Copyright Notice

This presentation is intended to be used only by the participants who attended the training session by Prakash Badhe, VishwaSoft Technologies.

Sharing/selling of this presentation in any form is NOT permitted.

Others found using this presentation or violation Of above terms is considered as legal offence.

Vishwasoft Technologies

Soap Web Services with java

Prakash Badhe

prakash.badhe@vishwasoft.in

Vishwasoft Technologies

Internet Network

- 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
- Interoperability and manageability across browsers and web servers because of standardized protocol HTTP and HTML language for graphics.

Web applications on Internet

- Internet applications works on client-server mode.(browser-Web server).
- Html is the language of communication.
- Http is the protocol of communication.
- The web applications support communications across different client-server operating systems and also) different networks and hardware platforms.
- The communication interface is limited..to http and html..

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.

Data Sharing challenge...

- How to share the data from web application to its clients ? Across different applications such as desktop, mobile, another web application as client,In platform/language independent way ?
- Can the client application from any platform(OS, Programming language etc.) call any non standard functions from the web application on the server ?
- How they will understand each others data formats ?
- How custom data types will be understood across the language platforms ?

Applications as services

- At abstract level if the applications are defined as platform independent services, then they can share the data and understand the data exchange...
- The SOA is the answer..
- Service Oriented Architecture…
- Web services are evolved as SOA..

Web Service Introduction

- A Web service is a software application designed to support **interoperable** machine-to-machine interaction over a network
- *Generally web services are small task /service operations supported by applications on the server/s.*
- *These tasks are reusable across different OS platforms and programming language applications.*
- *Web services operate in a distributed environment.*

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 features

- Adapts the loosely coupled Web programming model for use in applications that may not be 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 Services 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.
- How the communication happen ? - Remote Communication..
- How the data is shared in language/os independent way ? – platform neutral data formats
- Certain RPC Mode communication needed..

RPC in Distributed world

- RPC specifies invoking procedures on one machine remotely from another machine in the network.
- Microsoft has DCOM and Remoting standards.
- CORBA brokers support RPC across language platforms
- In Java RMI methods of object running on one machine are invoked from another machine in the network and results are passed to the caller.(same is followed in EJB)

Serialization and deserialization

- RPC involves object/parameters serialization and deserialization
- It is process of converting of an object/parameter into a stream of bytes and reconstructing the object/parameters from the serialized data.
- Java supports serialization process through JVM via interface ***Serializable,e***ach class that has to be serializable must implement the ***Serializable*** *interface****.***
- *This* process of encoding/decoding arguments, objects and results for transmission over the wire is also called as Marshalling and unmarshalling. .
- Thus objects/parameters to be marshaled or un marshaled over the network *must be serializable*
- *The data m*arshalling format must be understood by all applications involved in the communication.

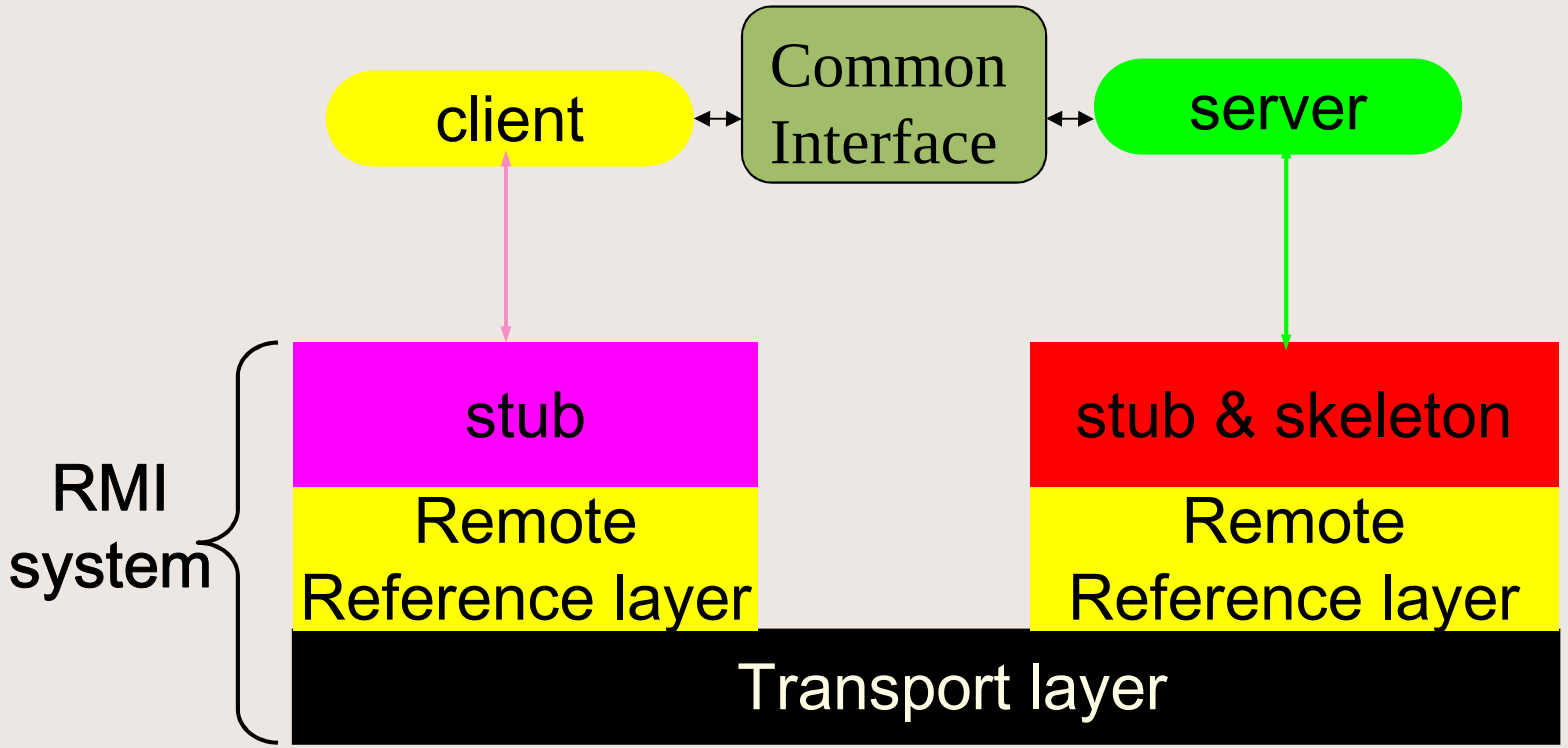
Object/Info sharing

- How a running object 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 implementing component of the running code will not be shared as happens in normal desktop standalone applications where the required components/classes are loaded from local systems/path/classpath etc.

Object 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 ?
- Java 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.
- Who provides this sharing and linkage?

RPC/RMI Architecture



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

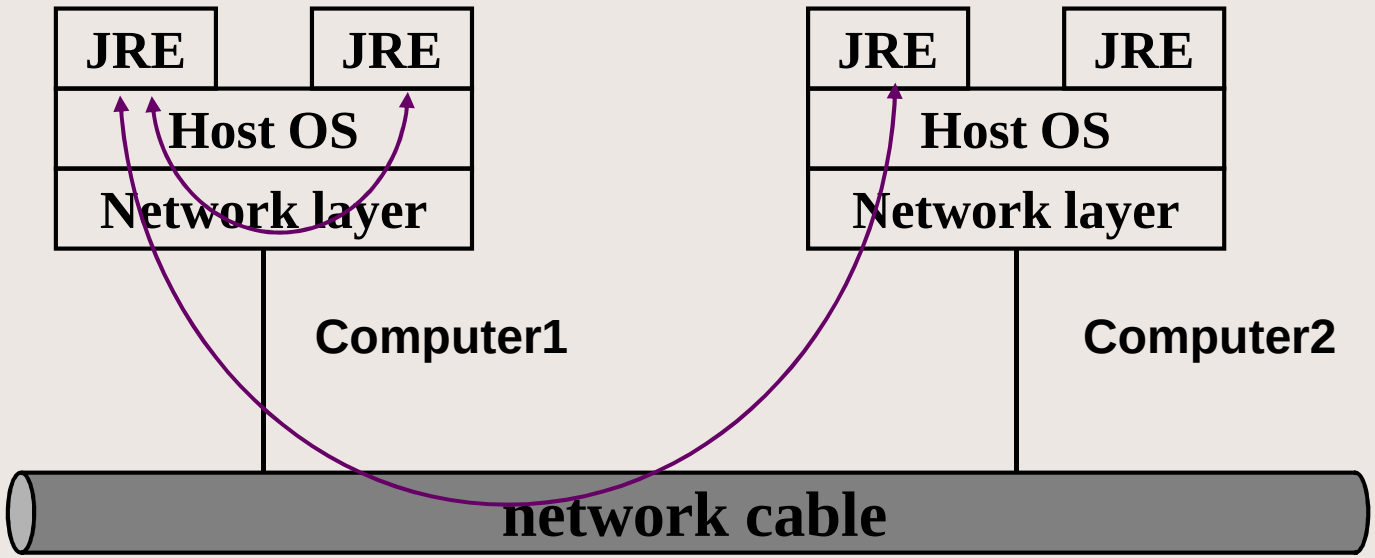
Stub and Skeleton class

- 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.

