IT1451-XML AND WEBSERVICE

S.JAYAPRAKASH, MCA.,MPhil.,MTech.,

ASSISTANT PROFESSOR

DEPT OF COMPUTER SCIENCE & ENGINEERING

PRCET

SEMESTER VIII

IT1451 – XML AND WEB SERVICES L T P C 3 1 0 4

UNIT I XML TECHNOLOGY FAMILY 9

XML – Benefits – Advantages of XML over HTML – EDI – Databases – XML based standards – Structuring with schemas – DTD – XML schemas – XML processing – DOM – SAX – Presentation technologies – XSL – XFORMS – XHTML – Transformation – XSLT – XLINK – XPATH – Xquery

UNIT II ARCHITECTING WEB SERVICES 9

Business motivations for web services – B2B – B2C – Technical motivations – Limitations of CORBA and DCOM – Service Oriented Architecture (SOA) – Architecting web services – Implementation view – Web services technology stack – Logical view – Composition of web services – Deployment view – From application server to peer to peer – Process view – Life in the runtime.

UNIT III WEB SERVICES BUILDING BLOCKS 9

Transport protocols for web services – Messaging with web services – Protocols – SOAP – Describing web services – WSDL – Anatomy of WSDL – Manipulating WSDL – Web service policy – Discovering web services – UDDI – Anatomy of UDDI – Web service inspection – Ad hoc discovery – Securing web services.

UNIT IV IMPLEMENTING XML IN E–BUSINESS 9

B2B – B2C applications – Different types of B2B interaction – Components of E -Business XML systems – EBXML – RosettaNet – Applied XML in vertical industry – Web services for mobile devices.

UNIT V XML CONTENT MANAGEMENT AND SECURITY 9

Semantic web – Role of meta data in web content – Resource description framework – RDF schema – Architecture of semantic web – Content management workflow – XLANG – WSFL – Securing web services

TEXT BOOKS

1. Ron Schmelzer and Travis Vandersypen, ―XML and Web Services unleashed‖, Pearson Education, 2002.

2. Keith Ballinger, ―. NET Web Services Architecture and Implementation‖,

Pearson Education, 2003.

REFERENCES

1. David Chappell, ―Understanding .NET A Tutorial and Analysis‖, Addison

Wesley, 2002.

2. Kennard Scibner and Mark C. Stiver, ―Understanding SOAP‖, SAMS publishing, 2000.

3. Alexander Nakhimovsky and Tom Myers, ―XML Programming: Web Applications and Web Services with JSP and ASP‖, Apress, 2002.

UNIT I

XML TECHNOLOGY FAMILY

XML – Benefits – Advantages of XML over HTML – EDI – Databases – XML based standards – Structuring with schemas – DTD – XML schemas – XML processing – DOM – SAX – Presentation technologies – XSL – XFORMS – XHTML – Transformation – XSLT – XLINK – XPATH – Xquery

XML:

As the Internet emerged and rapidly became a viable place to conduct business, communicate, and entertain, it became apparent that the need to exchange data in an open manner was still unmet. SGML provided a solution for exchanging data in a structured, standardized manner, but it was inappropriate for direct application on the Internet.HTML was a pure-Internet approach for displaying and presenting information in a plat- form-independent manner, but it was wholly inadequate for representing data structures. EDI had proven its merit in conducting electronic business transactions but was ill-suitedto being exchanged on the Internet and lacked the sophisticated features of either HTMLor SGML. It was obvious something more was needed.

In this environment, an initiative led Jon Bosak and supported by a group of SGML and industry notables, including Tim Bray, C. M. Sperberg-McQueen, Jean Paoli, and James Clark, sought to take some of the best features of SGML and ―put them on the Web.‖Their goal was to take the standard, generalized manner for marking up data and extendit with metadata while stripping out all the complexities and optional features that madeSGML too difficult to implement. On top of that, the new language would be designedinherently for the Internet and have the support of the Internet‘s top standards-setting body, the World Wide Web Consortium (W3C). Originally called Web SGML, this new language was later named the Extensible Markup Language (XML) .

Benefits of XML

The very nature of XML is that it is a structured document format that represents not only the information to be exchanged but also the metadata encapsulating its meaning.

A) XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

B) XML Simplifies Data Sharing

In the real world, computer systems and databases contain data in incompatible formats. XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications.

