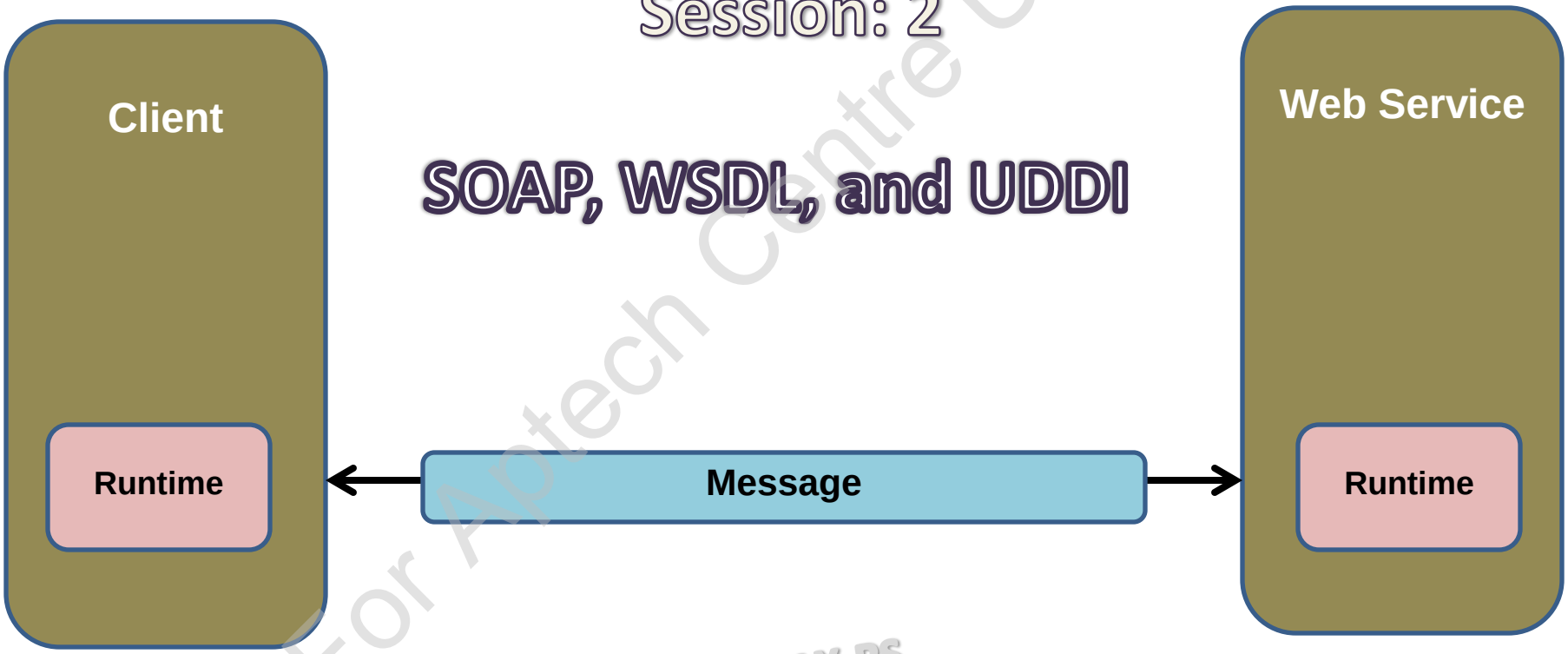
Developing Java Web Services

Session: 2

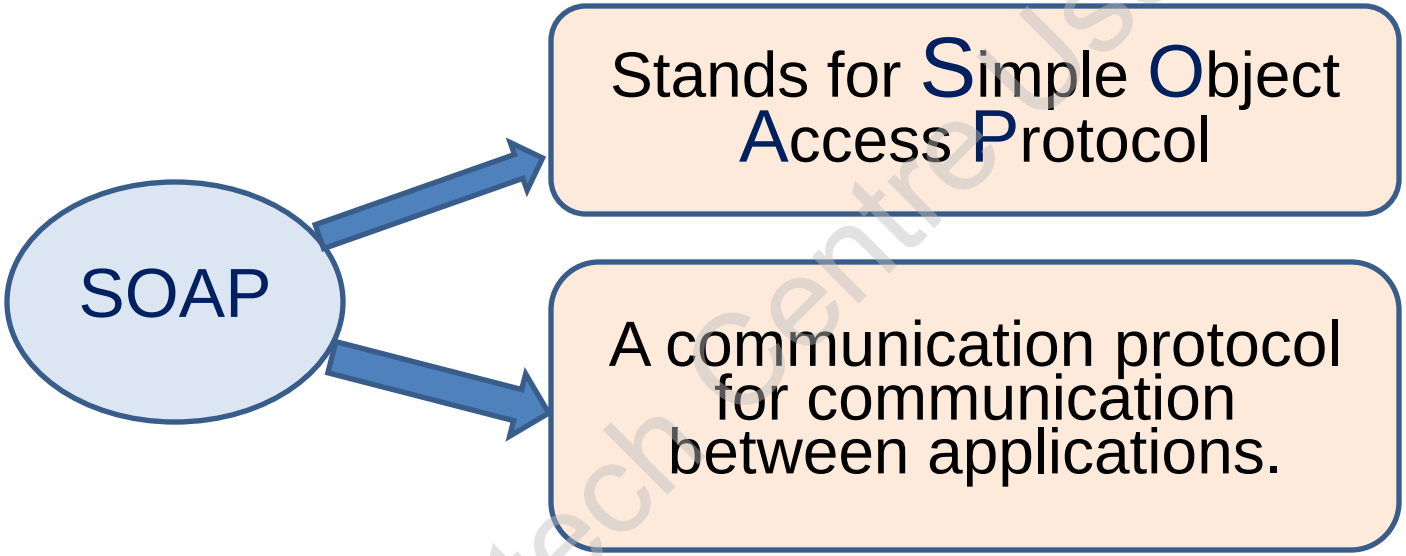
SOAP, WSDL, and UDDI



Objectives

- Explain the purpose and advantages of using SOAP in Web Service
- Describe the different parts of a SOAP message
- Explain SOAP messaging with attachment
- Define Document/Literal SOAP and RPC/Literal SOAP messaging modes
- Explain transportation of SOAP over HTTP protocol
- List sub-elements of SOAP fault element and describe SOAP fault codes
- Explain the purpose of a WSDL file and describe its elements
- Describe UDDI model
- Explain the ebXML Registry standards

Introduction to SOAP



Information Exchange Approaches

EDI

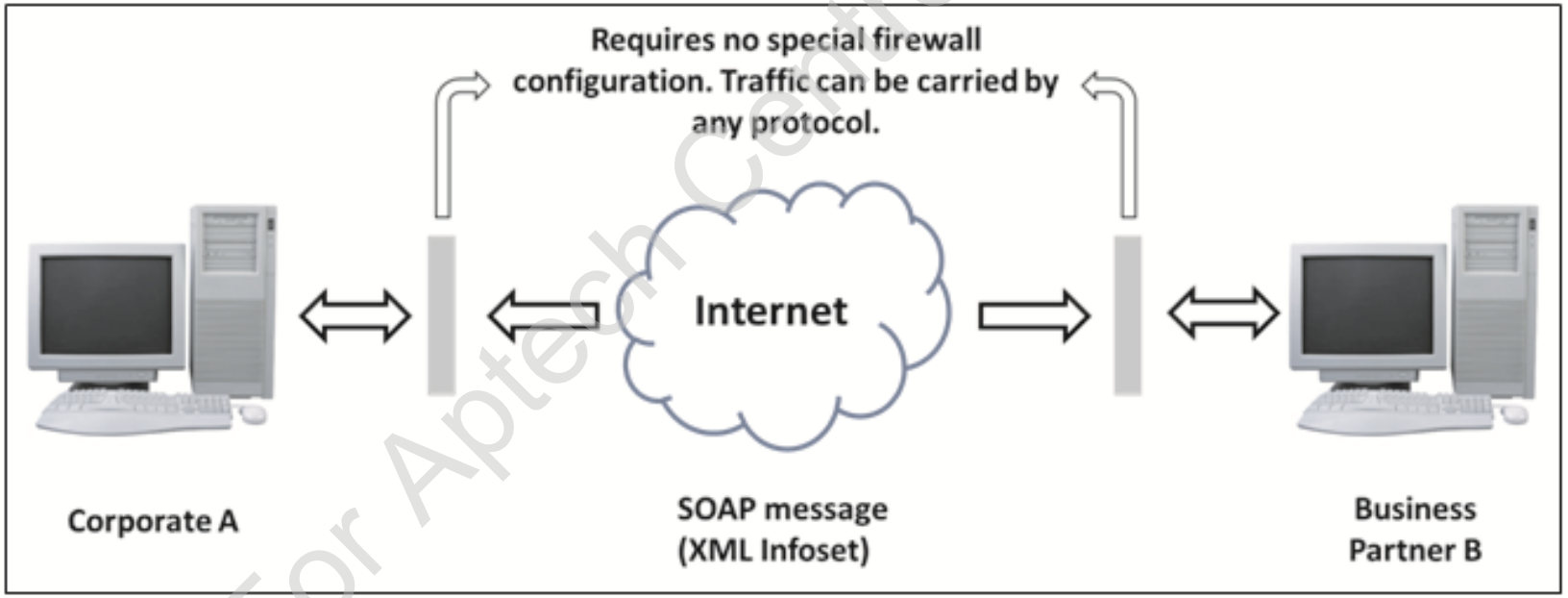
- To send an EDI document, perform the following steps:
  - Install translation software on your system. The translation software converts business documents into X12 format.
  - Set up a private wide area network to send and receive the documents.