Transport Layer

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

RMI communication



RMI Execution

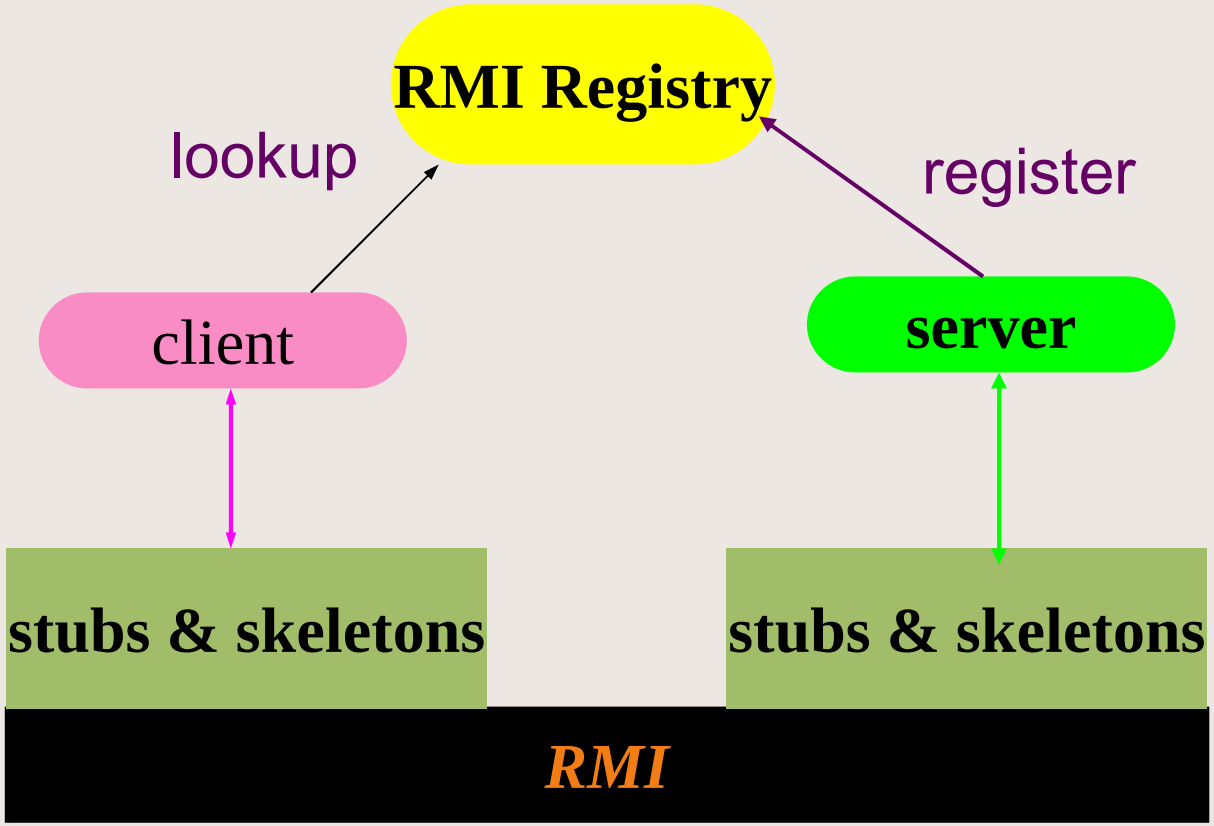
How the remote clients can find the Remote Server object and call the methods on it ?

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

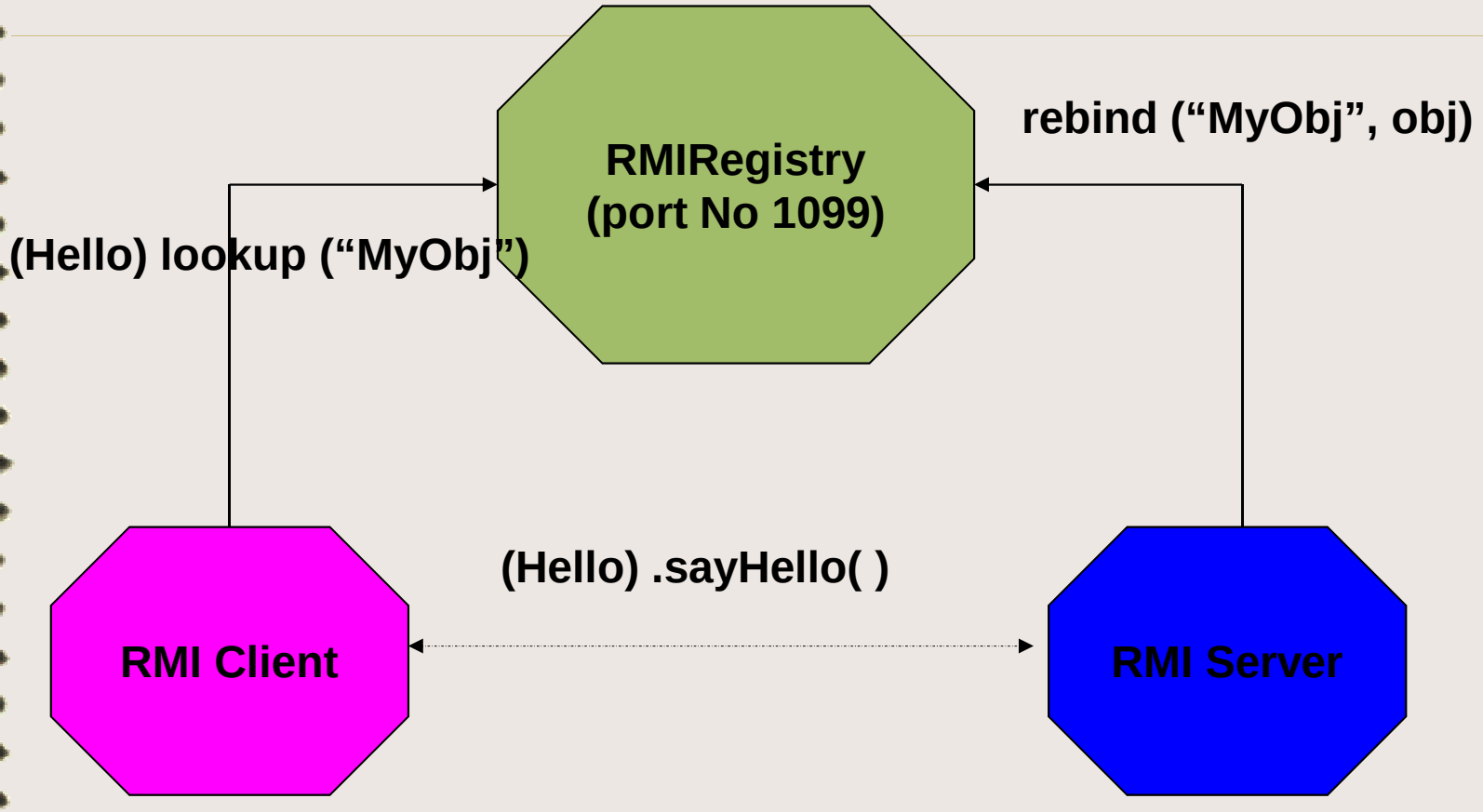
Essential operations: ***bind/rebind, unbind, lookup***

- –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.

RMI Execution



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.

