Web Services

Distributed Systems
Sistemi Distribuiti

Andrea Santi
a.santi@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011

Outline

Outline

Reference Material

- This presentation is rooted on some of the reference books on the topic [Erl, 2005, Richardson and Ruby, 2007]
- Most of the content of those slide has been re-adapted from the books [Erl, 2005, Richardson and Ruby, 2007] and integrated with new material according to the personal view of speaker about this topic
- Eventual mistakes/problems are the sole responsible of the lecturer
- For a more comprehensive picture regarding this topic the cited books [Erl, 2005, Richardson and Ruby, 2007] and some other on-line documentation [Corp., 2011] is a must (and recommended) read

Outline

Web Services: one of the Buzzwords of the 21th Century

- Web Services are currently causing a lot of confusion in the IT world
- IT professionals, researchers, anyone claim their own interpretation
  - causing a lot of trouble and misunderstanding
- Leading to a not so clear picture on this topic after more then a decade of debate

So, Web Services: What are they?

- Good question, and it raises lots of others
- When use them, and for what?
- Which architectural style should I use?
  - Service-Oriented Architecture vs Resource-Oriented Architecture
- A Web Site is a Web Service?
  - Even the answer to this question is now no more so clear [Richardson and Ruby, 2007] ...
- ...

Tentative definition

Web services (WS) are client and server applications that communicate via *message-based interactions* over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP) [Corp., 2011].

Main Features

- A WS encapsulate a unit of logic/functionality within a certain context
- Functionalities provided described by means of a proper *contract*
  - Explicit (SOA) Implicit (in most cases in ROA)
- Autonomy
- Loose coupling
- Composability
- Reusability
- Multi-vendor support and interoperability

Why studying WS in this course?

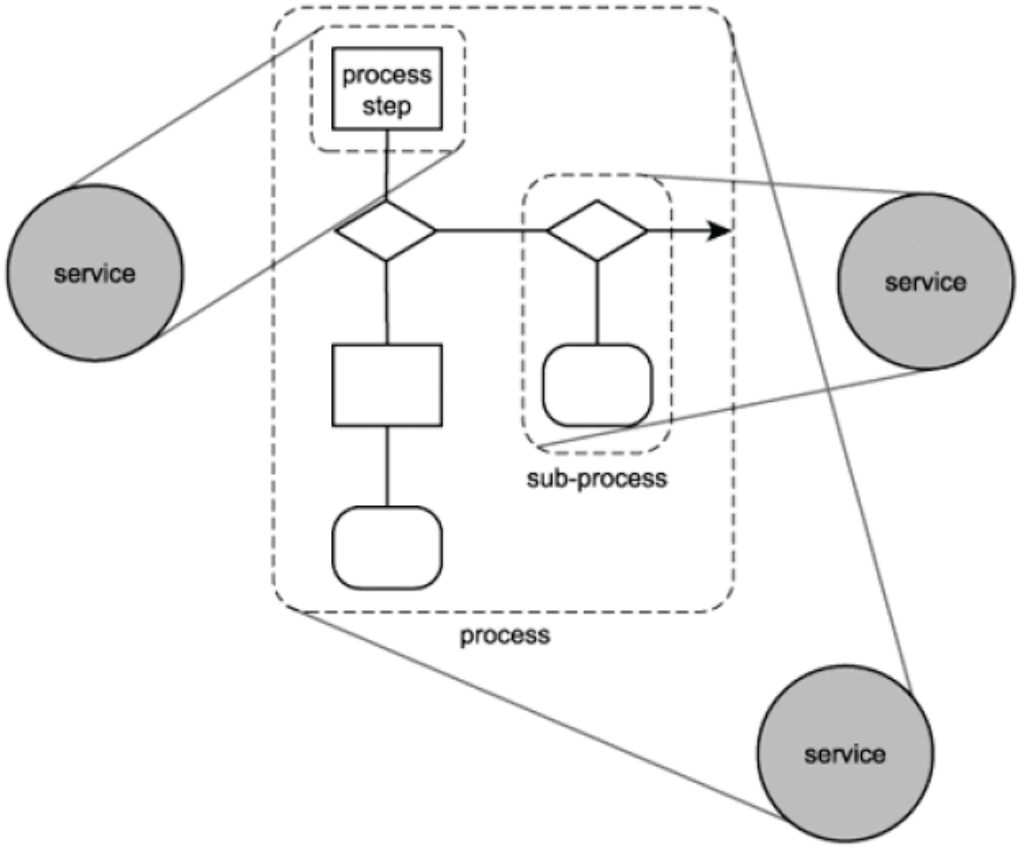
Web-Services are today the reference stack of protocols for building interoperable distributed systems

- Enabling technology for different styles of communication
  - Message passing
  - Remote Procedure Call (RPC)
- Enabling interoperability thanks to a set of well defined standards
  - Between vendor-diverse applications
  - Between legacy and new applications

Outline

How service encapsulate logic?

- Services encapsulate logic within a distinct context
- This context can be specific to a business task, a business entity, or some other logical grouping

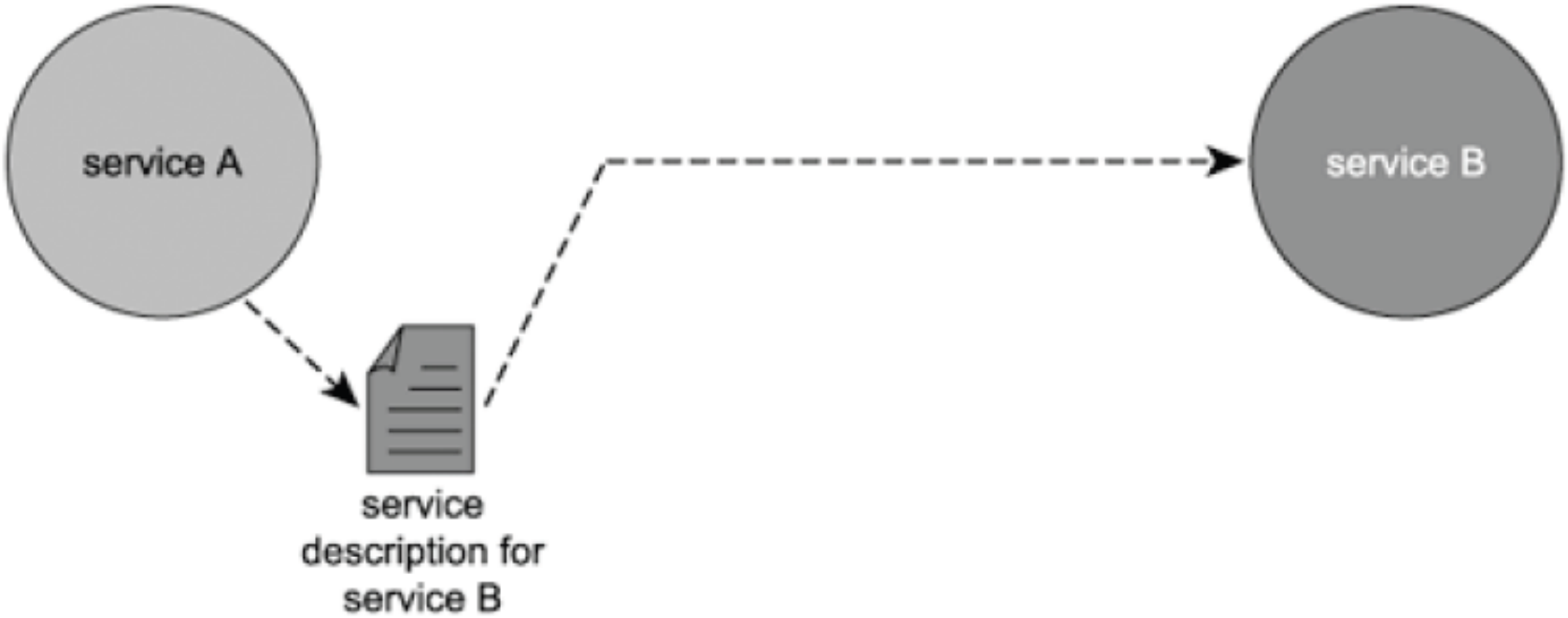


How service relates? 1/2

- Services relationship is based on an understanding that for services to interact, *they must be aware of each other*
- This awareness is achieved through the use of *service descriptions*
- A service description establishes (at least)
  - The name of the service
  - The data expected and returned by the service
- The manner in which services use service descriptions results in a relationship classified as *loosely coupled*

How service relates? 2/2

- Service A is aware of service B because service A knows service B's service description
- Knowing B's service description, service A has all of the information it needs to communicate with service B

