Apache CXF Web Services

Prakash Badhe

prakash.badhe@agilesoft.in

Distributed World

- RPC specifies invoking procedures on one machine remotely from another machine in the network.
- Java supports RPC by RMI technique.
- RMI-Remote Method Invocation
- In RMI methods of object running on one machine are invoked from another machine in the network and results are passed to the caller.
- CORBA allows different programming platforms on different OS systems to communicate and exchange data through Object Brokers.
- Microsoft DCOM supports data exchange and communication on Windows distributed platforms.

Data Serialization

- It is process of converting of an object/data structure into a stream of bytes
- Java supports serialization process through JVM via interface ***Serializable***.
- Each class that wants to be serializable must implement this ***Serializable*** *interface**.***
- *The* process of encoding arguments, objects and results for transmission over the wire is called as Marshalling.
- The reverse process of re-constructing objects, results from input bytes received over the network is called as Un Marshalling.
- Thus objects to be marshaled or un marshaled over the network *must be serializable*

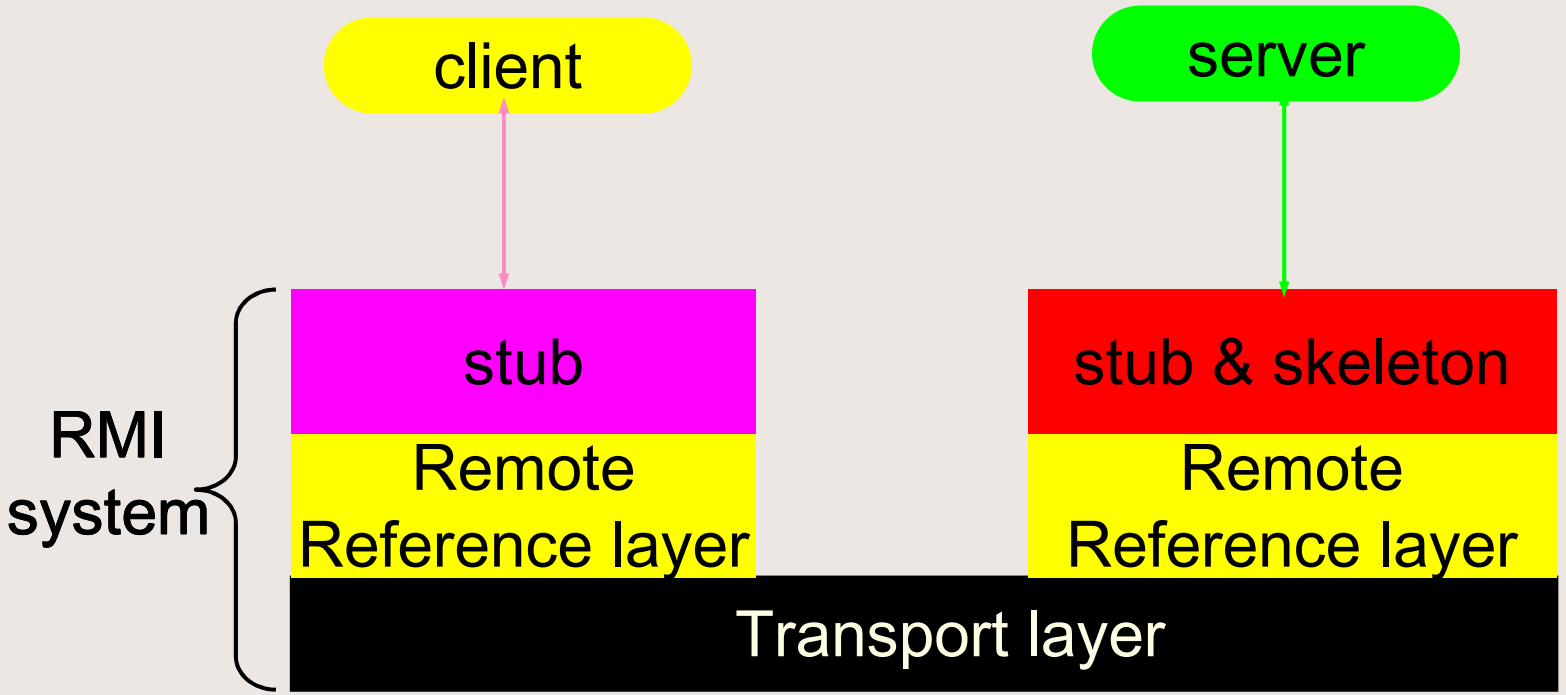
Data sharing

- How a running object/structure on one machine can be shared to another remote machine ?
- How the remote machine will get the reference to the object running on another machine ?
- Obviously the class/structure of the running object will not be shared as happens in normal desktop standalone applications where the required classes are loaded from local classpath.

Object /Data Sharing Solution..

- The object reference is shared without sharing the object implementation.
- How the client will be linked to this unknown object ?
- How client will be looking at this unknown object ?
- RMI provides sharing in form of common interfaces available on both the machines.
- Remote client will be getting the shared object in the form of the interface object without knowing the actual class which implements the interface

RMI Architecture



RMI Server and Client

- RMI Server is a program which implements the common interface and provides the sharing of this object over the network.
- Client program gets the shared object reference through interface and invokes the methods specified by the interface on this shared object.
- The methods invoked by remote client are executed on server program, and hence the name Remote Method Invocation.
- Who provides this sharing and linkage?

Stub layer

Stub class

- Acts as a proxy to remote server object on client side
- Communicates with the real object over the network by marshalling and unmarshalling the data to and from the server to the client.
- It actually communicates to stub and skeletons on server.
- The stub class is generated from corresponding remote class by the RMI compiler

The skeleton layer

Stub and Skeleton class

- Server side proxy.
- Carries on a conversation with the client side stub object.
- Reads the parameters for the method call from the remote link.
- Executes the remote method on server object.
- Writes the return value from the method back to the client stub.
- Generated or Inbuilt in the JVM.

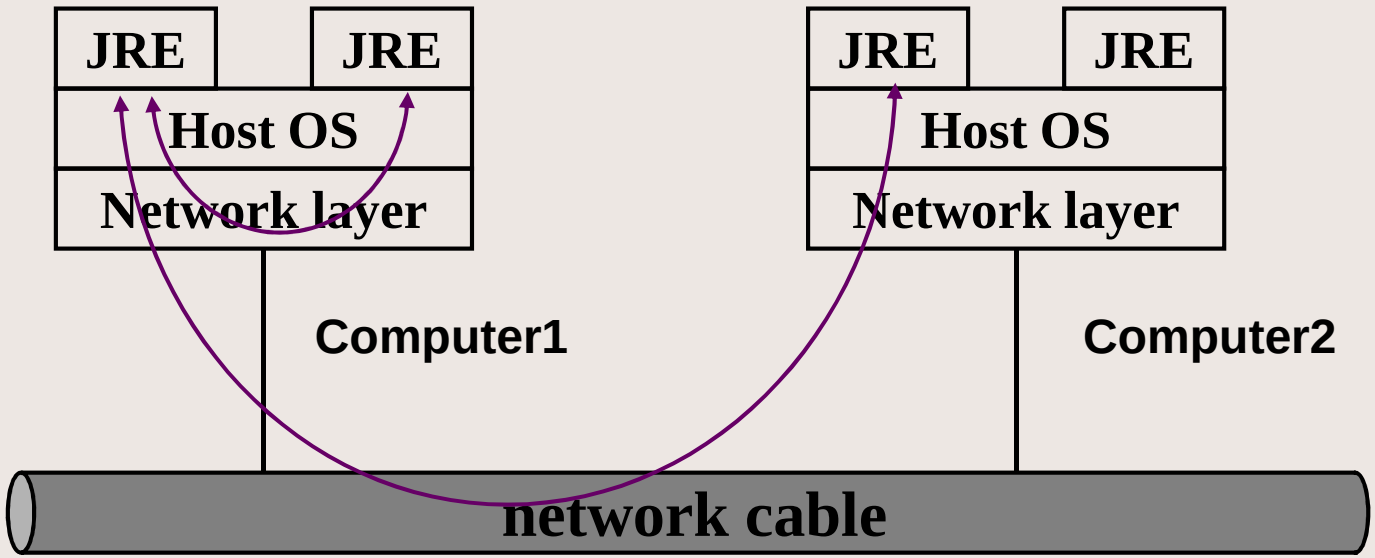
Remote reference layer

- Defines and supports the invocation semantics of the RMI connection
- Connections: unicast, point-to-point.
- Provides a ***RemoteRef*** object that represents the link to the remote service implementation object

Transport Layer

- Provides basic connectivity
  - makes the connection between JVMs
- Based on TCP/IP connections between machines in a network
- RMI-Higher level protocol