RPC and RMI

- The Remote Procedure Call (RPC) approach involves invoking remote methods. These remote methods allowed information exchange in the form of parameters and returned values.

SOAP

- SOAP provides interoperability between applications using XML and HTTP.
- XML is platform and language independent.
- A Java program on a Linux platform can easily interpret a SOAP message created by a C# program on a Windows platform.
- Following figure shows the use of SOAP:



Advantages of SOAP

Vendor Neutral

- Most vendors implement products as per the specifications or standards defined by W3C.

Transport Protocol Independent

- SOAP messages can be transmitted over HTTP, Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and Post Office Protocol (POP).

Platform Independent

- XML is platform independent. SOAP message are constructed using XML Infoset, hence, SOAP is also platform independent.

Language Independent

- SOAP messages are XML Infosets. Several languages, such as Java, Perl, C++ can be used to create these messages.

Interoperability

- SOAP transports XML Infosets using various protocols. This enables distributed applications to communicate without any issues.

Simple

- SOAP does not require any change in network infrastructure or security configurations and hence is simple to use.

SOAP 1.1 vs. SOAP 1.2 1-3

- Following table provides a comparison between the features provided by SOAP 1.1 and SOAP 1.2:

Feature	SOAP 1.1	SOAP 1.2
SOAP messages	Is based on XML 1.0	Is based on XML Infosets
SOAP elements	Allows additional elements to be used after the <body> element	Does not allow additional elements to be used after the <body> element
actor attribute	Indicates the receiver of a header element	Renamed to role and indicates the receiver of a header element
next role	Can be applied to the intermediary SOAP nodes	Can be applied to the intermediary SOAP nodes and the ultimate receiver
none role	Not supported	Used to indicate that the header block should not be processed by any of the SOAP nodes
ultimateReceiver role	Not supported	Used to indicate a header block must be processed only by the ultimate receiver of the SOAP message

SOAP 1.1 vs. SOAP 1.2 2-3

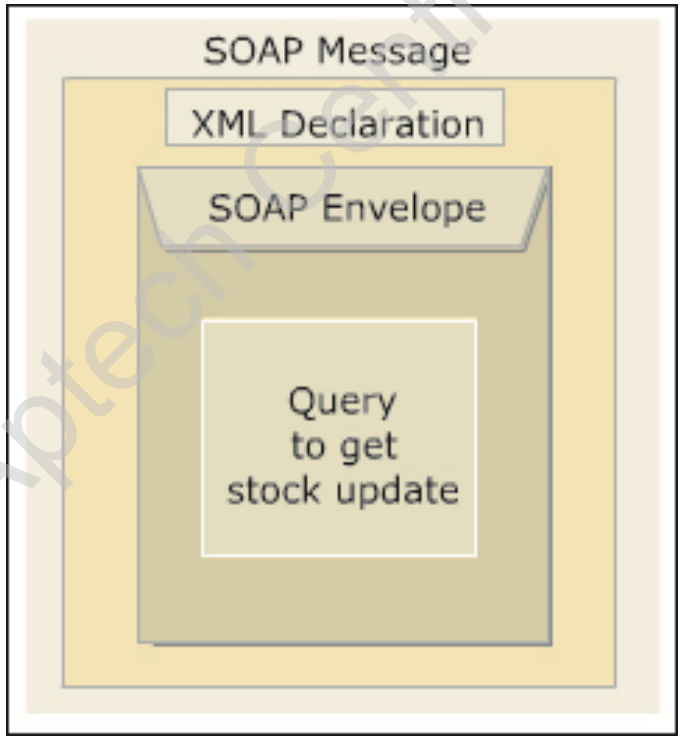
Feature	SOAP 1.1	SOAP 1.2
Fault codes	Supports the VersionMismatch, MustUnderstand, Client, and Server fault codes	Supports the DataEncodingUnknown, VersionMismatch, MustUnderstand, Sender (previously, Client), and Receiver (previously, Server) fault codes
Fault code extensions	Allows a dot notation to be used to indicate fault code extensions	Provides an XML-like structure for the fault code extensions
Fault structure	Only the root element was namespace qualified (e:fault); the other elements are faultcode, faultstring, faultfactor, and detail	All the elements are namespace qualified with e:fault as the root element and e:Code, e:Subcode, e:Value, e:Reason, e:Node, e:Role, and e:detail as the child elements
Fault semantics	Allows the body of the fault message to include multiple child elements	Allows the body of the fault message to include only one child element
Misunderstood header	Not supported	Used for providing additional information in the MustUnderstand faults

SOAP 1.1 vs. SOAP 1.2 3-3

Feature	SOAP 1.1	SOAP 1.2
Upgrade header	Not supported	Used to specify the supported envelope versions when the VersionMismatch fault is reported by a node
Binding Framework	Supports single binding to HTTP	Provides an abstract binding framework to support all protocols
encodingStyle attribute	Allows the attribute to be used on any element in the envelope	Allows the attribute to be used only on the child elements of the Body, Header, and fault Detail elements
Multi-reference values	Encodes multi-reference values in top-level elements	Allows in-place encoding of multi-reference values

SOAP Messages

- A SOAP message contains the following parts:
  - XML declaration
  - Envelope: Comprises Header and Body. The data is enclosed in the Body part of the SOAP message. For example, a request to receive stock update will be part of the Envelope.
- Following figure shows the SOAP message structure:



XML Declaration

- Following Code Snippet demonstrates the XML declaration tag of a SOAP message:

```
<? Xml version= "1.0" encoding = "UTF-8"?>
```

Note –

- It is not mandatory to have an XML declaration in a SOAP message.
- Unicode is an encoding standard that can represent the characters, digits, and symbols of world's major languages.
- UTF-8 is a Unicode character-encoding format created to provide backward compatibility with ASCII.
- UTF-8 and UTF-16 encoding formats can represent any character in Unicode character set.

SOAP Message Envelope 1-6

- The SOAP envelope contains a message from one application to be sent to another application.
- The envelope part of the SOAP message acts as a container for the message.
- The Envelope element:
  - is the root element of the message and is mandatory
  - acts as a processing node to the receiving application. For example, the <Envelope> tag indicates the start of a SOAP message and the </Envelope> tag indicates the end. Once the receiving application encounters the </Envelope> tag, it starts processing the message
  - contains two child elements, an optional <Header> element and a mandatory <Body> element

SOAP 1.1 allows additional elements to be added after the SOAP Body element, SOAP 1.2 does not allow these.

SOAP Message Envelope 2-6

- Following Code Snippet demonstrates the Envelope tag in a SOAP message:

```
<Envelope ...>
  <Header>
    0 or more headers
  </Header>
  <Body>
    message body
  </Body>
</Envelope>
```

SOAP Namespaces

- Following Code Snippet demonstrates a SOAP message with its elements qualified with respective namespaces:

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV= "http://www.w3.org/2003/05/soap-envelope"
xmlns:po= "http://www.flamingo.com/books/PO">
   ...
<!--SOAP Message goes here -->
      <po:orderDate> 2014/07/06 </po:orderDate>
   ...
</SOAP-ENV:Envelope>
```

SOAP Message Envelope 3-6

SOAP Message Header

- Header is the child element of Envelope element.
- Headers can also include information such as digital signatures for password-protected service, authentication, authorization, transaction management, and routing path.

role Attribute

- Following Code Snippet demonstrates a SOAP message with various header entries:

```
<SOAP-ENV: Envelope
xmlns:SOAP-ENV= "http://www.w3.org/2003/05/soap-envelope"
xmlns:mid= "http://www.flamingo.com/books/message-id"
xmlns:m= "http://www.flamingo.com/books/monitored-by"
xmlns:md = "http://www.flamingo.com/books/monitoring-date"
xmlns:mu = "http://www.flamingo.com/books/mark"
        <SOAP-ENV: Header>
        <mid:message-id
SOAP-ENV:role="http://www.flamingo.com/logger">
    11546544ea:b134534:f3sdas5342:4354
   </mid:message-id>
   <m:monitored-by SOAP-ENV:role = "
```

SOAP Message Envelope 4-6

```
http://www.w3.org/2003/05/soap-envelope/role/next">
      <node>
         <time> 1078753670000   </time>
         <identity>austria</identity>
      </node>
  </m:monitored-by>
<md:monitoring-date SOAP-ENV:role = " http://www.w3.org/2003/05/soap-
envelope/role/none">
...
  </md:monitoring-date>
  <mu:mark SOAP-ENV:role = "http://www.w3.org/2003/05/soap-
envelope/role/ultimateReceiver">
    ...
   </mu:mark>
  </SOAP-ENV: Header>
</SOAP-ENV: Envelope>
```

SOAP Message Envelope 5-6

mustUnderstand Attribute

- Following Code Snippet demonstrates an example of a header entry with the mustUnderstand attribute.