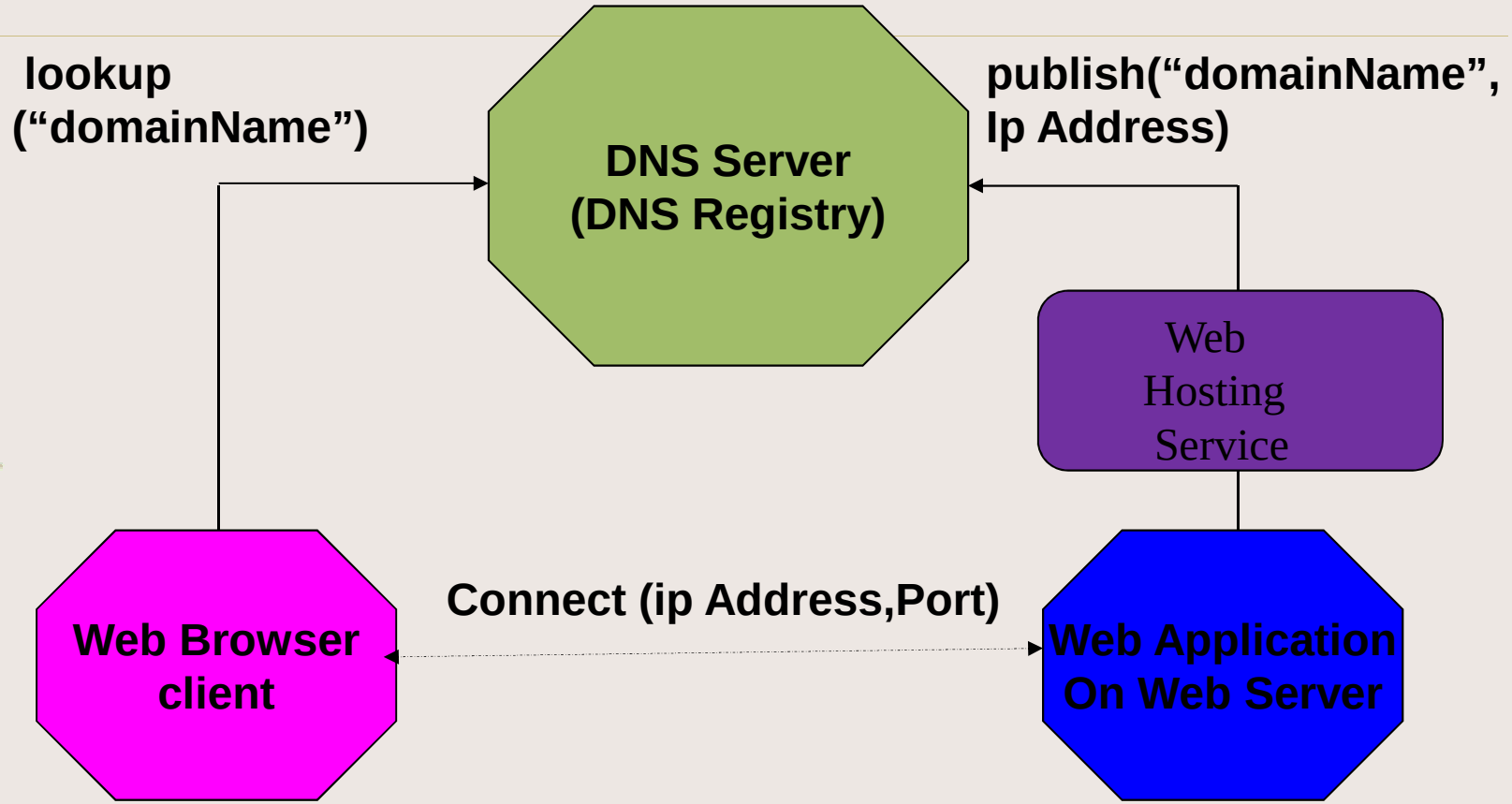
The client looks at this remote reference through common Remote interface.

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.

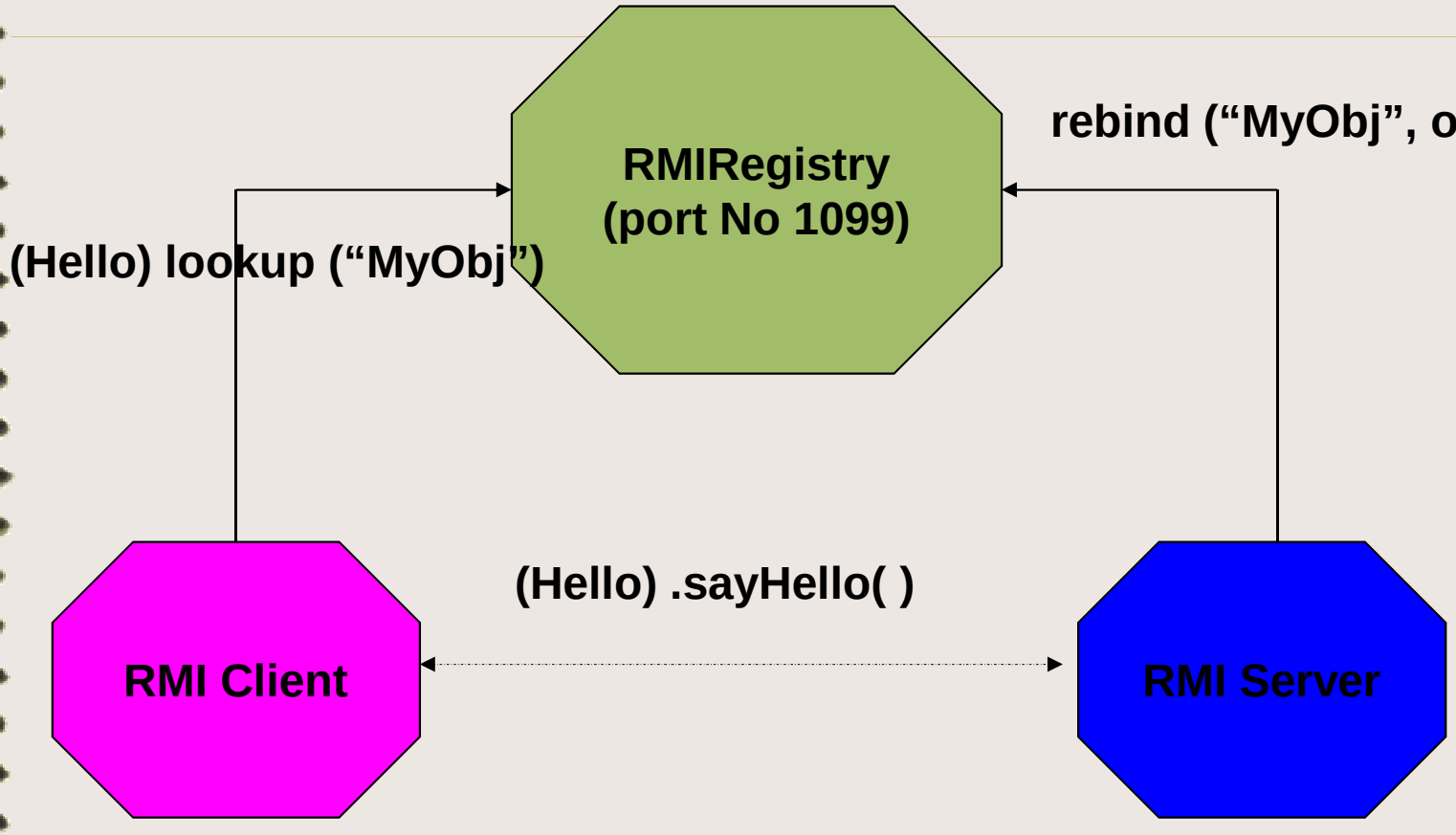
Remote method execution

- 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 handled properly they will be sent to the ***client*** as the result of execution.