RMI communication



RMI Execution

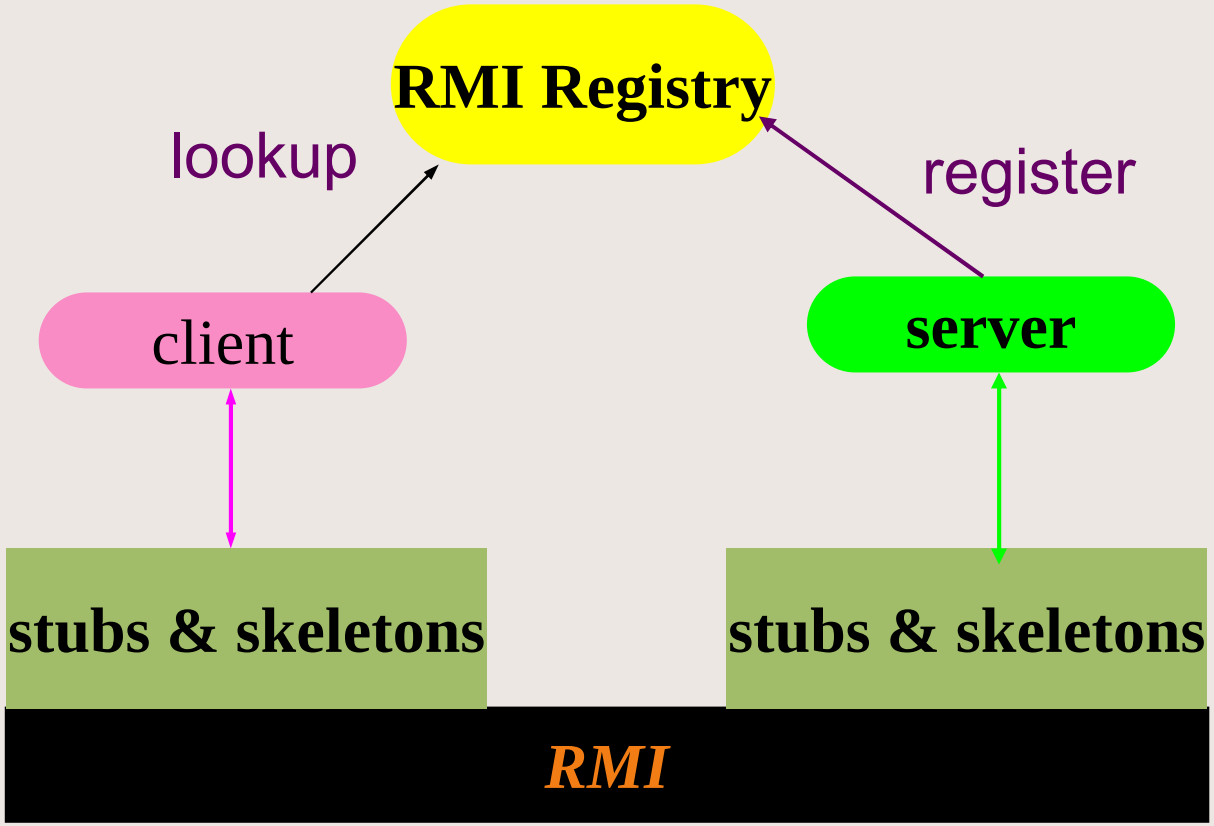
How the remote clients can find the Remote Server object and call the methods on it ?

Well, the RMIRegistry provides the information about the object to the client and a common interface acts as a link in-between them.

The remote interface is implemented by the server application and the object is exported over the RMI Link.

The client looks at the server object through the remote interface,the stub and skeleton classes provide the link of communication.

RMI Linking


RMI Registry
lookup
register
client
server
stubs & skeletons
stubs & skeletons
RMI

RmiRegistry

A **naming (Registry)** service

- –Maps names to remote object
- –Provides clients with a mechanism to find remote objects running on RMI servers

Essential operations: ***bind/rebind, unbind, lookup***

- –Bind adds a object entry to the registry
- –Unbind removes an object entry from the registry
- –Lookup allows clients to find the object's address using known object name.

Why Registry ?

- On the network implementation details must be hidden from the client.
- The client should never know where the application is running.
- The server program can change the location of objects without informing the clients, it can only update the registry.
- The client does not have to remember addresses of objects running on different machines.
- The registry is like yellow pages.

The protocol…

- The protocol used for communication is java RMI protocol.
- RMI is internally based on TCP/IP implementations.

RMI Applications

Basic Components of RMI based application:

- **Interface definitions for remote object service methods**
- **Implementations of these remote interface service methods on server side.**
- **stub and skeleton class files**
- **server(s) to host the services**
- **RMI naming service i.e. RmiRegistry**
- **client program to access the remote methods.**

Remote interface

The **Remote** interface specifies the interfaces whose methods can be invoked from a remote JVM.
User defined interfaces should extend this interface.

Every method in the user interface must declare that it throws ***java.rmi.RemoteException***

*Any objects that are to be sent over the network must be **serializable**.*

Server implementation

Java provides a base class for server side rmi implementation classes to extend.

UnicastRemoteObject:
supports the object export service for live objects
provides support for point-to-point active object references (invocations, parameters, and results) using TCP streams.

RMI Application Steps

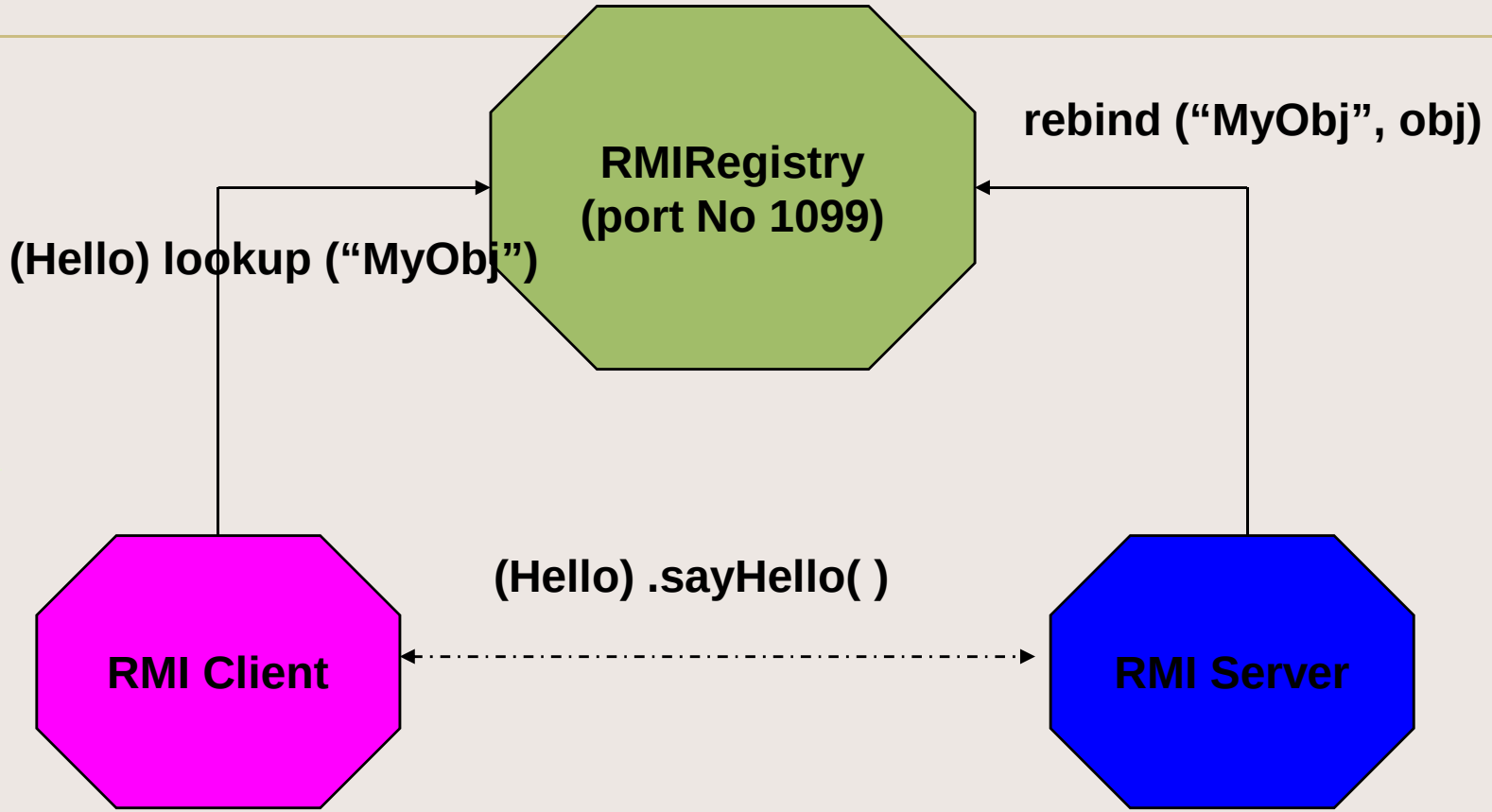
- ✓Define a common interface extending Remote and define the remote methods for remote client invocation.
- ✓Define the server class to implement this common interface i.e. implement the remote methods
- ✓In server hosting program create the server object and register with the running RmiRegistry by specifying a name.
- ✓Define the client program to query the RmiRegistry for the specified named object , get a reference and invoke the remote interface methods and process the result returned from server.

Build and Run the RMI application

- Compile the common interface.
- Compile server and client classes
- Generate stub and skeleton by invoking rmic on server class by – rmic Server on command line.
- Keep the stub in the client and server directory.

- To run…
- Run rmiregistry : start rmiregistry command
- Run Server class: java Server
- Run Client : java Client

RMI Execution



Registry access

The java.rmi.**Naming** class provides methods for storing and obtaining references to remote objects in the remote object registry.

The Naming class's methods take, as one of their arguments, a name that is URL formatted name of the Remote Object in String form: *//host:port/name*,where default protocol is **rmi** and the default port is ***1099***.

Remote client

Remote client should have common Remote interface and stub classes for remote objects at its end.

The client will look into the Naming service i.e RmiRegistry for Remote object registration and if registered, will get a remote reference for the object.

The client looks at this remote reference through common Remote interface.

On this remote reference the client will invoke the methods(these will be executed on server side) and get the results of execution if any.This is same as calling methods on local objects.

Remote method execution

- The remote method invoked by the rmi client is always **executed** at server end.
- If the method throws any Exceptions, where they will be thrown ? At ***server*** side and if not caught properly they will be sent to the ***client*** as the result of execution.

Internet applications

- Build systems incrementally: add clients and servers to Web-based systems as needed.
- Decentralized Management: Only central coordination required is the registration of DNS names
- High degree of interoperability, scalability and manageability across browsers and web servers because of standardized protocol HTTP and HTML language for graphics.

Web Programming Model

- Involves web client-server interaction
- Uses http protocol and html for data exchange
- Exchange Messages that carry MIME-typed data
- Http message attributes can be modified using headers of http
- The destination of a message is specified indirectly using a URL and this level of indirection can be leveraged to implement load balancing, session tracking and other tasks.
- Web applications are loosely coupled than the traditional distributed programming models like RPC, DCOM, and CORBA.