C) XML Simplifies Data Transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

D) XML Simplifies Platform Changes

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost. XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

E) XML Makes Your Data More Available

Different applications can access your data, not only in HTML pages, but also from XML data sources.With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

F) XML is Used to Create New Internet Languages

A lot of new Internet languages are created with XML.

Here are some examples:

- XHTML
- WSDL for describing available web services
- WAP and WML as markup languages for handheld devices
- RSS languages for news feeds
- RDF and OWL for describing resources and ontology
- SMIL for describing multimedia for the web

Advantages of XML over HTML

HTML was created to meet a very different need than XML. It is clear that XML will not now, or perhaps ever, completely replace HTML. Except of course with regard to the XML-enabled version of HTML, known as XHTML. HTML was designed as a language to present hyperlinked, formatted information in a Web browser. It has no capability to represent metadata, provide validation, support extensibility by users, or support even the basic needs of e-business. Fundamentally, the difference is that HTML is intended for consumption by humans, whereas XML is meant for both machine and human consumption.

AdvantagesofXMLoverEDI

EDIadoptionhasbeenfairlywidespread,eventhoughmainlyamonglarger-sizedbusinesses.ThecostofEDIimplementationandongoingmaintenancecanbemeasuredin thebillionsinaggregate.Millionsofdollarsintransactionsoccuronadailybasisusing EDI-mediatedmessages.Itwouldbeverydifficult,ifnotimpossible,touprootallthis activityandreplaceitwithexclusivelyXML-basedtransactions.Thesebusinesseshave somuchmoneyandtimeinvestedinANSIX12/EDIthattheywillbefairlyslowto adoptanewstandard,whichwouldnecessitatenewprocessingtechnology,mapping software,andback-endintegration.Forthem,itwouldseemthattheywouldneedto discardtheirexisting,workingtechnologyinfavourofanunprovenandstillimmature technology.

1) XMLisagoodreplacementforEDIbecauseitusestheInternetforthe dataexchange.
2) ComparedtoEDIandotherelectroniccommerceanddata-interchangestandards,XML offersseriouscostsavingsandefficiencyenhancementsthatmakeimplementationof XMLgoodforthebottomline
3) XML‘sbuilt-invalidity checking,low-costparsersandprocessingtools,Extensible Stylesheet Language(XSL)basedmapping,anduseoftheInternetkeepdownmuchof thee-commercechaincost.
4) TheuseoftheInternetitselfgreatlylowersthebarrierforsmallandmedium-sizedcompaniesthathavefoundEDItoocostlytoimplement
5) TheideathatXMLrepresentsanew,freshapproachto solvingmanylingeringproblemsinaflexiblemannerappealstomanyinseniormanagement.
6) XML syntax allows for international characters that follow the Unicode standard to be included as content in any XML element

AdvantagesofXMLoverDatabases

Relational and object-oriented databases and formats can represent data as well as metadata, but for the most part, their formats are not text based. Most databases use a proprietary binary format to represent their information. There are other text-based formats that include metadata regarding information and are structured in a hierarchical representation, but they have not caught on in popularity nearly to the extent that XML or even SGML has. Oneoftheprimaryissuesfacedbyalternatefileformatanddatabaselanguagesisthat processingtoolsarecustom,proprietary,orexpensive.Whentoolsarewidespread,they areusuallyspecifictotheparticularfileformatinquestion.OneofXML‘sgreatest strengthsisthatprocessingtoolshavebecomerelativelywidespreadandinexpensive,if notfree.

1.6) XML based standards

We have already discussed the advantages of the ―ML‖ in XML, but the ―X‖ presents advantages of its own. Extensibility, as applied to XML, is the ability for the language tobe used to define specific vocabularies and metadata. Rather than being fixed in describing a particular set of data, XML, in conjunction with its DTDs and schema, is able to define any number of documents that together form a language of their own.

Indeed, hundreds, if not thousands, of specific document vocabularies have been created based on XML to meet the different needs of healthcare, manufacturing, user interface design, petroleum refining, and even chess games. Text files and relational database schemas are rigid in that they are meant to represent the information contained within and nothing more. It would be a difficult proposition at best to add a new set of information to a text file or relational database management system (RDBMS). XML files, especially those created using an ―open content

model,‖ can easily be extended by adding additional elements and attributes. Whole classes of documents can be defined simply by sending a document with a new DTD or schema. Sharing a DTD and schema within a user community results in a joint specification—if not a de facto or explicit standard.

