+

PA165 Enterprise Java
2014-2015

SOAP, WS-* Web Services & Spring-WS

Bruno Rossi & Juha Rikkilä

Objectives and content

Objectives	Content
Get “the big picture” of SOAP and WS-* related web services Have a look at how contract-first development can be used with Spring-WS	■ Some Definitions ■ WSDL & SOAP ■ Creating a SOAP Web service ■ Web services development ■ Using Spring-WS

W3C Definition of Web Services

A Web service is a software system designed to support **interoperable machine-to-machine interaction over a network**. It has an interface described in a **machine processable format** (specifically **WSDL**). Other systems interact with the Web service in a manner prescribed by its description using **SOAP messages**, typically conveyed using **HTTP with an XML** serialization in conjunction with **other Web-related standards**.

Web Service Description Language (WSDL)

- The Web Service Description Language (WSDL) is a **technical description of a Web Service**
- It mentions all **interfaces available**, with the relevant information for the **invocation** (parameters, return type...)

It is possible to generate:

- the client code for accessing the Web Service
- A WSDL file from Java source code
- A Java source code skeleton from WSDL file

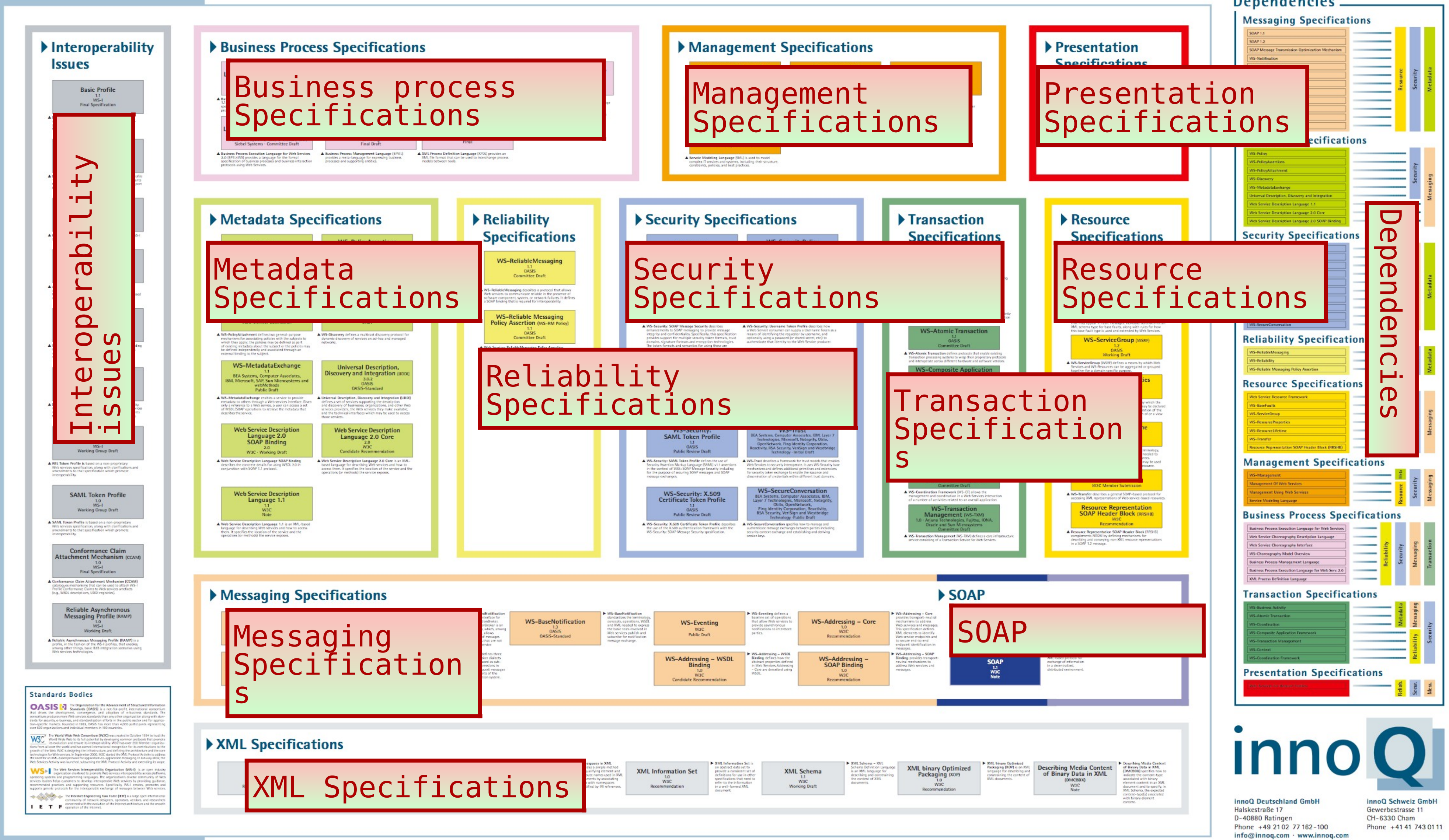
Thomas Erl definition

What are WS-* specifications

- The term "WS-*" has become a commonly used abbreviation that refers to the **second-generation Web services specifications**. These are **extensions to the basic Web services framework** established by first generation standards represented by WSDL, SOAP, and UDDI.
- The term "WS-*" became popular because the majority of titles given to second-generation Web services specifications have been prefixed with "WS-".