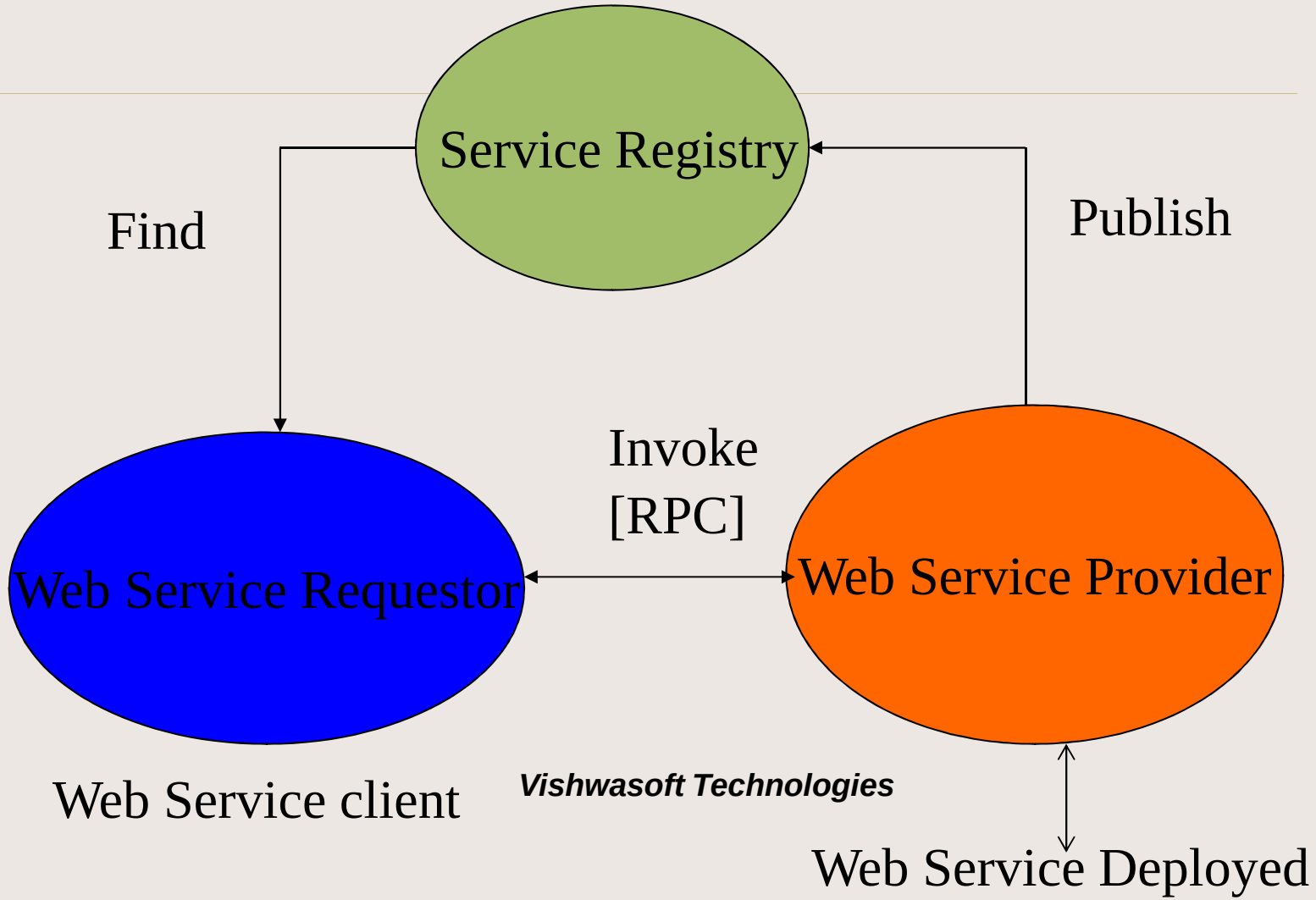
Web Application Process



RPC/RMI Architecture



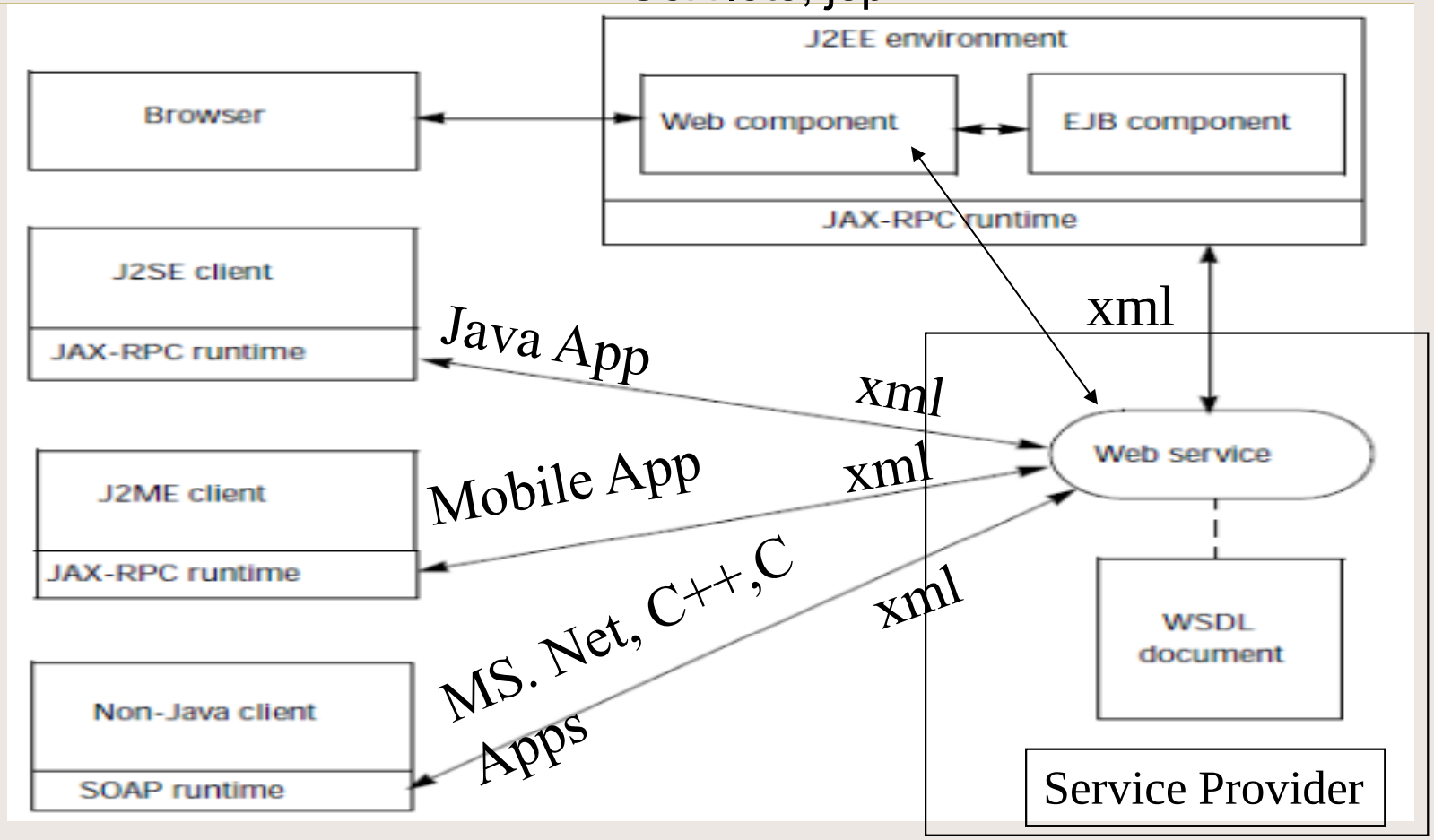
Web Services Conceptually



Web services in SOA World

- Service Oriented Architecture specifies
  - 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….