```
<!-- SOAP Message Structure  -->
<?xml version ="1.0" encoding="UTF-8" ?><SOAP-ENV:Envelope
    xmlns:SOAP-ENV= "http://www.w3.org/2003/05/soap-envelope">
<SOAP-ENV:Header xlmns:au="http://www.flamingo.com/books.authentication">
<au:Requestor mustUnderstand="true"
                  <name> John Smith </name>
          </au:Requestor>
               ...
        </SOAP-ENV:Header>
                ...
</SOAP-ENV:Envelope>
```

SOAP Message Envelope 6-6

Body Element

- The Body element is the mandatory element of Envelope element. The immediate child elements of Body element must be namespace-qualified. The Body element must be placed as follows:
  - If the Header element is not present, the Body element should be the immediate child of Envelope element.
  - If the Header element is present, the Body element should immediately follow the Header element.

Need of Attachment

- SOAP messages contain XML fragment. However, at times you may want to send data that is not XML.
- Messages that require binary data are converted to a Multipurpose Internet Mail Extensions (MIME) message format and then sent.
- A MIME message can contain multiple parts and supports binary data as well.

MIME Message Structure 1-3

MIME header

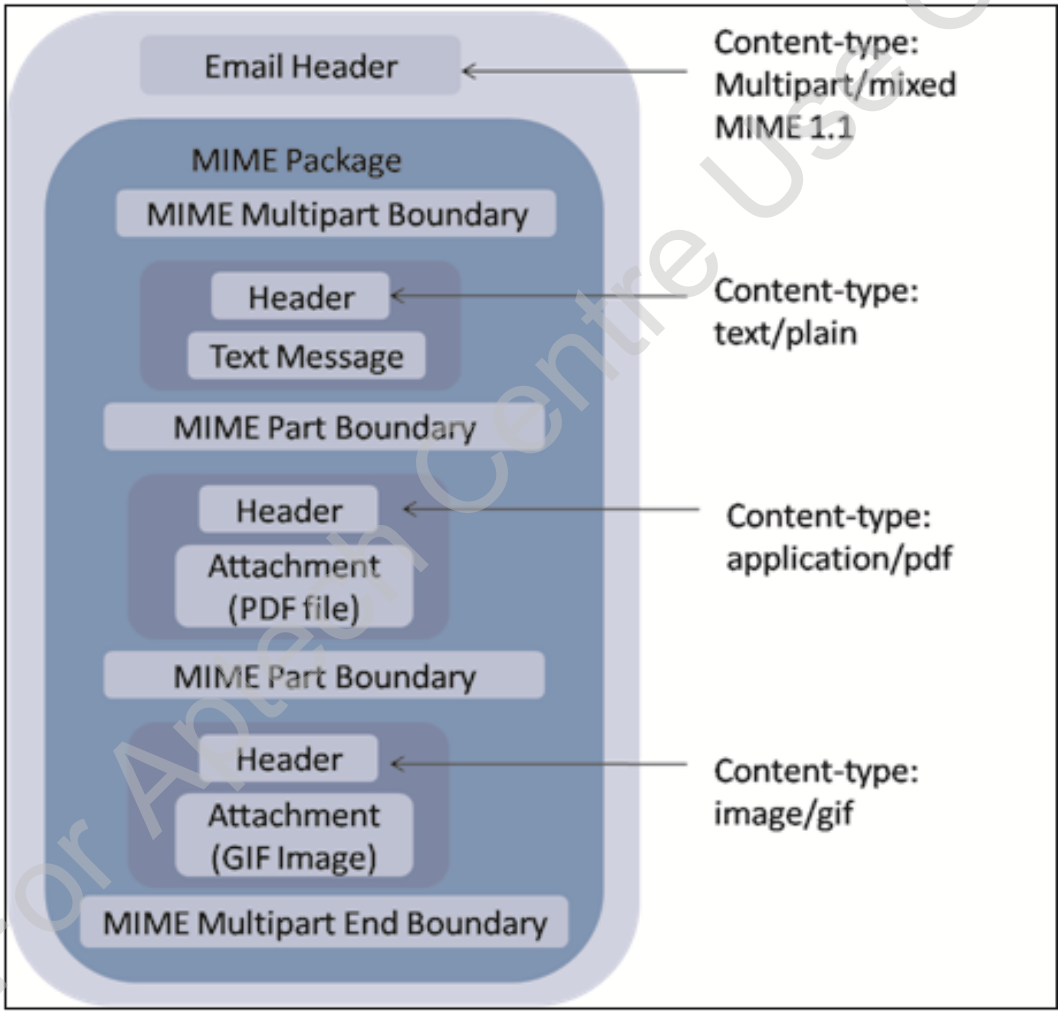
- Identifies the version of MIME and content type. A content type as Multipart/Mixed indicates that the email contains multiple parts of varying types

MIME package

- Can contain one or more MIME parts

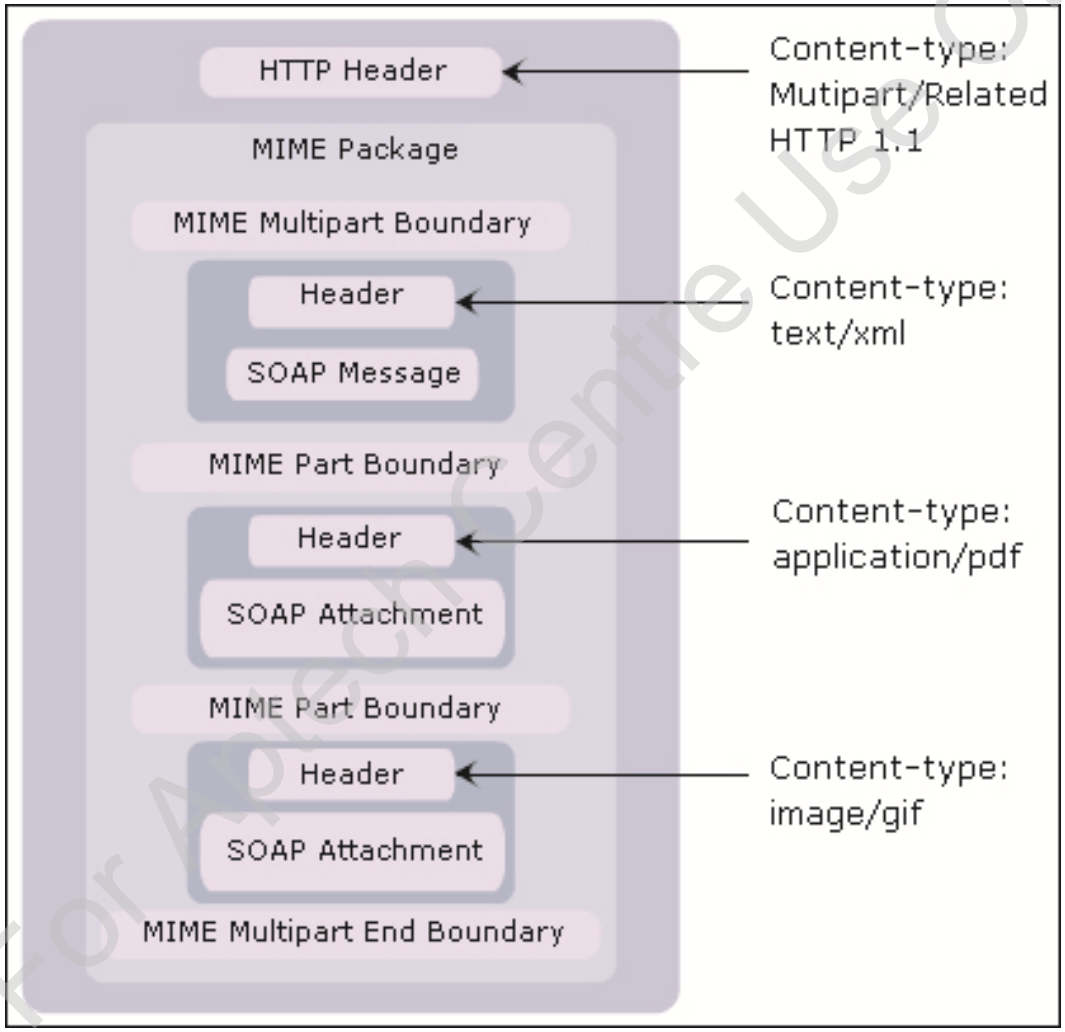
MIME Message Structure 2-3

- Following figure shows different types of message attachments:



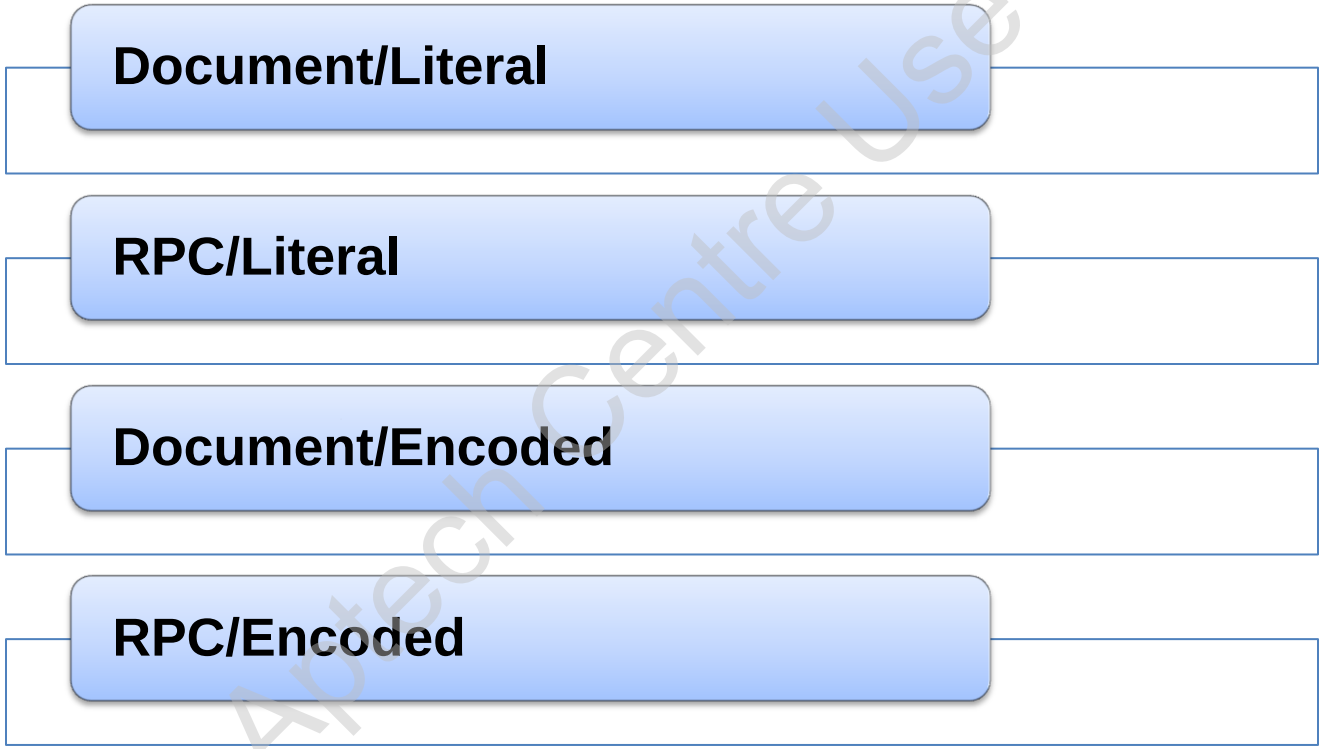
MIME Message Structure 3-3

- Following figure shows a SOAP message with two attachments:



SOAP Message Modes

- Based on the messaging styles and encoding types, SOAP defines following four messaging modes:



Document / Literal Messaging Mode

- Following Code Snippet demonstrates a SOAP message with a Body element with the details about the Spiderman Digital Versatile/Video Disc (DVD):

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap/envelope">
     ...
    <SOAP-ENV:Body>
      <d:dvd xmlns:d="http://www.flamingo.com/dvds/">
         <d:title>Spider-Man</d:title>
         <d:language>English</d:language>
         <d:discs>2</d:discs>
         <d:runtime>
              <d:minutes>121 </d:minutes>
          </d:runtime>
         <d:price>9.49</d:price>
      </d:dvd>
   </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

RPC / Literal Messaging Mode

- The RPC/Literal messaging mode is based on sending request messages and receiving response messages.
- Following Code Snippet demonstrates SOAP request message:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=" http://www.w3.org/2003/05/soap/envelope"
    ...
 <SOAP-ENV:Body>
  <m:GetOrderStatus xmlns:m="http://www.flamingo.com/methods">
   <m:OrderNo>34347</m:OrderNo>
    </m:GetOrderStatus>
    </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

- Following Code Snippet demonstrates SOAP response message:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=" http://www.w3.org/2003/05/soap/envelope"

<SOAP-ENV:Body>
  <m:GetOrderStatusResponse xmlns:m="http://www.flamingo.com/methods">
   <m:OrderStatus>Shipped on 2014-08-09 </m:OrderStatus>
    </m:GetOrderStatusResponse>
    </SOAP-ENV:Body>
 </SOAP-ENV:Envelope
```