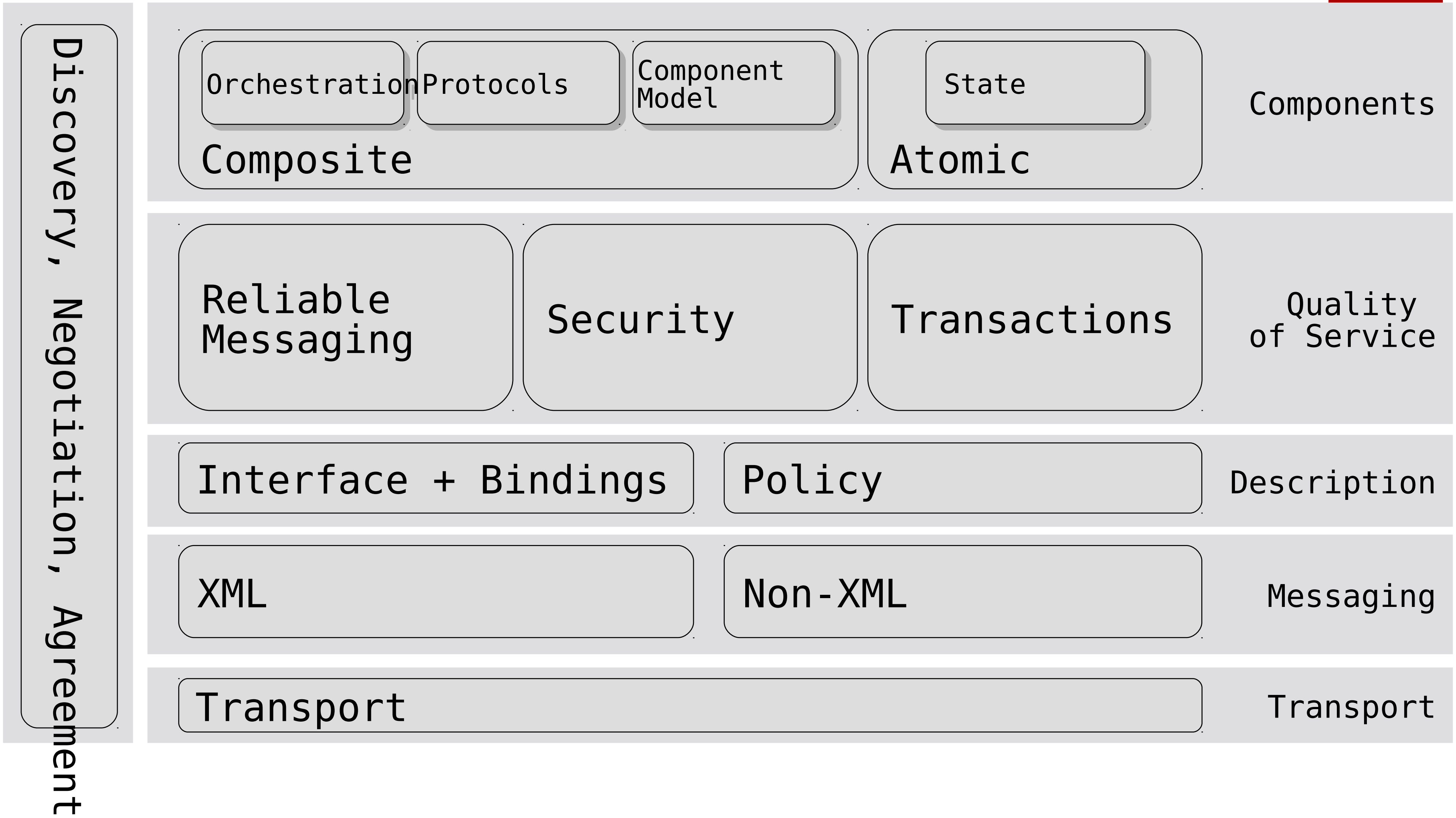
Thomas Erl definition

Web Services Standards Overview



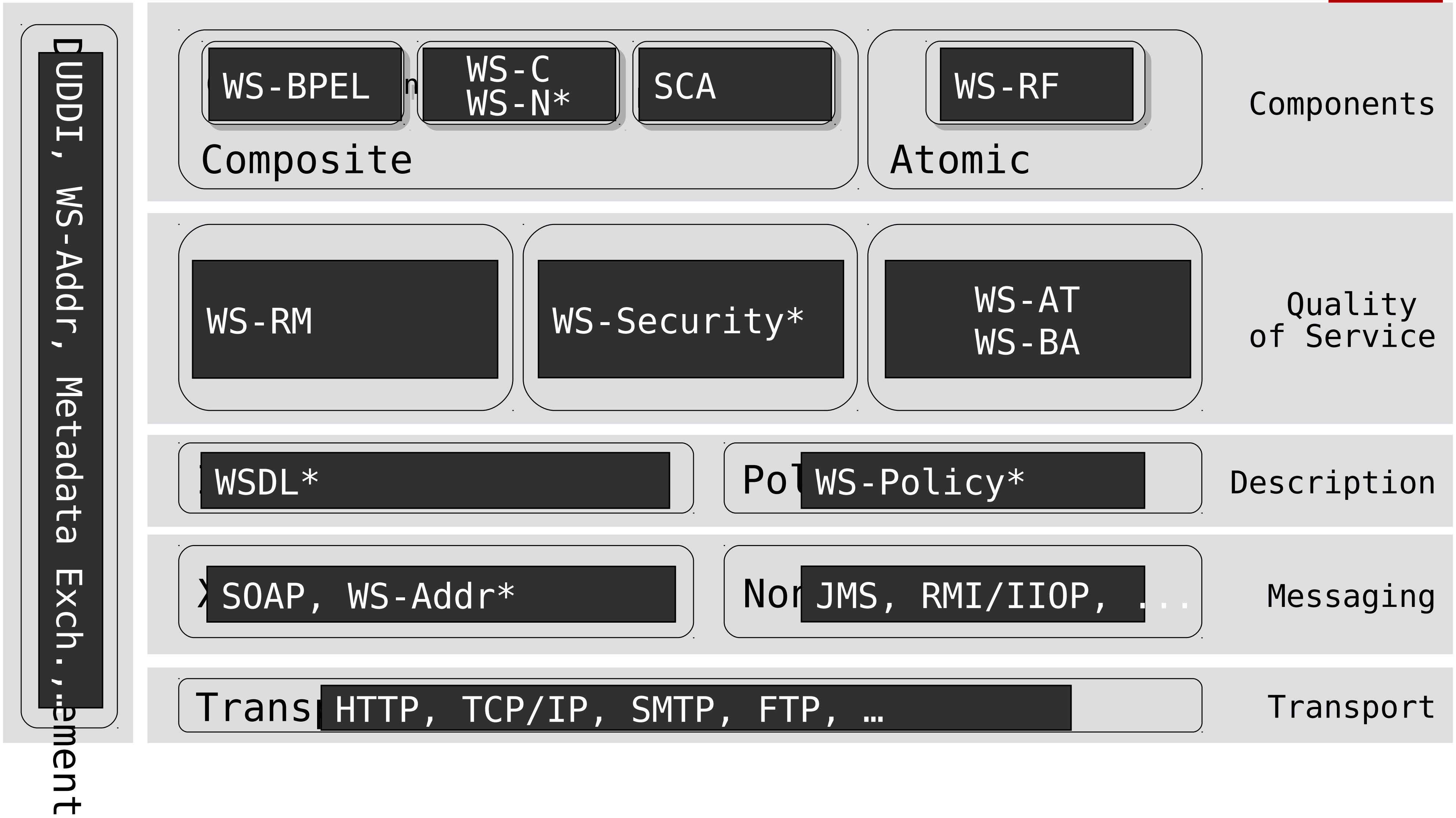
Web Services Standards for SOA

The Web Services Platform Architecture



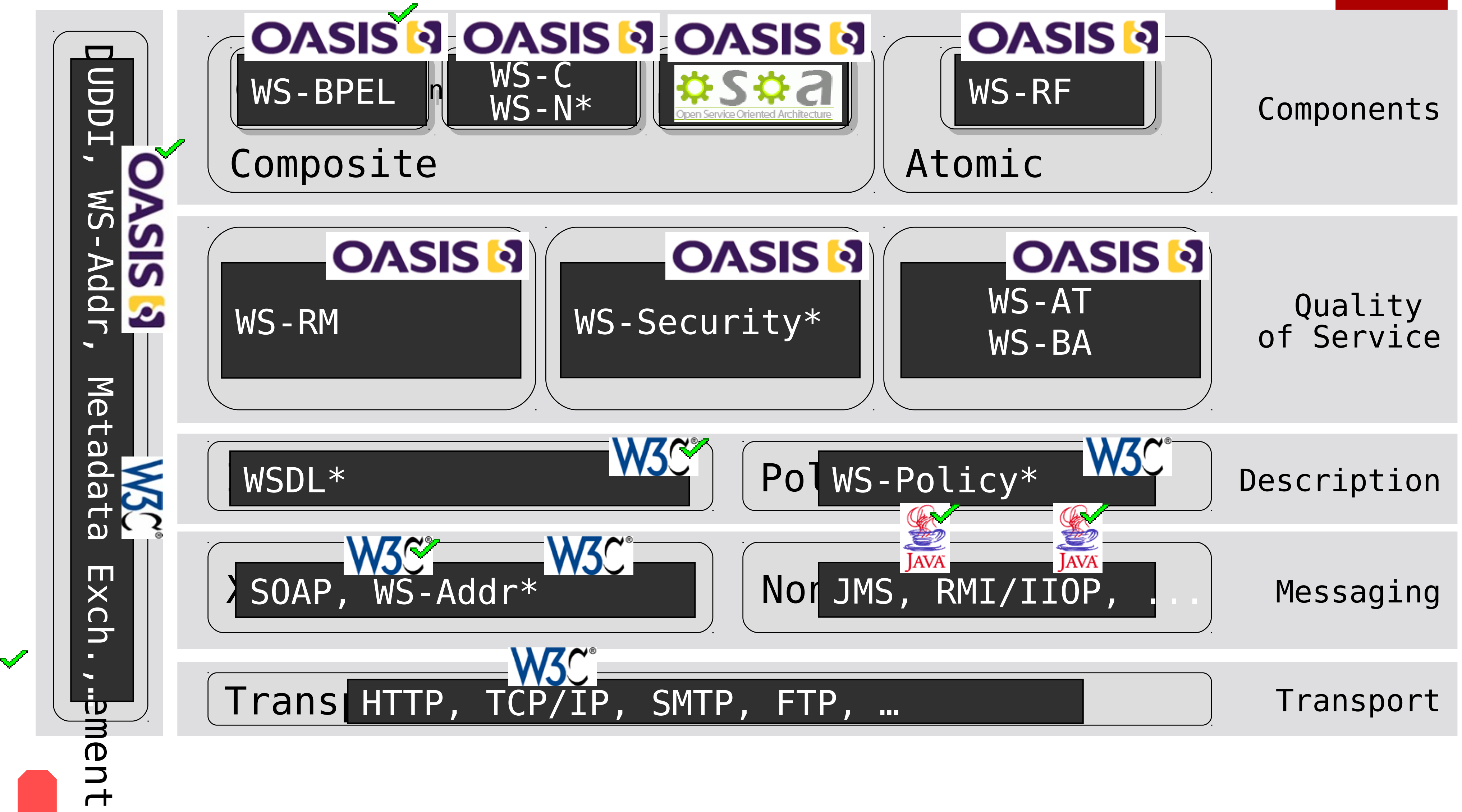
Web Services Standards for SOA

The Web Services Platform Architecture



Web Services Standards for SOA

The Web Services Platform Architecture



SOAP, in general terms

- Originally acronym for **Simple Object Access Protocol**, now a common name
- A **communication protocol**, designed to communicate via Internet
- Extends **HTTP for XML messaging**
- Provides data transport for Web services
- Exchanges **complete documents** or **call a remote procedure**
- Is used for **broadcasting a message**
- Is platform and **language independent**
- Is the XML way of defining what information gets sent and how

An historical overview:
https://kore.fi.muni.cz/wiki/index.php/PA165/WebServices_(English)

XML (Extensible Markup Language)

- Sets of rules for encoding documents to structure, store, and transport data in a convenient way
- Human-readable and machine-readable format
- XML 1.0 Specification produced by the W3C
- two current versions of XML.
  - XML 1.0, currently in its fifth edition, still recommended for general use
  - XML 1.1, not very widely implemented and is recommended for use only by those who need its unique features

+ XML, markup and content

- **markup**
  - begins with character “<” and ends with a “>”,
- strings of characters that are not markup are content
- **tag(s)**
  - start-tags; for example: <section>
  - end-tags; for example: </section>
  - empty-element tags; for example: <line-break />
- **element**, begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.

Schema and validation

- well-formed, and may be valid.
  - Document contains a reference to DTD,
  - DTD declares elements and attributes, and specifies the grammatical rules
- XML processors
  - re validating or non-validating
  - If error discovered it is reported, but processing may continue normally
- schema languages constrain
  - the **set of elements in a document**,
  - **attributes that are applied** to them,
  - the **order** in which they appear,
  - the **allowable parent/child relationships**