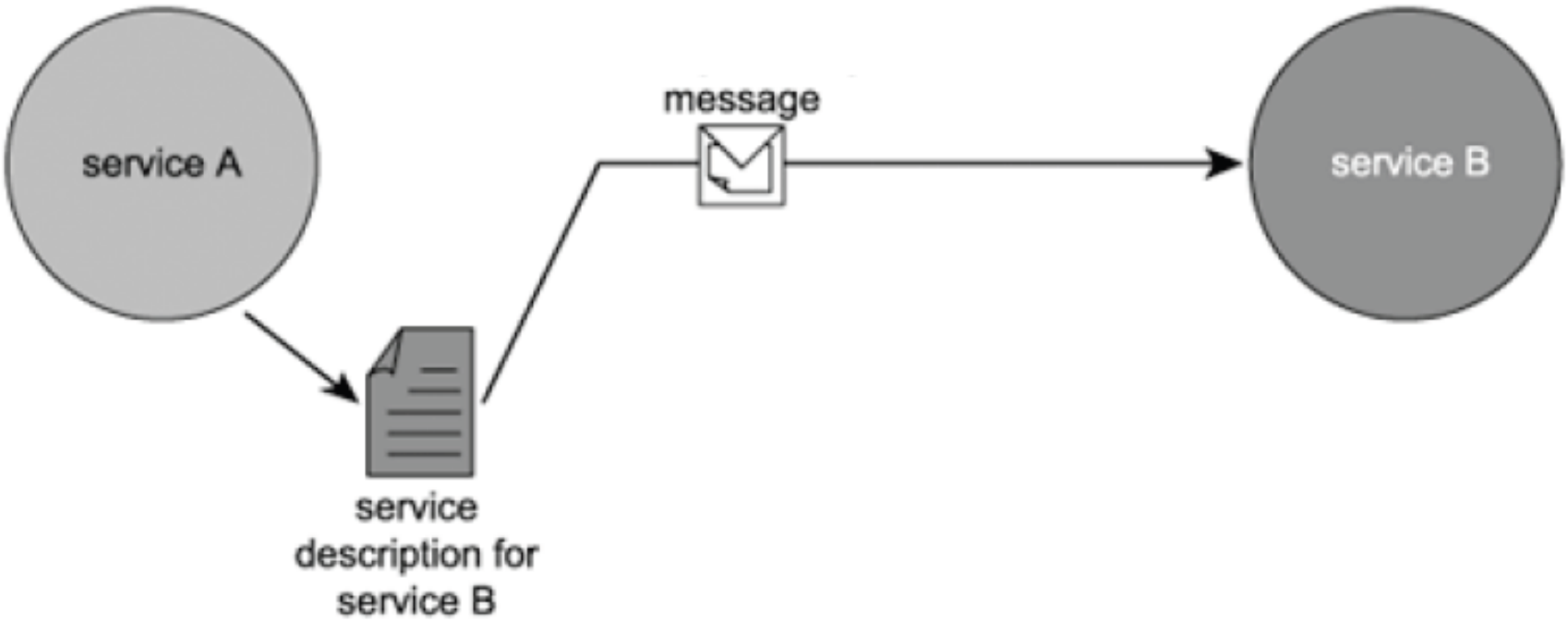


How service communicates? 1/2

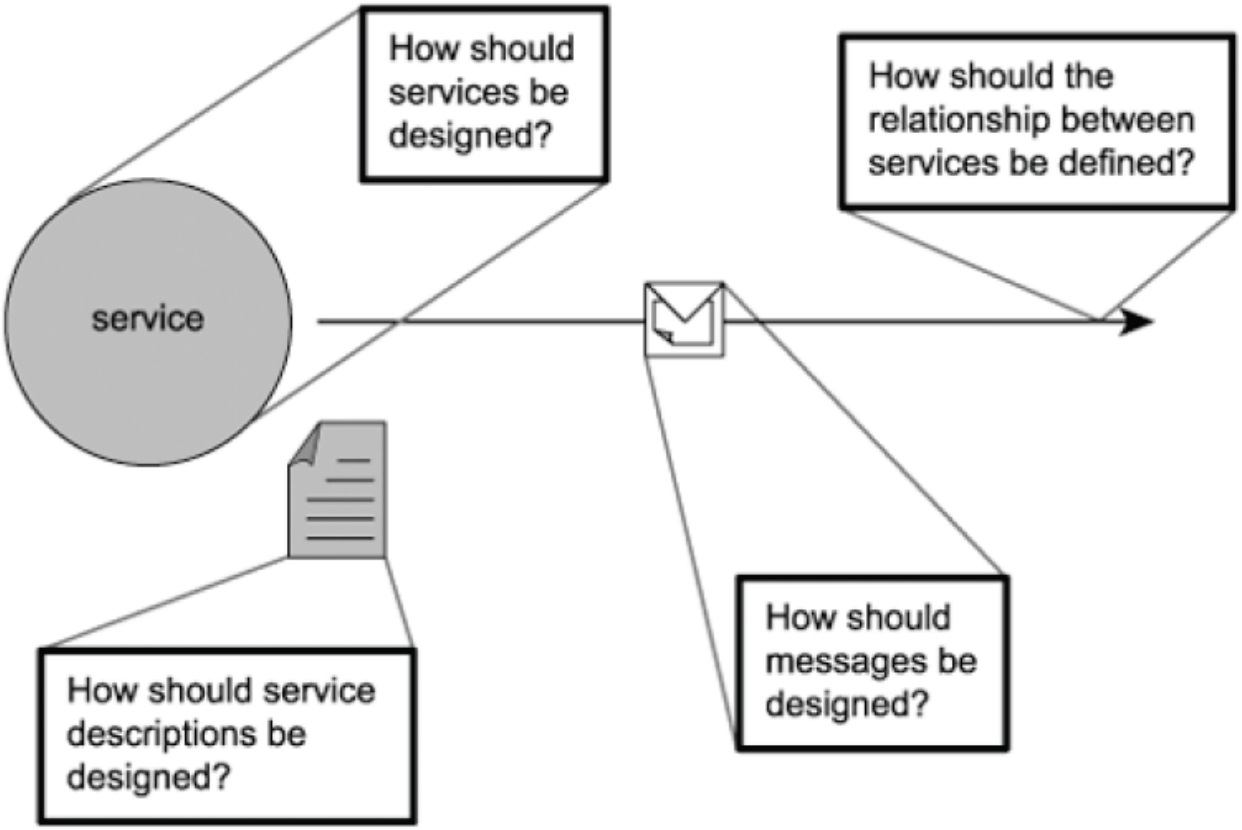
- Services communicates by means of proper *exchanges of messages*
- After a service sends a message on its way, it loses control of what happens to the message thereafter
- Supported style of communication
  - Asynchronous communication
  - Synchronous communication

How service communicates? 2/2

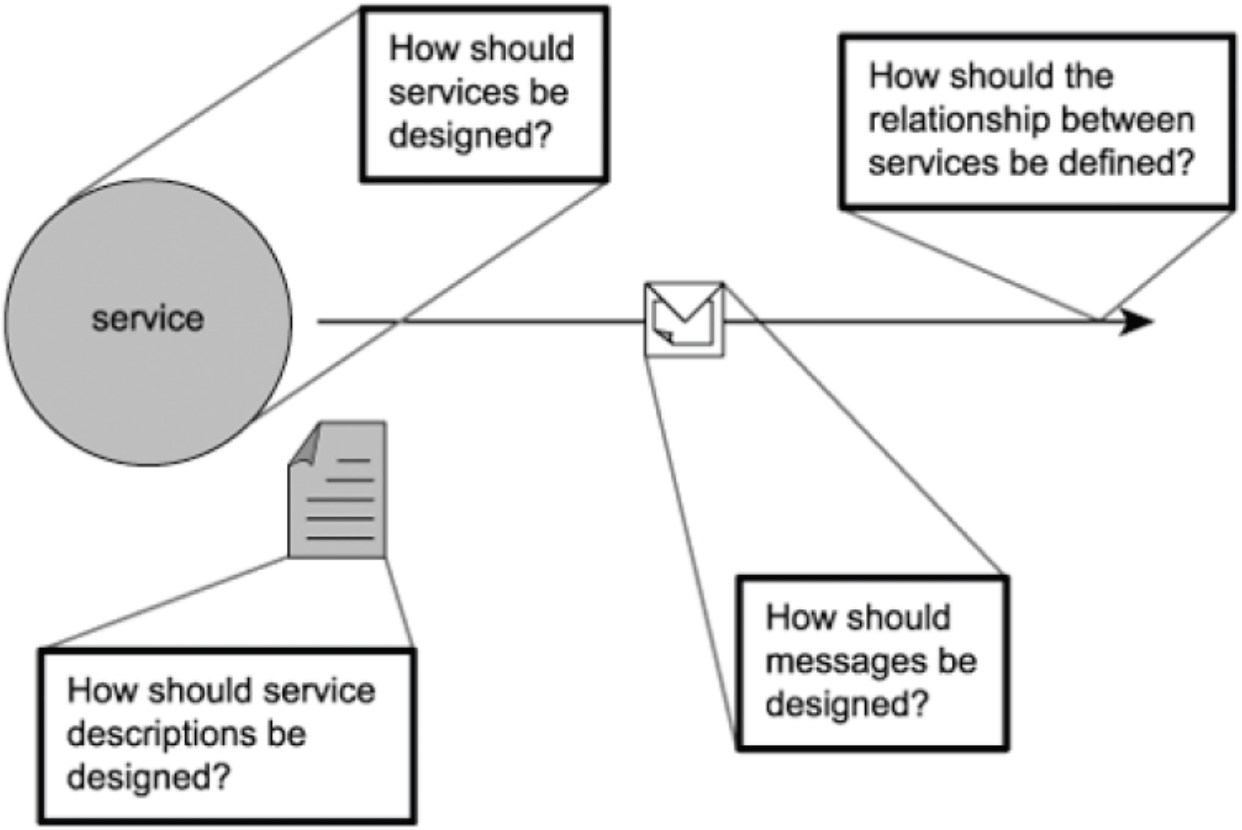
A simple communication example



How design all those things?



How design all those things?



Two different architectural approaches

- Service Oriented Architecture (SOA)
  - HTTP — as the *underlying* transport protocol
  - SOAP — as the *real* transport protocol
  - WSDL — for service description
  - XML — for formatting the messages exchanged
  - WS-* — A set of specification for handling high-level application features
- Resource Oriented Architecture (ROA)
  - HTTP — as the real transport protocol
  - XML — for formatting the messages exchanged
  - WADL — for service description (in early development stages)

Outline

What is a Service Oriented Architecture (SOA)?

A *"formal"* definition

SOA can be defined as an *open*, *agile*, *extensible*, *federated*, *composable* architecture *comprised of autonomous*, *QoS-capable*, *vendor diverse*, *inter-operable*, *discoverable*, and *potentially reusable* services [Erl, 2005]

Main features of the Service-Oriented Architectural model

- A service encapsulate a unit of logic within a certain context
- Loose coupling and message-based interactions
- Autonomy
- Composability
- Reusability
- Multi-vendor support and interoperability

Well, wait... something sound familiar...

- Do you find any similarities with the Web Service definition provided a few slide before?
- We are using two terms for referring to the same thing?
  - No...
- So, Web Services $\leftrightarrow$ SOA-based application?
  - Partially true but...

SOA & Web Services: let's make things clear 1/2

- SOA and Web Services are not synonyms!!!
  - The former it's a *definition* of an architecture (principles, features...)
  - The latter is a *concrete implementation* of the service-oriented architectural model
- Web Services are the *reference* framework providing a *concrete implementation* of the service-oriented architecture
- A WS-based application is not necessarily a SOA-based application
  - e.g. WS used just for enabling RPC
  - A SOA-based application must adhere to the basic SOA features (e.g. loose coupling, service autonomy, etc.)