What is a Web Service ?

- A Web service is a software system designed to support **interoperable** machine-to-machine interaction over a network
- *Generally web services are small task operations supported by applications.*
- *These tasks are reusable across different OS platforms and programming language applications.*

Web service in SOA World

- Service Oriented Architecture allows
- Any application can talk to any other independent of language and platform.
- SOA describes every application as a service.
- New applications are built by combining existing services together.
- Web service follows SOA….

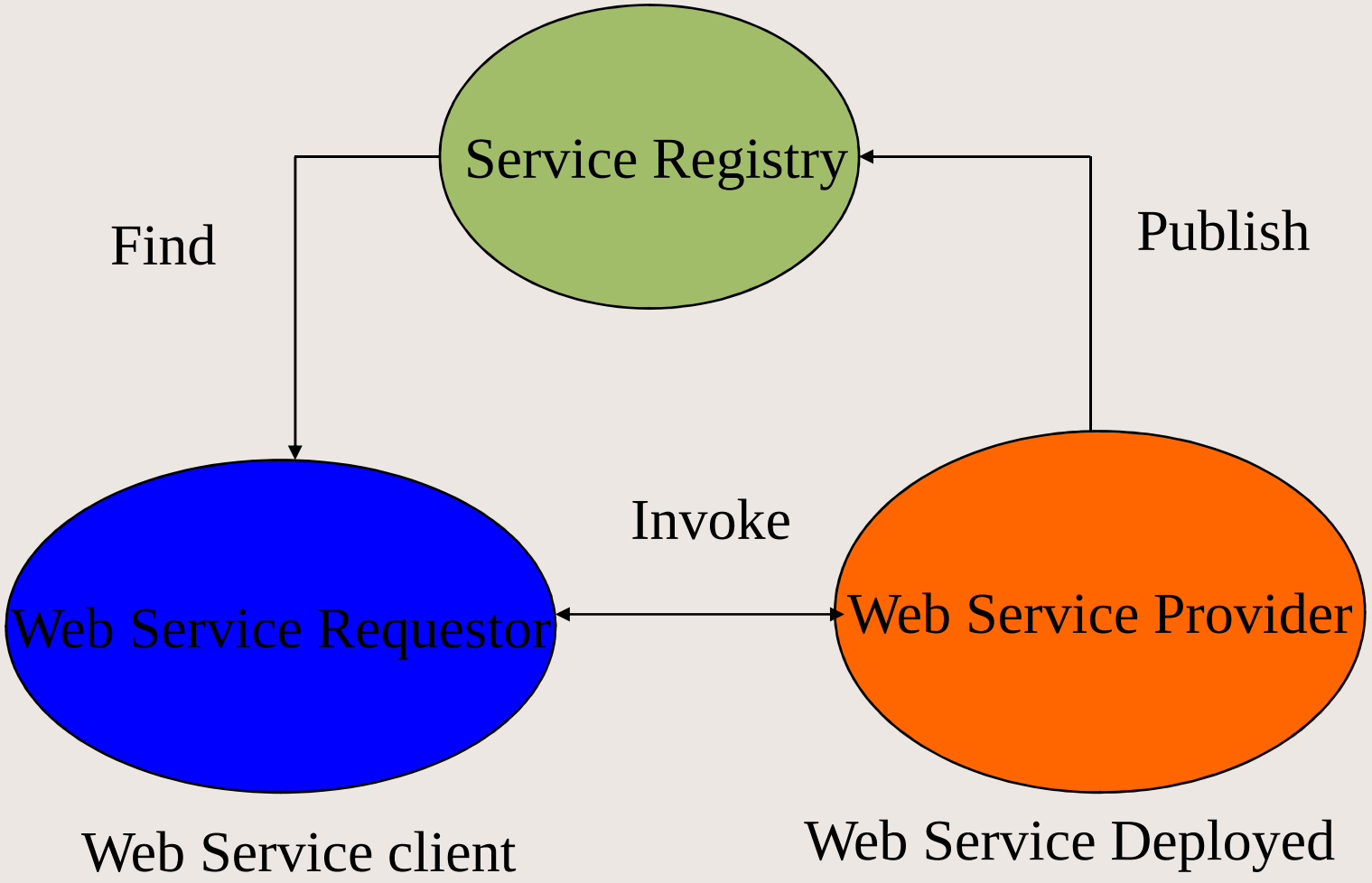
Web service features

- Adapts the loosely coupled Web programming model for use in applications that are not browser-based.
- Provides a platform for building distributed applications using software components that are..
  - running on different operating systems and devices,
  - written using different programming languages and tools from multiple vendors,
  - all developed and deployed independently

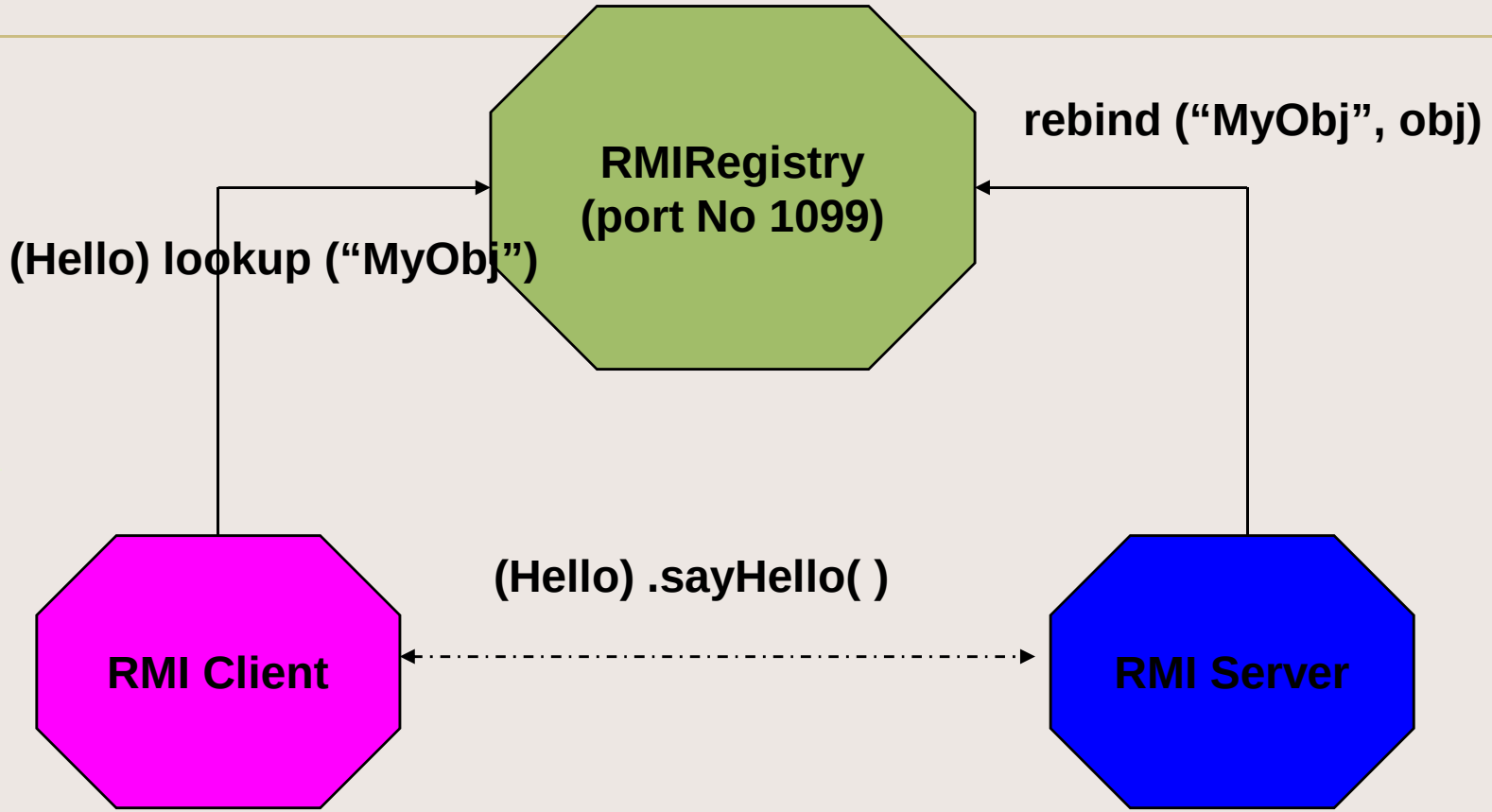
Web Service in networks

- The web services are standard mechanism for the applications to communicate with each other.
- The web services are self contained services.
- The web services are language independent and OS independent.
- Enables enterprise applications to be extended as Web services in a manner that is standard, easy, portable, reusable and interoperable

