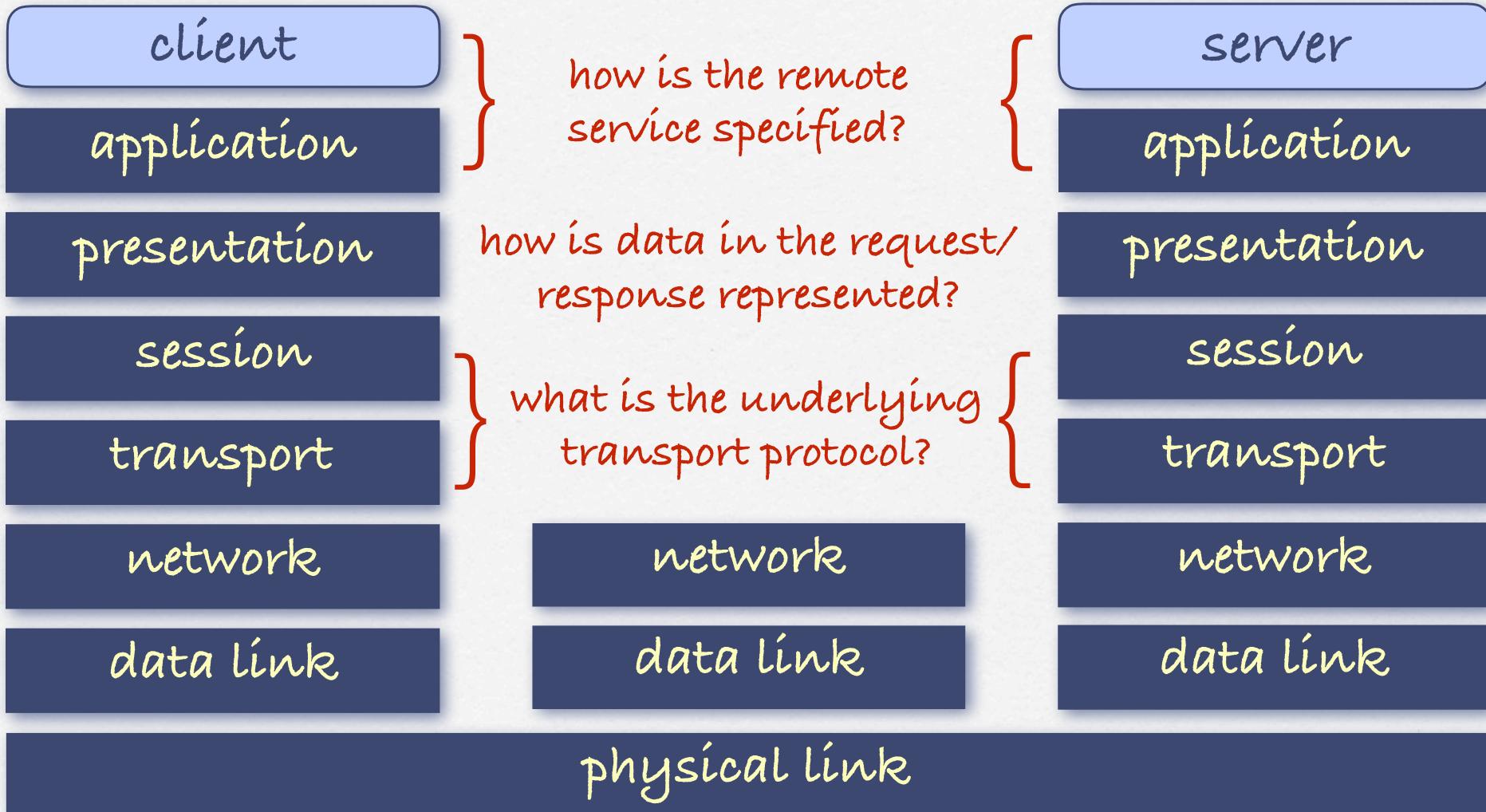
how is the remote service specified?