Clients for Web Service



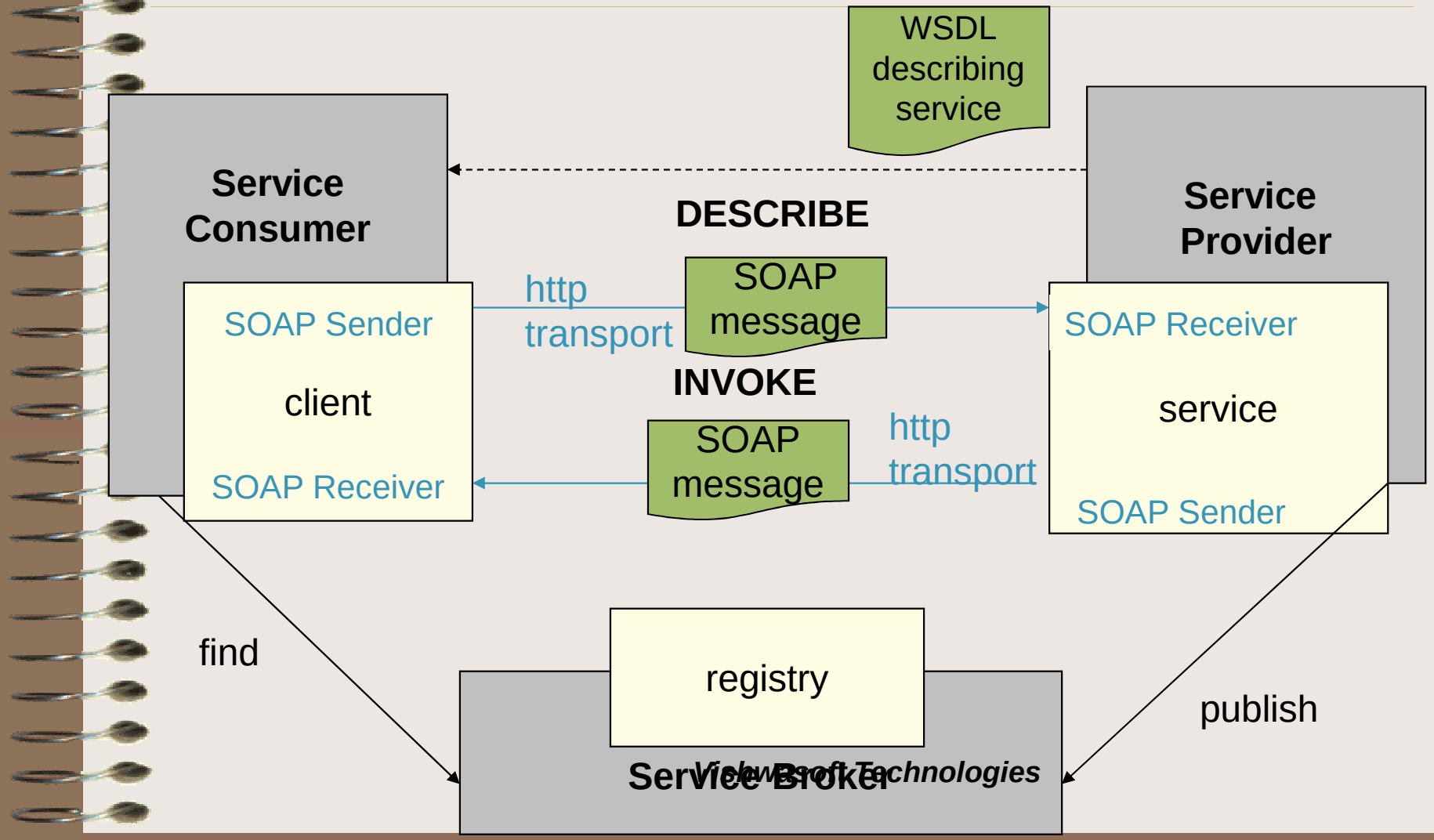
Web service data formats

- How the communication data formats are understood across different programming language platforms ? In a language independent way ? – XML
- How the protocol is implemented in language independent way ? – XML based communication intyerface
- XML is programming language and OS independent as java/C++/.Net.. And Windows/Unix/Solaris..
- A Web service is a software application identified by a URI, whose public interfaces and bindings are defined and described using XML

Web service in action

- The web service definitions are discovered by other software client applications.
- These applications then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed over internet transport protocol/s.

Communication in web services



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 in XML format.

XML Usage

- Web Services fundamentally use XML to standardize the behaviors and data.
- Key XML standards to understand
  - XML : extensible mark up across programming languages.
  - XML Schema specifies xml structure.
  - XML Namespace avoids naming conflicts and categorizes elements as per the functionality

Web Service Standards

- SOAP
- – Simple Object Access Protocol for web service and clients 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 specified in XML format.
- All these standards are specified by *w3c for interoperability and implemented across platforms.*

Service Definition Language

- 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

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 Components

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

Web Service client components

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

Java Standards for Web Services

- JAXP – Java API for XML Processing
- JAXB - Java API 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
- Apache Axis1.4
- SAAJ
- JAX-WS2.2
- Apache Axis2
- IBM, Oracle SOA suites libraries
- gSoap C++ library/framework
- Microsoft .Net Framework
- All these platforms implement and follow the soap protocol guidelines similar to web browsers-web servers following http guidelines..

Java Web Service Build Tools

- ANT
- NetBeans
- Eclipse
- IntelliJ
- MyEclipse

Web service Provider

- On server side you need a service provider to handle the requests ,pass the requests to business objects and generate the xml responses and send it to clients.
- The service provider in java web services is standardized in two categories and has to be implemented by web service frameworks.
  - Servlet End Point on web server.
  - EJB end point on Application server.

Java Web service Frameworks

- These frameworks will implement the end points on server and need configuration to get invoked and generate xml responses.
- They have soap implementations and understand and generate wsdl to describe the service to clients.
  - Apache Axis1.4 and Axis2 frameworks.
  - Sun jax-rpc implementation.
  - IBM,Oracle SOA suite
  - Jax-ws,metro frameworks.
- These frameworks also support client applications to invoke the web service operations.

Developing 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 (xml) and generate corresponding java classes(**top down** approach).

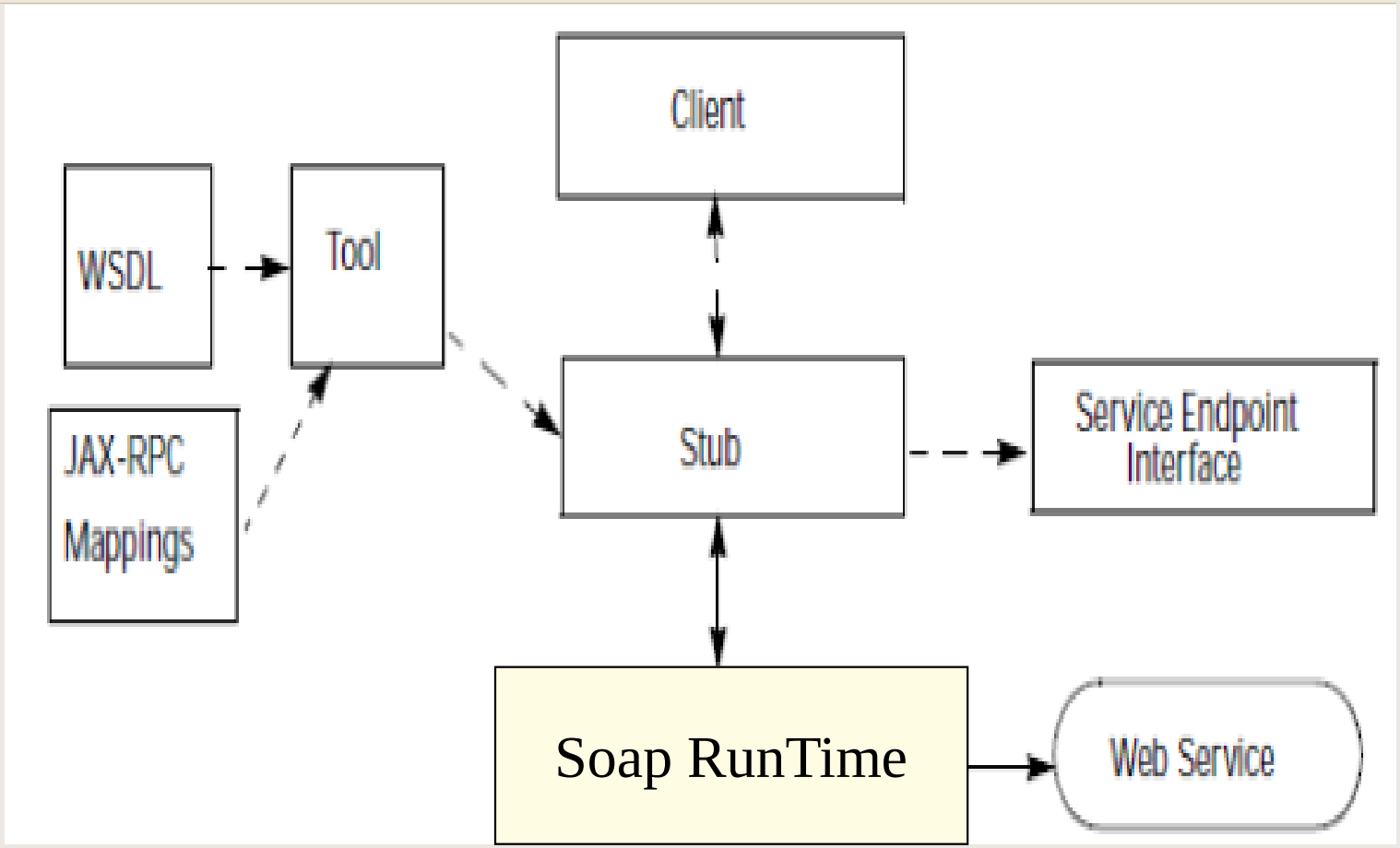
Build Web service with Axis1.4 and Eclipse

- Define a dynamic web application project.
- Add a business class e.g. Account with method as deposit and getInfo.
- Generate the web service endpoint by passing through the bottom up wizard.
- Build and deploy the web application.
- Run the web application and observe the wsdl

Web Service Client

- The client will need proxy to send the request to invoke an operation and get the result.
- The proxy makes the xml based RPC call to web service application on server.
- The Client will need soap library to exchange xml data.
- Define a java project and generate the proxies.
- Add the soap library and run the client.