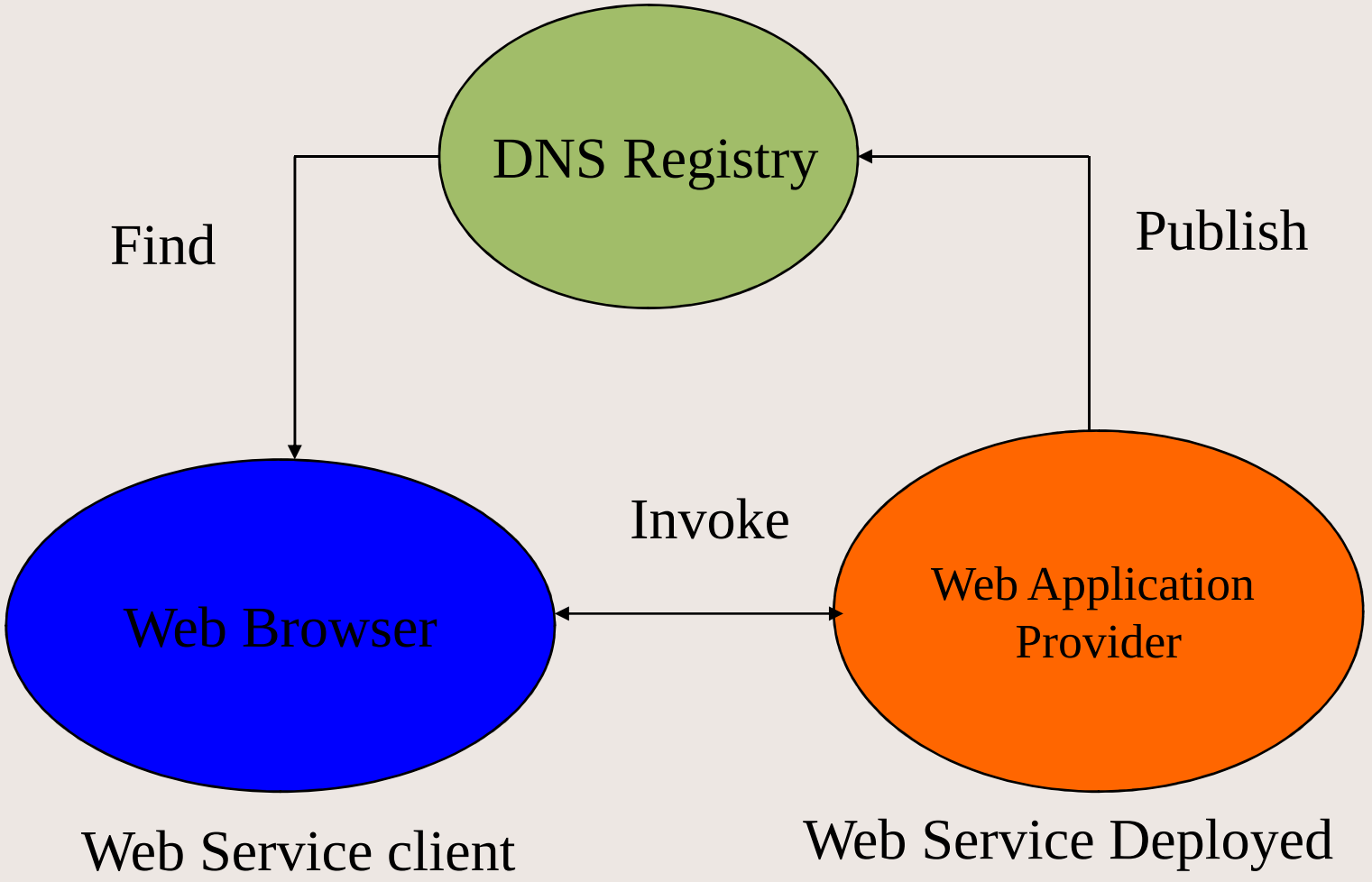
Web Service Conceptually



RMI Execution



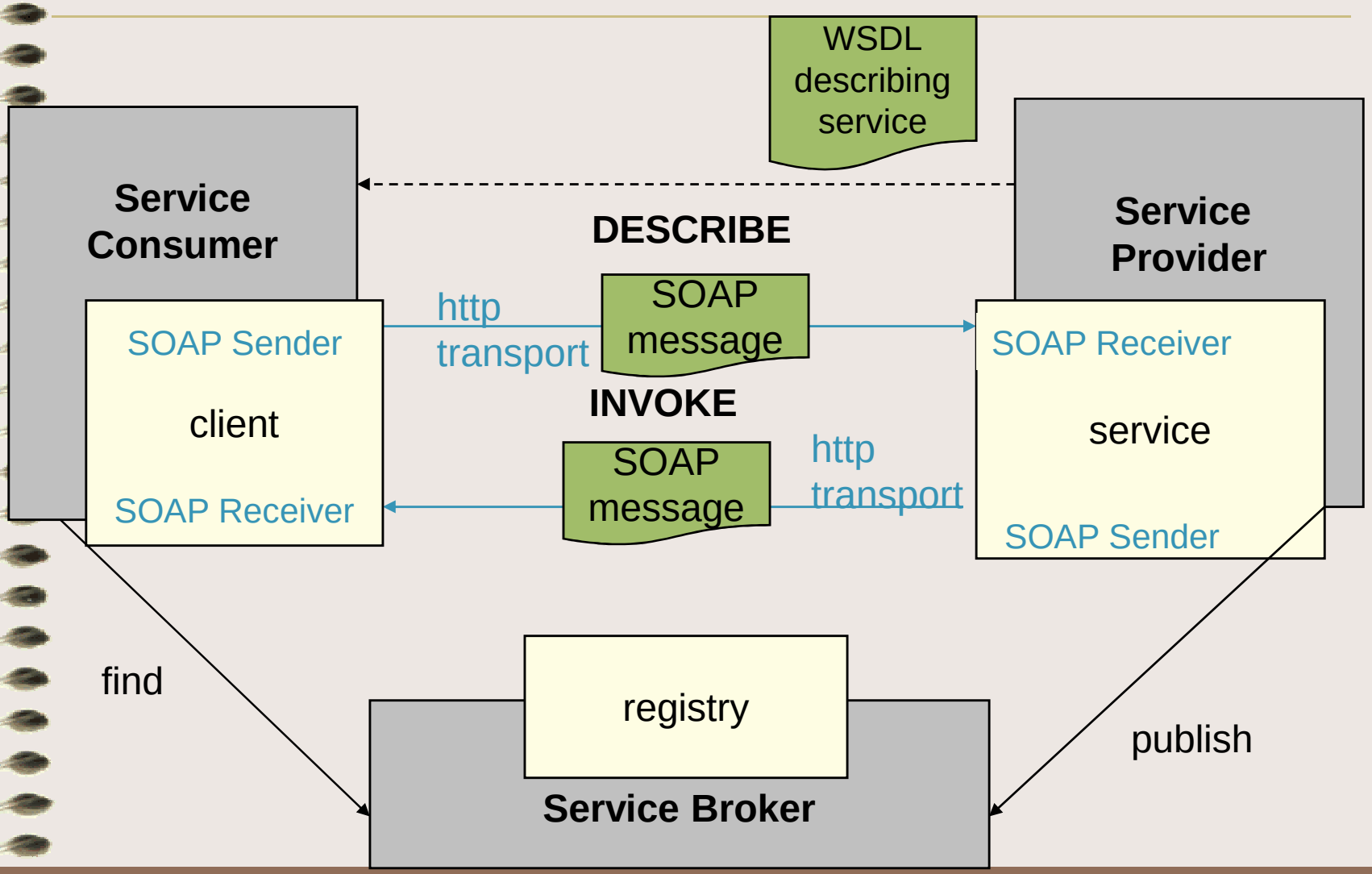
Web Browsers and server



Web Service Communication

- Web service and client use Simple Object Access Protocol (SOAP) messages instead of MIME messages (like in HTTP)
- Web services are not HTTP-specific, they can use any other *transport* protocol
- Web services provide metadata in *wsdl* that describes the messages they produce and consume.

Web Service in Action



Web Service Standards

- SOAP
- – Simple Object Access Protocol for web service and client to communicate.
- WSDL
- – Web Service Definition Language for web services to describe their services and other information.
- UDDI
- – Universal Description, Discovery, and Integration protocol for Web Services to discover web services.
- All these are built upon XML.

XML In Web Services

- Web Services fundamentally uses XML to standardize the behaviors and data.
- Key XML standards to understand
  - XML : extensible mark up
  - XML Schema specifies xml structure
  - XML Namespace avoids naming conflicts

Web service and web applications

Web Service

- XML/SOAP
- Program-to-program interaction
- Static or dynamic integration
- Re-usable service
- Interpret XML-based SOAP messages
- Transport protocols supported are HTTP ,SMTP,JMS.
- Self-describing with WSDL

Web applications

- HTML/HTTP
- User-to-program interaction
- Static integration of components
- Single use service
- Render MIME-type messages
- Use HTTP
- Use header information to change request/response behavior