SOA & Web Services: let's make things clear 2/2

- Why all this confusion then?
- SOA is intrinsically reliant on Web services so much so that Web services concepts and technology used to actualize service-orientation have influenced a number of the SOA characteristics identified before [Erl, 2005].
- But in reality the features we have described in "Web Service Fundamentals" are SOA founding features
  - Supported by the reference implementation of the service-oriented architectural model

Outline

Designing SOA-based Web Services

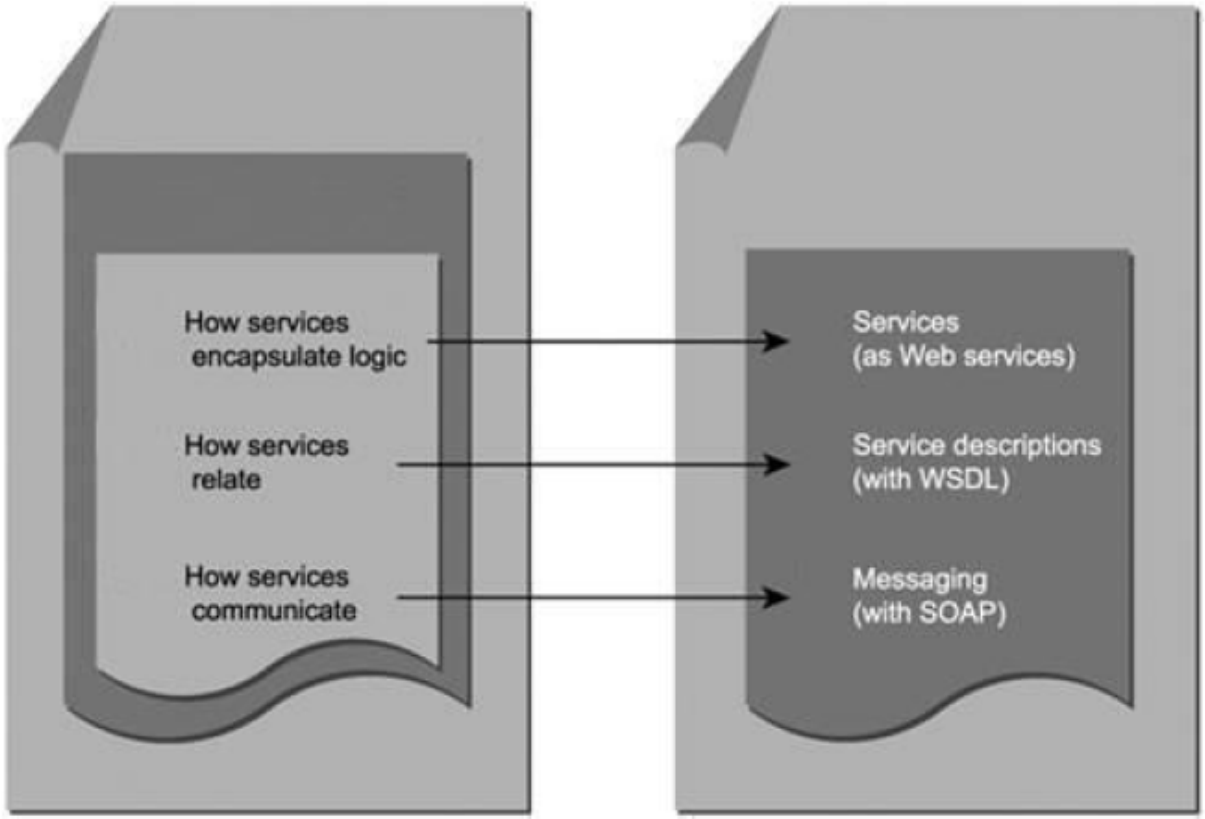


Figure: Mapping of SOA concepts into the WS framework [Erl, 2005]

An Hello World example

- Classical entry-level example
- One Web Service that prints in standard output the message "Hello X" where X is the person/thing to greet
- Try it on your PC starting from the material provided
  - Download it from the standard course website

Services (as Web Services)

Web Service (WS): main features

- Technology abstraction used for concretely implement a *service* in a service-oriented fashion
- It can be designed to duplicate the behavior and functionality found in proprietary distributed systems, or it can be designed to be fully SOA-compliant
  - therefore Web services are not necessarily inherently service-oriented

A Web Service can be associated with...

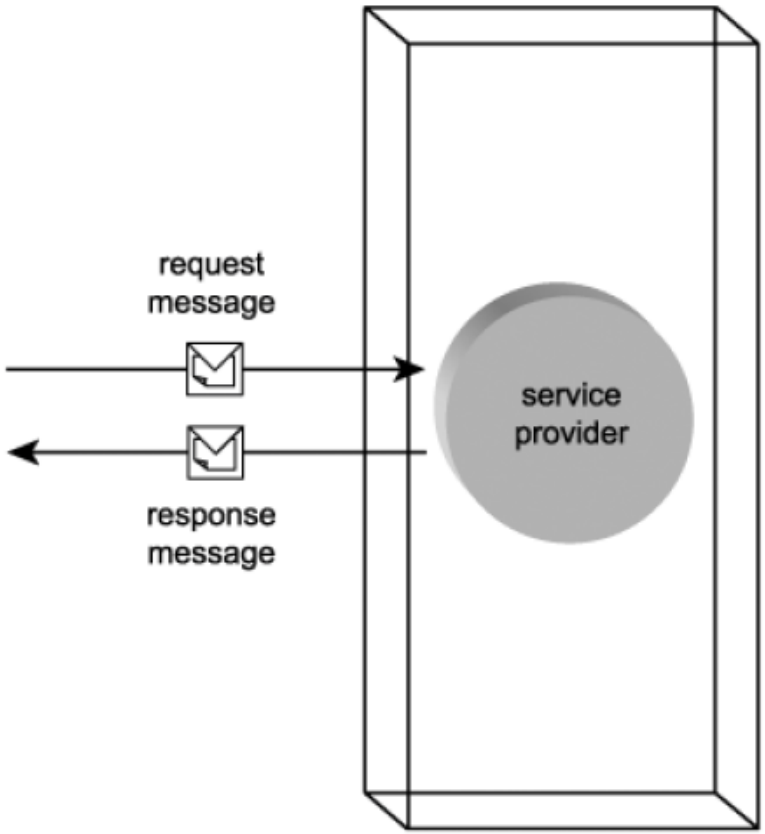
A service role — runtime classification depending on its responsibility in a given scenario (initiator - requestor - intermediary)

A service model — permanent classification depending the role played by the WS into an application (broker - utility service...)

Service provider role

A WS recipient of a request message is classified as a service provider

- The WS is invoked by an external source
- The WS provides a published service description (WSDL)



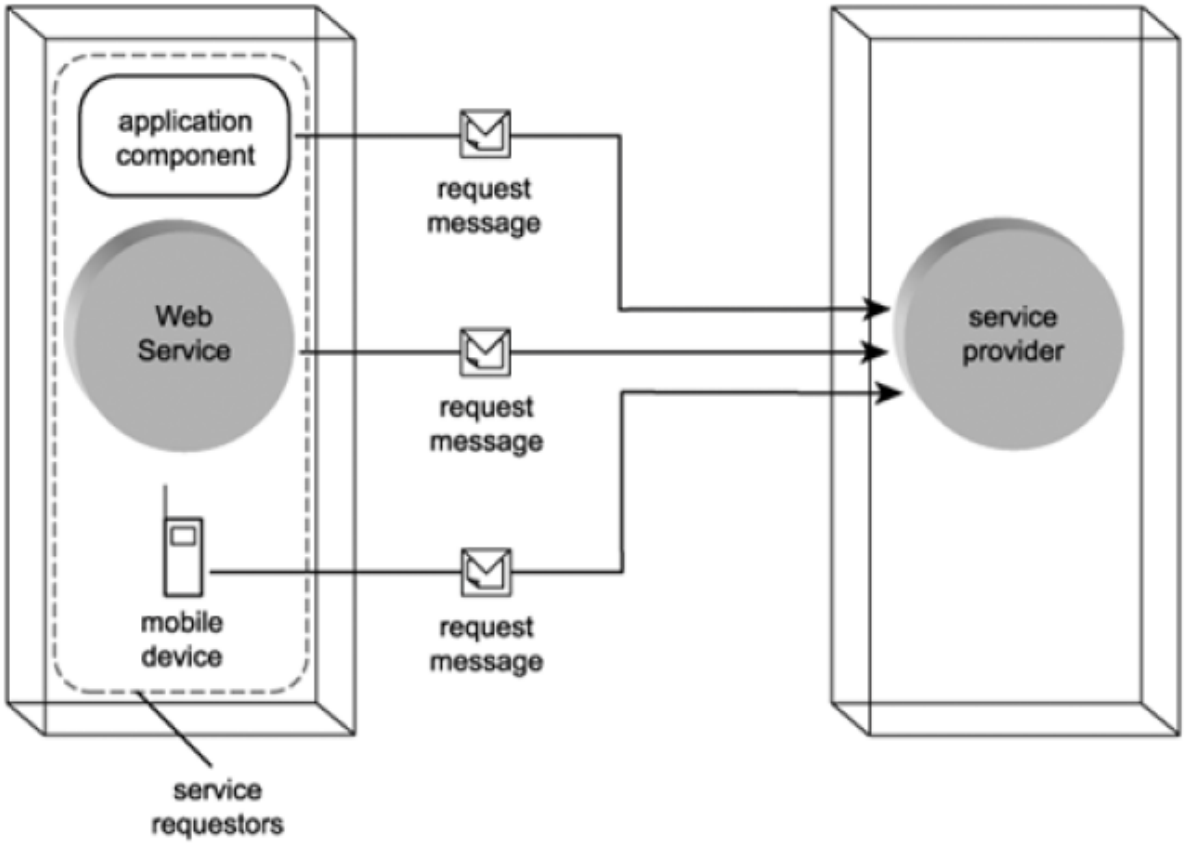
Service provider in our example

- The HelloService Web Service is the service provider
  - It provides the really basic greeting service
- Request an input message containing the person/thing to greet
- Provides as output a message containing the greeting