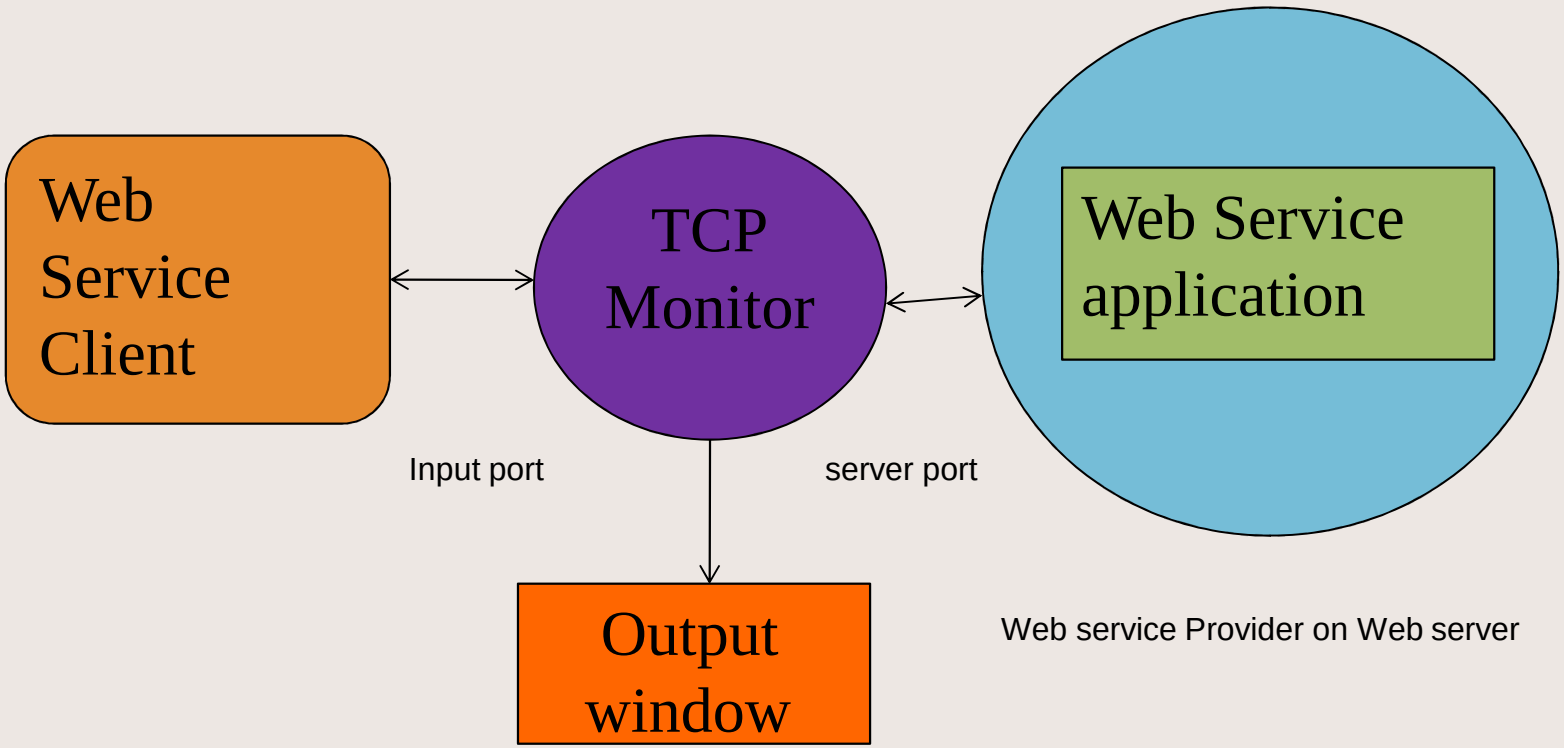
Static Stub Web Service Client



Monitor the communication

- TCP Monitor is java application that intercepts the service request and forwards it to actual application running on server.
- It accepts the response from application on server and forwards it to client.
- The client has to send the service request via the TCP Monitor input port.

TCP Monitor


Web
Service
Client
TCP
Monitor
Web Service
application
Input port
server port
Output
window
Web service Provider on Web server
Request and Response SOAP XML message

XML Processing

- The java environment has built-in xml processing support to parse xml and understand the xml structure and retrieve the data.
- The javax.xml is the basic package.

QName

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

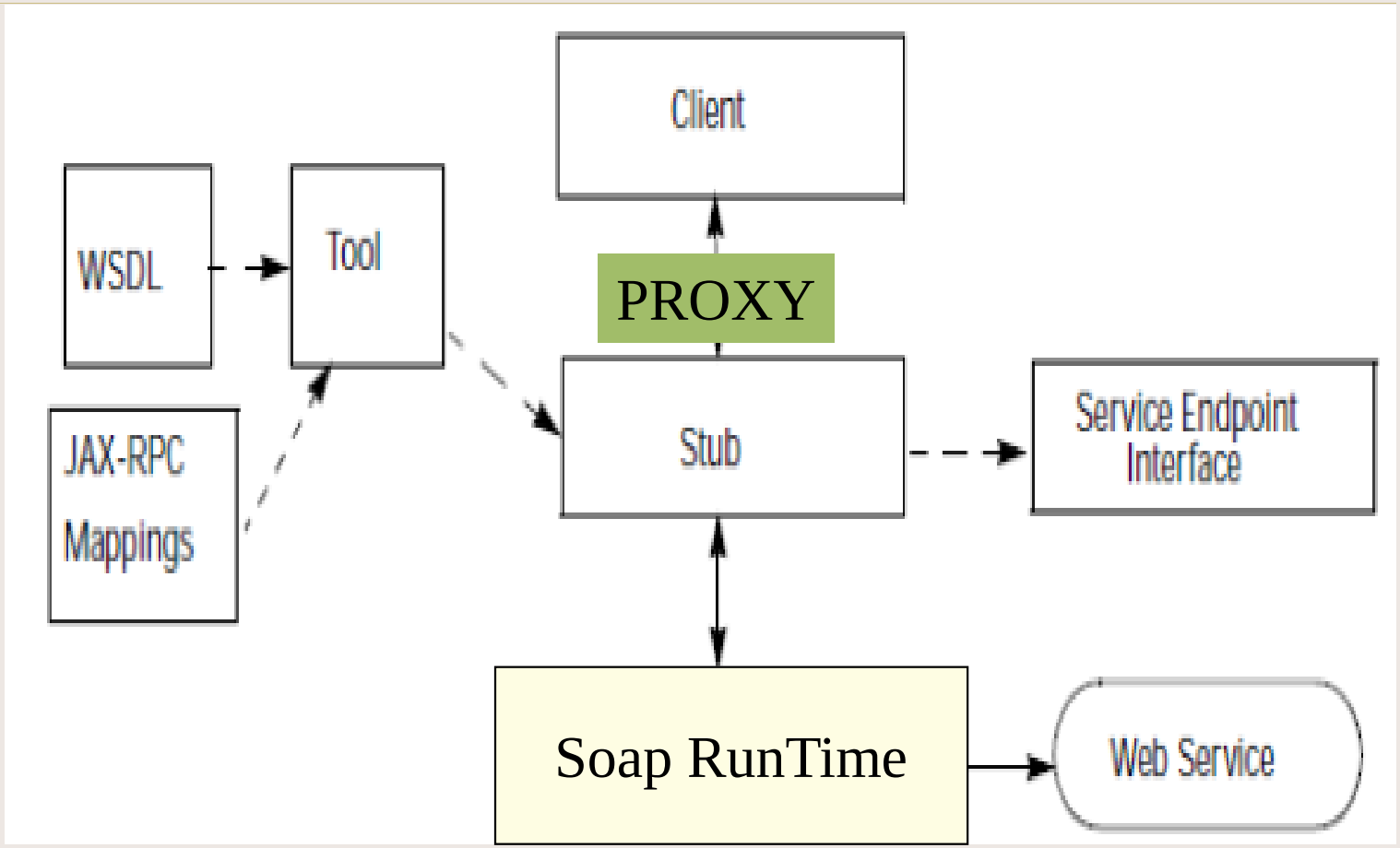
javax.xml.rpc. ServiceFactory

- The ServiceFactory is an abstract class that used for the creation of instances of the type javax.xml.rpc.Service implementation.
- This enables a J2SE based client to create a Service instance in a portable manner without using the constructor of the Service implementation class directly.
- This is extended by axis framework class as org.apache.axis.client.ServiceFactory.