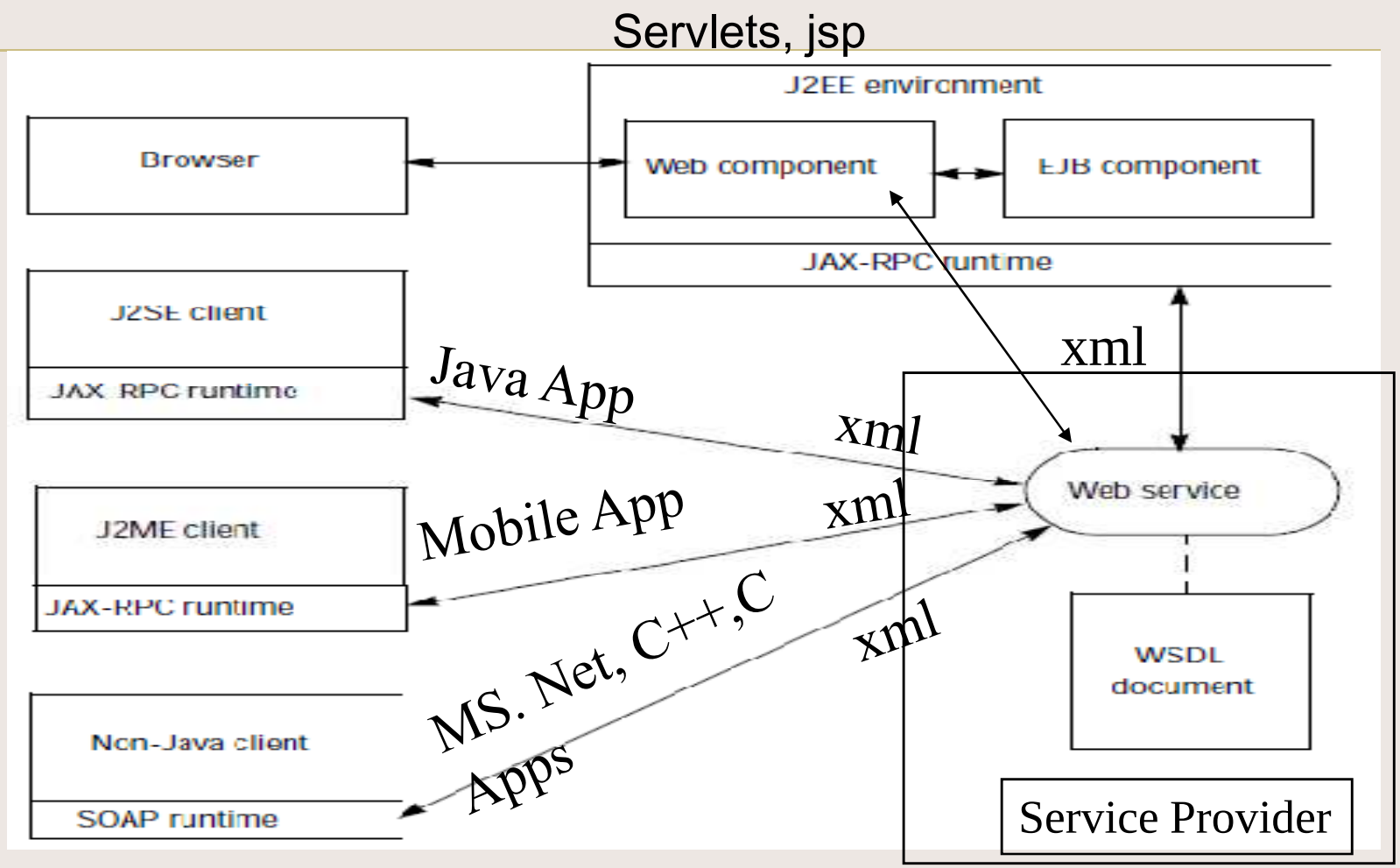
Web service Usage

- A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML.
- Its definition can be discovered by other software systems.
- These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed over internet transport protocol

Web Service Components

- What are the components required for Web service on server side ?
  - Common interface implementation
  - Required stub/skeleton layer components
  - Soap Runtime library
  - Web service Framework
  - Service Registry

Clients for Web Service



Java Support for Web Services

- JAXP – Java API for XML Processing
- JAXB - Java Architecture for XML Binding
- JAXR - Java API for XML Registries
- JAXM – Java API for XML Messaging
- JAX-RPC - API for XML-based RPC
- SAAJ - SOAP with Attachments API
- JAX-WS - Java API for XML Web Services

SOAP Run time

- JAX-RPC : SUN, Apache Axis1.4
- SAAJ : SUN ,Oracle
- JAX-WS
  - Metro
  - Sun Jax-ws reference
  - Apache CXF
- Apache Axis2
- gSoap C++ library/framework
- Microsoft .Net Framework
- IBM, Oracle Implementation libraries

JAX-WS Standard

- JAX-WS is a new technology designed to simplify the development of Web services and Web service clients in Java.
- It replaces earlier JAX-RPC standard.
- It provides a complete Web services stack that eases the task of developing and deploying Web services.
- JAX-WS includes the Java Architecture for XML Binding (JAXB) and SOAP with Attachments API for Java (SAAJ).

JAX-WS Features

- JAX-WS supports message (Document) oriented as well as RPC-oriented web services and web service clients
- Supports annotations for configuration.
- Server side framework for deploying web services.
- Supports synchronous and asynchronous calls to web services.

Developing a web service

- The Web service developed using one of two approaches
  - Define java interface and implementation classes and generate required components wsdl,xsd etc (**bottom up** approach).
  - Define wsdl and generate corresponding java classes using JAXB (**top down** approach)

Web Service attributes

- **endpointInterface** :The complete name of the service endpoint interface defining the service's abstract Web Service methods.
- **name** :The name of the Web Service.
- **portName** :The port name of the Web Service.
- **serviceName:**The service name of the Web Service.
- **targetNamespace:** used to define namespace for the wsdl:portType and associated XML elements if it is defined for on a service endpoint interface
- **wsdlLocation:** The location of a pre-defined WSDL that describes the service.

WebMethod annotations

- Customizes a method declared in interface that is exposed as a Web Service operation.
- The associated method must be public and its parameters return value and exceptions if any must be defined.

SOAPBinding.Style

- Defines the soap messaging style of communication.
  - SOAPBinding.Style.**DOCUMENT**
  - SOAPBinding.Style.**RPC**

Building a Web service

- Define java interface with abstract methods and annotations.
- Define a java class with ***annotations*** to implement interface methods.
- Create a web application directory structure.
- Define mapping in ‘web.xml’ and endpoint configurations in ‘sun-jaxws.xml’ in WEB-INF.
- Use JAX-WS tools to generate required server side classes.
- Put these class files in WEB-INF/classes directory.
- Make a ‘war’ file and deploy in web server.
- Put required JAX-WS jar libraries from JAX-WS/lib directory to server/lib directory.
- All these command line tasks can be automated using ‘ANT’

Apache CXF Tools

- **Java2Wsdl** : Generates the web service artifacts (i.e wsdl and other)from java classes.
- **Wsdl2Java** : Generates the web service artifacts (i.e Java code and other)from wsdl.
- **Jax-ws** annotations processor
- **Service EndPoint** configuration

Web Service Build Tools

- ANT
- NetBeans
- Eclipse
- IntelliJ
- MyEclipse

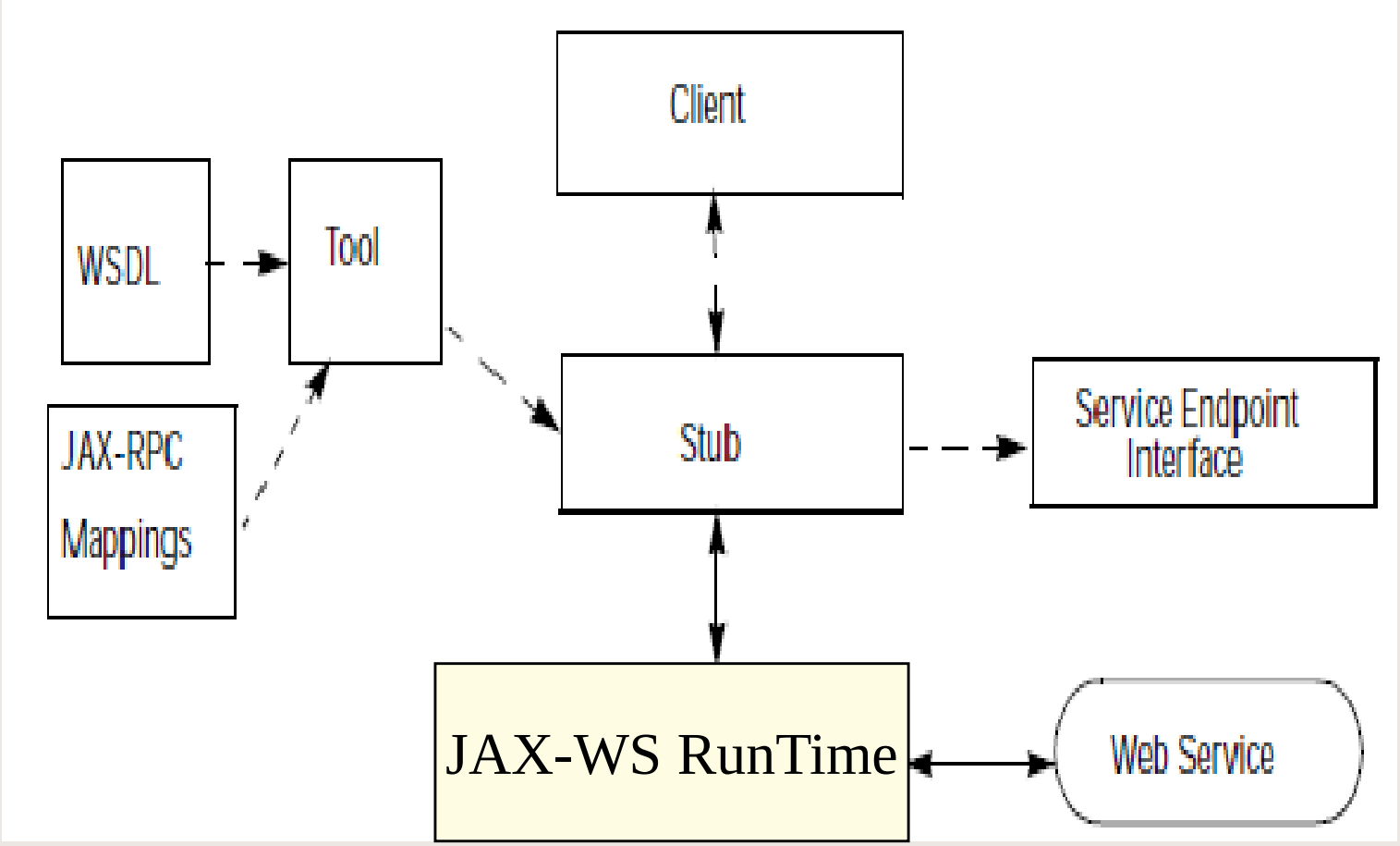
Clients to Web Service

- What are the components required for a Web service client ?
  - Common interface
  - Required stub layer components
  - Soap Runtime library
  - Web service URL information
  - Access to Service Registry

