Middleware for synchronous requests illustrated with WebServices

Sophie Chabridon and Chantal Taconet

ASR/CSC5002

septembre 2020

Revision : 528
Outline

1 Introduction ................................................................. 2
2 Synchronous middleware and the big picture ........................................ 12
3 WebService introduction ........................................................ 27
4 Interface definition language ..................................................... 32
5 Distributed Identification ......................................................... 40
6 First steps with WebServices ...................................................... 43
7 Conclusions ............................................................................ 55
Middleware for synchronous requests illustrated with WebServices
1 Introduction

1.1 Middleware for distribution ................................................................. 4
1.2 Goal : interoperability ................................................................. 5
1.3 Distribution models ................................................................. 6
1.4 Client-server models ................................................................. 7
1.5 Middleware for distributed objects history ........................................ 8
1.6 Synchronous vs asynchronous mode .............................................. 9
1.7 Asynchronous call, synchronous call, buffered message ...................... 10
1.8 Call-back and Inversion of control ................................................... 11
1.1 Middleware for distribution

Middleware is a software layer which provides:

- Programming interfaces (common API)
- Protocol for interoperability
  - With data exchange format

...to support distribution and heterogeneity.
1.2 Goal: interoperability

- Existing “legacy code”,
- Numerous languages,
- Several operating systems,
- Various hardware (e.g., little endian, big endian),
- Several network protocols

⇒ need for interoperability!
1.3 Distribution models

- Point to point message
- Point to multipoint message
- Event/action
- Publish/subscribe
- Client/server
- Mobile code
- Virtual shared memory
1.4 Client-server models

- **Procedural**
  - Remote Procedure Call - RPC

- **Object-oriented**
  - Remote Method Invocation (RMI, Common Object Request Broker Architecture CORBA)

- **Data-oriented**
  - SQL requests
  - REST (Representational State Transfer) - create, read, update, delete over HTTP

- Traditionnal **Web** (HTTP requests)

- **Web Services** (SOAP over HTTP)
1.5 Middleware for distributed objects history

- Comes from two technologies:
  - Objects (inheritance, encapsulation and polymorphism)
  - RPC or Remote Procedure Call (distribution, heterogeneity, data marshalling and unmarshalling)
1.6 Synchronous vs asynchronous mode

- Two entities (e.g., processus) communicate

- In **synchronous** mode: the two entities (client and server) are active at the same time, after a request, client is waiting for server response.

- In **asynchronous** mode: entities send messages, they don’t wait for responses, they don’t know when the message will be delivered.
1.7 Asynchronous call, synchronous call, buffered message

Asynchronous event (push)

Synchronous call

Buffered messages (pull)
1.8 Call-back and Inversion of control

**Synchronous call with callback**
A callback is first registered and later called asynchronously.

**Inversion of control**
The control flow is no more under the responsibility of the application but controlled by the framework.
2 Synchronous middleware and the big picture

**Structural Compositions**

![SCA Diagram]

**Activity Orchestrations**

![BPEL Diagram]

**Application servers**
- Life cycle (instantiate)
- Persistency

![JavaEE Diagram]

- **WebServices/JavaRMI**
- **RabbitMQ**
- **Publish/Subscribe**

**Synchronous Call**

![TCP/UDP Diagram]
2.1 Introduction of the distributed example

- Which distribution?
- Which abstractions (service, object)?
- Which middleware?
2.2 Principle of distributed objects

```java
interface Printer {
    JobInfo submitPrint();
}
```

Automatic generation
stub and skeleton
implementation

client
stub
skeleton
implementation

Télécom SudParis — INF — septembre 2020 — ASR/CSC5002
2.3 The stub and the skeleton

t=RPI.submitPrint();
I.submitPrint()
submitPrint()
client
proxy
marshalling
arguments
unmarshalling
arguments
marshalling
results
unmarshalling
results
stub
skeleton
implementation

message
message
2.4 Proxy Object and inheritance tree

- Proxy: Representative for remote access

```
interface

Printer Interface

submitPrint()

<<implements>>

Client

Printer Proxy

submitPrint()

delegates

Printer Implementation

submitPrint()
```
2.5 Proxy design pattern

- **Context**: A client needs access to a remote service provided by some entity (called the “servant”)

- **Problem**
  - Define an access mechanism that does not involve
    - Hard-coding the location of the servant into the client code
    - Deep knowledge of the communication protocols by the client