Service requestor role

The sender of a request message is classified as a service requestor

- The WS invokes a service provider by sending it a message
- The WS searches for the most suitable service provider studying available service descriptions



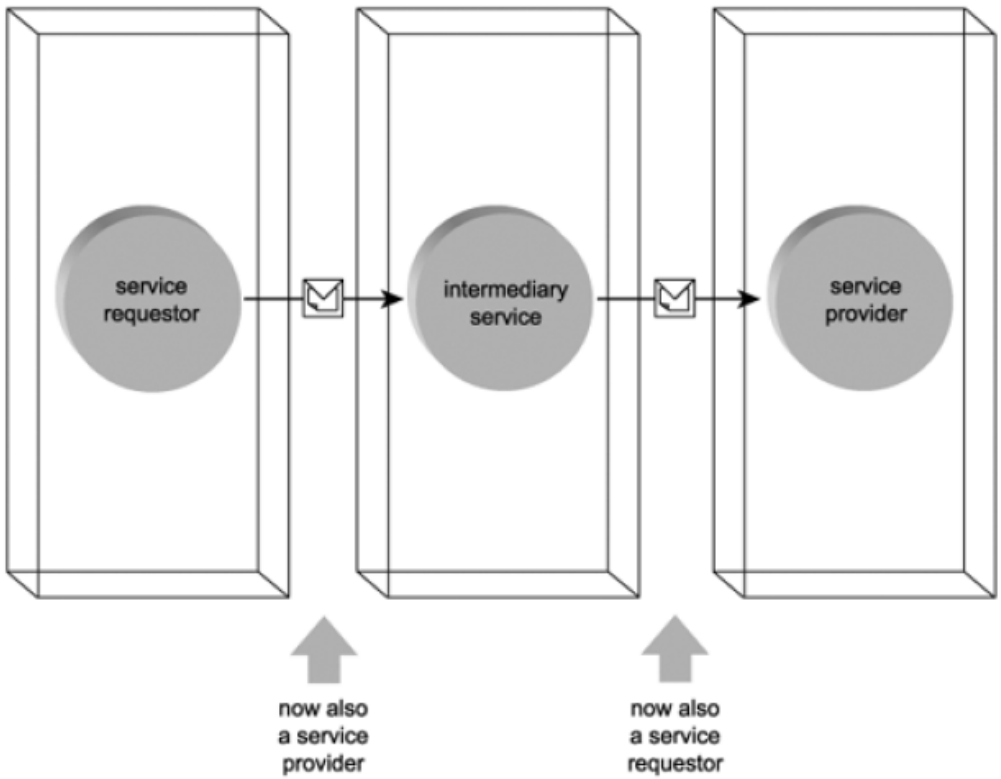
Service requestor in our example

- The Java application exploiting the HelloService Web Service
- Invokes the Web Service providing the appropriate input message
- Retrieve the desired response message

Service intermediary role

A message can be processed by multiple intermediate before its final destination

- Passive intermediaries: simply route messages
- Active intermediaries: route messages to a forwarding destination actively processing/altering the message contents



Simple Object Access Protocol (SOAP) 1/2

Definition

The standard transport protocol for messages processed by Web services

- HTTP is used as the underlying transport protocol for SOAP messages

- Originally designed to replace proprietary RPC protocols (i.e. serialization of object)
- Now, despite the name, the purpose is to define a standard message format
  - Important remark: others transport protocols can be used as well
- Extremely flexible and extensible
  - has been revised several times to accommodate more sophisticated features and message structures

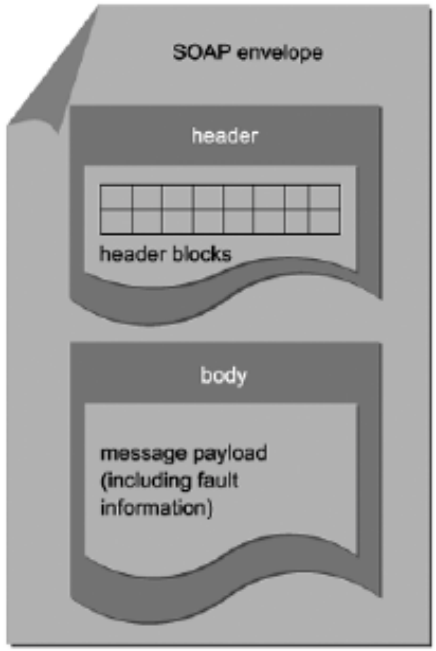
Simple Object Access Protocol (SOAP) 2/2

Structure of a SOAP message

envelope — the message container: house all parts of the message

header — dedicated to hosting meta-information (used by WS-* specifications, described next)

body — the message content (i.e. XML-formatted data)



The SOAP request message in our Example

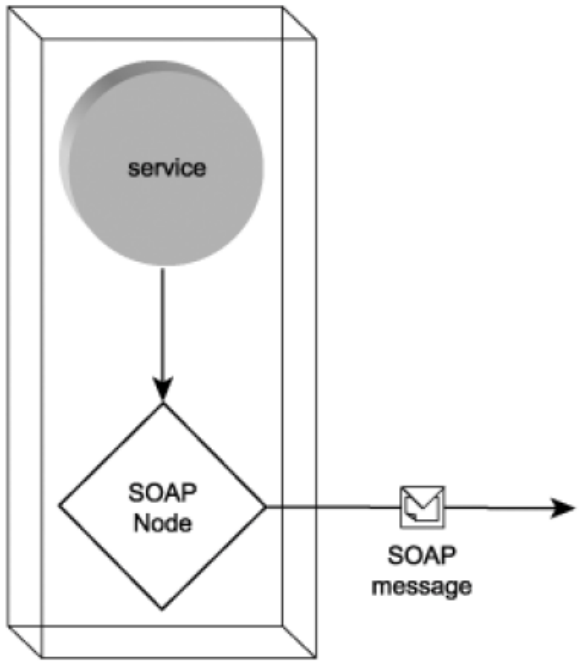
```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:sayHello
           xmlns:ns2="http://endpoint.helloservice/">
            <arg0>John</arg0>
        </ns2:sayHello>
    </S:Body>
</S:Envelope>
```