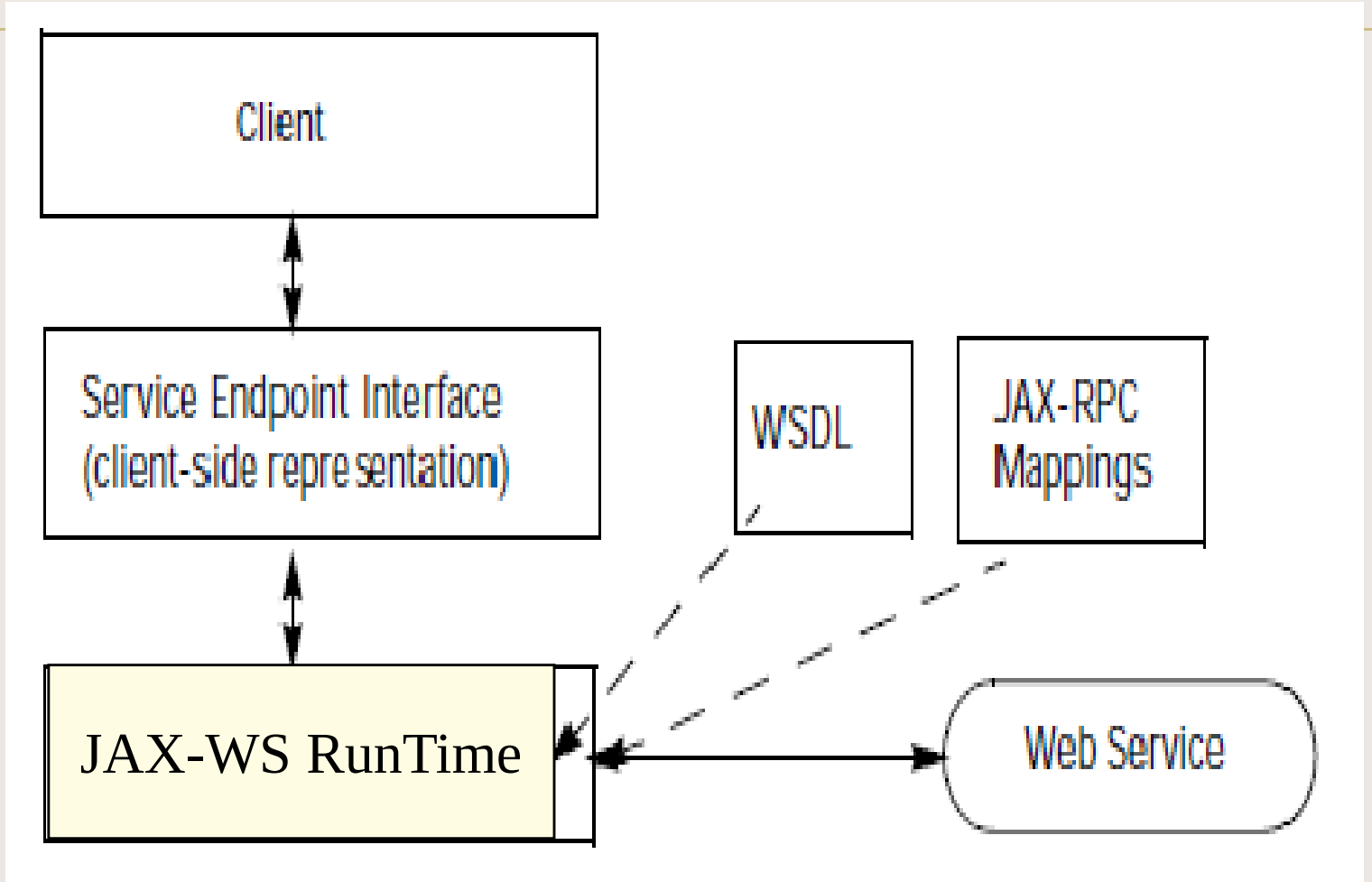
Web Service Clients

- Three types of clients supported by JAX-WS
  - **Static Stub Client** : Client program using stub components generated by JAX-WS:
  - **Proxy Client**: Client having only the web service interface and wsdl lookup to server generate runtime proxy for interface and invokes operations using interface type.
  - **Dynamic Dispatch Client**: Client program having nothing but JAX-WS components to generate runtime calls by sending soap messages.