XML Schema: **XSD (XML Schema Definition)**

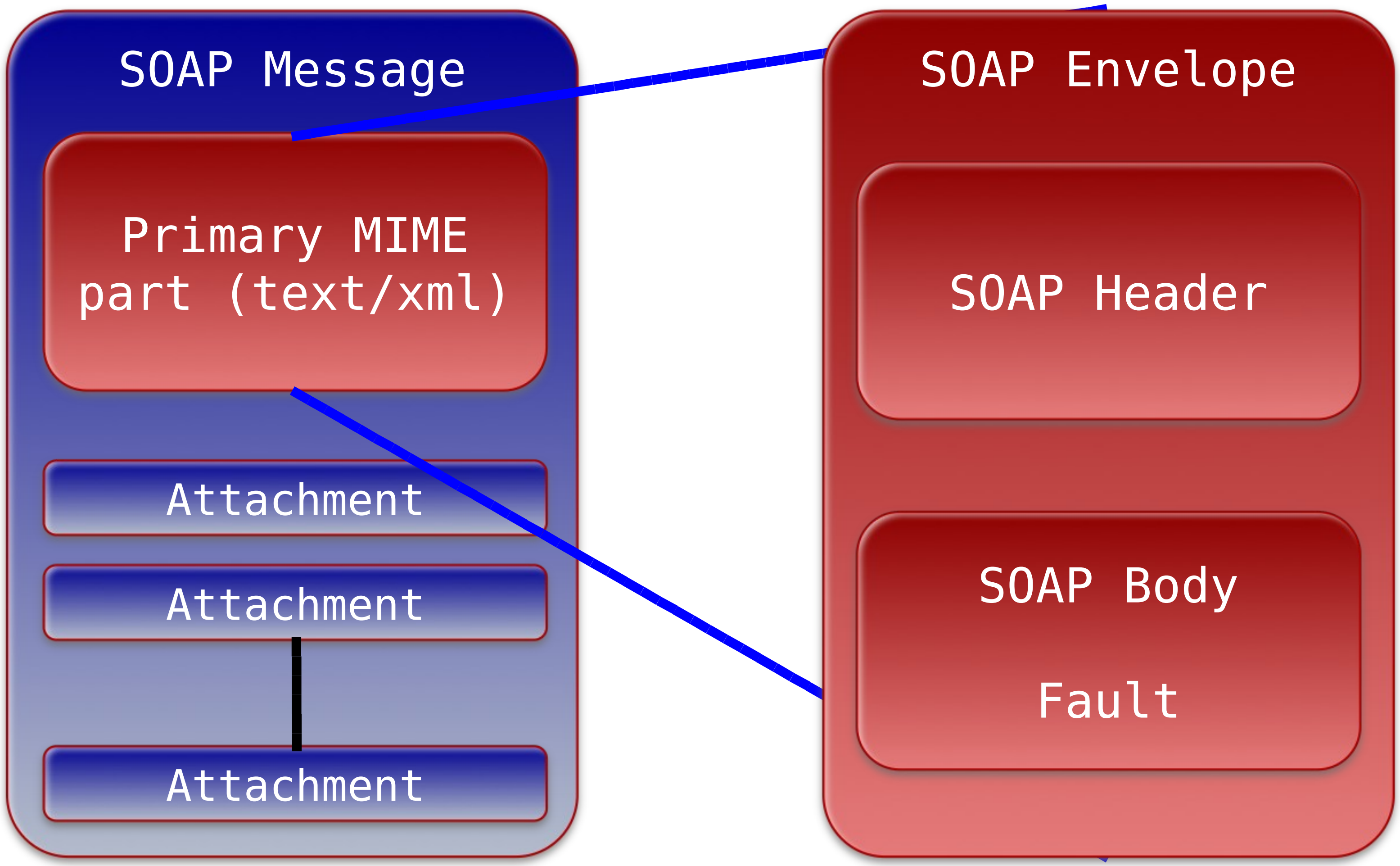
- schema language, described by the W3C
  - **(successor of DTD = Document Type Definition)**
  - XML schema is more powerful than DTDs

- XSDs use an XML-based format, so XML tools can be used process them.

XML Messaging

- SOAP 1.1 defined:
  - An **XML envelope for XML messaging**:
    - Headers + body.
  - An **HTTP binding for SOAP messaging**:
    - SOAP is “transport independent”.
  - A **convention for doing RPC**,
  - An **XML serialization format for structured dat**a.
- SOAP Attachments adds:
  - How to carry and reference data attachments using in a MIME envelope and a SOAP envelope.

SOAP Message



SOAP Message Envelope

- Encoding information
- Header
  - Optional
  - Contains context knowledge
    - Security
    - Transaction
- Body
  - Methods and parameters
  - Contains application data

+ A SOAP Request

```
POST /temp HTTP/1.1
Host: www.somewhere.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx
SOAPAction: "http://www…../temp"

<?xml version="1.0"?>
……………..
```

HTTP headers and the blank line

an XML document

"The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent. SOAP places no restrictions on the format or specificity of the URI or that it is resolvable. An HTTP client MUST use this header field when issuing a SOAP HTTP Request."
Note: in SOAP 1.2, the SOAPAction header has been replaced with the "action" attribute on the application/soap+xml media type (Content-Type: application/soap+xml; charset=utf-8). But it works almost exactly the same way as SOAPAction.

Source: Simple Object Access Protocol (SOAP) 1.1 specifications

XML message structure

Version number

```
<?xml version="1.0"?>
<soap:Envelope
   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">

   <soap:Header>
      ...
   </soap:Header>

   <soap:Body>
      ...
      <soap:Fault>
         ...
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

soap-encoding

SOAP encoding

- When SOAP specification was written for the first time, **XMLSchema was not available, so a common way to describe messages was defined**.
- Now SOAP encoding defines it's own namespace as http://schemas.xmlsoap.org/soap/encoding/ and a set of rules to follow.

- Rules of expressing application-defined data types in XML
- Based on W3C XML Schema
- Simple values
  - Built-in types from XML Schema, Part 2 (simple types, enumerations, arrays of bytes)
- Compound values
  - Structures, arrays, complex types

http://www.tutorialspoint.com/soap/soap_encoding.htm

WS-Addressing (1/2)

- WS-* specifications are inserted on top of SOAP messaging
- For example, looking at SOAP, there is no knowledge about where the message is going, or how to return the response or where to post an error message → this can be problematic in case of asynchronous communication
- WS-Addressing adds this information to the SOAP envelope

See https://jax-ws.java.net/nonav/jax-ws-21-ea2/docs/why-wsaddressing.html

WS-Addressing (2/2)

- Example

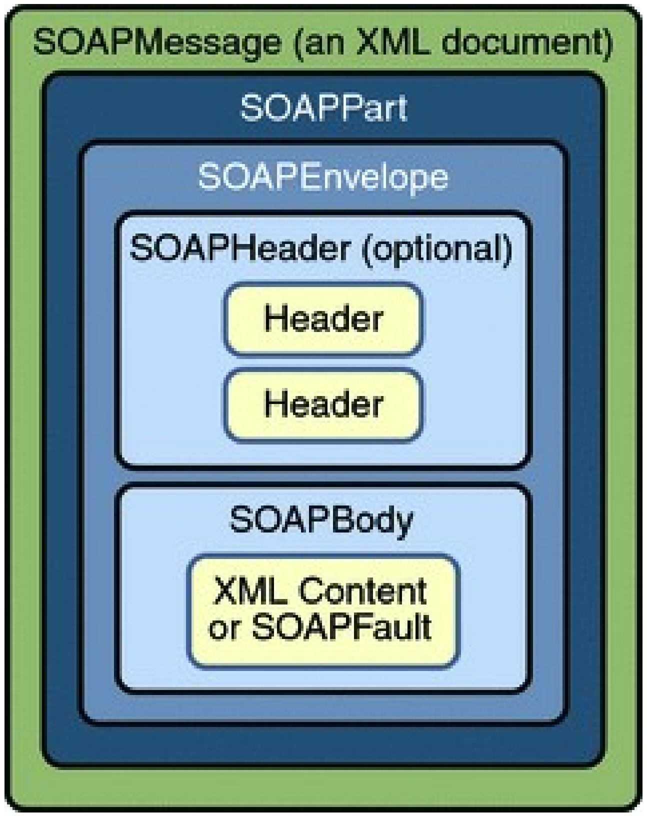
```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope”
xmlns:wsa="http://www.w3.org/2004/12/addressing">
<soap:Header>
        <wsa:MessageID>
            UniqueMessageIdentifier
        </wsa:MessageID>
        <wsa:ReplyTo>
          <wsa:Address>http://somereceiving.client</wsa:Address>
        </wsa:ReplyTo>

        <wsa:FaultTo>
            <wsa:Address>http://somereceiving.server/ErrorHandler</wsa:Address>
        </wsa:FaultTo>

        <wsa:To>http://somereceiving.server/HandlerURI </wsa:To>
        <wsa:Action>
           http://somereceiving.server/ACTION
       </wsa:Action>
    </soap:Header>

    <soap:Body>
       <!-- SOAP Request as usual here -->
    </soap:Body>
```

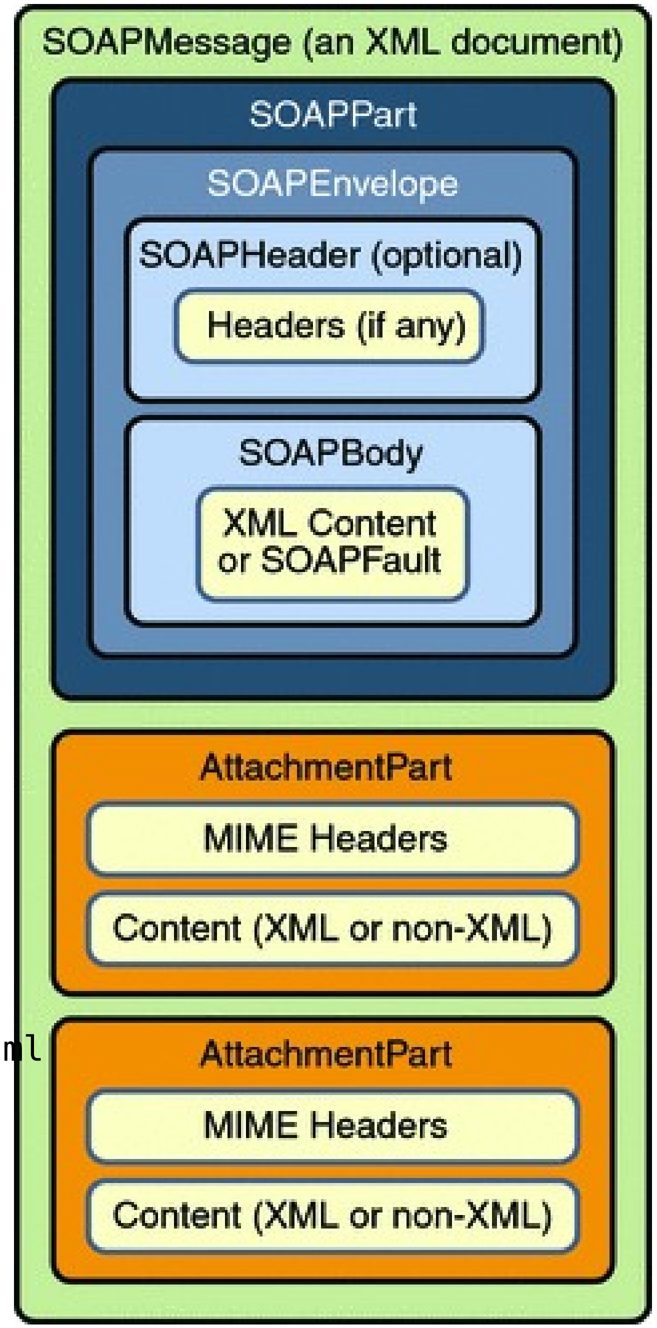
SOAP, “closer to the bit space”

- Summing up:
- SOAP, originally defined as **Simple Object Access Protocol**, is a protocol specification that is used to exchange information in a structured way – the protocol is used for implementation of **Web Services as it builds on top of an Application Layer protocol, Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP)**.
- SOAP is based on on Extensible Markup Language (XML) for its message format.
- SOAP is usually the **foundation layer of a web services protocol stack**, to provide a basic messaging framework.

SOAP with Attachments, SOAP with Attachments API for Java (SAAJ)

- **SOAP with Attachments (SwA)** or MIME for Web Services refers to the method of using Web Services to send and receive files using a combination of SOAP and MIME, primarily over HTTP.
- Note that SwA is not a new specification, but rather a **mechanism for using the existing SOAP and MIME facilities to perfect the transmission of files using Web Services invocations**.
- The **SOAP with Attachments API for Java** or **SAAJ** provides a standard way to send XML documents over the Internet from the Java platform.
- **SAAJ** enables developers to produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments note.
- Developers can also use it to **write SOAP messaging applications directly instead of using JAX-RPC (obsolete) or JAX-WS**





SOAP with Attachments API for Java
The Java EE 5 Tutorial
http://docs.oracle.com/javaee/5/tutorial/doc/bnbhf.html

Creating a SOAP Conection

```
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.MessageFactory;
……….

public SimpleSAAJ {
    public static void main(String args[]) {
       try {
           //Create a SOAPConnection
           SOAPConnectionFactory factory =
                    SOAPConnectionFactory.newInstance();

           SOAPConnection connection =
                    factory.createConnection();

           .................
           // Close the SOAPConnection
           connection.close();

       } catch (Exception e) {
           System.out.println(e.getMessage());
       }
   }
}
```

Creating a SOAP Message

```
…………….
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPBody;
import java.net.URL;
……………..