The SOAP response message in our Example

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:sayHelloResponse
          xmlns:ns2="http://endpoint.helloservice/">
            <return>Hello, John.</return>
        </ns2:sayHelloResponse>
    </S:Body>
</S:Envelope>
```

SOAP Nodes

- WS are self-contained units of processing logic, but they are reliant upon a physical communications infrastructure
- Every platform has its own implementation of SOAP communications
- In abstract, the programs that services use to transmit/receive SOAP messages are referred as *SOAP nodes*



Web Service Description Language (WSDL) 1/2

- XML-based language used for defining *service descriptions*
- A WSDL document define
  - The functionalities provided by the service
  - The service behavior

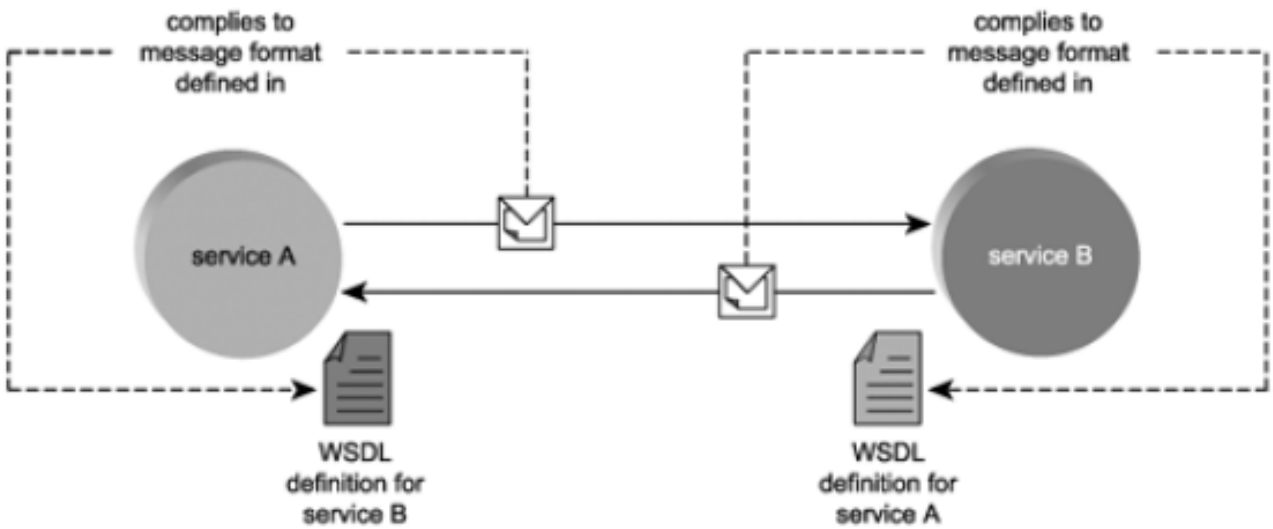


Figure: WSDL definitions enable loose coupling between services

Web Service Description Language (WSDL) 2/2

Parts of a WSDL document

A WSDL service description is composed of two parts

- An abstract description
- A concrete description

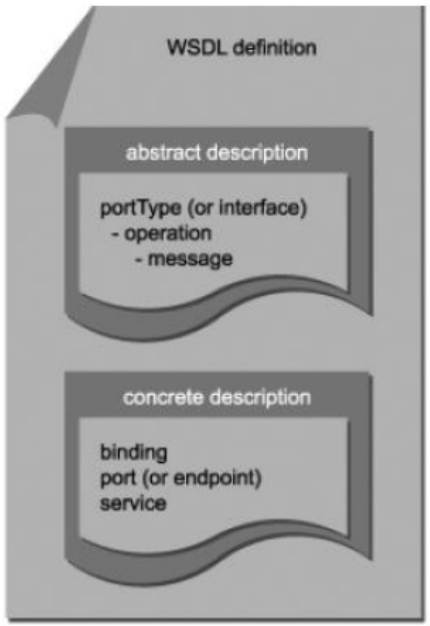


Figure: A WSDL document abstract representation

WSDL abstract description

Abstract description purpose

Establish the interface characteristics of the Web Service without any reference to

- The technology used for realize the Web Service
- The technology used for transmit/receive messages

Abstract description elements

portType is a high-level view of the service interface by sorting the *messages* a service can process into groups of functions known as *operations*

operation is a specific action performed by the service

message is the abstraction used for describe operation's input/output

WSDL concrete description

Concrete description purpose

Establish the physical connection (*binding*) of the WSDL abstract description to a physical transport protocol

Concrete description elements

binding describes the requirements (i.e. the transport protocol) for establishing a physical connection with the Web Service

service define the WS name and the set of service *port*s (i.e. all the possible service contact addresses)

port is the physical address at which a service can be accessed with a specific protocol

SOAP & WSDL

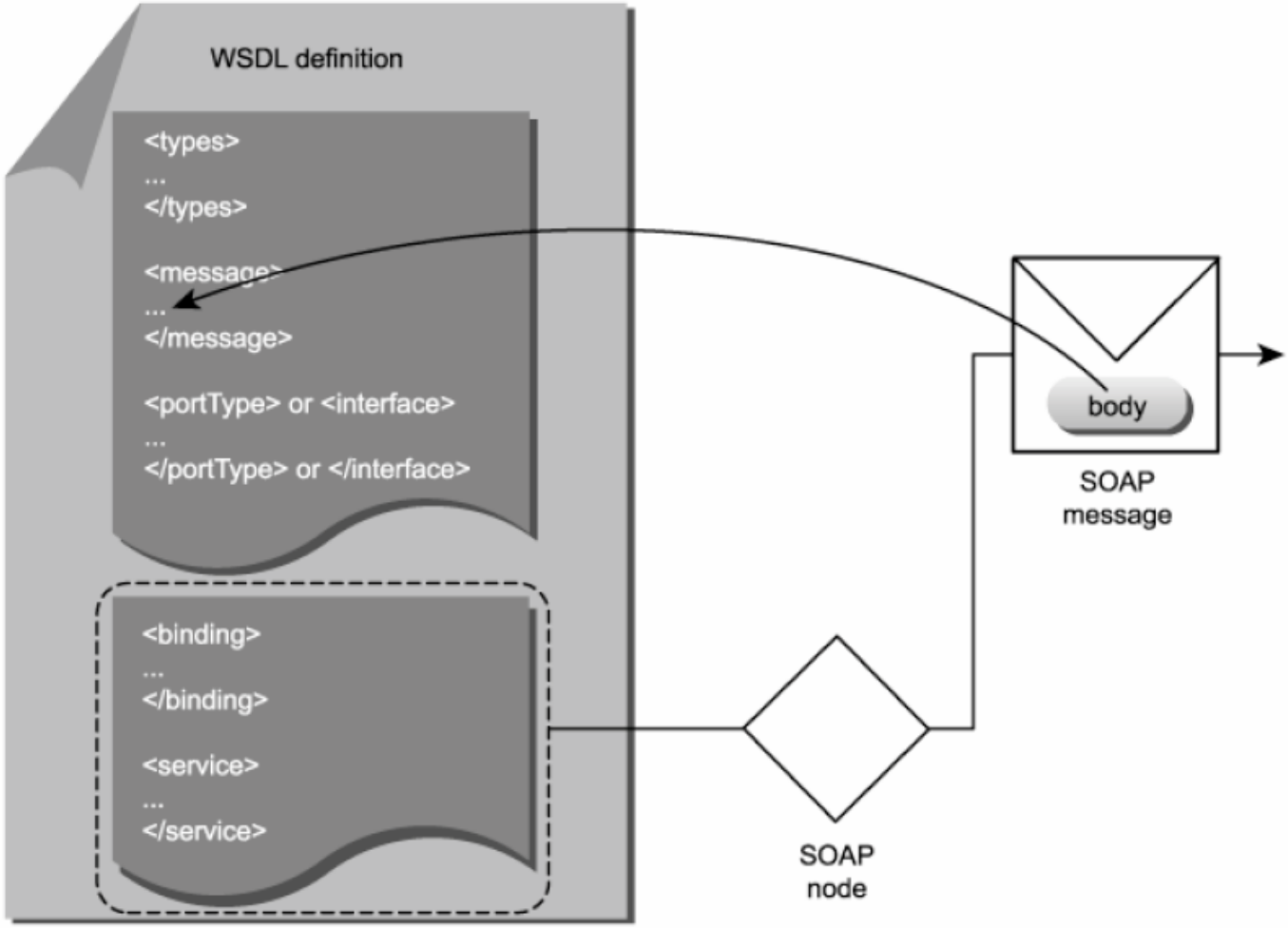


Figure: Relation between a SOAP message and its related WSDL document

The WSDL of the Hello Service 1/3

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Generated by JAX-WS ..>
<definitions xmlns:wsp1_2="http://schemas.xmlsoap.org/ws..."
  xmlns:tns="http://endpoint.helloservice/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://endpoint.helloservice/" name="HelloService">

<!-- Import of the XML data-types used -->
<types>
  <xsd:schema>
    <xsd:import namespace="http://endpoint.helloservice/"
      schemaLocation="http://localhost:8080/helloservice/HelloService?xsd=1" />
  </xsd:schema>
</types>

<!-- Messages definition-->
<message name="sayHello">
  <part name="parameters" element="tns:sayHello" />
</message>
<message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse" />
</message>
```

The WSDL of the Hello Service 2/3

```
<!-- PortType definition-->

<portType name="Hello">

    <operation name="sayHello">

    <!-- Definition of the input message for the Hello operation -->
    <input wsam:Action="http://endpoint.helloservice/Hello/sayHelloRequest"
        message="tns:sayHello" />

    <!-- Definition of the output message for the Hello operation -->
    <output wsam:Action="http://endpoint.helloservice/Hello/sayHelloResponse"
        message="tns:sayHelloResponse" />

  </operation>

</portType>
```

The WSDL of the Hello Service 3/3

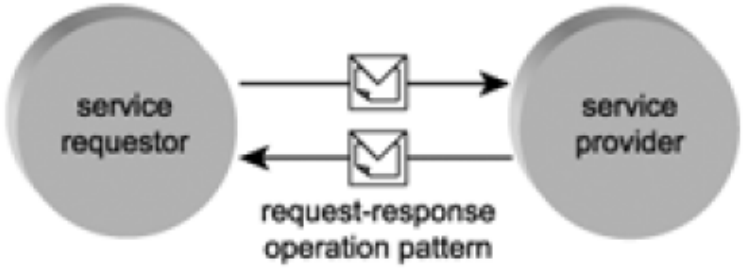
```
<!-- PorType binding definition -->
<binding name="HelloPortBinding" type="tns:Hello">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="sayHello">
    <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
  </operation>
</binding>

<!-- Service definition -->
<service name="HelloService">
  <port name="HelloPort" binding="tns:HelloPortBinding">
    <!-- Service address -->
    <soap:address location="http://localhost:8080/helloservice/HelloService" />
  </port>
</service>
</definitions>
```

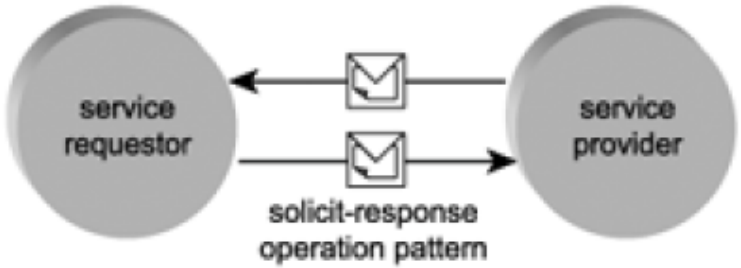
Message Exchange Pattern (MEPs)

- Definition of all the possible communication interaction dynamics between Web Service
- A group of already mapped out sequence for the exchange of messages
- Like design patter in software engineering but oriented to the *message exchange dynamics*

WSDL 1.1 supported MEPs 1/2

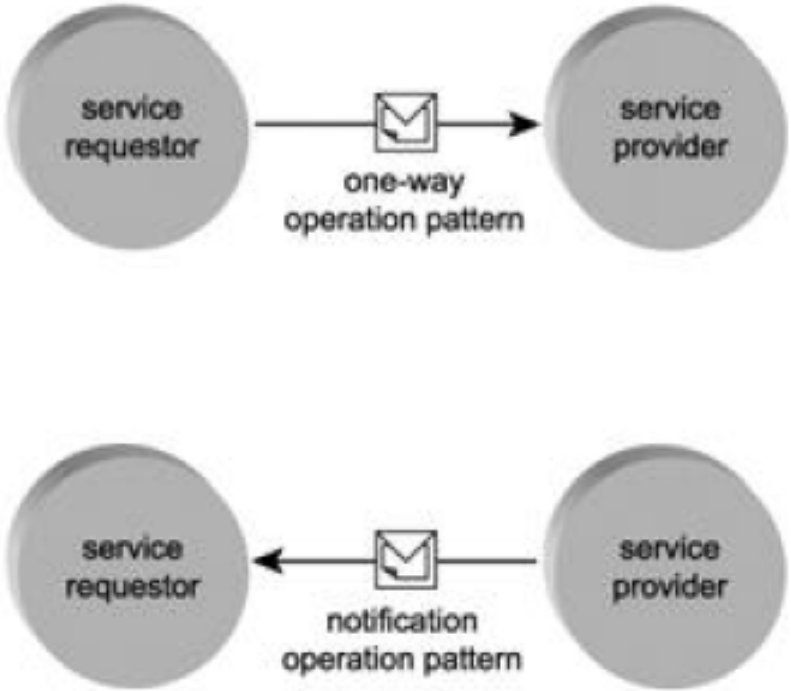
- Request-Response
- Solicit-Response





WSDL 1.1 supported MEPs 2/2

- One-way
- Notification



WSDL 2.0 supported MEPs

Old MEPs, but with new names

In-out	equivalent to the Request-Response pattern
Out-in	equivalent to the Solicit-Response pattern
In-only	equivalent to the One-way pattern
Out-only	equivalent to the Notification pattern