Structuring with schemas

A Simple XML Document:

We‘ll talk about a shirt. There‘s actually a lot we can talk about with regard to a shirt: size, color, fabric, price, brand, and condition, among other properties. The Following example shows one possi- ble XML rendition of a document describing a shirt. Of course, there are many other possible ways to describe a shirt, but this example provides a foundation for our further discussions.

```
<?xml version=‖1.0‖?>

<shirt>
    <model>Zippy Tee</model>
    <brand>Tommy Hilbunger</brand>
    <price currency=‖USD‖>14.99</price>
    <on_sale/>
    <fabric content=‖60%‖>cotton</fabric>
    <fabric content=‖40%‖>polyester</fabric>
    <options>
    <colorOptions>
    <color>red</color>
    <color>white</color>
    </colorOptions>
    <sizeOptions>
    <size>Medium</size>
    <size>Large</size>
    </sizeOptions>
    </options>
    <description> This is a <b>funky</b> Tee shirt similar to the Floppy Tee shirt </description>
</shirt>
```

XML Declaration:

The XML declaration is a processing instruction of the form <?xml ...?>. Although it is not required, the presence of the declaration explicitly identifies the document as an XML document and indicates the version of XML to which it was authored. In addition, the XML declaration indicates the presence of external markup declarations and character encoding. Because a number of document formats use markup similar to XML, the declaration is useful in establishing the document as being compliant with a specific version of XML without any doubt or ambiguity. In general, every XML document should use an XML declaration. As documents increase in size and complexity, this importance likewise grows.

Components of the XML Declaration:

Component	*Description*
<?xml	Startsthebeginningoftheprocessinginstruction(inthis case,fortheXMLdeclaration).
Version=‖xxx‖	DescribesthespecificversionofXMLbeingusedinthe document(inthiscase,version1.0oftheW3Cspecifica-

	tion).Futureiterationscouldbe2.0,1.1,andsoon.
standalone=‖xxx‖	Thisstandaloneoptiondefineswhetherdocumentsare allowedtocontainexternalmarkupdeclarations.This optioncanbesetto—yes‖or—no‖.
encoding=‖xxx‖	Indicatesthecharacterencodingthatthedocumentuses. Thedefaultis—US-ASCII‖butcanbesettoanyvaluethat XMLprocessorsrecognizeandcansupport.Themostcommonalternat e

ValidXMLDeclarations

```
<?xmlversion=‖1.0‖standalone=‖yes‖?>
<?xmlversion=‖1.0‖standalone=‖no‖?>
<?xmlversion=‖1.0‖encoding=‖UTF-8‖standalone=‖no‖?>
```

Document Type Declaration

A Document Type Declaration names the document type and identifies the internal con- tent by specifying the root element, in essence the first XML tag that the XML-process- ing tools will encounter in the document. A DOCTYPE can identify the constraints on the validity of the document by making a reference to an external DTD subset and/or include the DTD internally within the document by means of an internal DTD subset.

GeneralFormsoftheDocumentTypeDeclarations:

```
<!DOCTYPENAMESYSTEM—file‖>
<!DOCTYPENAME[]>
<!DOCTYPENAMESYSTEM—file‖[]>
```

ComponentsoftheDocumentTypeDeclaration

Component	*Description*
<	ThestartoftheXMLtag(inthiscase,thebeginningofthe DocumentTypeDeclaration).
!DOCTYPE	ThebeginningoftheDocumentTypeDeclaration.
NAME	Specifiesthenameofthedocumenttypebeingdefined. ThismustcomplywithXMLnamingrules.
SYSTEM	Specifiesthatthefollowingsystemidentifierwillberead andprocessed.
—file‖	Specifiesthenameofthefiletobeprocessedbythesys- tem.
[	StartsaninternalDTDsubset.
]	EndstheinternalDTDsubset.
>	TheendoftheXMLtag(inthiscase,theendofthe

DocumentTypedeclaration).

MarkupandContent:

In general, six kinds of markup can occur in an XML document: elements, entity references, comments, processing instructions,marked sections, and Document Type Declarations.