Transport Protocols

SOAP

- Is an XML-based protocol.
- Follows the HTTP request and response model to transmit client request and obtain Web Service response.

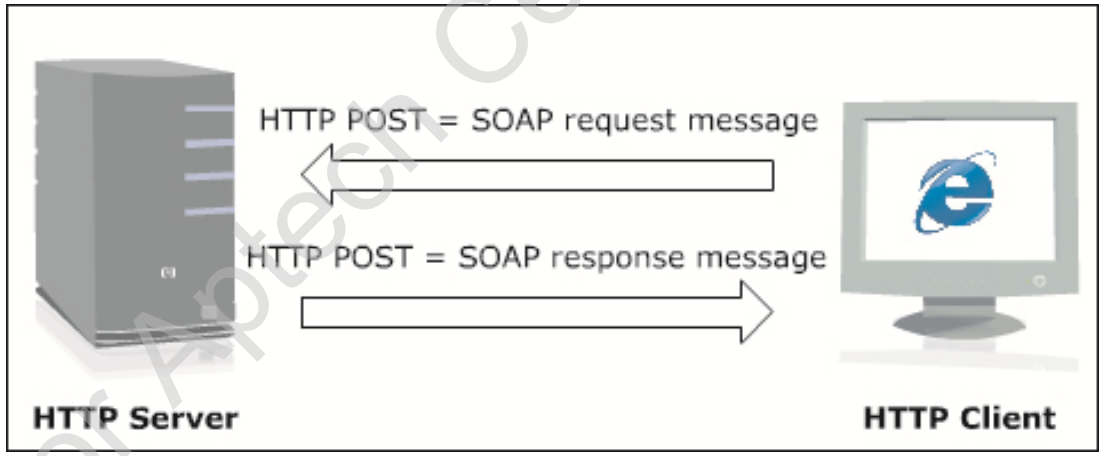
SOAP messages over HTTP help in exchange of messages between clients and Web services, all running on different platforms, and at various locations on the Internet.

SOAP HTTP Binding 1-2

- SOAP messages are transmitted as a payload of an HTTP message.
- A payload of an HTTP message contains form data such as username, password, credit card number, and so on.

HTTP

- Is a request-response protocol. A SOAP message is carried as a payload in an HTTP POST message.
- Following figure shows the HTTP POST message:



SOAP HTTP Binding 2-2

- Features of HTTP 1.1 are as follows:
  - Requires a Host header. This header allows a message to be routed through proxy servers and helps the Web server to distinguish between various sites on the same server.
  - Supports persistent connections, which allows multiple request/response on the same HTTP connections.
  - Provides the OPTIONS method that helps determine the capabilities of the HTTP server.
  - Provides the entity tag, which expands on the caching support.
  - Provides additional conditional headers such as If-Unmodified-Since, If-Match, and If-None-Match.

SOAP Request over HTTP

- A SOAP message sent using HTTP POST message comprises the HTTP header and the SOAP message.
- Following Code Snippet demonstrates SOAP request over HTTP:

```
POST/orderstatus HTTP/1.1
Host:www.flamingo.com:80
Content-Type: text/xml; charset=utf-8
COntent-Length:482
SOAPAction:"http://www.flamingo.com/books/getOrderStatus"
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=" http://www.w3.org/2003/05/soap/envelope">
    ...
    <SOAP-ENV:Body>
         <m:GetOrderStatus
           xmlns:m="http://www.flamingo,com/methods">
            <m:OrderNo>34347</m:OrderNo>
    </m:GetOrderStatus>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

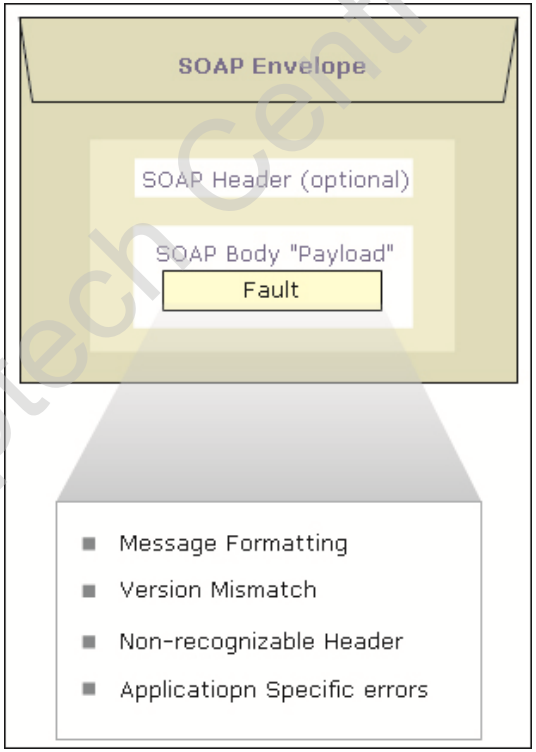
SOAP Response over HTTP

- A SOAP message sent using HTTP also contains the HTTP header and the SOAP message.
- Following Code Snippet demonstrates a SOAP response message embedded within an HTTP POST message:

```
HTTP/1.1 200 OK
Connection:close
Content-Length: 659
Content-Type:text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV: Envelope
    xmlns:SOAP-ENV=" http://www.w3.org/2003/05/soap/envelope">
  ...
  <SOAP-ENV:Body>
   <m:GetOrderStatusRespose
        xmlns:m="http://www.flamingo.com/methods">
           <m:OrderStatus>
                Shipped on 2014-08-09
           </m:OrderStatus>
    </m:GetOrderStatusResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fault Element

- A SOAP fault is generated when an error occurs in transmission of message.
- In SOAP, errors are generated by the nodes in the message path while processing a message.
- The causes for errors in SOAP are improper message formatting, version mismatch, trouble processing a header and application specific errors.
- Following figure shows SOAP message with Fault element:



Fault Subelements

- Following table describes the Fault subelements:

Fault Subelement	Description
env:Code	Identifies an error in a SOAP message. This subemelement contains two child elements e:Value and e:Subcode
env:Value	Indicates a value that identifies the type of error that occurred.
env:Subcode	Indicates additional error information.

Fault Codes

- Following table describes the fault code:

Fault Code	Description
VersionMismatch	Indicates that receiving application received a different version of SOAP message.
MustUnderstand	Indicates that receiving node or application was unable to process the header entry targeted for it. **Note:** If the mustUnderstand attribute in the message received was set to 1, the receiving node sends a fault message.
Sender	Indicates an error in the message or its data. That is, it indicates an error on the message sender's end.
Receiver	Indicates that the intermediary node or the ultimate receiver was unable to process the message.
DataEncodingUnknown	Indicates that the received messages are using an unrecognized value of the encodingStyle attribute.