New MEPs, introduced by WSDL 2.0

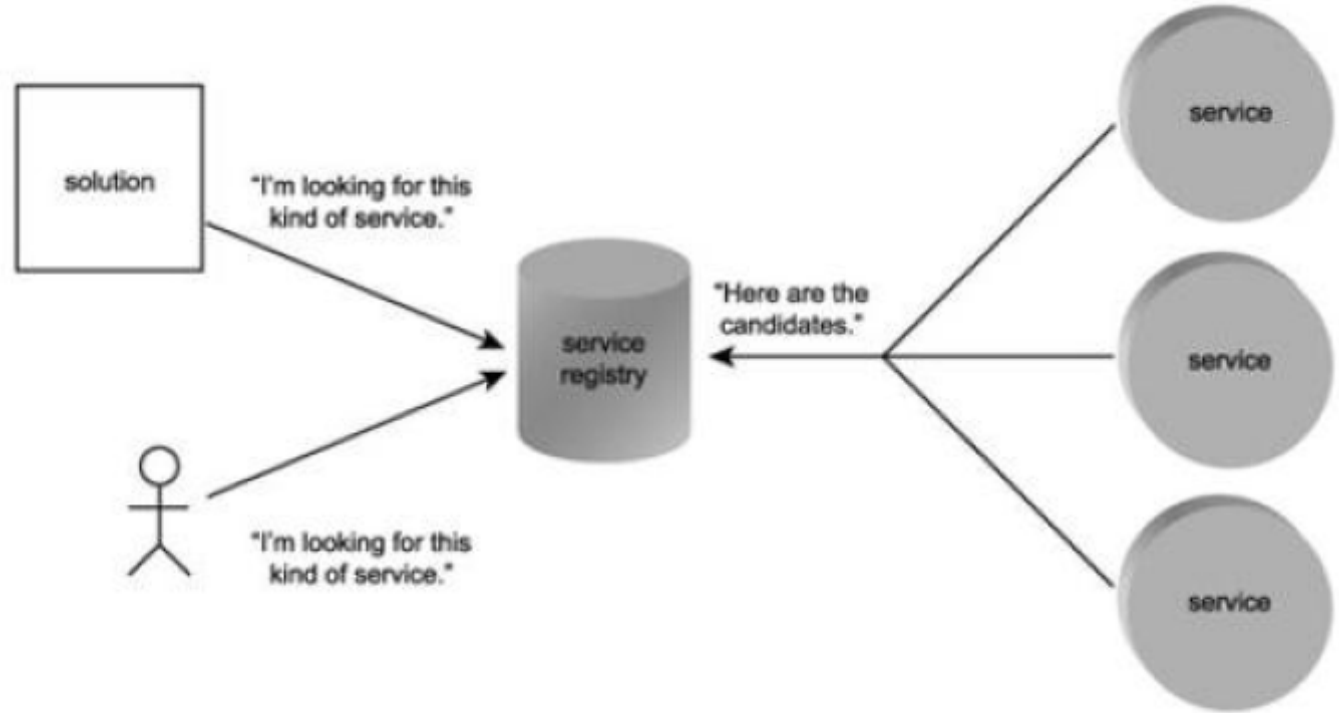
Variations of the basic four MEPs, in addition provides optional in/out message or fault response message

- Robust in-only
- Robust out-only
- In-optional-out
- Out-optional-in

Universal Description Discovery and Integration (UDDI)

OASIS standard that *try* to address the issues related to service discovery and composition

- Functionalities advertising by registering the WS's WSDL into the UDDI registry
- Service requestors search functionalities offered by Web Services simply querying the registry



UDDI problems and limitations

Main problems: no semantics issues are considered!

- Without addressing semantic issues Web Service discovery and composition can not be successfully handled
- UDDI service advertising/discovery only rely upon syntactic aspects
  - Full signature-match for an operation is required
  - Otherwise how infer that a functionality (i.e. a WS operation) such as *rent a vehicle* is the same of *rent a car*?

UDDI isn't so much widespread yet

For taking advance of WS discovery and composition by means of UDDI are required

- A widespread diffusion of the public UDDI registries
- The registration of a high number of WSs

Outline

Web Service Tools overview

- Brief overview of the architecture of two main Web Service stack implementations
  - Java Metro (GlassFish)
  - Apache Axis2 [The Apache Software Foundation, 2004]
- JAX-WS specification [Sun Microsystems, 2004]
  - standard Java-based programming model supported by both

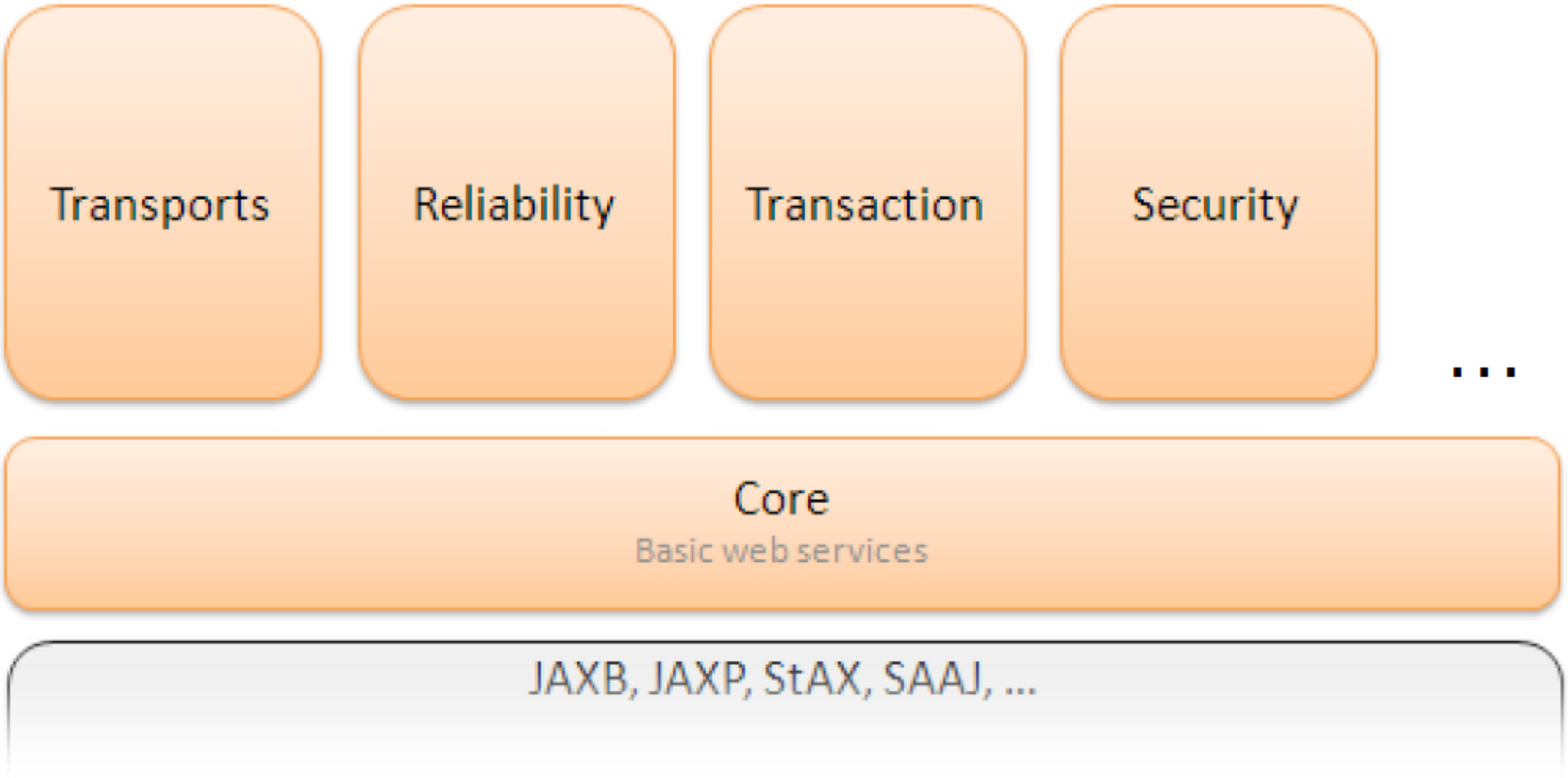
Java Metro

- High-performance, extensible, easy-to-use web service stack.
- Proposed as a *one-stop shop for all your web service needs*
  - from the simplest hello world web service...
  - ... to reliable, secured, and transacted web service that involves .NET services
- Part of the GlassFish Application Server
  - but it can be also used outside GlassFish

Java Metro and WSIT

- Metro includes WSIT (Web Services Interoperability Technologies)
  - previously known as Project Tango
- It includes implementations of:
  - WS-Trust
  - WS-SecureConversation
  - WS-SecurityPolicy
  - WS-ReliableMessaging
  - WS-AtomicTransactions/Coordination
  - WS-MetadataExchange
  - SOAP over TCP
- Interoperability between the Java platform and Windows Communication Foundation (WCF) (aka Indigo) in .NET 3.0 and .NET 3.5

Metro functionalities



Axis 2

- Java platform for creating and deploying web services applications
  - Born from the Apache implementation of the SOAP specification
- First version: Axis
  - RPC-perspective on Web Services
- New version: Axis 2
  - Web Services in the SOA perspective

Axis 2 features

- Flexible, efficient and configurable architecture
  - supporting SOAP 1.1, SOAP 1.2, REST style of Web services .
  - the same business logic implementation can offer both a WS-* style interface as well as a REST/POX style interface simultaneously
- Modular and XML-oriented
  - it is carefully designed to support the easy addition of plug-in *modules* that extend their functionality for features such as security and reliability
- Modules currently available:
  - WS-ReliableMessaging (Apache Sandesha2)
  - WS-Coordination and WS-AtomicTransaction (Apache Kandula2)
  - WS-Security (Apache Rampart)
  - WS-Addressing (part of Axis2 core)

API Standardizations: JAX-WS 2.0

- It is a specification...
  - so different implementations (e.g. Axis2, Java Metro,..)
- ... of a programming model (= set of API)
  - Java-based
- ...that aims at simplifying application development through support of a standard, annotation-based model to develop Web Service applications and clients in Java
- Document-centric messaging model, replacing the remote procedure call programming model as defined by JAX-RPC
  - SOA perspective