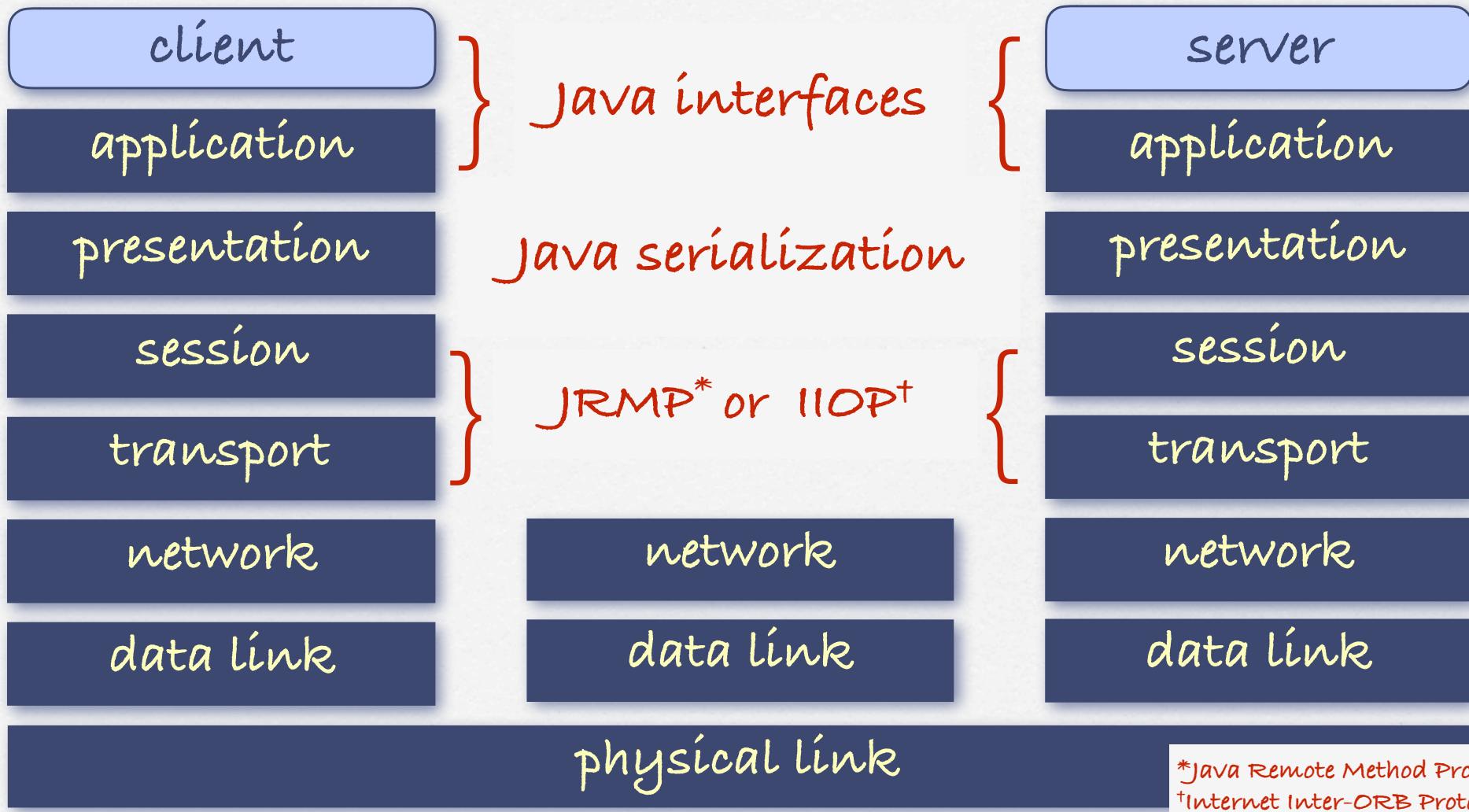
how is data in the request/response represented?