- **Desirable properties**
  - Access should be efficient at run-time and secure
  - Programming should be simple: No difference between local and remote access

- **Constraints**: Distributed environment (no single address space)

- **Solutions**
  - Use a local representative of the server on the client side that isolates the client from the communication system and the servant
  - Keep the same interface for the representative as for the servant
  - Define a uniform proxy structure to facilitate automatic generation
### 2.5.1 Sequence diagram of Proxy

- **c:** Client
- **p:** Proxy
- **s:** Servant

1. **Service request**
   - Pre-processing (e.g., marshalling)
2. **Result**
   - Post-processing (e.g., unmarshalling)

- **Interface I**
1. Description of the interface in **IDL**

2. **IDL** compiler creates the stub and the skeleton

3. Write both **client** and **server** implementations
2.7 Multi-languages (or multi-ORBs, or multi-OSs)

Interface IDL

```
interface Printer {
    Tache submitPrint ();
};
```

compile IDL -> Java

```
stub
Java
```

```
skeleton
Java
```

printer.submitPrint();

compile IDL -> C++

```
stub
C++
```

```
skeleton
C++
```

client Java

```
client Java
```

```
implementation
C++
```

```
class PrinterImpl {
};
```
2.8 Distribution implications

- Objects/service implementation are in different spaces (not the same process, not the same computer . . . ):
  - Assign a **unique identifier** to each object/service in different spaces
  - Localize objects/service implementations
  - Transports requests and replys
  - Use of a neutral network format for the data
2.9 Invocation sequence diagram
2.10 Middleware for synchronous requests: main concepts

Middleware for synchronous requests illustrated with WebServices
2.11 Inherent complexity of distribution

- No global state
- Poor debugging tools
- Partial failures, network partition
- Requests in parallel (concurrency management)
- Trusting the caller (authentication)
2.12 Main distributed object middleware

CORBA (OMG) 1991
Java RMI (Sun) 1997
DCOM (Microsoft) 1998
.net (Microsoft) 2001
WebService (w3C) 2001
2.13 Middleware history
3 WebService introduction

3.1 Web Services specified by W3C ................................................................. 28
3.2 Two classes of WebServices ................................................................. 29
3.3 Web Service Overview ................................................................. 30
3.4 Some WebServices implementations .................................................. 31
3.1 Web Services specified by W3C

- World Wide Web Consortium
  - International standards organization for the World Wide Web created in 1994
  - 2011: 322 members
  - Famous standards: CSS, CGI, DOM, HTML, OWL, SOAP, XML, XPath, WSDL

http://www.w3c.org
3.2 Two classes of WebServices

- **REST Web Services**
  - Web resources manipulated through a uniform set of operations
    - POST (create resource), GET (read resource), PUT (update resource), DELETE (delete resource) - (CRUD)

- **Arbitrary Web Services**
  - Services expose an arbitrary set of operations
3.3 Web Service Overview

Middleware for synchronous requests illustrated with WebServices

SOAP
ORB 1
WSDL
IP
TCP
HTTP
ORB2
Specialized Web Server
WADL
REST WS
client
stub
message
submitPrint()
t=rPl

skeleton
implementation
i.submitPrint()

ORBIT

HTTP
TCP
IP
Arbitrary WS

SOAPO
HTTP
TCP
IP

HTTP
TCP
IP
REST WS

HTTP
TCP
IP
SMTP
Arbitrary WS

HTTP
TCP
IP
SMTP
Arbitrary WS
3.4 Some WebServices implementations

Some platforms among a list of many many Web services platforms (http://en.wikipedia.org/wiki/List_of_web_service_frameworks)

JDK (>1.6) JAVA Implémentation de JAX-WS (Java API for XML Web Services)
METRO JAVA Uptodate implementation of JAX-WS (Oracle)
glassfish JAVA Implementation of JAX-WS + application server + JavaEE (Oracle)
gsoap C, C++ free software
php/soap php free software
.net C#, VB Microsoft
4 Interface definition language

4.1 Interface definition language ................................................................. 33
4.2 RMI IDL ................................................................. 34
4.3 WSDL : Web Service Definition Language - Concepts ................................... 35
4.4 WSDL 1.1 vs WSDL 2.0 ................................................................. 36
4.5 WSDL PrintService example ................................................................. 37
4.6 WSDL : PrintService - text file excerpt ............................................... 38
4.7 Design methodologies ................................................................. 39
4.1 Interface definition language