          //Create a SOAPMessage
          SOAPMessageFactory messageFactory =
          MessageFactory.newInstance();
          SOAPMessage message = messageFactory.createMessage();
          SOAPPart  soapPart = message.getSOAPPart();
          SOAPEnvelope envelope = soapPart.getEnvelope();
          SOAPHeader header = envelope.getHeader();
          SOAPBody body = envelope.getBody();
          header.detachNode();
```

Populate a SOAP Message

```
//Create a SOAPBodyElement
 Name bodyName = envelope.createName("GetElement"
                  "n", "http://localhost");
SOAPBodyElement bodyElement = body.addBodyElement(bodyName);

//Insert Content
  Name name = envelope.createName("symbol");
  SOAPElement symbol = bodyElement.addChildElement(name);
  symbol.addTextNode("Smith");
```

This will produce the SOAP envelope:

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <n:GetElement xmlns:n="http://localhost">
      <symbol>Smith</symbol>
    </n:GetElement>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

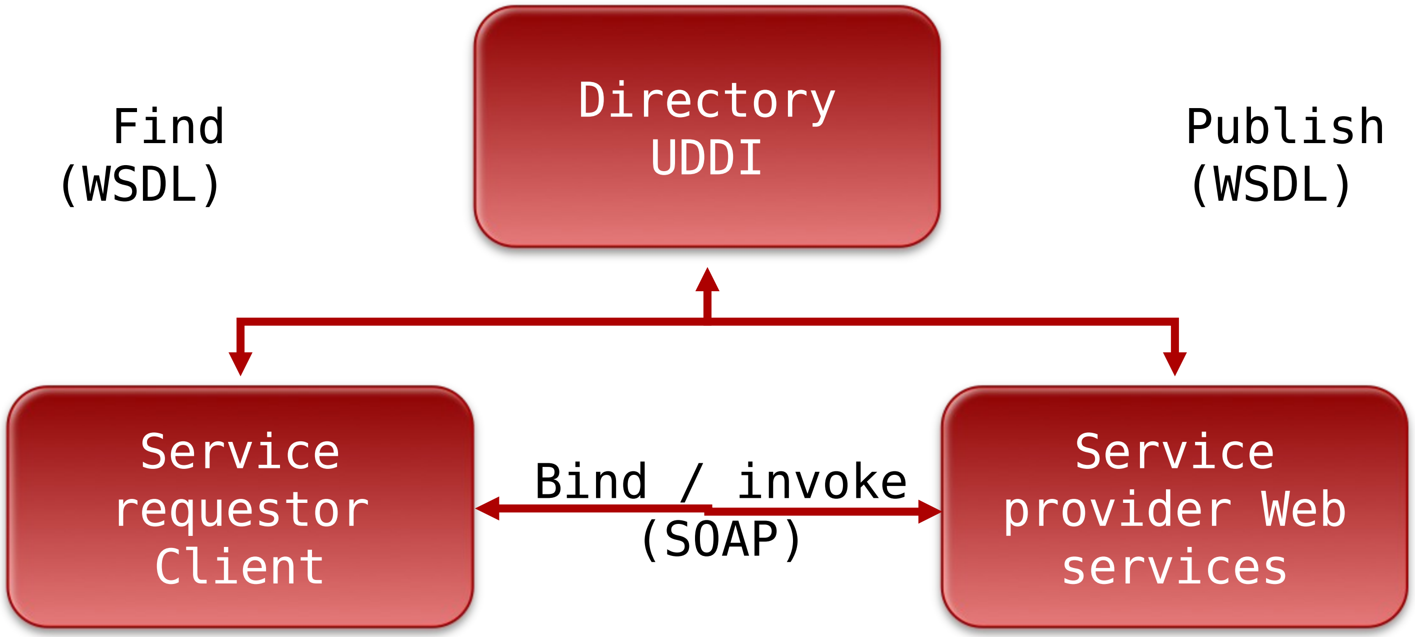
That you can send with