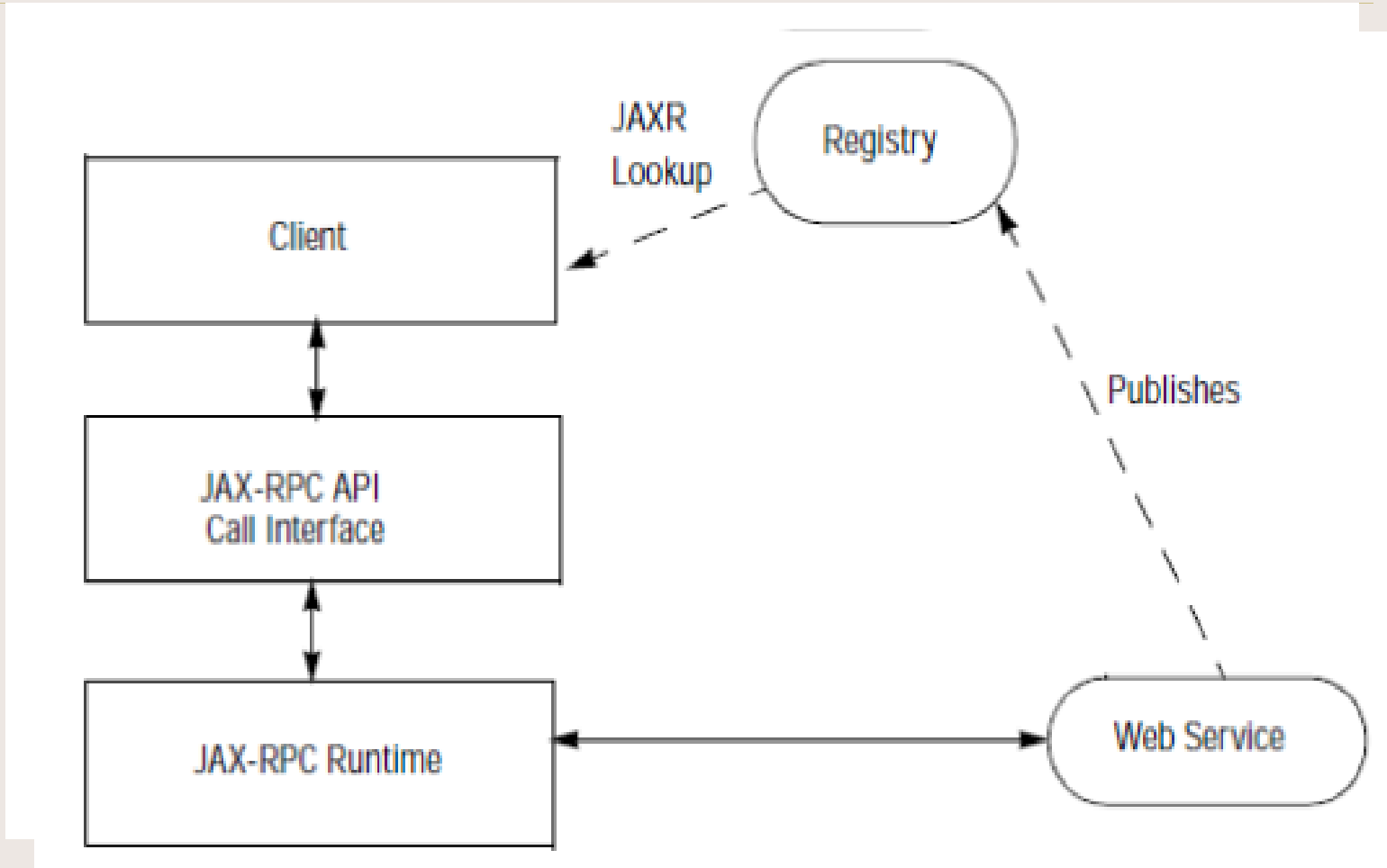
javax.xml.rpc.Service

- The service interface implementation defines dynamic proxy for the web service endpoint.. i.e the web service implementation.
- This endpoint is the port type of service that is used to invoke the service through proxy interface.
- The getPort method returns either an instance of a generated stub implementation class or a dynamic proxy for the port type.
- The client uses this dynamic proxy to invoke operations on the target service endpoint.

Client with Proxy



Dynamic Client



Dynamic Call

- The service also returns an customized instance of the type javax.xml.rpc.Call for the dynamic invocation of a remote operation on the target service endpoint without using the proxy interface.
- The javax.xml.rpc.Call interface supports the dynamic invocation of a service endpoint.
- This Call instance is customized to make the actual call to the web service.

Call customize

- call.setTargetEndpointAddress(new java.net.URL(endpoint));
- call.setOperationName(methodName);
- call.addParameter("param1", XMLType.XSD_STRING, ParameterMode.IN);
- call.setReturnType(XMLType.XSD_INT);
- Object result = call.invoke(new Object [] {"Dynamic Result values...."});//invoke the operation the web service implementation and return the result to client..

Soap Message structure

- A SOAPMessage object consists of a SOAP part and optionally one or more attachment parts.
- The SOAP part for a SOAPMessage object is a SOAPPart object, which contains information used for message routing and identification, and which can contain application-specific content. All data in the SOAP Part of a message must be in XML format.
- A new SOAPMessage object contains the following by default:
  - A SOAPPart object
  - A SOAPEnvelope object
  - A SOAPBody object
  - A SOAPHeader object

Exchange custom data types

- How the user defined objects such as Account, Employee class instances are sterilized as custom parameters as part of service request/response.
- Those custom types need to be translated to xml at transmission end and get converted back to objects type by the receiver end.
- The ComplexType element defined in xml schema allows to define custom types with their properties.
- This types will also be mentioned in in WSDL as custom data types.

Attachment support

- In addition to the SOAPPart object, a SOAPMessage object may contain zero or more AttachmentPart objects, each of which contains application-specific data.
- The SOAPMessage interface supports methods for creating AttachmentPart objects and also for adding them to a SOAPMessage object.
- The soap receiver on client side object can examine its contents by retrieving individual attachment parts.
- The soap message attachment can be anything from simple text to an image file, or xml/non-xml files.

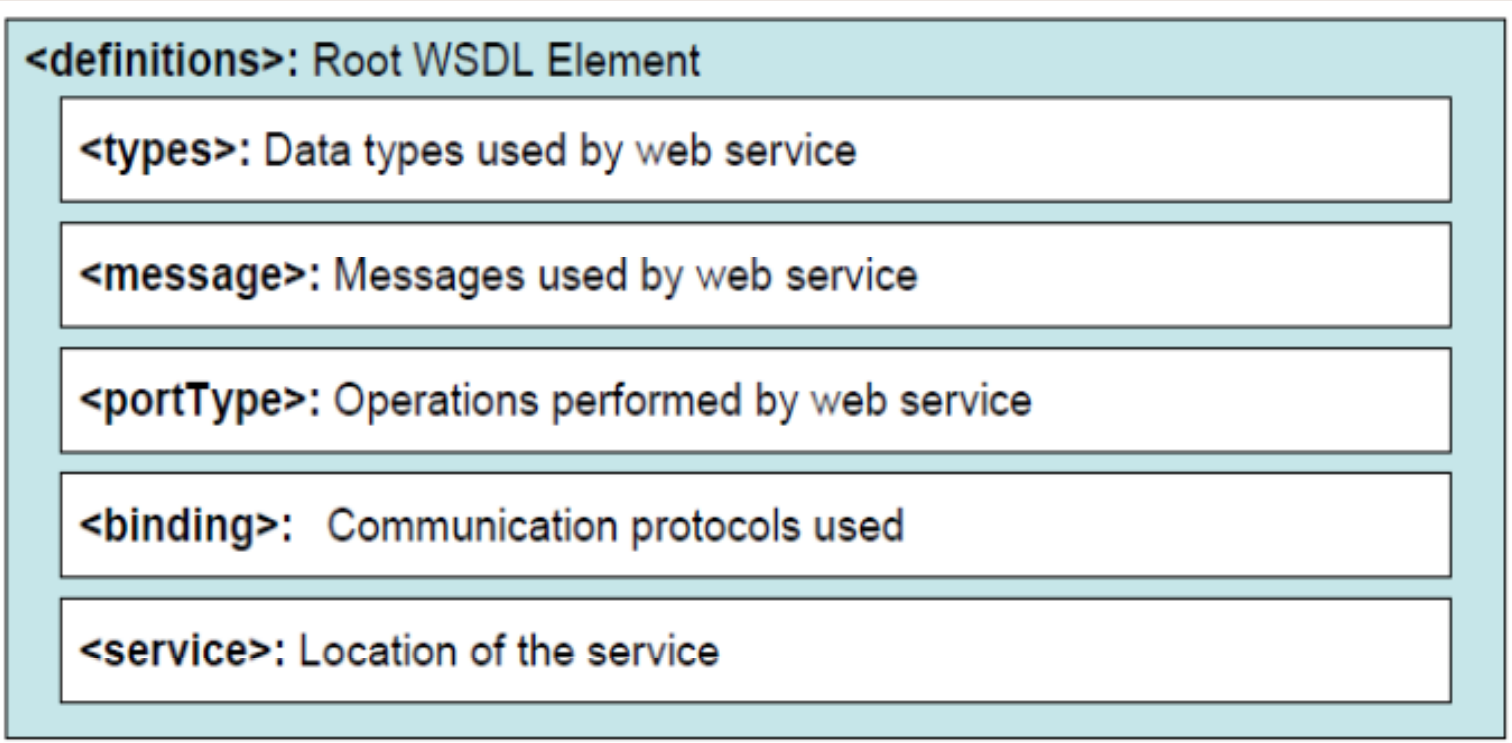
Sending attachments

- The Apache AXIS 1.4 does not have attachment support with SAAJ.
- We will have to add mail.jar and activation.jar to service application and client classpath.
- Tweak the web service application in the serviceConfig.wsdd deployment file.