SOAP Fault over HTTP

- An error in a SOAP message is communicated to the sender using a SOAP fault message.
- Following Code Snippet demonstrates an HTTP response message indicating mismatch in the version of SOAP message received:

```
HTTP/1.1 500 Internal Server Error
Connection: close
Content-Length:659
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
   <SOAP-ENV:Fault>
       <Code>SOAP-ENV:VersionMismatch</Code>
       <Reason>Not a SOAP 1.2 message. </Reason>
          <Detail>Version MisMatch </Detail>
    </SOAP-ENV:Fault>
 </SOAP-ENV: Body>
</SOAP-ENV:Envelope>
```

Basics of WSDL

WSDL is a specification defined to describe information about a Web Service in XML format.

WSDL provides the common methods to represent the data types passed in the service, the functions that are available in the service, and the mapping of service onto the network protocol.

WSDL 2.0 divides the description of the abstract functionality offered by a service from the specific description of the service description such as ‘how’ and ‘where’ that functionality is offered.

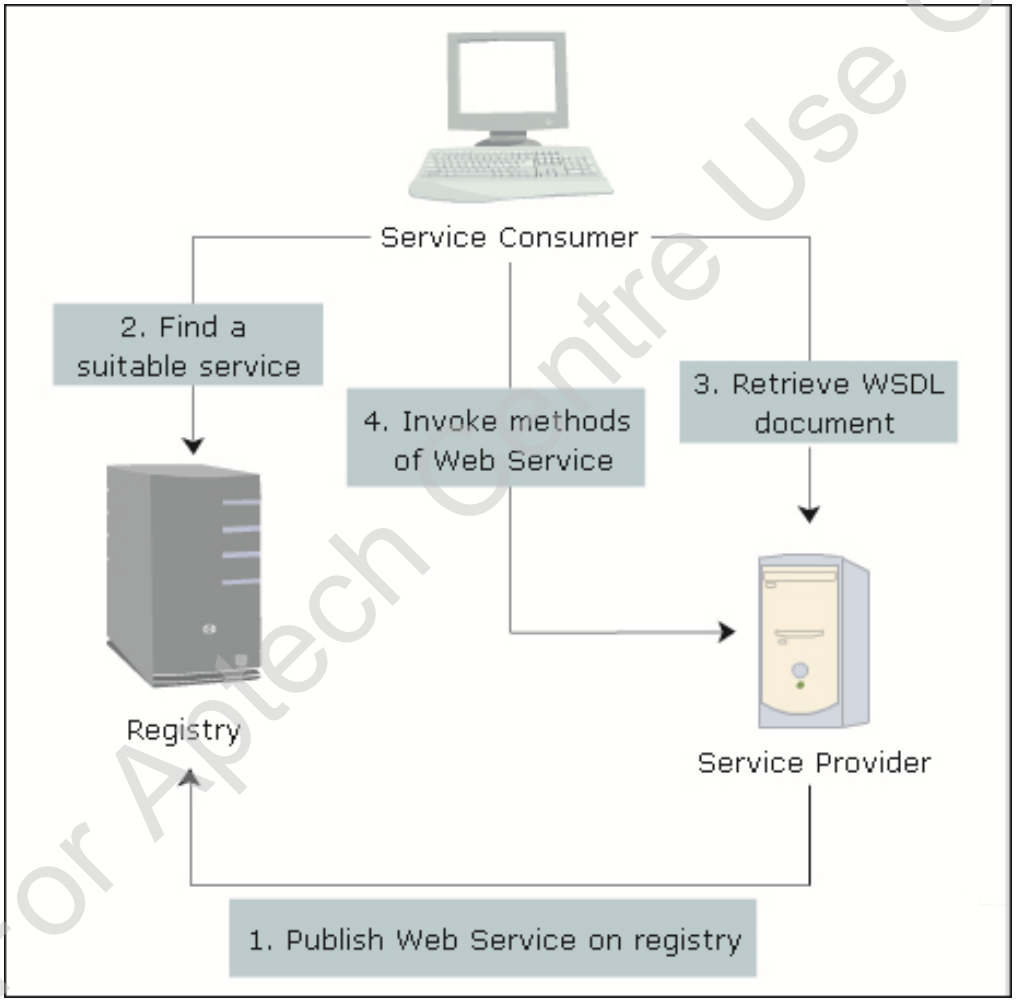
WSDL for Describing Web Services

- A typical WSDL document contains the following information about the Web Service:

  - Available methods
  - Type of protocol to be used
  - Parameters and return type of methods
  - Location of Web Service

WSDL for Service Providers and Consumers

- Following figure shows the WSDL for service providers and consumers:



WSDL Document Structure 1-4

- Following table describes the WSDL document elements:

Elements	Description
description	• Acts as a container for the elements types, interface, binding, and service. • Defines the name of the Web Service and also one or more namespaces used by its child elements.
types	• Defines the data type of the information exchanged between applications. • Is mandatory only if the data type is other than the built-in data types of XML Schema. Example of XML schema's built-in types are string, integer, and so on.
interface	• Describes the operations that the Web service includes and the input/output messages that are exchanged for each operation. • Describes the fault messages.
binding	• Describes how the input and output messages of each operation will be transmitted over the Internet from one application to another.
service	• Defines the address for invoking the Web Service. In other words, it provides the URL of the service provider's server on which the Web Service is hosted. The service consumer's application uses this URL to invoke the methods of the Web Service.
import	• Used to import XML Schemas or WSDL document. This is an optional element.

WSDL Document Structure 2-4

- Following Code Snippet shows a sample WSDL document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<description targetNamespace="http://ws.soap.syskan.com/"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:tns="http://ws.soap.syskan.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/ns/wsdl"
xmlns:wsoap="http://www.w3.org/ns/wsdl/soap">
<types>
<xsd:schema>
<xsd:import
namespace="http://ws.soap.syskan.com/"
 schemaLocation="http://localhost:8080/SOAPWebService/SOAPWS?xsd=1"/>
```

WSDL Document Structure 3-4

```
</xsd:schema>
</types>
<interface name="SOAPWS">
<operation name="add"
pattern="http://www.w3.org/ns/wsdl/in-out">
<input element="tns:add"
wsam:Action="http://ws.soap.syskan.com/SOAPWS/addRequest"/>
<output element="tns:addResponse"
wsam:Action="http://ws.soap.syskan.com/SOAPWS/addResponse"/>
</operation>
</interface>

<binding name="SOAPWSPortBinding"
interface="tns:SOAPWS"
type="http://www.w3.org/ns/wsdl/soap"
wsoap:version="1.1"
wsoap:protocol="http://www.w3.org/2006/01/
soap11/bindings/HTTP/">
<operation ref="tns:add">
<input/>
```

WSDL Document Structure 4-4