To produce stub and skeletons, middleware use Interface Definition Languages

- Independent of implementation languages
- Independent of hardware and operating systems,
- Independent of implementation (processus, databases, objects...
4.2 RMI IDL

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface PrinterOperations extends Remote {
    PrinterInfo getPrinterInfo() throws RemoteException;
    JobInfo submitPrint(String fileName)
        throws PrintDenied, RemoteException;
};
```

- RMI client/server are written in JAVA/JAVA
- JAVA interface is the interface definition language
4.3 WSDL : Web Service Definition Language - Concepts

- A **service** may have several **ports** (operations are grouped in ports)
- **Bindings** define protocols, and URL to access a port
- Each **operation** is then defined by **request**, **response** and **fault messages**

*WSDL 1.1 (source wikipedia)*
4.4 WSDL 1.1 vs WSDL 2.0

**WSDL 1.1**
- **definitions**
  - **types**
  - **message**
  - **portType**
    - **operation**
      - input
      - output
  - **binding**
    - **service**
      - **port**

**WSDL 2.0**
- **description**
  - **types**
  - **interface**
    - **operation**
      - input
      - output
  - **binding**
    - **service**
      - **endpoint**

---

a. source wikipedia
4.5 WSDL PrintService example

Middleware for synchronous requests illustrated with WebServices

PortType

Operations

Binding

Service

PortTypeOperationsService Binding

SOAP 1.1

SOAP 1.2

HTTP POST

PrintService

PrinterServiceSOAP

http://localhost:....

PrinterService

submitPrint

input parameters

output return

fault fault

getPrinterInfo

input getPrinterInfoRequest in

output return

submitPrintRequest

submitPrintResponse

submitPrintFault

getPrinterInfoResponse

Télécom SudParis — INF — septembre 2020 — ASR/CSC5002
4.6 WSDL : PrintService - text file excerpt

```xml
<wSDL:types> ... </wSDL:types>
<wSDL:message name="submitPrintRequest">
  <wSDL:part element="tns:submitPrintRequest" name="parameters"/>
</wSDL:message>
<wSDL:message name="submitPrintResponse">
  <wSDL:part element="tns:submitPrintResponse" name="return"/>
</wSDL:message>
<wSDL:message name="submitPrintFault">
  <wSDL:part name="fault" element="tns:submitPrintFault"></wSDL:part>
</wSDL:message>
<wSDL:portType name="PrinterService">
  <wSDL:operation name="submitPrint">
    <wSDL:input message="tns:submitPrintRequest"/>
    <wSDL:output message="tns:submitPrintResponse"/>
    <wSDL:fault name="fault" message="tns:submitPrintFault"></wSDL:fault>
  </wSDL:operation>
  <wSDL:operation name="getPrinterInfo">
    <wSDL:input message="tns:getPrinterInfoRequest"></wSDL:input>
    <wSDL:output message="tns:getPrinterInfoResponse"></wSDL:output>
  </wSDL:operation>
</wSDL:portType>
<wSDL:binding name="PrinterInterfaceSoap11Binding"> ... </wSDL:binding>
<wSDL:service name="PrinterInterface"> ... </wSDL:service>
</wSDL:definitions>
```
4.7 Design methodologies

- **Bottom-Up strategy** (for example JAX-WS)
  1. Write implementation classes/interface with java annotations
     (@webservice, @webmethod)
  2. WSDL generating tool to expose some methods as a Web Service

- **Top-Down strategy**
  1. Write WSDL
  2. Produce the class skeleton (methods to be completed)
5 Distributed Identification

5.1 Distributed Identification ................................................. 41
5.2 Web Services URLs .......................................................... 42
5.1 Distributed Identification

- Because of distribution, objects/services don’t share the same space
- How middleware are able to:
  - Provide unique identifiers
  - Locate distributed entities
  - Transport messages to their destination
5.2 Web Services URLs

- For each **service port**: a URL
- **NB**: the URL is included in the interface (defined in WSDL)