what is the underlying transport protocol?

The OSI model | Reminder

client		server
application	how is the remote service specified?	application
presentation	how is data in the request/response represented?	presentation
session		session
transport	what is the underlying transport protocol?	transport
network	network	network
data link	data link	data link
physical link		

dop lab

The OSI model | Java RMI

client		server
application	Java interfaces	application
presentation	Java serialization	presentation
session		session
transport	JRMP* or IIOP†	transport
network	network	network
data link	data link	data link
physical link		

*Java Remote Method Protocol
†Internet Inter-ORB Protocol

Web Services © Benoît Garbinato

dop lab

The OSI model | CORBA

client	Interface Definition Language (IDL)	server
application		application
presentation	language-specific mappings	presentation
session	Internet Inter-ORB Protocol (IIOP)	session
transport		transport
network	network	network
data link	data link	data link
physical link		

dop lab

Service specification with Java RMI

```java
public interface CalendarService extends Remote {
    public DayCalendar createCalendar(String name) throws RemoteException, CalendarException;
    public DayCalendar getCalendar(String name) throws RemoteException, CalendarException;
    public ArrayList getCalendars() throws RemoteException;
    public boolean exists(String name) throws RemoteException;
}
```

```java
public class CalendarServer extends UnicastRemoteObject implements CalendarService  {
    ...
    public DayCalendar createCalendar(String name) throws RemoteException, CalendarException {
        if (calendars.containsKey(name)) throw new CalendarException(name + "\" already exists.");
        DayCalendar newCal= new DayCalendarImpl(name);
        calendars.put(name, newCal);
        return newCal;
    }
    public DayCalendar getCalendar(String name) throws RemoteException, CalendarException {
        if (!calendars.containsKey(name)) throw new CalendarException(name + "\" does not exist.");
        return ((DayCalendar) calendars.get(name));
    }
    public ArrayList getCalendars() throws RemoteException {
        return new ArrayList(calendars.values());
    }
    public boolean exists(String name) throws RemoteException {
        return calendars.containsKey(name);
    }
}
```

Service specification with CORBA IDL

IDL Type	Java Type
module	package
boolean	boolean
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long, unsigned long	long
float	float
double	double
fixed	java.math.BigDecimal
enum, struct, union	class
sequence, array	array
...	...
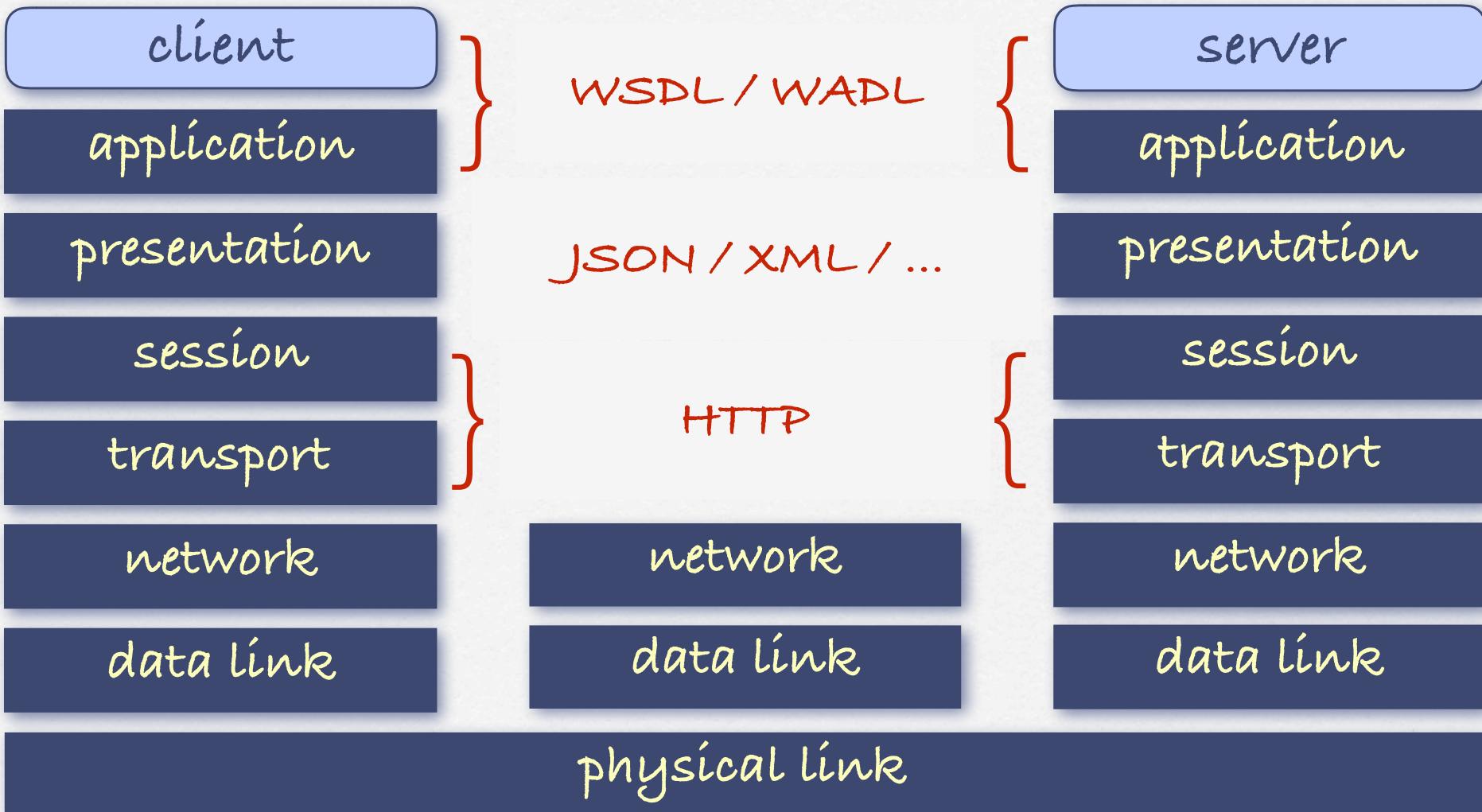
```
module employee {
  struct EmployeeInfo {
    wstring name;
    long number;
    double salary;
  };


  exception SQLError {
    wstring message;
  };


  interface Employee {
    EmployeeInfo getEmployee (in wstring name) raises (SQLError);
    EmployeeInfo getEmployeeForUpdate (in wstring name) raises (SQLError);
    void updateEmployee (in EmployeeInfo name) raises (SQLError);
  };
};
```