```
<output/>
                </operation>
</binding>
        <service name="SOAPWS" interface="tns:SOAPWS">
                <endpoint name="SOAPWSPort"
binding="tns:SOAPWSPortBinding"
address="http://localhost:8080/
SOAPWebService/SOAPWS"/>
        </service>
</description>
```

What is UDDI 1-3

Web Service Registry

- A dictionary consists of searchable collection of words in one or more specific languages.
- Similarly, a service registry consists of searchable collection of descriptions of Web services.

Universal Description, Discovery, and Integration (UDDI)

- Is a platform-independent, XML based registry for businesses worldwide to list the business on the Internet.
- Provides standard mechanisms for businesses to describe and publish their Web Services, discover published Web Services, and use them.

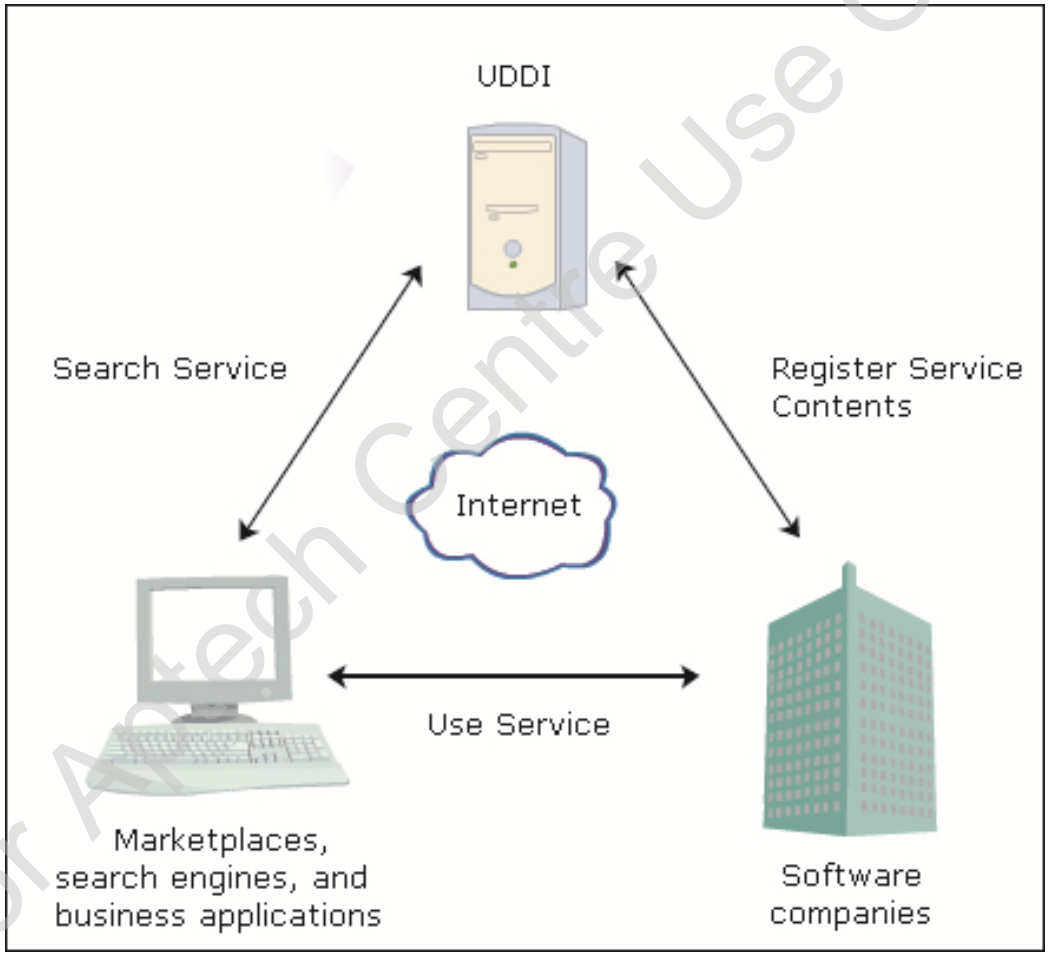
What is UDDI 2-3

To use the registry following steps are required:

- UDDI contains references to specifications called as Technical Models, or tModels.
- The tModels describe the working of Web Services. They are built upon a programming model and schema that are platform and language independent.
- The software companies populate the registry by describing various tModels and specifications common to a business.
- UDDI programmatically assigns a Unique Universal Identifier (UUID) to each tModel and business registration.
- Market places, search engines, and business applications query the registry to discover services of other companies and integrate this data with each other over the Web.

What is UDDI 3-3

- Following figure shows the UDDI registry:



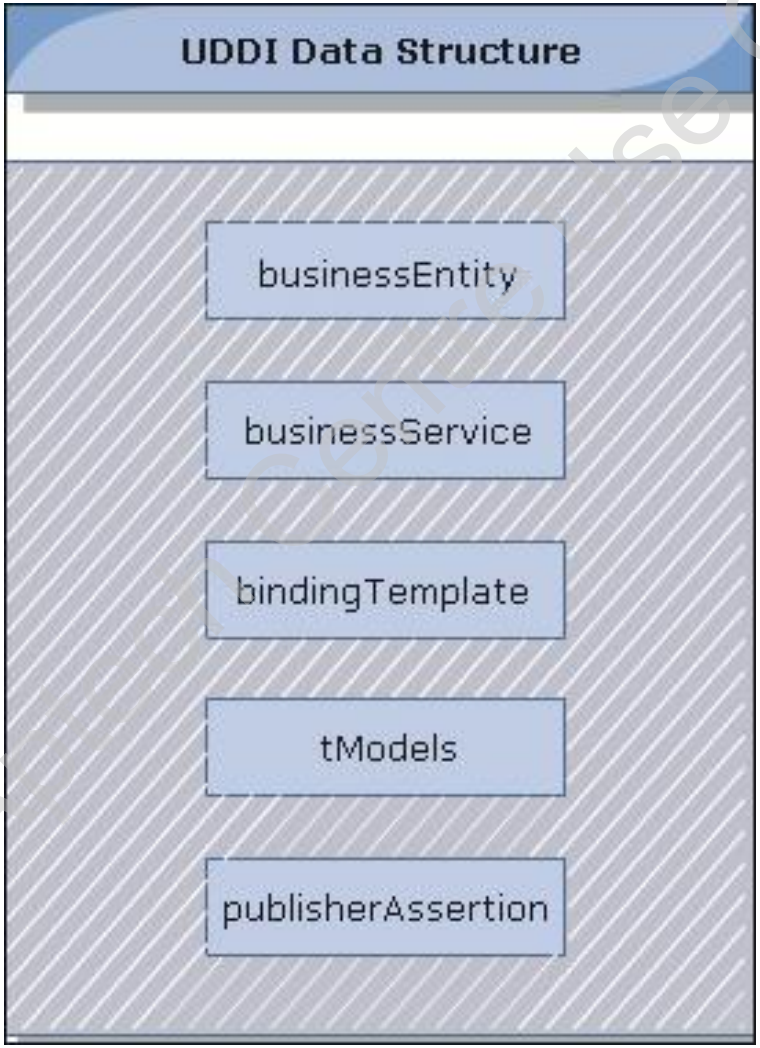
UDDI Data Structure 1-3

- UDDI provides five core data structures as follows:

businessEntity	• Represents all the information about the business or the organization that provides the Web Service.
businessService	• Represents a Web Service.
bindingTemplate	• Represents the binding of a Web Service with its URL and its tModels.
tModel	• Provides information about a Web Service. It specifies the name of the service, brief description of the service, and a unique code that is used to identify a service in the registry.
publisherAssertion	• Represents a relationship between two business entities or service providers.

UDDI Data Structure 2-3

- Following figure show the UDDI data structure:



UDDI Data Structure 3-3

- Features of UDDI version 3 are as follows:
  - It serves as a meta service to find Web services by enabling robust queries against rich metadata.
  - It provides specification for building flexible, interoperable XML Web services registries useful in both private and public deployments.
  - It consists of multi-registry topologies.
  - It provides increased security features and improved WSDL support.
  - It provides a new subscription API and core information model advances.
  - It offers clients and developers a broad and complete blueprint of a description and discovery foundation for a diverse set of Web services architectures.

UDDI Publisher API 1-2

Following table describes some of the methods available under UDDI Publisher API:

Methods	Description
Add_publisherAssertions	Adds relationship assertions to the existing set of assertions
save_business	Adds or updates one or more businessEntity entries
save_service	Adds or updates one or more businessService entries
save_binding	Adds or updates one or more bindingTemplate entries
save_tModel	Adds or updates one or more tModel entries
delete_business	Deletes one or more businessEntity entries
delete_service	Deletes one or more businessService entries
delete_binding	Deletes one or more bindingTemplate entries
delete_tModel	Deletes (or hides) one or more tModel entries
delete_publisherAssertions	Deletes specific publisher assertions from the assertion collection managed by a specific publisher

UDDI Publisher API 2-2

Methods	Description
get_authToken (deprecated)	Logs you into the registry
discard_authToken (deprecated)	Logs you out of the registry
find_relatedBusinesses(deprecated)	Finds matching publisherAssertion entries
get_publisherAssertions	Gets a list of publisherAssertion entries
get_registeredInfo	Gets an abbreviated list of businesses and tModels currently being managed by a given publisher
get_assertionStatusReport	Gets a summary of publisherAssertion entries
set_publisherAssertions	Saves the entire set of publisher assertions for an individual publisher

UDDI Inquiry API 1-2

Following table describes some of the methods available under the UDDI Inquiry API:

Methods	Description
find_business	Finds matching businessEntity entries
find_service	Finds matching businessService entries
find_binding	Finds matching bindingTemplate entries
find_tModel	Finds matching tModel entries
find_relatedBusinesses	Finds information about businessEntity registrations that are related to a specific business entity whose key is passed in the inquiry
get_businessDetail	Gets businessEntity entries
get_serviceDetail	Gets businessService entries
get_bindingDetail	Gets bindingDetail entries
get_tModelDetail	Gets tModel entries

UDDI Inquiry API 2-2

Methods	Description
get_operationalInfo	Gets operational information related to one or more entities in the registry
get_registeredInfo (deprecated)	Gets an abbreviated list of businessEntity and tModel entries
add_publisherAssertions (deprecated)	Adds one or more publisherAssertion entries
set_publisherAssertions (deprecated)	Updates one or more publisherAssertion entries
delete_publisherAssertions (deprecated)	Deletes one or more publisherAssertion entries

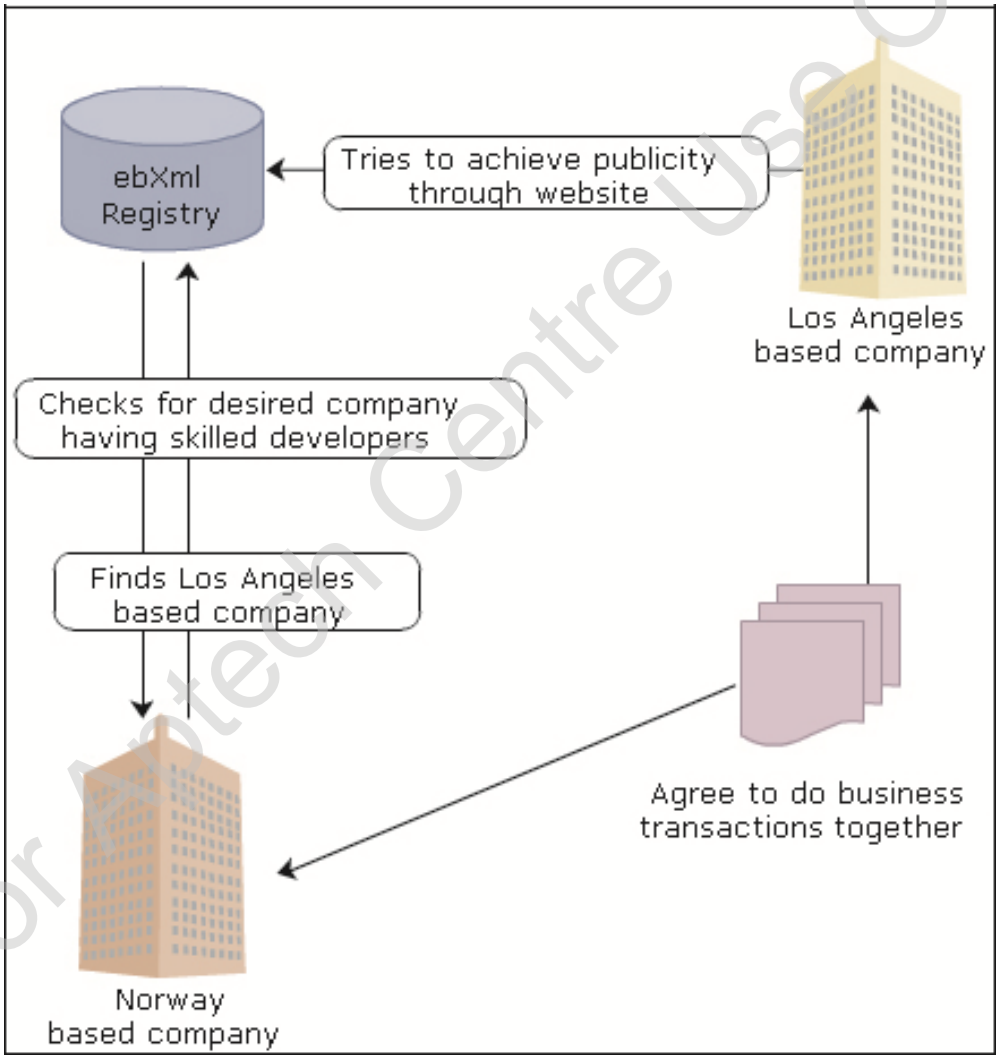
Electronic Business Scenario 1-2

Consider the case of two companies situated far away from each other and looking forward to start some business. Following are the details:

1. A small company in Los Angeles has a few skilled mobile application developers.