```xml
<wSDL:service name="PrinterService">
  <wSDL:port
    <!-- name of the port-->
    name="PrinterInterfaceHttpSoap11Endpoint"
    <!-- binding has been defined previously (contains protocol and data format) -->
    binding="PrinterServiceSoapBinding"
      <wSDLsoap:address
        <!-- port URL-->
        location="http://localhost:9000/PrinterService" />
  </wSDL:port>
</wSDL:service>
```
6 First steps with WebServices

6.1 Development steps ................................................................. 44
6.2 The printer manager example ................................................ 45
6.3 Automatic generation of stub and skeleton ................................. 46
6.4 Service side: describe the interface of the service ...................... 47
6.5 Service side: provide an implementation of the service .............. 48
6.6 JAX-WS annotations ............................................................... 50
6.7 Service side: write the publisher ............................................. 51
6.8 Start the service ..................................................................... 52
6.9 Client side: generate the stub ............................................... 53
6.10 Client side in java ................................................................. 54
6.1 Development steps

■ On the server side
  ♦ Define the interface of the service with WSDL or with a java interface
  ♦ Translate the WSDL to java interface or java interface to WSDL with an appropriate tool
  ♦ Deploy the implementation in a Web server (or write a server which instantiates the service)

■ On the client side
  ♦ Generate the Web Service stub from the URL of the deployed WSDL
  ♦ Write and test the client
6.2 The printer manager example

PrinterService (interface)

PrinterServer (server)

PrinterClient

Printer

hp
6.3 Automatic generation of stub and skeleton

```xml
<?xml ...?>
<definitions>
  <type> <message> <operation> <binding> ....
  <portType name="PrinterService">
    <operation name="submitPrint">
      @webservice
    </operation>
  </portType>
</definitions>
```

- `wsgen` (JAXB XML schema to java classes)
- `wsimport` (JAXB java classes to XML schema)
- `javac` (includes JAX–WS compiler)

```
interface PrinterService {
    JobInfo submitPrint (in string fileName);
};
```
6.4 Service side: describe the interface of the service

For example with a java interface

```java
package printer;
import javax.jws.WebService;
@WebService
public interface PrinterService {
    public PrinterInfo getPrinterInfo();
    public JobInfo submitPrint (String fileName) throws PrintDenied;
}

public final class PrinterInfo {
    private String printerName = null;
    private String hostName = null;
    private int currentJob = (int)0;
    ...
}
```
6.5 Service side: provide an implementation of the service

```java
package printer;
import java.net.*; import java.io.*;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.WebParam;

@WebService(endpointInterface="printer.PrinterService")
public class Printer {
    private short currentJob;
    private String printerName;
    private String hostName;
    public Printer(String name) {
        printerName = name;
        try {
            hostName = InetAddress.getLocalHost().getHostName();
        } catch (UnknownHostException u) {
            hostName = "unknown computer";
        }
        System.out.println(hostName + " : Printer " + name + " instantiation ");
    }
    public Printer() {this("hp");}

    @WebMethod()
    public PrinterInfo getPrinterInfo() {
        return new PrinterInfo(printerName, hostName, currentJob);
    }
```
@WebMethod()
public JobInfo submitPrint (@WebParam(name = "fileName", targetNamespace = ")String fileName) \nthrows PrintDenied {
    // Open the file and get its size
    System.out.println("Printing document : " + fileName + " on " + printerName);
    File file = new File(fileName);
    if (file == null) //throw new PrintDenied(new PrinterInfo(printerName, hostName, currentJob), \n            fileName + " can't open file");
            throw new PrintDenied(fileName + " can't open file");
    if (!file.isFile()) //throw new PrintDenied(new PrinterInfo(printerName, hostName, currentJob), \n            fileName + " : is not a file");
            throw new PrintDenied(fileName + " : is not a file");
    if (currentJob > 5) {
        System.out.println("I am definitely overloaded, you must restart !");
        throw new PrintDenied("overloaded");
    }
    currentJob++;
    System.out.println("... " + " size " + file.length() + " Current Job : " + currentJob);
    return new JobInfo(currentJob, (int)file.length());
}
6.6 JAX-WS annotations

- **@WebService** Marks a Java class as implementing a Web Service, or a Java interface as defining a Web Service interface.

```java
// Link the class and the interface
@WebService(endpointInterface="printer.PrinterService")
public class Printer {

  // @WebMethod : Customizes a method that is exposed as a Web Service operation.
  // @OneWay : the web method has only an input message and no output
  // @WebParam : Customizes the mapping of an individual parameter to a Web Service message part and XML element
  // @WebResult : Customizes the mapping of the return value to a WSDL part and XML element.