dop lab

The OSI model | Web Services

client		WSDL / WADL		server
application	}		{	application
presentation		JSON / XML / ...		presentation
session	}		{	session
transport		HTTP		transport
network		network		network
data link		data link		data link
physical link				

doplab

Web service types

There exists two types of web services:

☐ **General-purpose web services**, which provide support to remotely call any kind of operations

☐ **RESTful web services**, which focus on state transfer

REST = Representational State Transfer

General-purpose web services

☐ They provide support to remotely call any kind of operations

☐ They rely on the Web Services Description Language (WSDL)

☐ They rely on the Simple Object Access Protocol (SOAP), an XML standard defining a message architecture and format

☐ In Java, JAX-WS is the technology that encapsulates (part of) the complexity of defining and using general-purpose web services

dop lab

Using JAX-WS (1)

```xml
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://service/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://service/" name="HelloWorld">
<types>
<xsd:schema>
<xsd:import namespace="http://service/" schemaLocation="http://localhost:8080/HelloWorld/HelloWorld?xsd=1"/>
</xsd:schema>
</types>
<message name="hello">
<part name="parameters" element="tns:hello"/>
</message>
<message name="helloResponse">
<part name="parameters" element="tns:helloResponse"/>
</message>
<portType name="HelloWorld">
<operation name="hello">
<input wsam:Action="http://service/HelloWorld/helloRequest" message="tns:hello"/>
<output wsam:Action="http://service/HelloWorld/helloResponse" message="tns:helloResponse"/>
</operation>
</portType>
<binding name="HelloWorldPortBinding" type="tns:HelloWorld">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="hello">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="HelloWorld">
<port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
<soap:address location="http://localhost:8080/HelloWorld/HelloWorld"/>
</port>
</service>
</definitions>
```

WSDL

dop lab

hello Method invocation

Method parameter(s)

Type	Value
java.lang.String	Tim

Method returned
java.lang.String : "**Hello Tim !**"

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

SOAP Response
```xml
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:helloResponse xmlns:ns2="http://service/">
            <return>Hello Tim !</return>
        </ns2:helloResponse>
    </S:Body>
</S:Envelope>
```