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



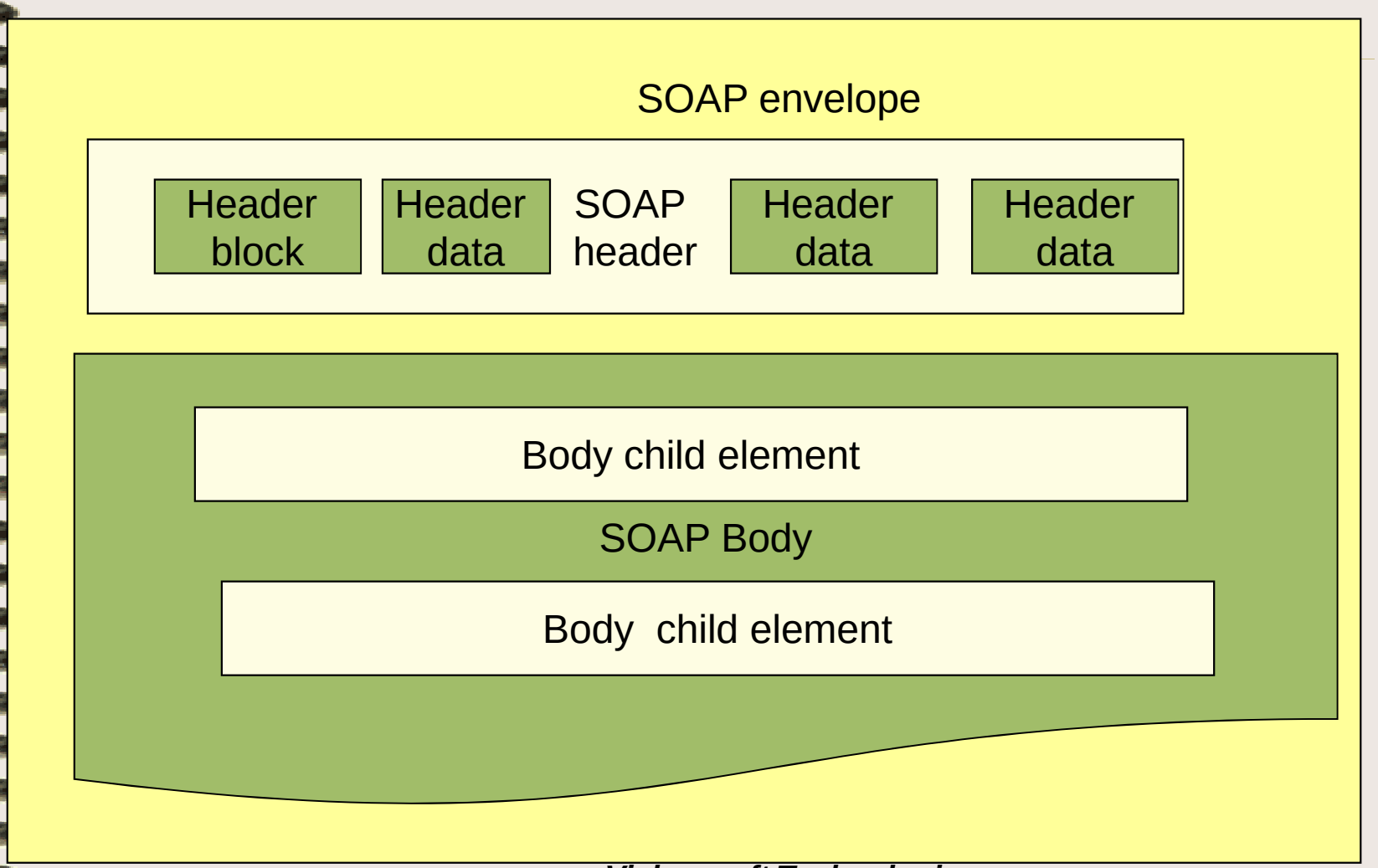
WSDL Bindings

- 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 rules define how the data should be converted from binary form to text and back again.
- SOAP XML Serialization rules define how the XML data should be structured for SOAP messages.
- 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.

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 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)

Introducing JAX-WS

- 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
- 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.

Jax-ws 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’

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.

SOAPBinding.Style

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

WSDL to Java for web service

- The applications may define first the wsdl as per the requirements and then build the web service.
- Use the '**wsimport**' tool to generate the required java classes from wsdl in jax-ws and deploy the web service.

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.

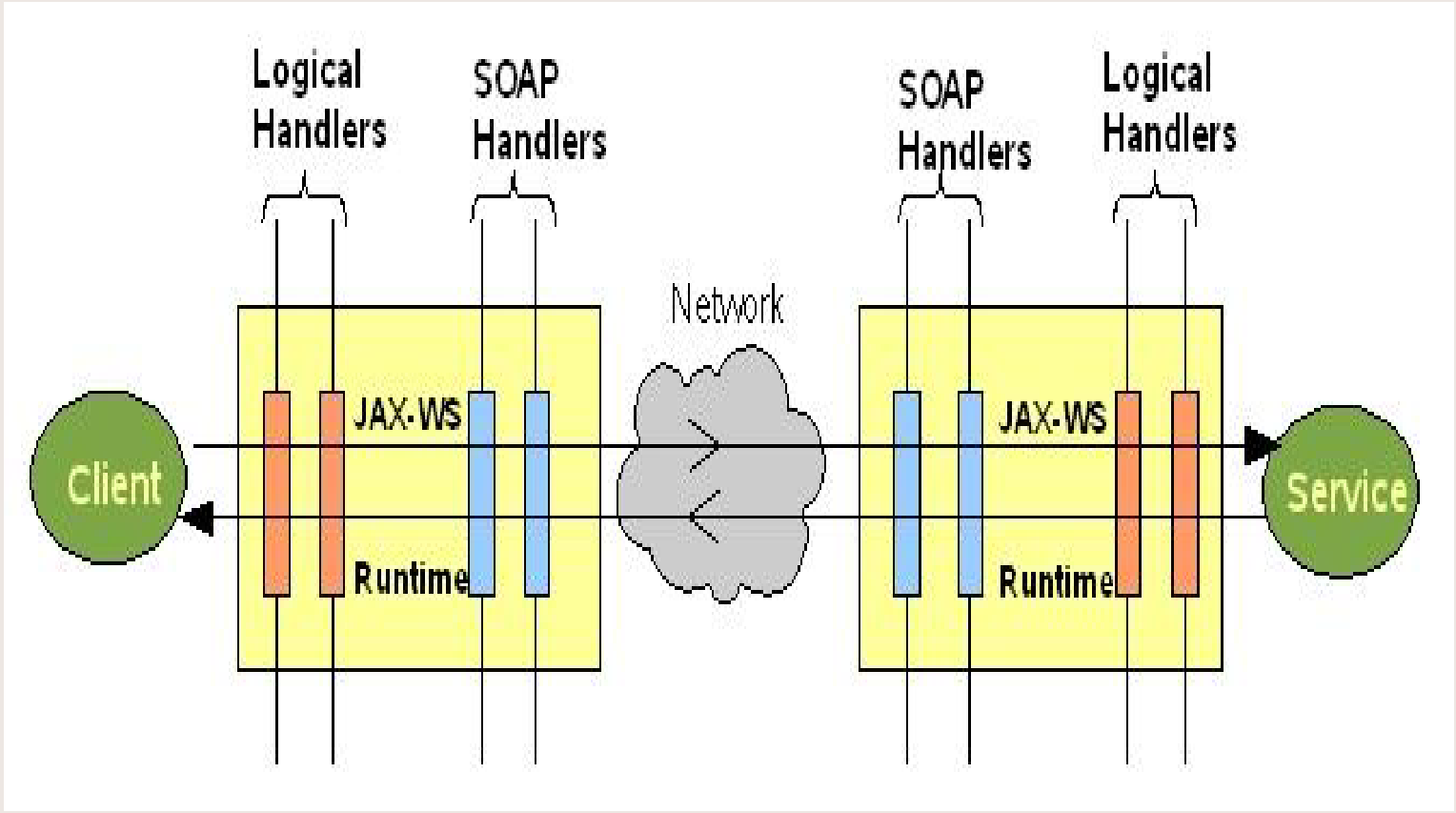
Handlers

- 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.