```
java.net.URL endpoint = new URL("localhost/addr");
SOAPMessage response = connection.call(message, endpoint);
```

Some Remarks

- SOAP is not *“what it used to be”*, the name remained, but the content has changed
- **SOAP term** is often used as **synonym** for **WS* web service architecture**, though it is one element of it
- **SOAP is not just one element of WS***, it is used in other context as well, even parallel with ReST web services.
- **SOAP is often hidden from the developer**, build into tools in such a way that developer does not have to deal with it at a detailed level.

Use of web services



SOAP, WSDL, UDDI, and XML in all of them

UDDI (Universal Description, Discovery and Integration)

- UDDI is a **platform-independent, Extensible Markup Language (XML)-based registry** by which businesses worldwide can list themselves, plus a mechanism to register and locate web service applications.
- It is a standard supported by the Organization for the Advancement of Structured Information Standards (OASIS)
- In the original plans for the discoverability of web services, a central role should have been played by UDDI

Public Registries *(well, it used to be...)*

- IBM Registration: https://uddi.ibm.com/ubr/registry.html
  - inquiryURL= https://uddi.ibm.com/ubr/inquiryapi
  - publishURL= https://uddi.ibm.com/ubr/publishapi
- HP Registration: http://uddi.hp.com
  - inquiryURL = http://uddi.hp.com/ubr/inquire
  - publishURL = https://uddi.hp.com/ubr/publish
- Microsoft Registration: http://uddi.rte.microsoft.com
  - inquiryURL= http://uddi.microsoft.com/inquire
  - publishURL=https://uddi.rte.microsoft.com/publish
- SAP Registration: http://udditest.sap.com
  - inquiryURL=http://uddi.sap.com/UDDI/api/inquiry/
  - publishURL=https://uddi.sap.com/UDDI/api/publish/

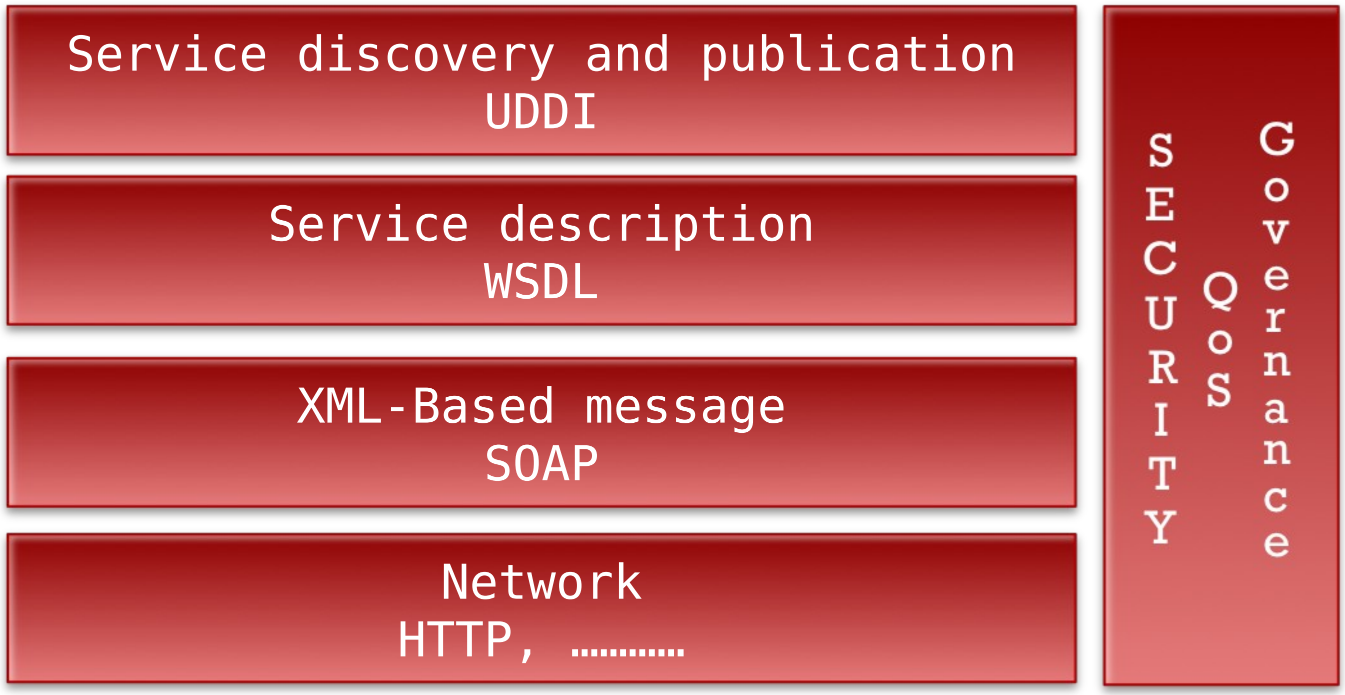
UDDI has not been as successful as its creators had expected. IBM, Microsoft, and SAP **closed** their public UDDI nodes in 2006.

The OASIS UDDI Specification Technical Committee has been **dismantled** as well.

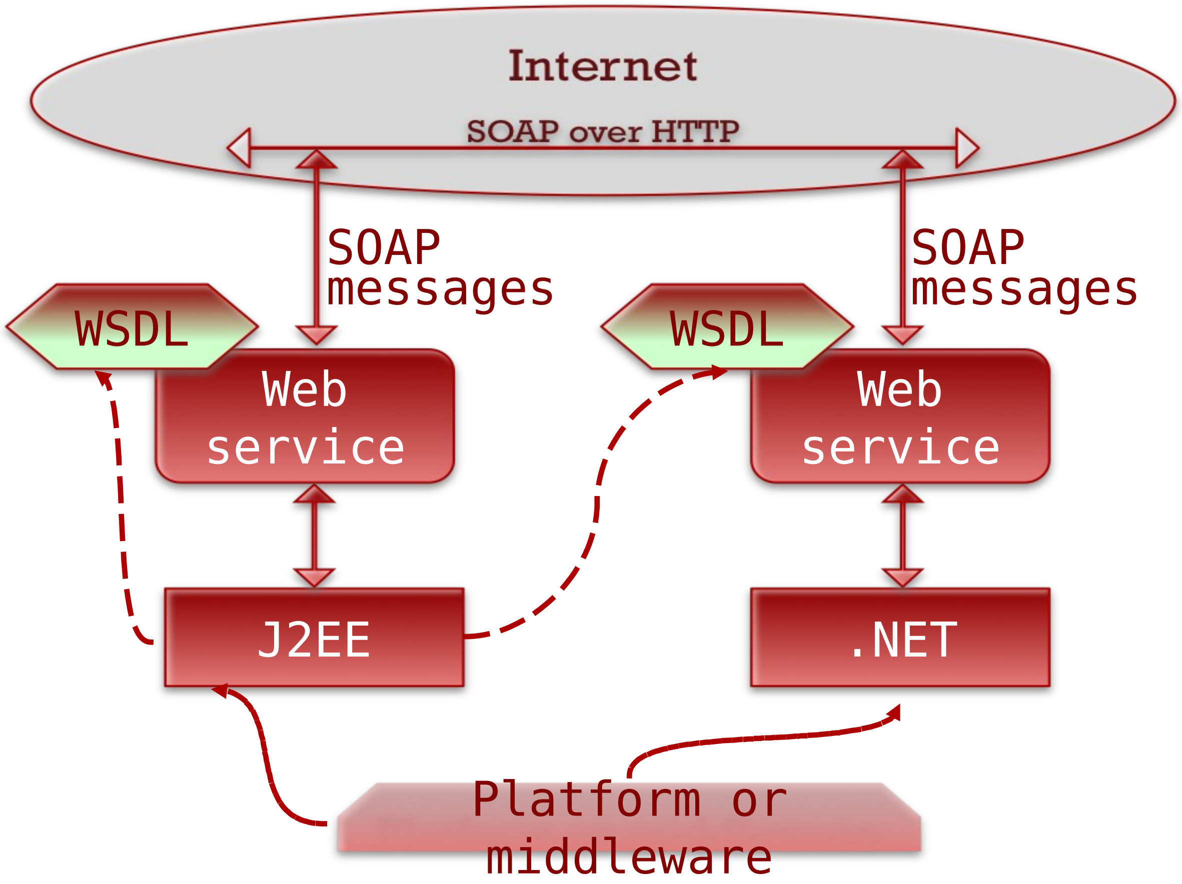
Microsoft **removed** UDDI services from the Windows Server operating system.

UDDI systems **are most commonly found inside companies**, where they are used to dynamically bind client systems to implementations. However, much of the more advanced functionalities are not used.

Enabling technologies



WS*

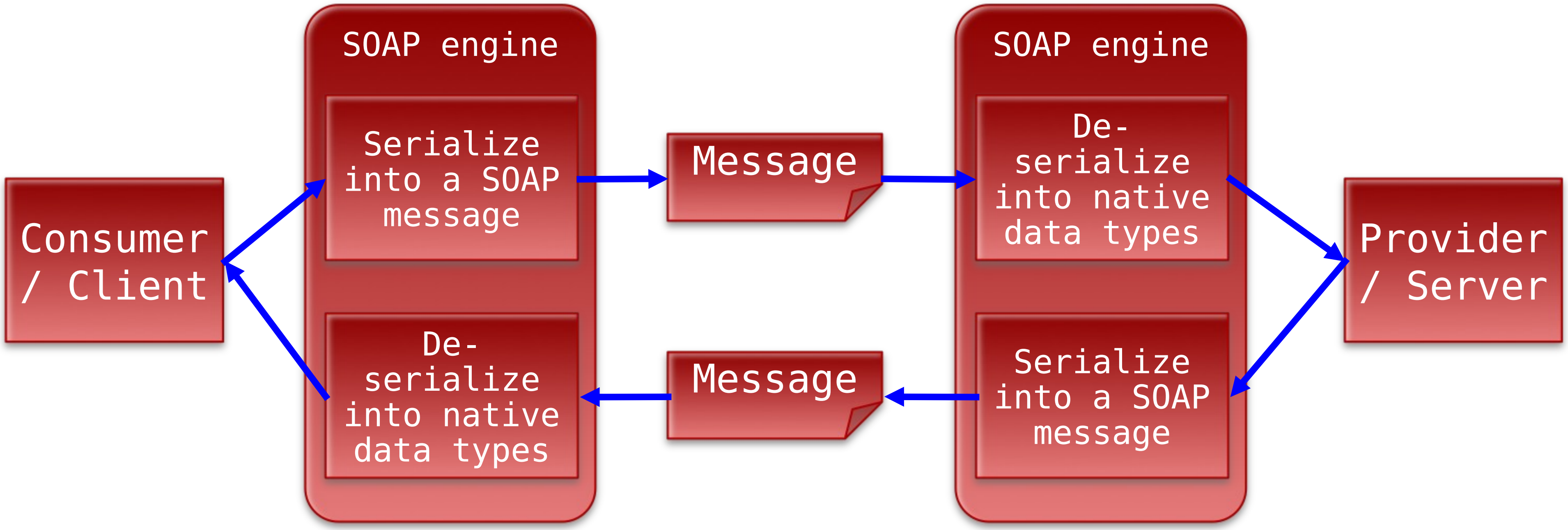


- clear specifications of the service interface and the data types in use
- communication protocol independent (platform, programming language)
- interoperability.

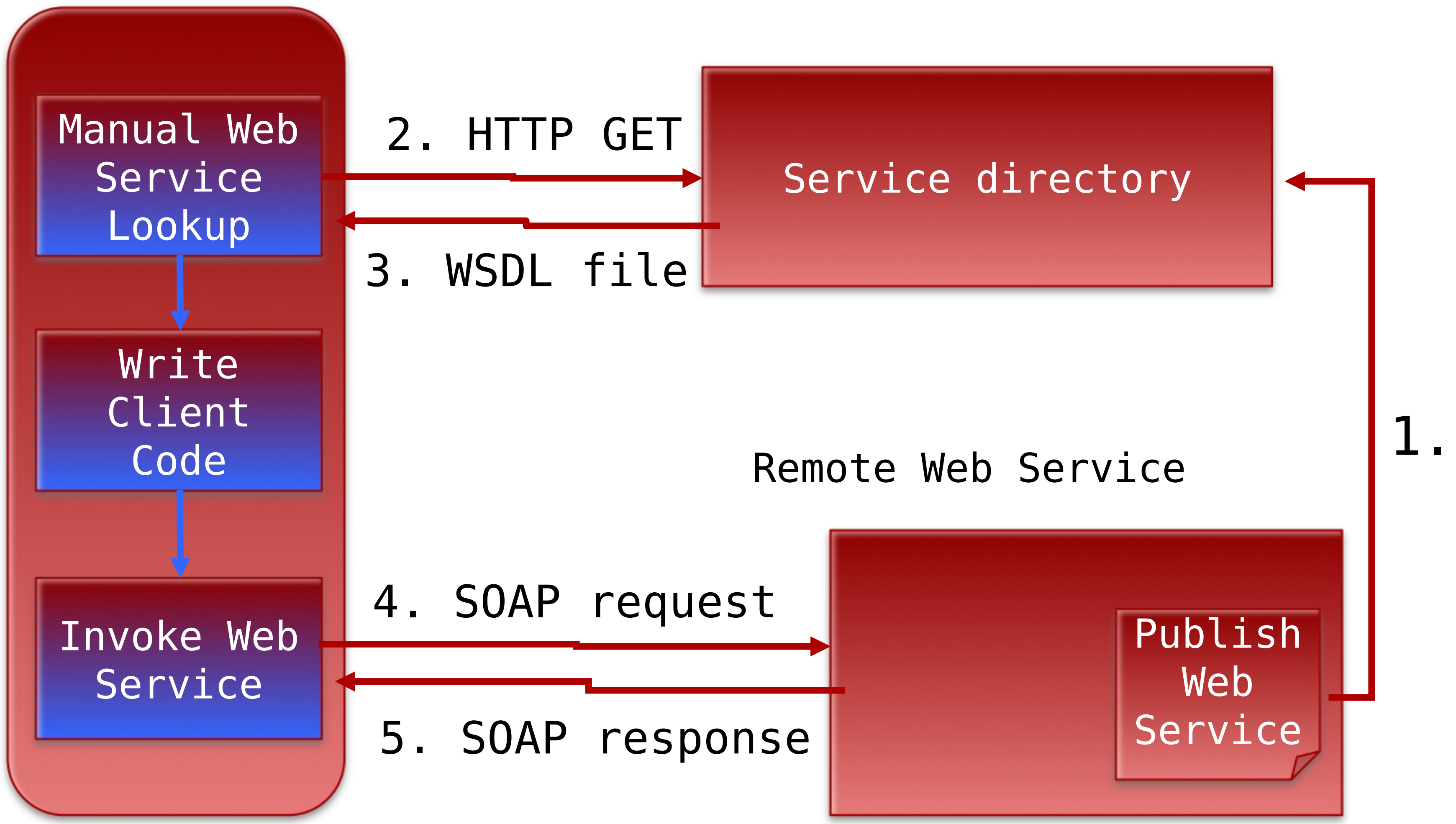
SOAP engines

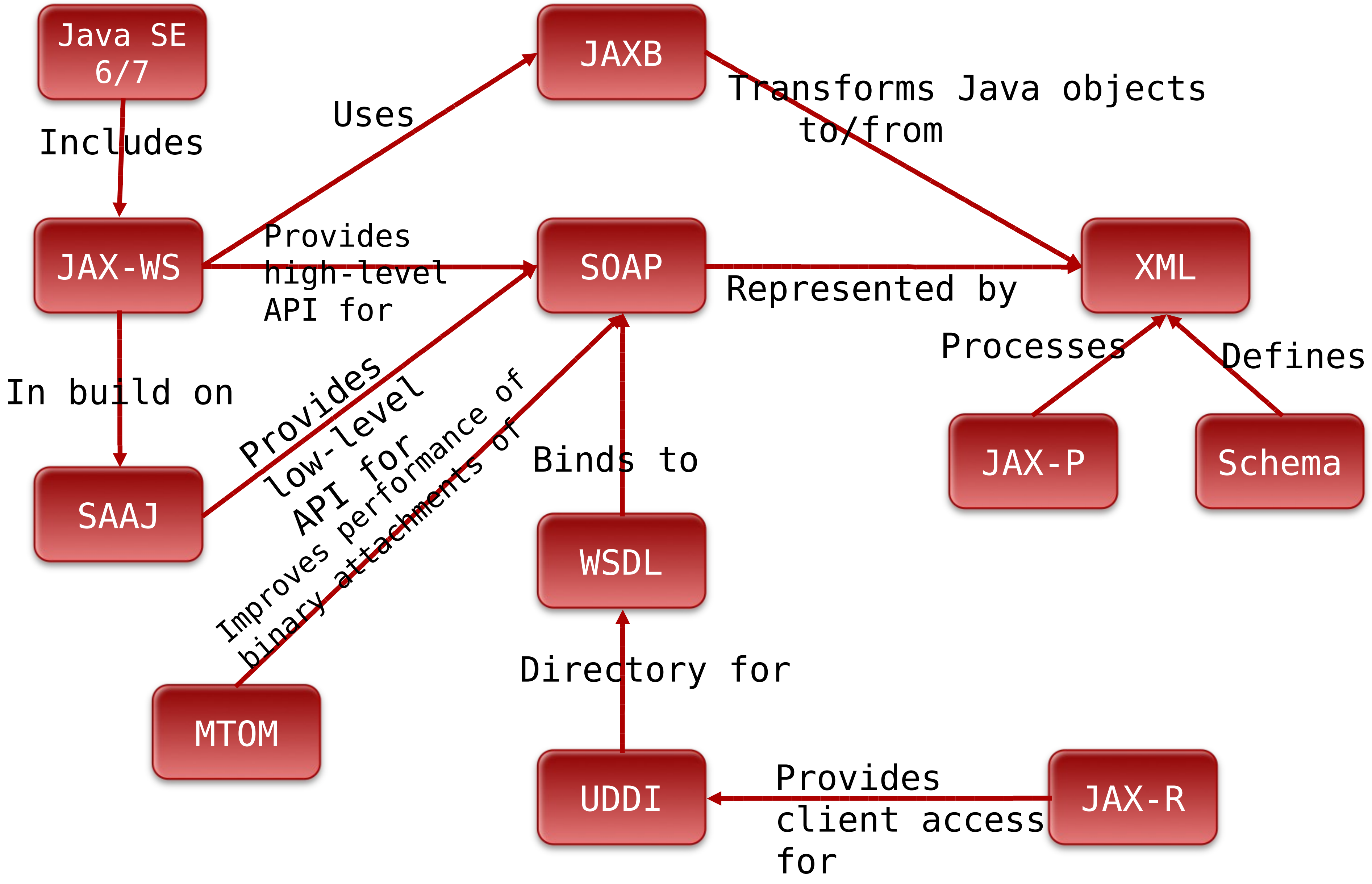
A **SOAP engine** is a framework used in servers and clients that facilitates:

1. Serializing objects from a programming language into SOAP messages
2. De-serializing SOAP messages into objects in a programming language, i.e. creating appropriate data types and populating these with the message content.


Consumer / Client
SOAP engine
Serialize into a SOAP message
De-serialize into native data types
Message
Message
SOAP engine
De-serialize into native data types
Serialize into a SOAP message
Provider / Server

Simple Web Service Invocation




Java SE 6/7
Includes
JAX-WS
Uses
JAXB
Transforms Java objects to/from
XML
Provides high-level API for
SOAP
Represented by
In build on
SAAJ
Provides low-level API for
Improves performance of binary attachments of
MTOM
Binds to
WSDL
Directory for
UDDI
Provides client access for
JAX-R
Processes
JAX-P
Defines
Schema

An example (1/7) Implementing a simple web service with Java

1. Create the "service endpoint interface"
   - Interface for web service
2. Create the "service implementation"
   - Class that implements the service
3. Create the "service publisher"

- Java supports web services in core Java
  - JAX-WS (Java API for XML-Web Services)
- In full production mode, one would use a Java application server such as Tomcat, Glassfish, etc.

An example (2/7)
Service Endpoint Interface