WSDL

dop l a b

Using JAX-WS (3)

```java
@WebService(serviceName = "HelloWorld")
@Stateless()
public class HelloWorld {

  @WebMethod(operationName = "hello")
  public String hello(@WebParam(name = "name") String name) {
    System.out.println("I received a call with " + name + " as parameter");
    return "Hello " + name + " !";
  }
}
```
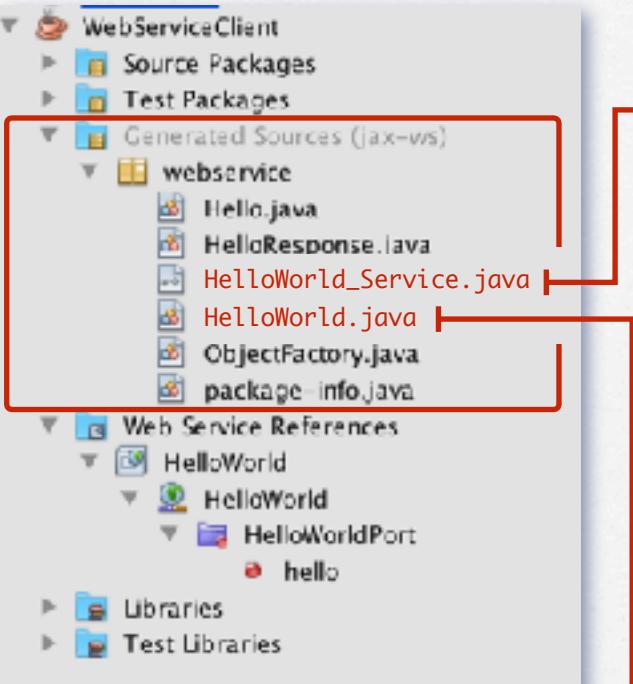
```java
public class Client {

  public static void main(String[] args) {
    String response = hello("Tim");
    System.out.println("response = " + response);
  }

  private static String hello(java.lang.String name) {
    webservice.HelloWorld_Service service = new webservice.HelloWorld_Service();
    webservice.HelloWorld port = service.getHelloWorldPort();
    return port.hello(name);
  }
}
```

dop l a b

```
WebServiceClient
  ▶ 📁 Source Packages
  ▶ 📁 Test Packages
  ▼ 📁 Generated Sources (jax-ws)
    ▼ 📁 webservice
        Hello.java
        HelloResponse.java
        HelloWorld_Service.java
        HelloWorld.java
        ObjectFactory.java
        package-info.java
  ▼ 📁 Web Service References
    ▼ 📁 HelloWorld
      ▼ 📁 HelloWorld
        ▼ 📁 HelloWorldPort
            hello
  ▶ 📁 Libraries
  ▶ 📁 Test Libraries
```

```java
@WebServiceClient(name = "HelloWorld",
                  targetNamespace = "http://service/",
                  wsdlLocation = "http://localhost:8080/HelloWorld/HelloWorld?wsdl")
public class HelloWorld_Service
    extends Service
{

    private final static URL HELLOWORLD_WSDL_LOCATION;
    private final static WebServiceException HELLOWORLD_EXCEPTION;
    private final static QName HELLOWORLD_QNAME = new QName("http://service/", "HelloWorld");
    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://localhost:8080/HelloWorld/HelloWorld?wsdl");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        HELLOWORLD_WSDL_LOCATION = url;
        HELLOWORLD_EXCEPTION = e;
    }
    public HelloWorld_Service() {
        super(__getWsdlLocation(), HELLOWORLD_QNAME);
    }
    ...
    @WebEndpoint(name = "HelloWorldPort")
    public HelloWorld getHelloWorldPort() {
        return super.getPort(new QName("http://service/", "HelloWorldPort"), HelloWorld.class);
    }
    ...
}
```

```java
@WebService(name = "HelloWorld",
            targetNamespace = "http://service/")
@XmlSeeAlso({
    ObjectFactory.class
})
public interface HelloWorld {
    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "hello", targetNamespace = "http://service/",
                    className = "webservice.Hello")
    @ResponseWrapper(localName = "helloResponse", targetNamespace = "http://service/",
                     className = "webservice.HelloResponse")
    @Action(input = "http://service/HelloWorld/helloRequest",
            output = "http://service/HelloWorld/helloResponse")
    public String hello(
        @WebParam(name = "name", targetNamespace = "")
        String name);

}
```

dop l a b

RESTful web services (1)

- ☐ They focus on state transfer, usually from/to some persistent storage, e.g., a relational database

- ☐ They manipulate state as resources accessed using Uniform Resource Identifiers (URIs) and four HTTP verbs as CRUD operations:

HTTP	CRUD
PUT	CREATE
GET	READ
POST	UPDATE
DELETE	DELETE

dop lab

RESTful web services (2)

- ☐ They do not formally require data to be represented using a particular format, although JSON and XML are very often used for structured data today