Quick Overview of JAX-WS 2.0

- Simpler way to develop/deploy Web services (w.r.t. JAX-RPC)
  - Plain Old Java Object (POJO) can be easily exposed as a Web service
  - No deployment descriptor is needed - use Annotation instead
  - Layered programming model
- Part of Java SE 6 and Java EE 5 platforms
- Integrated data binding via JAXB 2.0
- Protocol and transport independence

Server-side: two basic ways for building Web Services

- Starting from a WSDL file (top-down approach)
  - Generate classes using ws import
    - WS interface
    - WS implementation skeleton class
  - Add business logic to the WS implementation class
  - Build, deploy, and test
- Starting from a POJO (bottom-up approach)
  - Annotate POJO
  - Build and deploy
  - WSDL file generated automatically

Server-side: an example starting from a POJO

```
import javax.jws.WebService;

@WebService
public class Calculator {
  public int add(int a, int b) {
    return a+b;
  }
}
```

- **@WebService** annotation
  - all public methods become web service operations
- WSDL/Schema generated automatically

Client-side programming 1/2

- The process for creating a web service client application will always start with an existing WSDL document
- Point a tool (e.g. wsimport) at the WSDL for the service
  - wsimport http://example.org/calculator.wsdl
- The tool generates the corresponding Java source code for the described interface
  - JAXB used for providing WSDL $\leftrightarrow$ Java data-binding
- Call new on the service class
- Get a proxy using a get*ServiceName*Port method
- Invoke any remote operations

Client-side programming 2/2

```
CalculatorService svc = new CalculatorService();
Calculator proxy = svc.getCalculatorPort();
int answer = proxy.add(35, 7);
```

- No need to use factories
- The code is fully portable
- XML is completely hidden from programmer

Principal annotations

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

@WebMethod Customises a method that is exposed as a Web Service operation

@WebParam Customises the mapping of an individual parameter to a Web Service message part and XML element

@WebResult Customises the mapping of the return value to a WSDL part and XML element

Outline

The SOA/WS evolution

The first WS generation introduced the framework building blocks and the basic specifications: WSDL, SOAP, UDDI...

The second generation of Web Service

With the second generation of WS has been introduced a set of specification (WS-*) for the managing of advanced functionalities:

WS-Coordination provides the rules for coordinating complex activities (AtomicTransactions , BusinessActivities) between WSs

WS-Security framework is a set of security specifications that provides authentication, authorization, data integrity and so on...

WS-BPEL define a language for specifying business process behavior based on Web Services

Many others WS-MetadataExchange, WS-Choreography, WS-Federation..

WS-Coordination

Main features

- Define a general-purpose framework for managing complex activities
- Rooted on a general model for coordinating the common part of different complex activities
  - i.e different coordination activities can be coordinated using the same coordination model
- Aspects related to a particular coordination type are defined into a separated specification

Supported coordination types

Currently only two coordination types are supported

- WS-AtomicTransaction
- WS-BusinessActivities

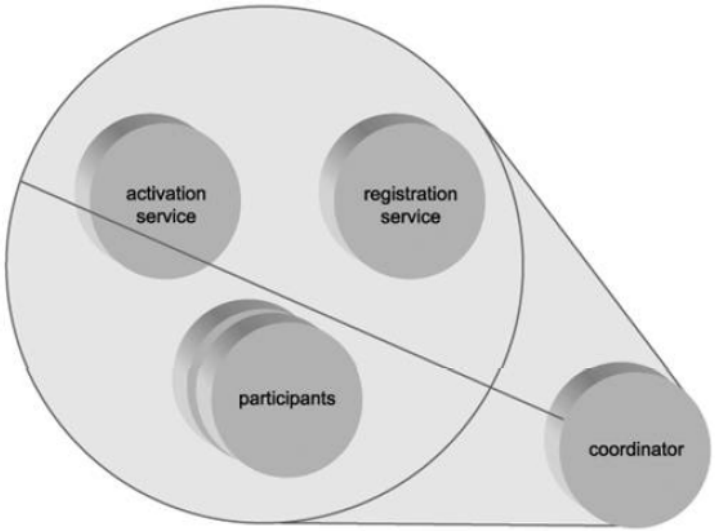
WS-Coordination general model

Service involved

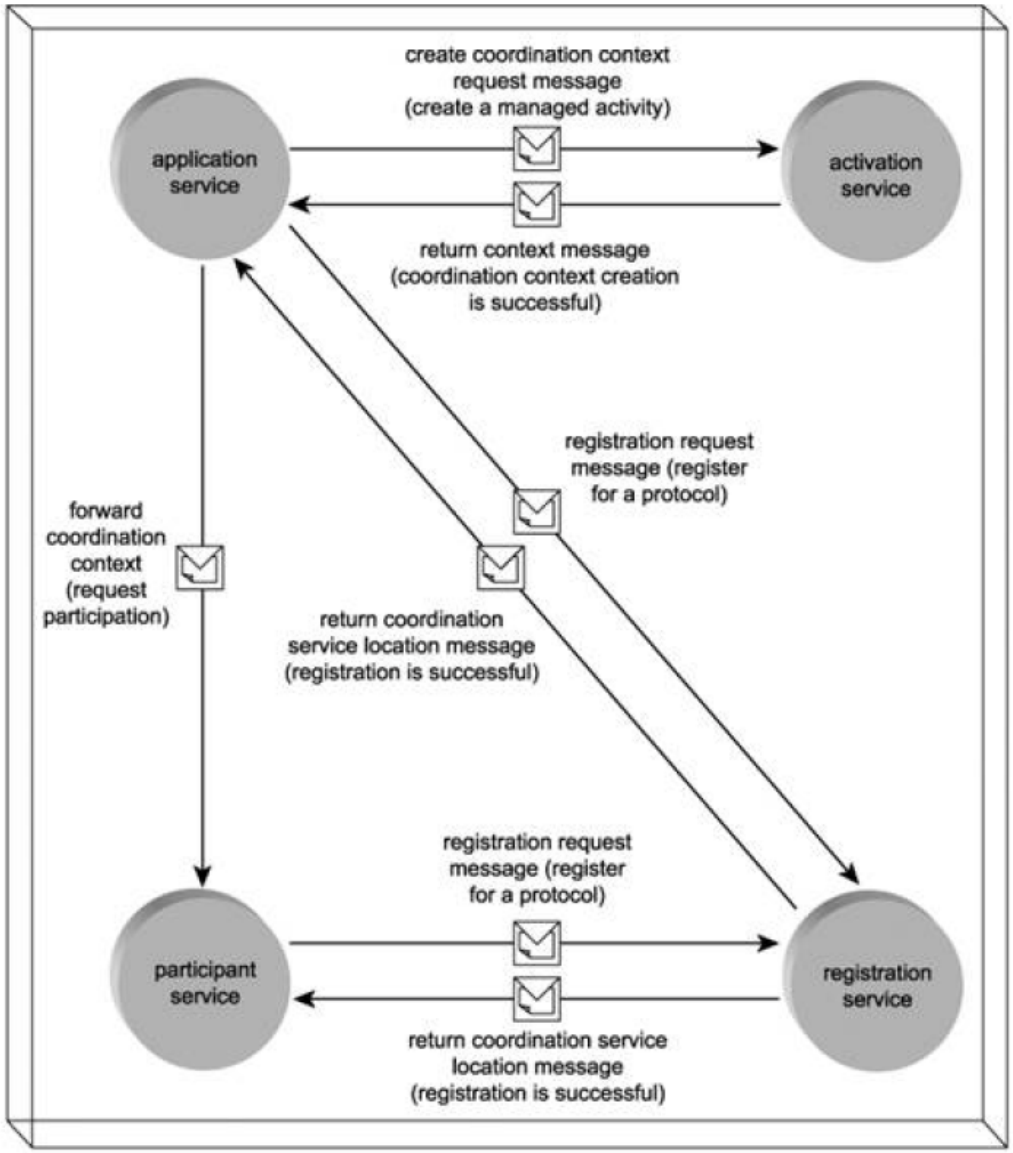
Activation Service responsible of the coordination-context's creation (i.e. the identifier of the coordination activity)

Registration Service register and keep track of the participants of a complex activity

Coordinator Service manages the coordination of an activity w.r.t. a particular coordination type



WS-Coordination dynamics example

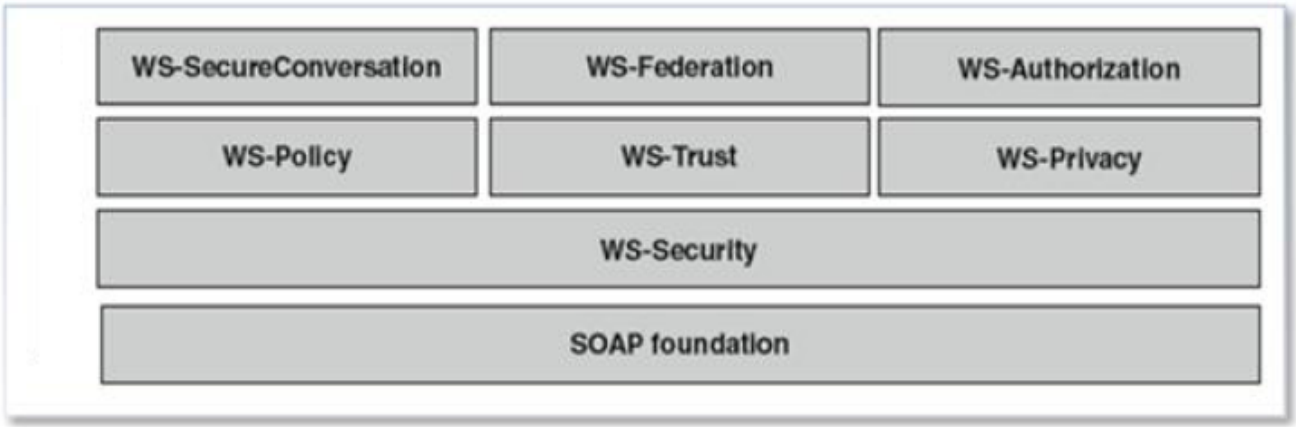


WS-Security framework

A set of WS specifications that address almost all the issues related to Web Service security

Specifications belonging to the security framework

- WS-Security
- WS-Policy
- WS-Trust
- WS-SecureConversation
- Others...