2. To establish a good international reputation, the company tries to achieve through Website.
3. Another small company in Norway is involved in mobile application development, but it does not have skilled developers.
4. When the Norway based company gets a new project, it looks for appropriate company.
5. The Norway based company checks the ebXML Registry for the desired company and finds the company in Los Angeles.
6. A contract is negotiated and both companies agree to do business together.
7. The Los Angeles based company sends the first mobile application back to Norway, which is accepted by Norway based company.
8. In return, the company transfers the money to Los Angeles.

Electronic Business Scenario 2-2

Following figure shows an E-business scenario:



Electronic Business Process 1-2

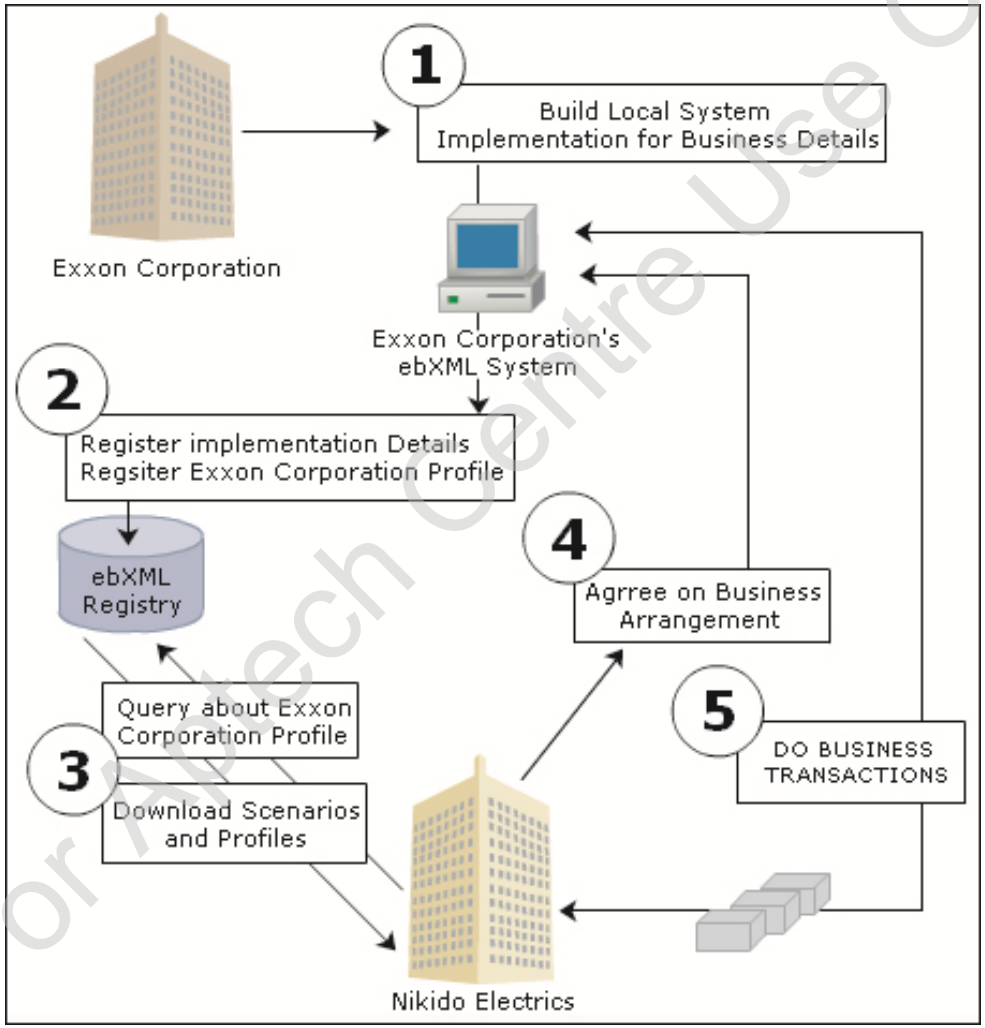
Consider a scenario of how two companies, Exxon Corporation, a Los Angeles-based company and Nikido Electrics, a Norwegian company could implement a business that adheres to ebXML processes.

Following are the steps both the companies must do to do electronic business in ebXML format:

1. Exxon Corporation creates a Collaboration Protocol Profile (CPP) by defining itself in an ebXML compliant application.
2. Exxon Corporation submits its CPP to the registry and tries to achieve publicity through its Website.
3. Nikido Electrics, already registered at the ebXML registry/repository, is looking for new trading partners. It starts a new query and receives the CPP of Exxon Corporation. Using the CPP of Exxon Corporation and that of Nikido Electrics, ebXML derives a document called Collaboration Protocol Agreement (CPA).
4. Nikido Electrics contacts Exxon Corporation directly and sends the newly created CPA for acceptance.
5. After signing the contract, both the companies do electronic business directly following the business processes defined in the CPA.

Electronic Business Process 2-2

Following figure shows the use of ebXML:

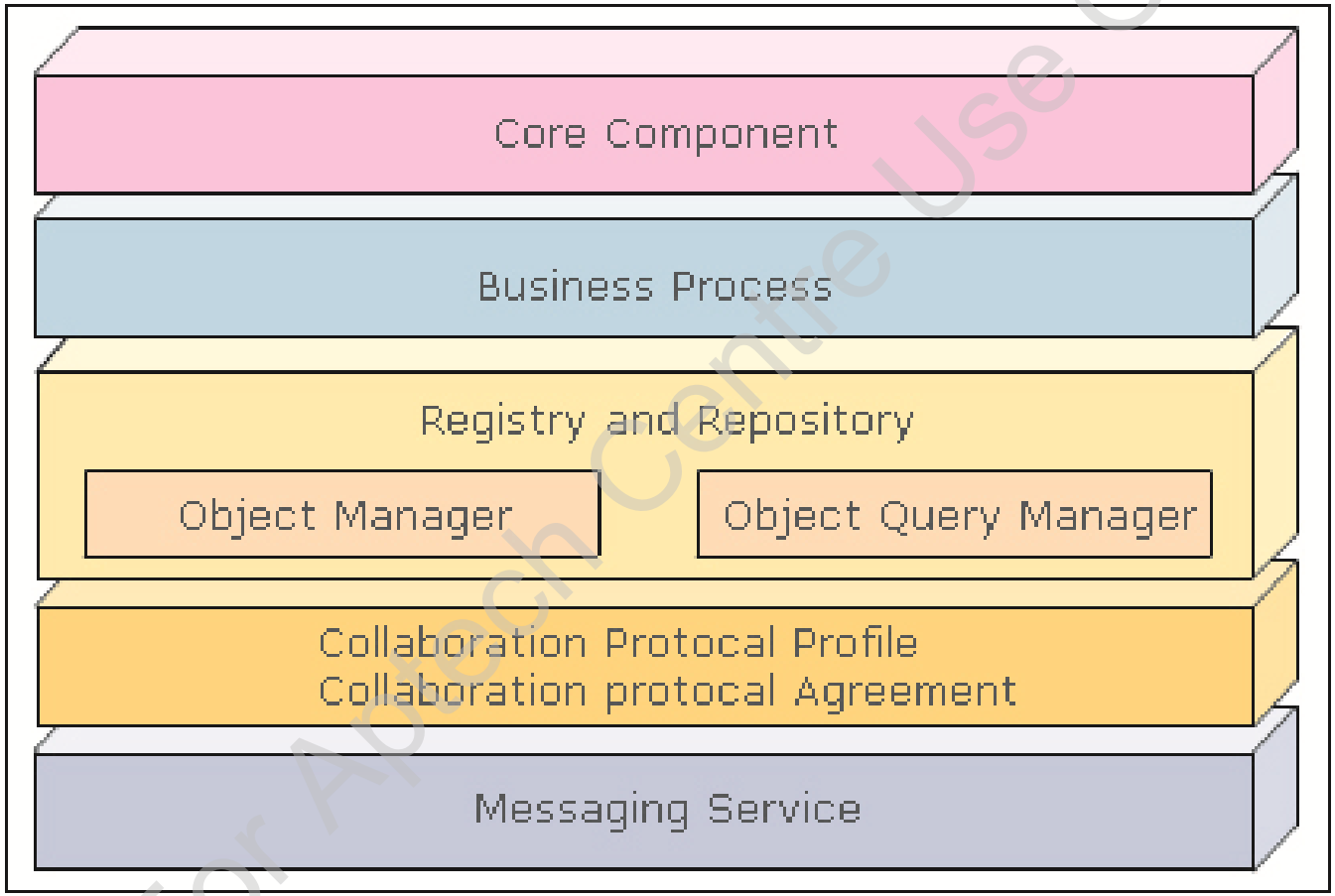


ebXML Registry and Repository 1-2

- The ebXML architecture consists of components such as:
  - ebXML Registry and Repository
  - ebXML Business Processes
  - ebXML Collaboration Protocol Profiles
  - Collaboration Protocol Agreements
  - ebXML Core Components
  - ebXML Messaging Service
- The clients communicate with the registry using the following two interfaces:
  - **Object Manager:** Provides the methods to create new objects within the registry and affect state transitions on existing objects.
  - **Object Query Manager:** Provides the methods to find and access the objects created within the registry.

ebXML Registry and Repository 2-2

Following figure shows the ebXML architecture:



ebXML Business Processes

- The second component of ebXML architecture is Business Process.
- Following are the main parts of Business Collaboration:

Business Collaboration Specification	• Set of roles interacting through a set of specialized protocol by exchanging Business Documents between business partners
Business Transaction	• An atomic unit of work in a trading arrangement between two business partners
Business Document Flows	• In Business Transaction, by default, each requesting role has one Business Document
Choreography	• Defines which Business Transaction follows which Business Transaction
Patterns	• Set of predefined transaction interactions

ebXML CPP and CPA 1-3

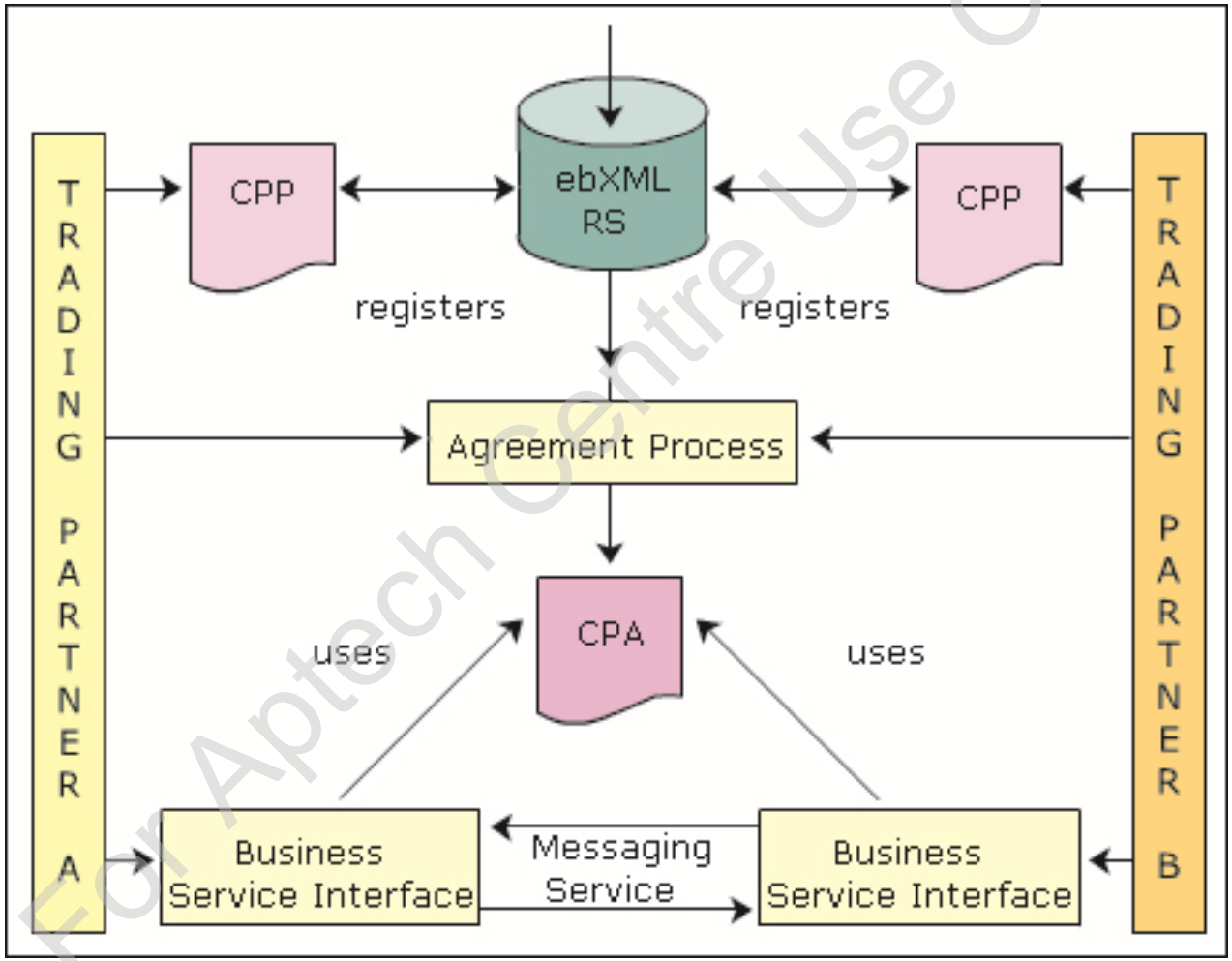
- The third component of ebXML architecture includes the Collaboration-Protocol Profile (CPP) and Collaboration-Protocol Agreement (CPA) documents.
  - **CPP**: Is an XML document that contains information about a business and the way it exchanges information with other businesses . Is an intersection of two CPP documents.
  - **CPA**: Is derived from two or more CPPs.

ebXML CPP and CPA 2-3

- The working of CPP and CPA is as follows:
  1. Two companies or trading partners create CPPs by adding information about the company.
  2. Both companies submit their CPPs to the ebXML registry.
  3. The CPPs are stored in the repository.
  4. A company then queries the ebXML registry to get a suitable CPP from a potential trading partner.
  5. Using the CPP of the two companies, ebXML derives the CPA.
  6. The company sends the CPA to the potential trading partner for acceptance.
  7. Upon agreement, the CPA gets registered in ebXML registry, and both the companies start the business.
  8. After registering the CPA in the ebXML registry, both companies configure their Business Service Interface (BSI) software with the newly created CPA.
  9. After the configuration, the BSI software is used to carry out electronic business.

ebXML CPP and CPA 3-3

- Following figure shows ebXML CPP and CPA:



Core Components and Messaging Service

- Two components of ebXML architecture are: Core Components and the Messaging Service.

Core Component	ebXML Messaging Service (ebXML MS)
• Is the last entity that gets referenced in a CPP document. • Is also called Aggregated Component. • Is a reusable building block containing information about a business concept. For example, for a concept such as purchase order, the core components are date of purchase order, sales tax, and total amount.	• Provides the message exchange functionality within the ebXML infrastructure. • Is an entity used by the Business Service Interface software to send and receive XML messages from one point to another • Is based on TCP/IP, FTP, HTTP, and SMTP protocols • Uses packages data in SOAP messages and transmits them using HTTP, TCP/IP, FTP, or SMTP protocol

Java API for XML Registries (JAXR) 1-2

- JAXR is an abstraction-based API therefore; Java programmers can use this API to develop registry client programs that can be used across different target registries.
- Following Code Snippet shows how to use UDDI to publish and query a Web service:

```
public class publishWebService{
public static void main(String[] args) throws
JAXRException{

//Configure the ConnectionFactory to use JAXR Provider for UDDI
System.setProperty("javax.xml.registry.
ConnectionFactoryClass", "com.ibm.xml.registry.
uddi.ConnectionFactoryImpl");
ConnectionFactory connFac = ConnectionFactory.newInstance();

//Configure URLs for UDDI inquiry and publish APIs.
Properties p = new Properties();
p.setProperty("javax.xml.registry.queryManagerURL", query_url);
p.setProperty("javax.xml.registry.lifeCycleManagerURL", publish_
url);
connFac.setProperties(p);

//Create a connection to the UDDI registry
Connection conn = connFac.createConnection();
```

Java API for XML Registries (JAXR) 2-2

```
//Specify credentials for accessing UDDI registry.
PasswordAuthentication passwdAuth = new
PasswordAuthentication("Aptech_user", new char[]
{'p', 'a', 's', 's', '@', '1', '2', '3' });
Set userCredentials = new HashSet();
userCredentials.add(passwdAuth);
conn.setCredentials(userCredentials);
//Retrieve the javax.xml.registry.BusinessLifeCycleManager interface
RegistryService regServ = conn.getRegistryService();
BusinessLifeCycleManager bLCM = regServ.getBusinessLifeCycleManager();
//Create an Organization named "Aptech".
Organization org = bLCM.createOrganization("Aptech");
//Add the Organization to a Collection
Collection coll = new ArrayList();
coll.add(org);
//Save the Organization to the UDDI registry.
BulkResponse bRes = bLCM.saveOrganizations(coll);
//Obtain the Organization's Key from the response.
if (bRes.getExceptions() == null) {
Collection res = bRes.getCollection();
Key orgaKey = (Key)res.iterator().next();
System.out.println("\nOrganization Key = " + orgKey.getId());
   }
 }
}
```

Summary

- SOAP overcomes the problems of EDI and RPC by using XML and HTTP.
- The structure of a SOAP message consists of Envelope element, Header element and body element.
- Document/Literal messaging mode is used to transmit data in XML fragments, whereas RPC/Literal messaging mode is used to transmit method call and the values returned by a method.
- Transport Protocols deals with transportation of SOAP messages over HTTP.
- Errors in SOAP messages are sent as fault messages over HTTP.
- A WSDL document is an XML document that adheres to the WSDL XML schema. It generally consists of six elements namely definitions, types, message, portType, binding and service.
- An ebXML registry is similar to the UDDI registry only with the difference that along with storing information about the WSDL document it also stores the document too.