```
package example.echo;   // echo server
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService      // This is a Service Endpoint Interface
@SOAPBinding(style = Style.RPC) // Needed for the WSDL
public interface EchoServer {
    @WebMethod    // This method is a service operation
        String EchoMessage(String strMsg); }
```

An example (3/7)

Service Implementation

```
package example.echo;
import javax.jws.WebService;
/**
   * The @WebService property endpointInterface links this class
   * to example.echo.EchoServer.
*/
@WebService(endpointInterface = "example.echo.EchoServer")
public class EchoServerImpl implements EchoServer {
        public String EchoMessage(String Msg) {
                String capitalizedMsg;
                System.out.println("Server: EchoMessage() invoked...");
                System.out.println("Server: Message > " + Msg);
                capitalizedMsg = Msg.toUpperCase();
                return(capitalizedMsg);
        }
}
```

An example (4/7)
Service Publisher

```
package example.echo;
import javax.xml.ws.Endpoint;

public class EchoServerPublisher {
	public static void main(String[ ] args) {
	Endpoint.publish("http://localhost:8080/ws", new
	EchoServerImpl());
	}
}
```

An example (5/7) Deploying and testing

1. Compile the Java code
2. Run the publisher
   - java example.echo.EchoServerPublisher
3. Testing the web service with a browser
   - URL: http://localhost:8080/ws?wsdl

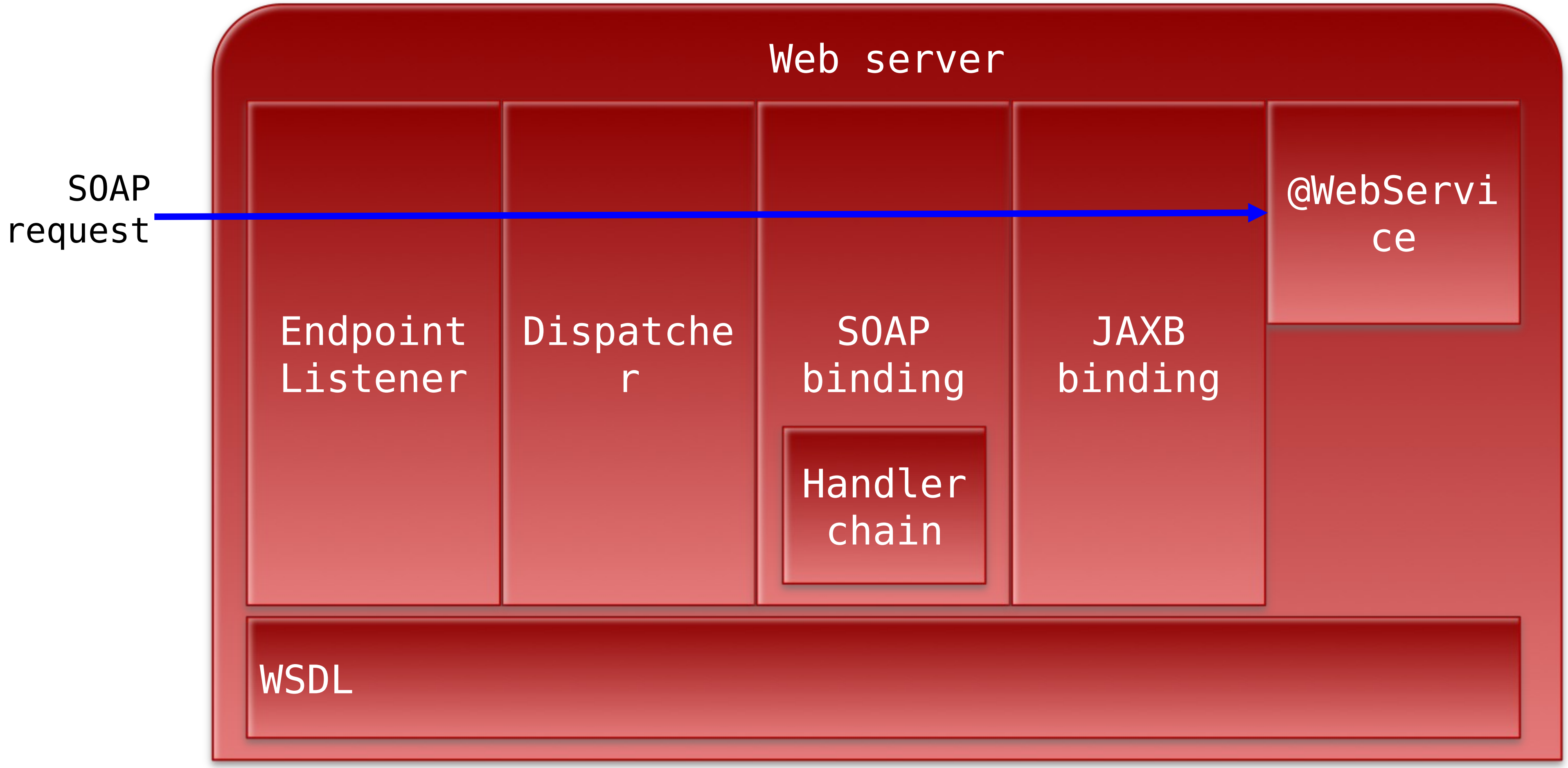
An Example (6/7)
WSDL for echo service