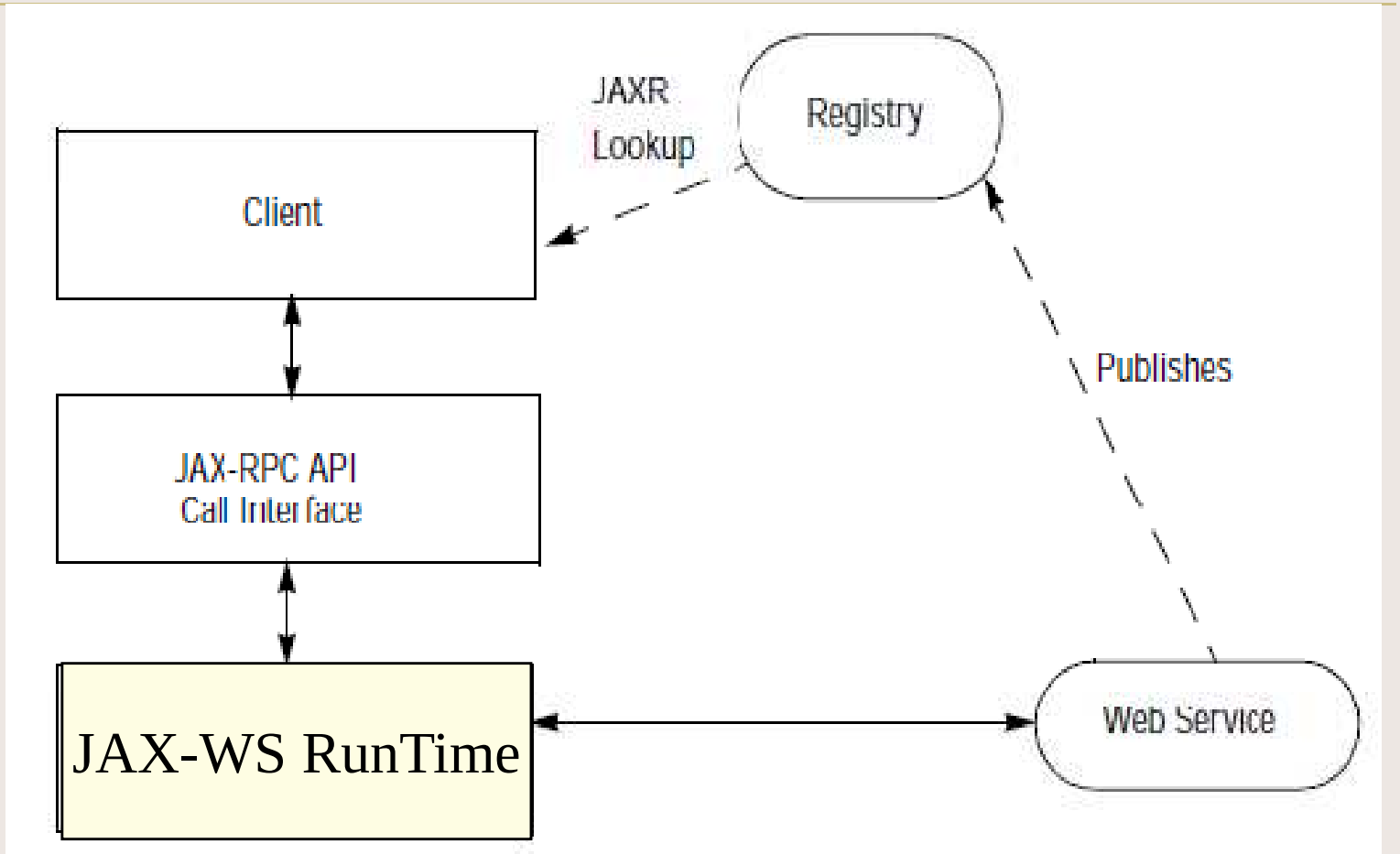
Static Stub Web Service Client



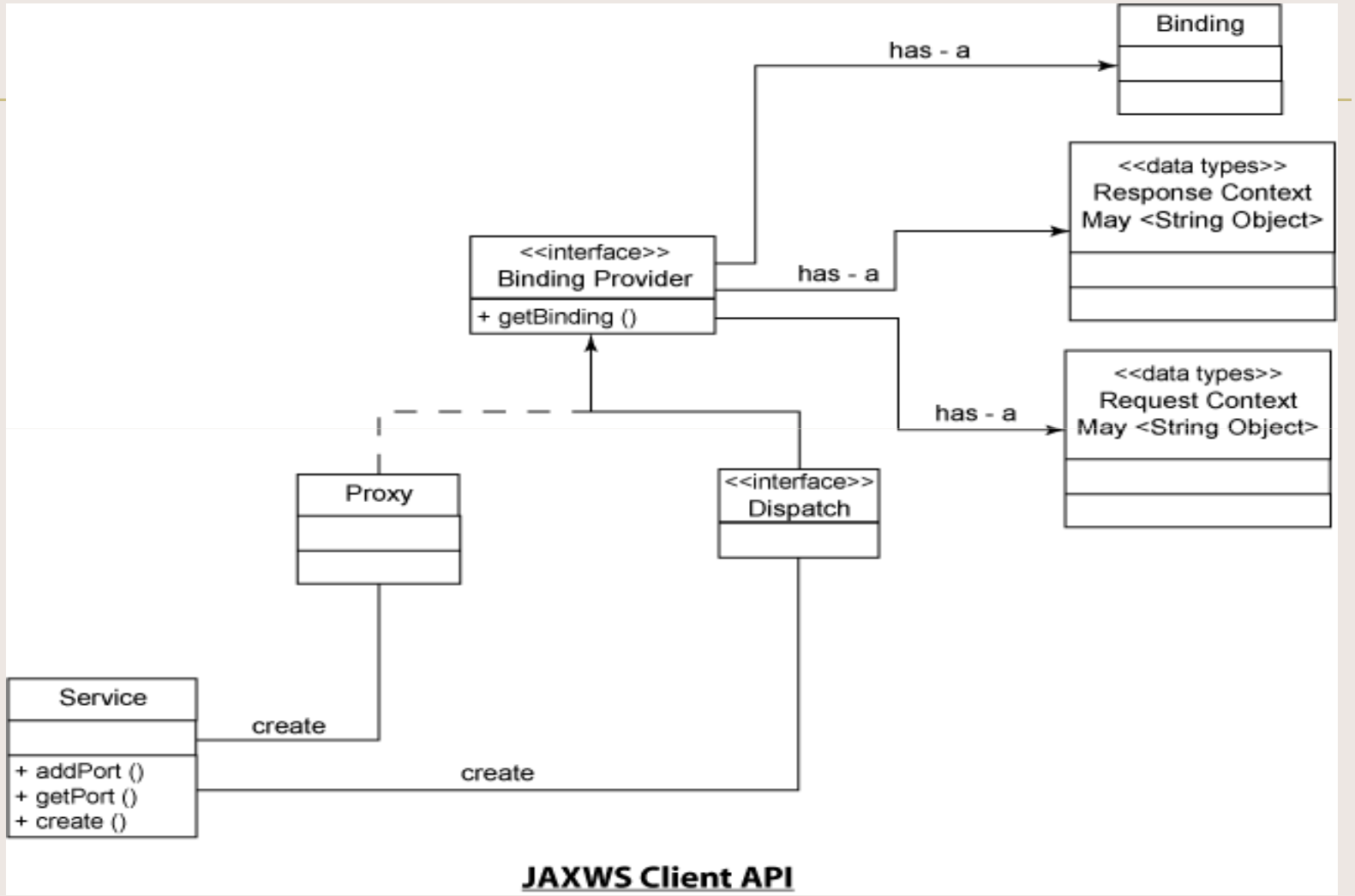
Proxy Client



Dynamic Client



JAX-WS Client API



Jax-ws Service

- Defined in package javax.xml.ws.
- The ‘Service’ objects provide the client view of a Web service.
- Service used to create proxies for a target service endpoint.
- Service also used to create instances of javax.xml.ws.Dispatch for dynamic message-oriented invocation of a remote operation.

QName

- Defined in javax.xml.namespace package.
- QName used to represent a **namespace qualified name** as defined in the XML specifications.

Web Service Discovery

- The web service definitions can be dynamically discovered by other software systems independent of platform and programming language.
- These systems then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.

UDDI

- Enables enterprises to quickly and dynamically discover and invoke Web Services both internally and externally
- Standards based specification for service description and discovery
- Programmatic descriptions of businesses and services are supported.
- UDDI Supports
  - Cataloging
  - Searching (humans or machines)
  - UDDI is xml based standard
  - UDDI is XML Registry

Web Service Definition

- How the web service clients know about the operations supported by particular web service ?
- Web services are self describing by the Web Service Definition language standard.
- WSDL has XML Schema for describing Web Services
- WSDL is in XML format for describing services operations in network, operating styles and service bindings.
- The operating styles can be either document-oriented or rpc-procedure-oriented information.
- Defines Web Services as collection of network services or ports.
- Allows both the operations and the messages (data) on the to be defined abstractly in XML

WSDL Format

- WSDL is an XML-based language that defines formal descriptions of the Web Services.
  - which interactions does the service provide?
  - which arguments and results are involved in the interactions?
  - which network addresses are used to locate the service?
  - which communication protocol should be used?
  - The messages are represented in which data formats ?

WSDL Structure

<definitions>: Root WSDL Element

- **<types>:** Data types used by web service
- **<message>:** Messages used by web service
- **<portType>:** Operations performed by web service
- **<binding>:** Communication protocols used
- **<service>:** Location of the service

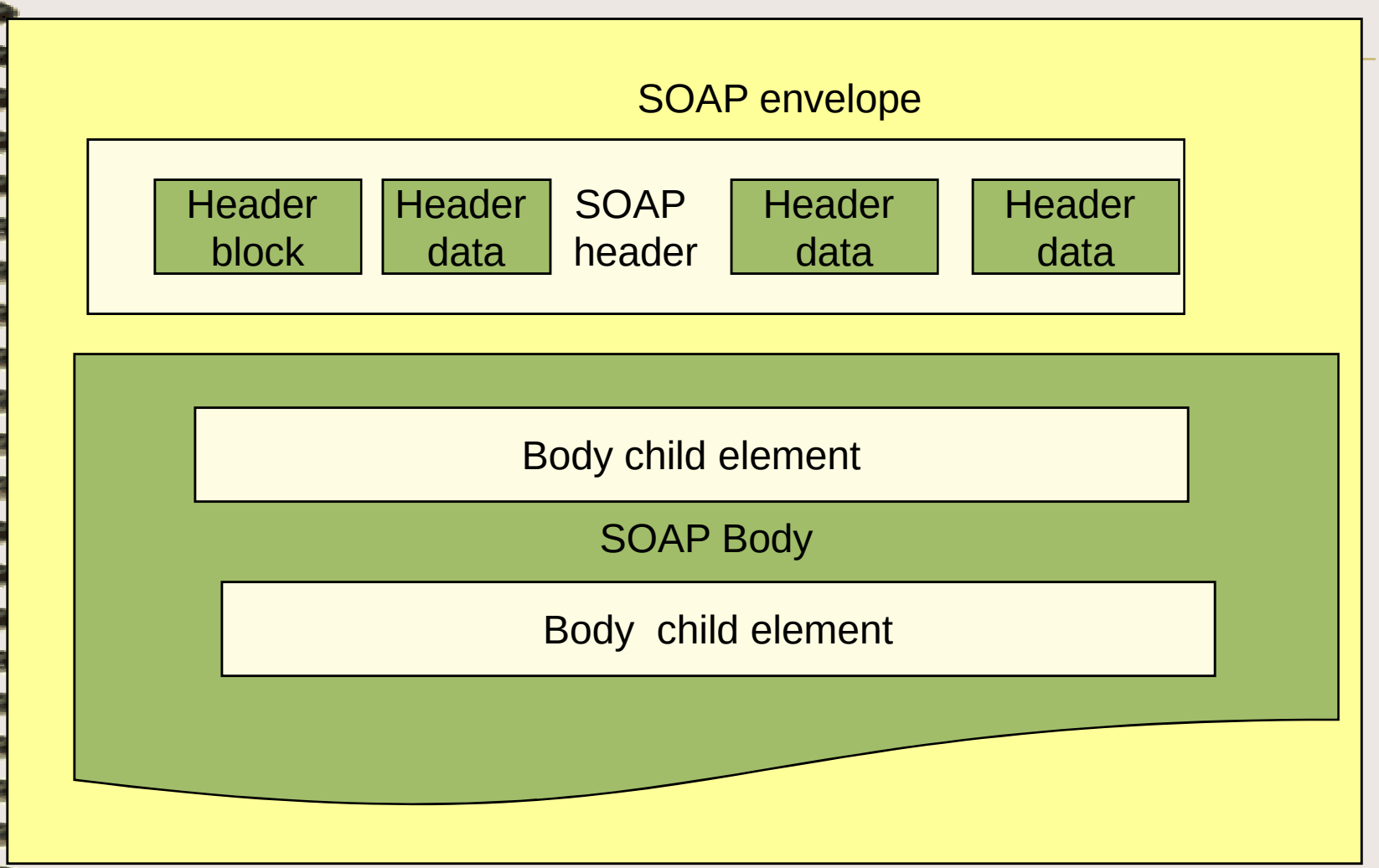
WSDL Bindings

- A WSDL document describes a Web service.
- A WSDL binding describes how the service is bound to the SOAP messaging protocol.
- A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding.
- A SOAP binding can also have an encoded use or a literal use. This gives four style/use models:
  - RPC/encoded
  - RPC/literal
  - Document/encoded
  - Document/literal

SOAP Messaging

- SOAP is a lightweight XML protocol for exchanging structured and typed information
- One way message based
- Exchange of information in a distributed environment
- Self-describing data representation format for requests and responses as XML messages

SOAP Message Structure



SOAP XML


```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<n:alertcontrol xmlns:n="http://server.org/alertcontrol">
                    <n:priority>1</n:priority>
                    <n:expires>2001-06-22</n:expires>
</n:alertcontrol>
</env:Header>

    <env:Body>
        <m:alert xmlns:m="http://server.org/alert">
        <m:msg>Pick up Me at school at 2pm</m:msg>
        </m:alert>
    </env:Body>

 </env:Envelope>
```

SOAP Transport Protocols

- **HTTP** : The web service deployed on web server uses http for transporting soap xml messages between client and server.
- **SMTP**: The soap messages can be transported over SMTP in mail exchange server implementations.
- **JMS**: The soap messages get transported over Java Messaging Service in J2EE environments

Soap Messaging Style

- Soap supports two messaging styles
- RPC Oriented :
  - The incoming request message is treated as remote method invocation request
  - The method is invoked on server side
  - Results are returned to client in xml format
- Document Oriented
  - Messaging is done by exchanging xml format documents.

SOAP Document Style

- There are two SOAP message styles called document and rpc.
- **Document style** : indicates that the SOAP body simply contains an XML document.
- The sender and receiver must agree on the format of the document ahead of time.
- The agreement between the sender and receiver is typically negotiated through literal XML Schema definitions. Hence, this combination is referred to as document/literal.

SOAP RPC Style

- **RPC** :Remote Procedure Call style indicates that the SOAP body contains an XML representation of a method call on a remote object.
- RPC style uses the names of the method and its parameters to generate structures that represent a method's call stack
- These structures can then be serialized into the SOAP message according to a set of encoding rules.
- The SOAP specification defines a standard set of encoding rules for mapping to xml format.
- Since RPC is traditionally used in conjunction with the SOAP encoding rules, the combination is referred to as rpc/encoded.

SOAP Encoding

- The Soap encoding/usage rules define how the wsdl xml data bindings are translated to soap xml message.
- Normally, data can be carried in XML as an attribute or as element content. The SOAP serialization rules only allow for data to be carried as element content.
- Encoding rules include strict type specifications while literal rules allow loosely defined type specifications.