- ☐ They do not formally require an interface definition language, although it is now common practice to use the Web Application Definition Language (WADL) in Java

- ☐ In Java, JAX-RS is the technology that encapsulates (part of) the complexity of defining and using RESTful web services and Jersey is the reference implementation of JAX-RS

dop l a b

Using JAX-RS (1)

```xml
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.10.4 2014-08-08 15:09:00"/>
<doc xmlns:jersey="http://jersey.java.net/" jersey:hint="This is simplified WADL with user and core resources only. To get full
WADL with extended resources use the query parameter detail. Link: http://localhost:8080/CustomerDB/webresources/application.wadl?
detail=true"/>
<grammars>
<include href="application.wadl/xsd0.xsd">
<doc title="Generated" xml:lang="en"/>
</include>
</grammars>
<resource path="entities.customer">
<method id="findAll" name="GET">
<response>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/xml"/>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/json"/>
</response>
</method>
<method id="create" name="POST">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/xml"/>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/json"/>
</request>
</method>
<resource path="{id}">
<param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" style="template" type="xs:int"/>
<method id="edit" name="PUT">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/xml"/>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/json"/>
</request>
</method>
<method id="remove" name="DELETE"/>
<method id="find" name="GET">
<response>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/xml"/>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="customer" mediaType="application/json"/>
</response>
</method>
</resource>
</resource>
...
</application>
```

Using JAX-RS (2)

```java
@Entity
@Table(name = "CUSTOMER")
@XmlRootElement
@NamedQueries({
   @NamedQuery(name = "Customer.findAll",
              query = "SELECT c FROM Customer c"), ...)})
public class Customer implements Serializable {
 @Id
  @Basic(optional = false)
  @NotNull
  @Column(name = "CUSTOMER_ID")
  private Integer customerId;
  @Basic(optional = false)
  @NotNull
  @Size(min = 1, max = 10)
  @Column(name = "ZIP")
  private String zip;
  ...
 public Integer getCustomerId() {
    return customerId;
  }
  public void setCustomerId(Integer customerId) {
    this.customerId = customerId;
  }
  public String getZip() {
    return zip;
  }
  public void setZip(String zip) {
    this.zip = zip;
  }
  ...
}
```

```java
@Stateless
@Path("entities.customer")
public class CustomerFacadeREST extends AbstractFacade<Customer> {
 @PersistenceContext(unitName = "CustomerDBPU")
  private EntityManager em;
  public CustomerFacadeREST() {
    super(Customer.class);
  }
  @POST
  @Override
  @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public void create(Customer entity) {
    super.create(entity);
  }
  @PUT
  @Path("{id}")
  @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public void edit(@PathParam("id") Integer id, Customer entity) {
    super.edit(entity);
  }
  @DELETE
  @Path("{id}")
  public void remove(@PathParam("id") Integer id) {
    super.remove(super.find(id));
  }
  @GET
  @Path("{id}")
  @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public Customer find(@PathParam("id") Integer id) {
    return super.find(id);
  }
  @GET
  @Override
  @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public List<Customer> findAll() {
    return super.findAll();
  }
  ...
}
```

Using JAX-RS (3)

```java
public class Main {
  public static void main(String[] args) {
    CustomerDBClient client = new CustomerDBClient();

    String response = client.countREST();
    System.out.println("Client response is : " + response);
    ...
    client.close();
  }
}
```

```java
public class CustomerDBClient {
  private WebTarget webTarget;
  private Client client;
  private static final String BASE_URI = "http://localhost:8080/CustomerDB/webresources";

  public CustomerDBClient() {
    client = javax.ws.rs.client.ClientBuilder.newClient();
    webTarget = client.target(BASE_URI).path("entities.customer");
  }
  public void create_XML(Object requestEntity) throws ClientErrorException {
    webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML)
          .post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
  }
  public void create_JSON(Object requestEntity) throws ClientErrorException {
    webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON)
          .post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
  }
  public String countREST() throws ClientErrorException {
    WebTarget resource = webTarget;
    resource = resource.path("count");
    return resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
  }
  public void remove(String id) throws ClientErrorException {
    webTarget.path(java.text.MessageFormat.format("{0}", new Object[]{id})).request().delete();
  }
  ...
  public void close() {
    client.close();
  }
}
```