WS-Security and WS-Policy

WS-Security

Enable applications to conduce secure SOAP message exchange ensuring

- Message integrity
- Message confidentiality
- Message authenticity

Rely upon a set of existing specification:XML-Encription, XML-Signature..

WS-Policy

- Define a general purpose model and corresponding syntax to describe the policies of a Web Service...
  - ...also security policies can be defined
- A policy can describe service requirements, capabilities..

WS-Trust and WS-SecureConversation

WS-Trust

Enable applications to construct trusted SOAP message exchanges

- Trust represented through the exchange and brokering of security tokens
- The specification provides a protocol by which: issue, renew and validate security tokens

WS-SecureConversation

Enable secure message exchanges between two or more Web Services

- Built on top of WS-Security and WS-Trust
- Use of security contexts, and derived keys from them, to enable a secure conversation

Semantic Web in a nutshell

Tim Berners-Lee vision

I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A *Semantic Web*, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The *intelligent agents* people have touted for ages will finally materialize [Berners-Lee and Fischetti, 1999]

- Goal: *use* and *reason upon* all the available data on the internet automatically
- By extending the current web with knowledge - semantic information - about the content (i.e. data about the data, meta-data)

Semantic Web Service

Introduction

- Researches area, in the ambit of the Semantic Web, that aims to introduce semantics issue into the world of Web Service
- Objective: enable the WS to communicate via *machine-readable* data
- Match regarding *concepts*, *not simply the signature*
  - WS composition/discovery driven by the meaning of the required data/functionalities

Foundations

- Ontologies: rigorous and formal description of a domain (OWL)
- Definition of the WS behavior (OWL-S, WSMO)
  - by means of IOPE (Input, Output, Preconditions, Effects)
- *Software agents* able to find/compose the most suitable WSs w.r.t the user goal

My personal opinion

A key topic but with the current research efforts is only possible grasp the surface of the problem (the same for all the Semantic Web stuff...)

Outline

RESTFul Web Services: why? 1/2

- In ten years the Web has changed the way we live, but it's got more change left to give
- Rooted on three main technologies

  HTTP as the transport protocol

  XML (HTML/XHTML) for data representation

  URIs for referring to *resources*

- These technologies are powerful enough to give us the Web and the applications we use on it
- It's time to seriously start applying its rules to distributed programming

RESTFul Web Services: why? 2/2

Web's potential for distributed programming has been overlooked

The Web is a simple, ubiquitous, yet *overlooked* platform for distributed programming [Richardson and Ruby, 2007]

- Most of today's web services have nothing to do with the Web
  - In opposition to its simplicity, they espouse a heavyweight architecture for realising distributed applications
- It has to be that way?
- It's time to put the "web" back into "Web Services"

REST

The original definition

*R*epresentational *S*tate *T*ransfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system [Fielding, 2000]

- Data and functionality are considered *resources*
  - Accessed using URIs
- The resources are acted upon well-defined operations
  - HTTP methods: GET, POST, PUT, DELETE
- Client/server architecture designed to use a *stateless* communication protocol (HTTP)
- Clients/servers exchange representations of *resources* by using a standardized interface and protocol

RESTful Web Service 1/3

- A RESTful WS is based on the Resource-Oriented Architecture
  - See [Richardson and Ruby, 2007] for details
- A RESTful Web Service exposes a set of resources identifying the targets of the interaction with its clients
- URIs provide an addressing space for resources and service discovery
- Uniform interface: Resources manipulation via fixed HTTP methods

PUT	creates a new resource
GET	retrieves the current state of a resource in some representation
DELETE	deletes an existing resource
POST	transfers a new state onto a resource

RESTful Web Service 2/3

- Self-descriptive messages
  - Resources are decoupled from their representation
  - Content accessible in a variety of formats
    - HTML, XML, plain text, PDF, JPEG, JSON, ...
- Meta-data about the resource is available and used for
  - Caching control
  - Transmission errors detection
  - Appropriate representation format negotiation
  - Authentication or access control

RESTful Web Service 3/3

- Every interaction with a resource is stateless
  - so again, request messages are self-contained
- Stateful interactions on the concept of *explicit state transfer*
  - Clients manipulate resource state by sending a representation as part of a PUT or POST request
  - Server manipulates client state sending representations in response to the client's GET requests
  - This is where the name *Representational State Transfer* comes from
- State can be embedded in response messages to point to valid future states of the interaction

RESTful Web Service Tools for Java

- JAX-RS specification (recommended)
  - Standard Java programming model (= set of API) for RESTful Web Services
  - Several implementations exist
    - See [Little, 2008] for a comparison
    - Jersey is the GlassFish implementation [Jersey, 2011]
- JAX-WS
  - Exploiting WSDL 2.0 for defining the REST Web Services
  - Usable, but not so used

JAX-RS in a nutshell

- Really similar (in the spirit) to its *brother* JAX-WS
- Provides an annotation-based model to simplify the development a restful Web Service
  - ROA perspective
- Plain Old Java Object (POJO) can be easily exposed as a Web service

JAX-RS in action

```
@Path("helloworld")
public class HelloWorld {
    @Context
    private UriInfo context;
    /** Creates a new instance of HelloWorld */
    public HelloWorld() {
    }

    /**
     * Retrieves representation of an instance of helloWorld.HelloWorld
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("text/html")
    public String getHtml() {
        return "<html><body><h1>"+ msg +"Hello, World!!</h1></body></html>";
    }

    /**
     * PUT method for updating or creating an instance of HelloWorld
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource.
     */
    @PUT
    @Consumes("text/html")
    public void putHtml(String content) {
    }
}
```

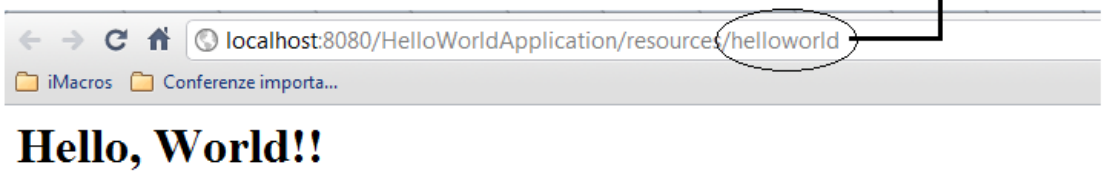


Figure: A REST Web Service printing in output the classical "Hello world!".

Others RESTful Web Service Tools

- Ruby on Rails
  - http://rubyonrails.org/
- Microsoft WCF
  - http://msdn.microsoft.com/en-us/netframework/aa663324
- Python
  - http://www.djangoproject.com/
- ...

Outline

Summing up

- Web Services are one of the reference technology for building distributed systems
- Two different architectural style exist
  - SOA vs ROA [Pautasso et al., 2008]
- An ongoing *"holy war"* between the two style
  - With strong supporters/experts in both sides
  - Often driven by not so strong/valid arguments
  - Difficult to provide a rigorous evaluation
- The question is: *which architecture should I use?*

SOA vs ROA (my opinion)

- I'm not the prophet able to end the holy war
- I can try to give my answer, in the most rigorous way
  - Taking inspiration from other relevant considerations and evaluations [Pautasso et al., 2008, Richardson and Ruby, 2007]
  - Adding my personal experience and vision on the topic
- So, don't take the next few slide as a well established dogma

SOA benefits

- SOA is weighted by standards designed to promote interoperability
  - WSDL for describing the WS functionalities/interface
  - WS-* for high level functionalities support
- Therefore better suited for
  - Enterprise and B2B solutions
  - Composition and integration of WS/existing application
- Most mature tools support (for now)

ROA benefits

- The main advantage of ROA is ease of implementation, agility of the design, and the lightweight approach to things
- REST is a lightweight solution as simple as the Web
  - No standards at all (except HTTP, XML, URI)
- Lower entry barrier
- Simplicity is its siren call
  - Being heard even in the far corners of corporate data centers

Concluding 1/2

- There is not a real winner yet
- A lot of developers and WS have turned to the ROA side
  - Because it seems faster, cheaper and easier
- But standard-less development can require more investment
  - To maintain and manage
  - In learning data formats (are you using XML? JSON? CSV?)
  - In learning service descriptions

Concluding 2/2

- Use ROA when
  - you needs something up-and-running quickly...
  - ...with good performance and low overhead
  - Web Services easily exploitable by simple clients
    - e.g. AJAX/Javascript-based
- Use SOA when you need a distributed application with
  - formal and explicit definition of Web Services contacts
  - support for high level functionalities (WS-*)

References I


Berners-Lee, T. and Fischetti, M. (1999).
Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor.
Harper San Francisco.


Corp., S. M. . O. (2011).
The Java EE 6 tutorial, part 3: Web Services.
`http://download.oracle.com/javaee/6/tutorial/doc/bnayk.html`.


Erl, T. (2005).
Service-Oriented Architecture: Concepts, Technology, and Design.
Prentice Hall PTR, Upper Saddle River, NJ, USA.

References II

Fielding, R. T. (2000).
Architectural Styles and the Design of Network-based Software Architectures.
PhD thesis, University of California, Irvine, CA, USA.

Jersey (2011).
Jersey home page.
http://jersey.java.net/.

Little, M. (2008).
A comparison of JAX-RS implementations.
http://www.infoq.com/news/2008/10/jaxrs-comparison.

References III

Pautasso, C., Zimmermann, O., and Leymann, F. (2008).
RESTful Web Services vs. "big" Web Services: Making the right architectural decision.
In *17th International Conference on World Wide Web (WWW '08)*, pages 805–814, New York, NY, USA. ACM.

Richardson, L. and Ruby, S. (2007).
RESTful Web Services.
O'Reilly.

Sun Microsystems (2004).
JAX-WS reference site.
`https://jax-ws.dev.java.net/`.

References IV


The Apache Software Foundation (2004).
Axis 2 reference site.
http://ws.apache.org/axis2.

Web Services

Distributed Systems
Sistemi Distribuiti

Andrea Santi
a.santi@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011