More on Encoding

- The SOAP encoding rules are based on XML Schema specifications.
- These specifications cover everything from simple data types, such as integers, to complex data types, such as structures with nested structures conversion in xml.

SOAP Encoding Rules

- Three rules evolved
  - Encoded
  - Literal
  - Literal wrapped

SOAP Style/encoding

- RPC-Encoded : Not supported by WS-I and limited interoparability.
- RPC-Literal: supported by WS-I and interoparability across platforms.
- Document-Enoded: Not supported.
- Document-Literal : supported with interoparability across platforms
- Document-Lieral Wrapped : Major std combination to support interoparability.

SOAP Faults

- Standard way to describe errors
- The faults inside env:Body elements
- In single env:Fault
- env:Node identifies node which generated fault
  - Absence indicates “ultimate recipient”
- env:Code
  - env:Value
  - env:Subcode
- env:Reason
  - env:Text
- env:Detail
  - Application specific

SOAP Fault Example


```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:rpc='http://www.w3.org/2003/05/soap-rpc'>
<env:Body>
  – <env:Fault>
      • <env:Code>
          – <env:Value>env:Sender</env:Value>
          – <env:Subcode>
              » <env:Value>rpc:BadArguments</env:Value>
          – </env:Subcode>
      • </env:Code>
      • <env:Reason>
          – <env:Text xml:lang="en-US">Processing error</env:Text>
          – <env:Text xml:lang="cs">Chyba zpracování</env:Text>
      • </env:Reason>
      • <env:Detail>
          – <e:myFaultDetails xmlns:e="http://shippingservice.org/faults">
              <e:message>Unknown destination</e:message>
              <e:errorcode>999</e:errorcode>
          – </e:myFaultDetails>
      • </env:Detail>
  – </env:Fault>
</env:Body>
</env:Envelope>
```

SOAP Faults on MustUnderstand


```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope>
<env:Header> <env:NotUnderstood qname="t:transaction"
     xmlns:t="http://shippingservice.org/transaction"/>
 </env:Header>
<env:Body>
     <env:Fault>
          <env:Code>
               <env:Value>env:MustUnderstand</env:Value>
          </env:Code>
          <env:Reason>
          <env:Text xml:lang="en-US">Header not understood</env:Text>
          <env:Text xml:lang="fr">En-tête non compris</env:Text>
          </env:Reason>
      </env:Fault>
</env:Body>
</env:Envelope>
```

SOAP Processing Model

- SOAP messages are sent from one **sender** node passing through zero or more intermediaries
- Three roles
  - **next:** each SOAP intermediary or end destination must act in this role
  - **none:** SOAP nodes must not act in this role
  - **ultimateReceiver:** destination acts in this role
- Header blocks targeted to specific roles using **Role** attribute
- If **mustUnderstand="true"** SOAP receiver must understand or generate SOAP fault
- Header blocks processed by intermediaries are generally removed before forwarding
  - Override with **relay** attribute
  - Allows targeting of headers to specific intermediaries (but **mustUnderstand** would then generally be turned off)

Type Mapping

- WSDL is an xml document and soap messages are also in xml format.
- What if the web service is sending some java object to .Net/C++ client ?
- Some additional type mapping is to be provided that will map application objects to xml data types which may be defined using xml schemas and namespaces.
- In JAX-WS this type mapping is handled internally by JAXB components dynamically.

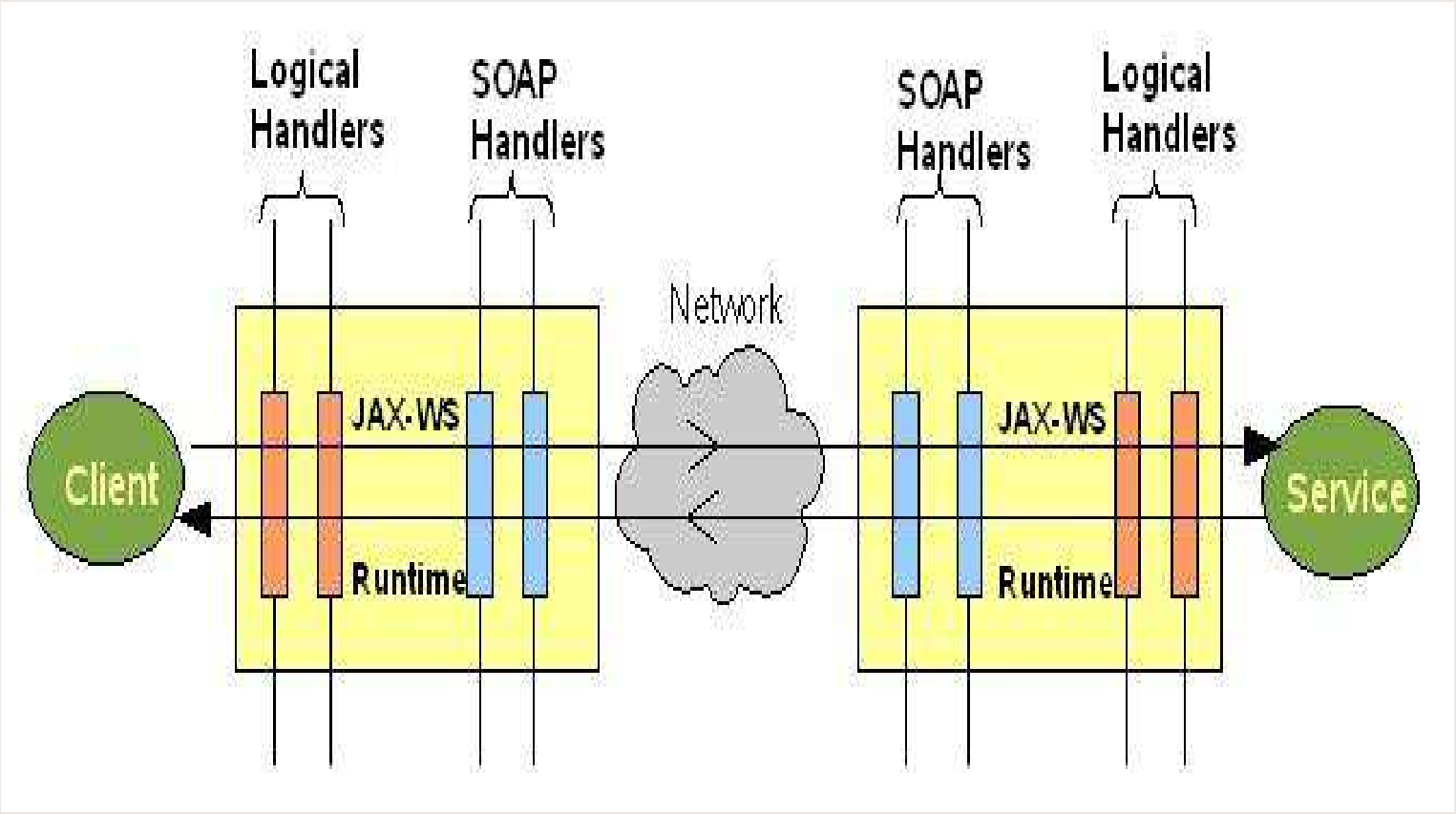
Handlers

- Normally clients send request to web service and web service sends response to clients directly.
- What if we want to add some component in-between the request to web service or in the response to web client before it reaches the destination ?
- Handlers support theses features.

Message Handlers

- Handlers are message interceptors that can be easily plugged in to the JAX-WS runtime to do additional processing of the inbound and outbound messages.
- JAX-WS defines two types of handlers
- **Logical handlers**: Logical handlers are protocol independent and cannot change any protocol specific parts (like headers) of a message. Logical handlers act only on the body part of the message.
- **Protocol handlers:** Protocol handlers are specific to a protocol and may access or change the protocol specific aspects of a message.

Handlers in Message



Asynchronous calls

- Normally web services follow a request-response paradigm where client sends a request and waits for response till the service sends it back.
- What if the web service client wants to use the web service asynchronously ?
- In this case the client may interact with a web service in a non-blocking, asynchronous approach.

Asynchronous with Polling and Callback

- The client may get an polling agent from service as a result of call which will be used by the client to poll for certain results on server side.
- In second approach the client will specify a callback listener and send the request and continue doing its work without waiting for the response. The service will call back the listener whenever the response is ready for the client.

JAX-WS Async

- **Async Polling** :The client application can invoke the async polling operation on the stub and check for a response on the returned Response object.
- The response is available when Response.isDone() returns true.
- **Async Callback** :client application provides an AsyncHandler by implementing the javax.xml.ws.AsyncHandler<T> interface.

Sending large binary data

- XML data is sent in lined inside the SOAP envelope.
- This gets quite in-efficient when the data size is more, for example a SOAP service that exchanges images/songs etc.
- XML Binary Optimized Packaging (XOP) defines how an XML binary data can be optimally transmitted over the wire.
- Message Transmission and Optimization Mechanism (MTOM) standard specifies how XOP packaging can be used to send the binary data optimally over the network.

MTOM in JAX-WS

- MTOM feature is disabled in jaxws by default.
- It can be enabled on the client and server.
- Once enabled all the XML binary data, XML elements of type xs:base64Binary and xs:hexBinary is optimally transmitted.
- Currently MTOM works only with proxy port clients.

Enable MTOM

- MTOM is enabled by annotation in service implementation class as @javax.xml.ws.soap.MTOM
- MTOM can be also be enabled on an endpoint by specifying enable-mtom attribute to true on an endpoint element in ‘sun-jaxws.xml’ deployment descriptor

MTOM-Java Mappings

MIME Type	Java Type
image/gif	java.awt.Image
image/jpeg	java.awt.Image
text/plain	java.lang.String
text/xml or application/xml	javax.xml.transform.Source
/	javax.activation.DataHandler