```
<definitions targetNamespace=”http://localhost/" name="EchoServerImplService">
<types/>
<message name="EchoMessage”> <part name="arg0" type="xsd:string"/> </message>
<message name="EchoMessageResponse”><part name="return"
type="xsd:string"/></message>

<portType name="EchoServer">
      <operation name="EchoMessage">
            <input message="tns:EchoMessage"/>
            <output message="tns:EchoMessageResponse"/>
      </operation>
</portType>

<binding name="EchoServerImplPortBinding" type="tns:EchoServer">
      <soap:binding transport=”http://schemas.xmlsoap.org/soap/http" style="rpc”/>
      <operation name="EchoMessage">
      <soap:operation soapAction=""/>
      <input> <soap:body use="literal" namespace="http://echo.example/"/> </input>
      <output> <soap:body use="literal" namespace="http://echo.example/"/> </output>
      inding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
      </operation>
</binding>

<service name="EchoServerImplService">
<port name="EchoServerImplPort" binding="tns:EchoServerImplPortBinding”>
      <soap:address location=”http://localhost:8080/ws"/>
</port>
</service>
</definitions>
```

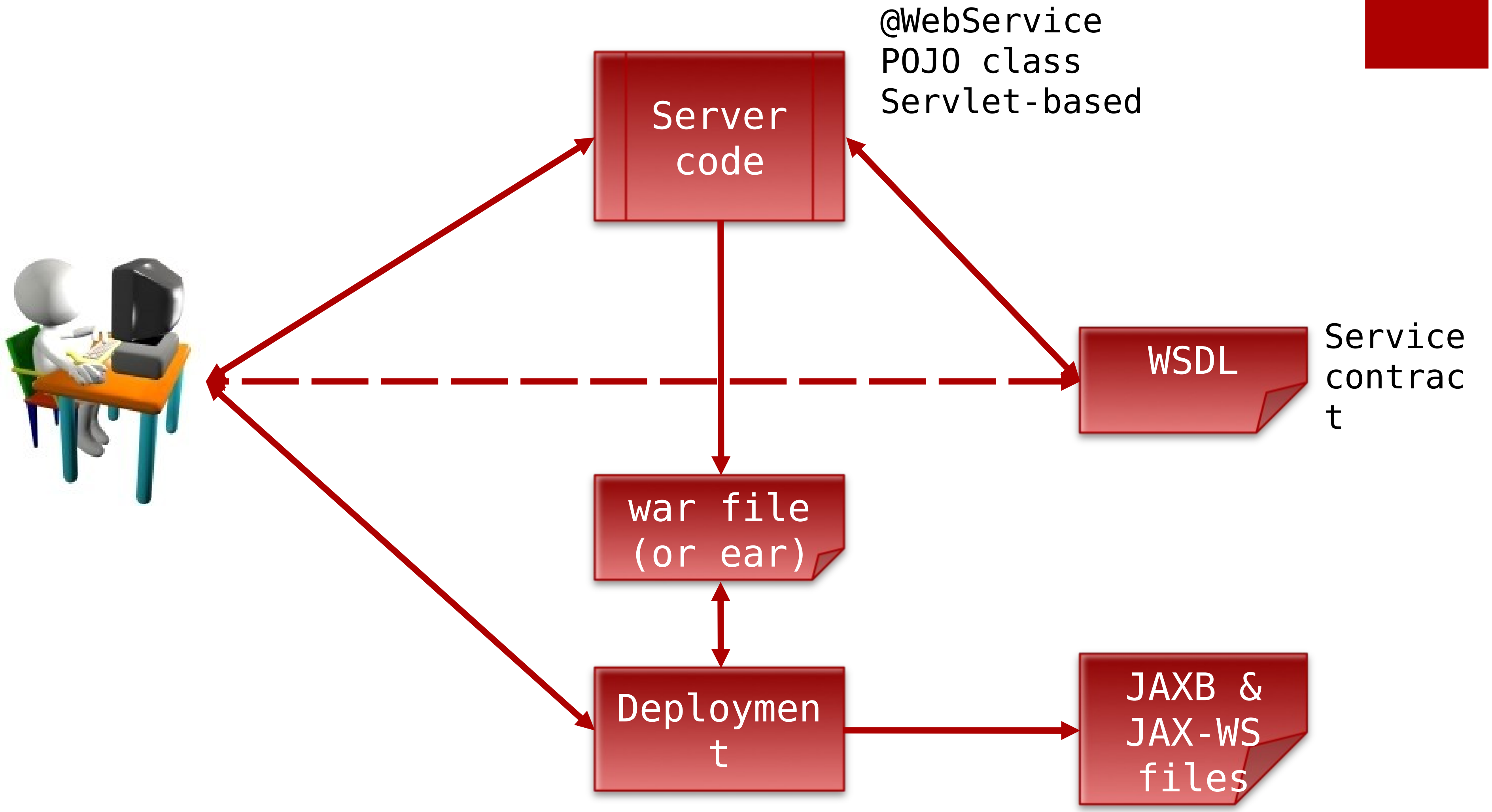
An Example (7/7)
EchoClient

```
package example.echo;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;

class EchoClient {
      public static void main(String argv[ ]) throws Exception {
            if (argv.length < 1) {
            System.out.println("Usage: java EchoClient \"MESSAGE\"");System.exit(1);}

            String strMsg = argv[0];
            URL url = new URL(”http://localhost:8080/ws?wsdl");
                  // Qualified name of the service:
            QName qname = new
            QName(“http://localhost/”,“EchoServerImplService");
            Service service = Service.create(url, qname);
                  // Extract the endpoint interface, the service "port".
            EchoServer eif = service.getPort(EchoServer.class);
            System.out.println(eif.EchoMessage(strMsg));
      }
}
```

Server side

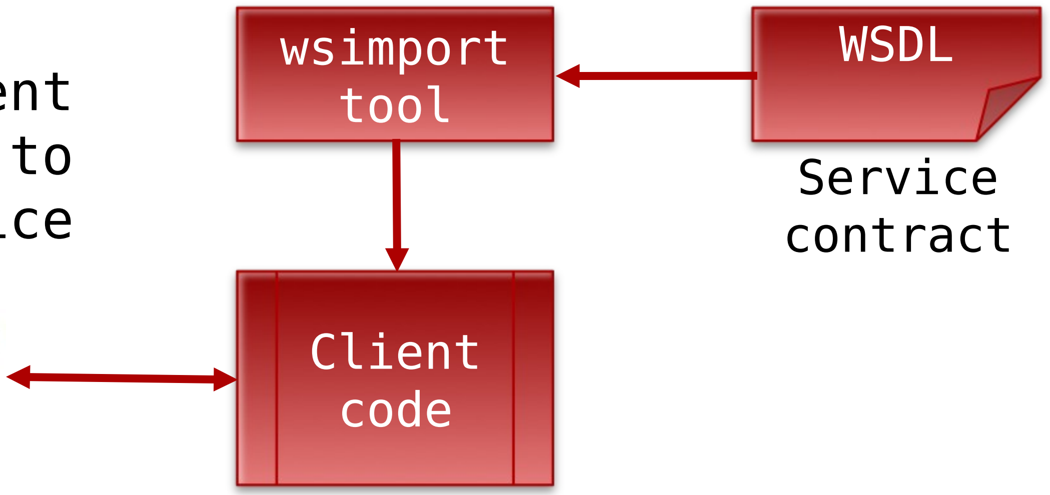

Web server
SOAP request
Endpoint Listener
Dispatcher
SOAP binding
Handler chain
JAXB binding
@WebService
WSDL

Developing a Web Service



Client-side programming

You develop Client which uses proxy to call Web Service

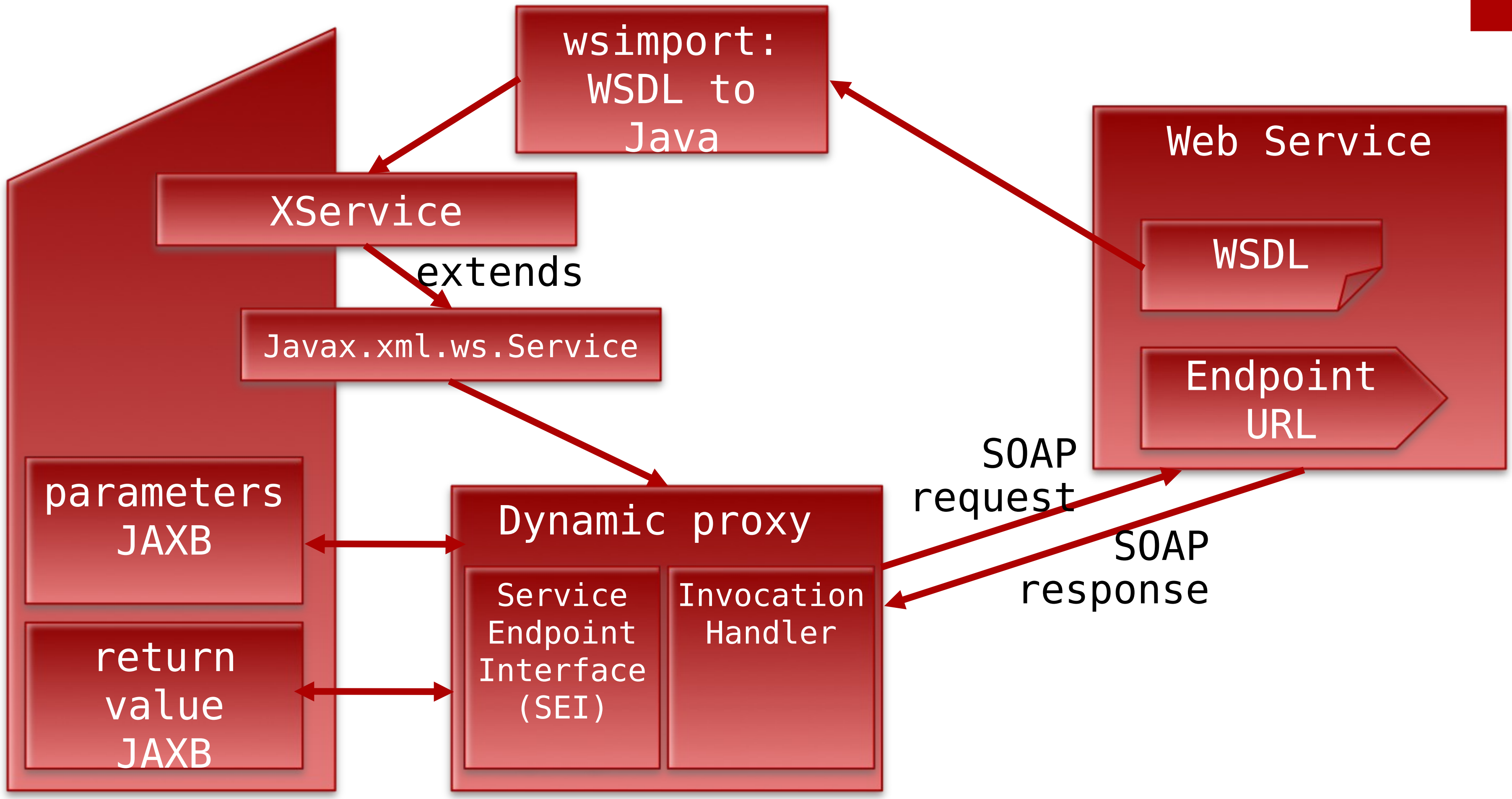


@WebService Dynamic proxy

Client-side programming

- Usually two ways:
- **Contract last**: first you create the code for your web service, then the contract (WSDL) is generated based on the code
- **Contract first**: you start with the creation of the contract for the web service and then source code templates are generated based on the contract

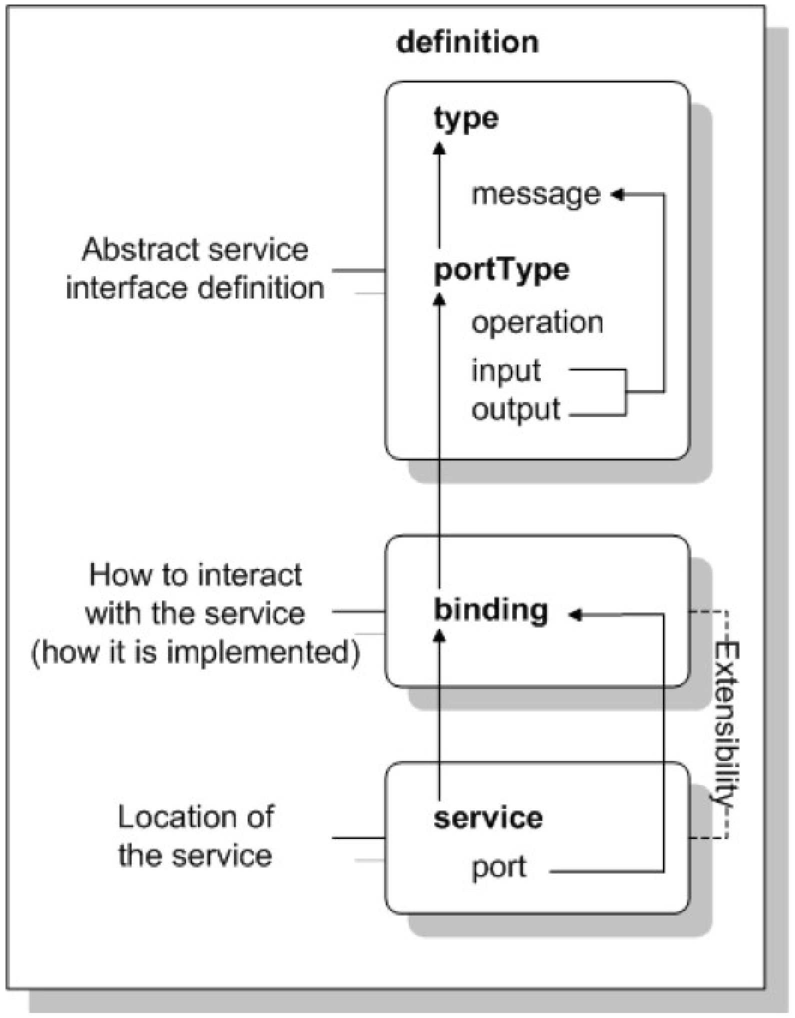
Client side


wsimport: WSDL to Java
XService
extends
Javax.xml.ws.Service
Web Service
WSDL
Endpoint URL
SOAP request
SOAP response
Dynamic proxy
Service Endpoint Interface (SEI)
Invocation Handler
parameters JAXB
return value JAXB

WSDL

- A WSDL describes the point of contact for a service provider, also known as the service endpoint or just endpoint.
- Provides a formal definition of the endpoint interface
- requestors wishing to communicate with the service provider know exactly how to structure request messages
- Establishes the physical location (address) of the service.

WSDL elements



- **<types>**, the data types of input and output data, used by the web service
- **<message>**, messages to be exchanged, used by the web service
- **<portType>**, the operations input and output exposed by the web service. Note: parameters are represented as messages
- **<binding>**, the coupling and protocols used by the web service. This is were for example SOAP can be used as protocol
- **<port>** service location and binding

+ Web Service Example

A Web service AddFunction with operation addInt is known through its WSDL:

```
<wsdl:message name="addIntResponse">
        <wsdl:part name="addIntReturn" type="xsd:int" />
 </wsdl:message>
<wsdl:message name="addIntRequest">
         <wsdl:part name="a" type="xsd:int" />
         <wsdl:part name="b" type="xsd:int" />
 </wsdl:message>
<wsdl:portType name="AddFunction">
     <wsdl:operation name="addInt" parameterOrder="a b">
          <wsdl:input message="impl:addIntRequest" name="addIntRequest" />
          <wsdl:output message="impl:addIntResponse" name="addIntResponse" />
     </wsdl:operation>
 </wsdl:portType>
```

```
// possible implementation of WS using
the wsimport tool:
// AddFunction.jws
public class AddFunction {
  int addInt(int a, int b){
    return(a+b);
  }
}
```

Generating a WSDL file from a Java class

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

```
javac –cp . Calculator.java
java2wsdl –cp . –tn calculator –stn calculator –cn Calculator
```

-cp = classpath; -tn target namespace; -stn schema target namespace; -cn class name

Generating the service code skeleton from the WSDL file

```
wsdl2java -ss -sd -uri Calculator.wsdl
```

-ss = server side; -sd = service descriptor

- A src directory is created with the source code for our server side files
- A resources directory is created with the WSDL file for the service and a service descriptor (services.xml) file
- A build.xml file is created in the current directory, which will be used to create the ws deployment file

Using WSDL

- WSDL allows to define a contract between client and server
  - Tool support to generate code from WSDL or WSDL from code.
- Allows to have standard service interfaces.
- It can also be used for dynamic discovery with UDDI registries

+ Summary

- WS* standards are unevenly taken into use
  - Service orientation is well accepted
  - Several competing solutions, most notably WS* vs REST that complement each other
  - Successful and accepted standardization process in technical interfaces

- Many technical complexities still remains

- Emergence of new new solutions is frequent although standards are mature and widely adopted

Spring-WS

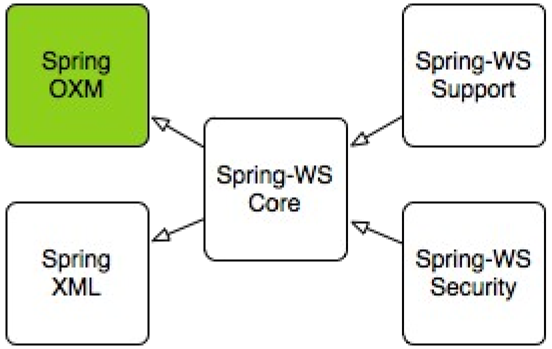
- A Spring “sub-project” that allows to simplify WS-* development
- You can reuse as such your Spring application context and configuration in your application in your SOA application
- Plus, you get access to various WS-* standards
- Note that Spring-WS only supports **“contract first”** development

Spring-WS - Configuration

```
<dependencies>
    <dependency>
  <groupId>org.springframework.ws</groupId>
        <artifactId>spring-ws-core</artifactId>
        <version>2.2.0.RELEASE</version>
    </dependency>
</dependencies>
```

Maven dependency

```
<beans xmlns="http://www.springframework.org/schema/beans">
    <bean id="webServiceClient" class="WebServiceClient">
        <property name="defaultUri"
value="http://localhost:8080/WebService"/>
    </bean>
</beans>
```



Webservice client bean

Spring-WS-Core depends
On Spring's Object/XML Mapping support
(OXM) module and on Spring XML module

- See http://projects.spring.io/spring-ws/

+ Spring-WS – Service Client

```
public class WebServiceClient {

    private static final String MESSAGE =
        "<message xmlns=\"http://tempuri.org\">Hello World</message>";

    private final WebServiceTemplate webServiceTemplate = new
WebServiceTemplate();

    public void setDefaultUri(String defaultUri) {
        webServiceTemplate.setDefaultUri(defaultUri);
    }

    // send to the configured default URI
    public void simpleSendAndReceive() {
        StreamSource source = new StreamSource(new
           StringReader(MESSAGE));
        StreamResult result = new StreamResult(System.out);
        webServiceTemplate.sendSourceAndReceiveToResult(source,
        result);
    }
}
```

```
<beans xmlns="http://www.springframework.org/schema/beans">
    <bean id="webServiceClient" class="WebServiceClient">
        <property name="defaultUri" value="http://localhost:8080/WebService"/>
    </bean>
</beans>
```

- See http://projects.spring.io/spring-ws/

+ Spring-WS – Endpoints

```
@Endpoint
public class BookEndpoint {
     private static final String NAMESPACE_URI = "http://muni.cz/pa165/soa";
     private final BookRepository bookRepository;

     @Autowired
     public BookEndpoint(BookRepository bookRepository) {
          this.bookRepository = bookRepository;
     }

     @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getBookRequest")
     @ResponsePayload
     public GetBookResponse getBook(@RequestPayload GetBookRequest request) {

            GetBookResponse response = new GetBookResponse();

            response.setBook(bookRepository.getBookByTitle(request.getTitle()));
            return response;
     }

}
```

Spring-WS

- Let's look at Spring-WS documentation, in particular:
- MessageDispatcher & MessageDispatcherServlet
- Automatic WSDL exposure
- Endpoints & Endpoint Mapping
- Interceptors
- Testing in Spring-WS

http://docs.spring.io/spring-ws/docs/2.2.0.RELEASE/reference/htmlsingle