  @WebMethod(operationName="add") // for different names in WSDL
  @WebResult(name="return")
  public int addNumbers(
      @WebParam(name="num1") int number1,
      @WebParam(name="num2") int number2);

  @OneWay
  public void checkIn(String name);
```
package printer;
import javax.xml.ws.Endpoint;

public class PrinterServer {

    protected PrinterServer() throws Exception {
        // START SNIPPET: publish
        System.out.println("Starting Server");
        Printer printer = new Printer();
        String address = "http://localhost:9000/PrinterService";
        Endpoint.publish(address, printer);
        // END SNIPPET: publish
    }

    public static void main(String args[]) throws Exception {
        new PrinterServer();
        System.out.println("Server ready...");

        Thread.sleep(360 * 60 * 1000);
        System.out.println("Server exiting");
        System.exit(0);
    }
}
6.8 Start the service

- Compile everything
- Starts the publisher
  
  ```java
  java -cp class printer.PrinterServerPublisher
  ```

- In a navigator, verify that the service is deployed
  
  ✦ http://localhost:9000/PrinterService

- And have a look on the wsdl
  
  ✦ http://localhost:9000/PrinterService?wsdl
6.9 Client side: generate the stub

- Produce the stub from the URL
  http://localhost:9000/PrinterService?wsdl with wsimport

```
wsimport -d class -s generated -keep http://localhost:9000/PrinterService?wsdl
```
6.10 Client side in java

- Write, compile and test a client

```java
import printer.*;
public class PrinterClient {
    public static void main(String[] args) {
        try {
            if (args.length < 1) {
                System.out.println("usage: java ClassName + " file_name");
                System.exit(-1);
            }
            String fileToPrint = args[0];
            PrinterService printerService = new PrinterService_Service().getPrinterPort();
            PrinterInfo printerInfo = printerService.getPrinterInfo();
            System.out.println("Printer :" + printerInfo.getPrinterName() + " \\
                               printInfo.getHost Name() + " \n\t on machine " + \\
                               printInfo.getHost Name() + " \n\t running task : " + printerInfo.getCurrentJob() + "\n");
            JobInfo jobInfo = printerService.submitPrint(fileToPrint);
            System.out.println("\n submitPrint(" + fileToPrint + ")" + "Job " + jobInfo.getJobNumber() + "; \\
                               + jobInfo.getJobSize()+ " bytes");
        } catch (Exception e) {System.out.println(e);}
    }
}
```
7 Conclusions

7.1 Comparison CORBA/RMI/WebServices ............................................ 56
7.2 Conclusions ................................................................................. 57
# 7 Conclusions

## 7.1 Comparison CORBA/RMI/WebServices

<table>
<thead>
<tr>
<th>Origin</th>
<th>CORBA</th>
<th>RMI</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>multiples</td>
<td>multiples</td>
<td>multiples</td>
</tr>
<tr>
<td>Prog. language</td>
<td>multiples</td>
<td>Java</td>
<td>multiples</td>
</tr>
<tr>
<td>IDL</td>
<td>IDL CORBA</td>
<td>interface Java</td>
<td>WSDL</td>
</tr>
<tr>
<td>data presentation</td>
<td>CDR</td>
<td>Java serialisation</td>
<td>SOAP Envelope</td>
</tr>
<tr>
<td>protocole</td>
<td>IIOP / GIOP</td>
<td>JRMP ou IIOP</td>
<td>SOAP Protocol above HTTP, SMTP, ...</td>
</tr>
<tr>
<td>connexions</td>
<td>connected</td>
<td>connected</td>
<td>short connexions</td>
</tr>
<tr>
<td>object references</td>
<td>IOR (location independant)</td>
<td>symbolic names/IP+port</td>
<td>URL (included in WSDL)</td>
</tr>
<tr>
<td>naming service</td>
<td>NS, trading</td>
<td>RMI registry, JNDI</td>
<td>UDDI, WSIL</td>
</tr>
<tr>
<td>main advantages</td>
<td>services/efficiency</td>
<td>easy to use in java</td>
<td>messages auto described</td>
</tr>
<tr>
<td>main difficulties</td>
<td>complex to learn</td>
<td>Java/Java</td>
<td>lack of services</td>
</tr>
</tbody>
</table>
7.2 Conclusions

■ Granularity of distribution variable (object, service),
■ Complexity of distribution
■ Synchronous request middleware is the necessary foundation to build higher level middleware
  ♦ Middleware services (e.g. name service, yellow pages)
  ♦ Message Oriented Middleware (MOM) (asynchronous middleware)
  ♦ Application servers
  ♦ Component middleware
  ♦ Compositions and orchestrations
References


... and omg.org, w3c.org