XML BASED STANDARDS:

1) XPATH

XPath is a syntax for defining parts of an XML document. XPath uses path expressions to navigate in XML documents. XPath contains a library of standard functions. XPath is a major element in XSLT. XPath is a W3C Standard

2) XSD

It defines elements that can appear in a document. defines attributes that can appear in a document. It defines which elements are child elements. defines the order of child elements. It defines the number of child elements. It defines whether an element is empty or can include text. It defines data types for elements and attributes. It defines default and fixed values for elements and attributes

3) XSL

XSL describes how the XML document should be displayed! XSL consists of three parts:XSLT - a language for transforming XML documents, XPath - a language for navigating in XML documents, XSL-FO - a language for formatting XML documents

4) XSLT

A common way to describe the transformation process is to say that XSLT transforms an XML source-tree into an XML result-tree.XSLT stands for XSL Transformations. XSLT is the most important part of XSL. XSLT transforms an XML document into another XML document. XSLT uses XPath to navigate in XML documents. XSLT is a W3C Recommendation.

XML DOCUMENT STRUCTURE:

XML document includes the following

- The xml declaration
- The document type declaration
- The element data
- The attribute data
- The character data or XML content

STRUCTURING WITH SCHEMAS:

– TWO TYPES OF SCHEMAS : SIMPLE TYPE, COMPLEX TYPE

SIMPLE TYPE: A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

RULES FOR XML STRUCTURE:

All XML elements must have a closing tag. XML tags are case sensitive, All XML elements must have a proper nesting, All XML Documents must contain a single root element, Attribute values must be quoted, Attributes may only appear once in the same start tag, Attribute values cannot contain references to external entities, All entities except amp,lt,gt,apos,and quot must be declared before they are used.

SIMPLE TYPE:

XML Schema has a lot of built-in data types. The most common types are:

- **xs:string**
- **xs:decimal**
- **xs:integer**
- **xs:boolean**
- **xs:date**
- **xs:time**

Example:

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

And here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

COMPLEX TYPE:

A complex element is an XML element that contains other elements and/or attributes. Look at this simple XML document called "note.xml":

```
<?xml version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget to submit the assignment this monday!</body>
</note>
```

The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):

```
<!ELEMENT note (to, from, heading, body)><!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"  elementFormDefault="qualified">
<xs:element name="note">
<xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
```

```
<xs:element name="heading" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

DTD:

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.

TWO TYPES OF DTD

- INTERNAL DTD
- EXTERNAL DTD

INTERNAL DTD:

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
Example XML document with an internal DTD:
<?xml version="1.0"?>
<!DOCTYPE note [ <!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)> ]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget to prepare for the UNIT TEST this weekend</body></note>
```

EXTERNAL DTD:

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename">
This is the same XML document as above, but with an external DTD
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget to prepare for the UNIT TEST this weekend!</body>
</note>
And this is the file "note.dtd" which contains the DTD:
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>


<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
```

<!ELEMENT body (#PCDATA)>

XML SCHEMAS:

XML Schema is an XML-based alternative to DTDs. An XML Schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

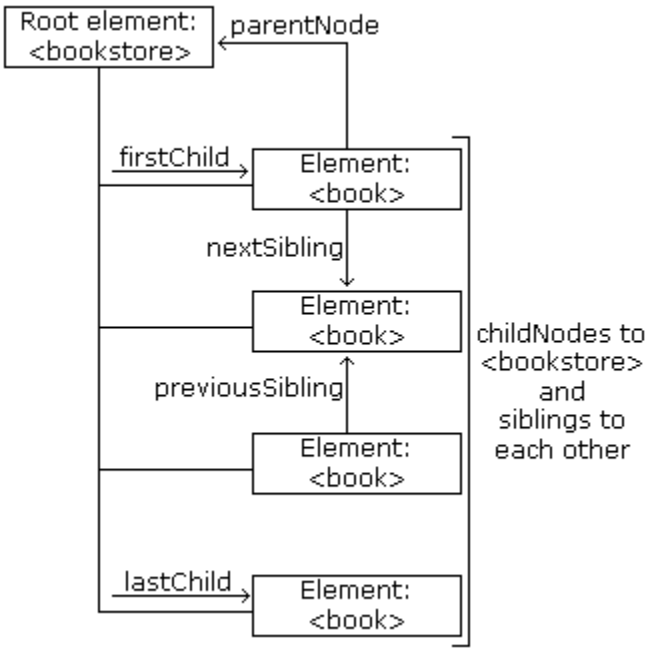
An XML Schema defines elements that can appear in a document, defines attributes that can appear in a document, defines which elements are child elements , defines the order of child elements, defines the number of child elements, defines whether an element is empty or can include text, defines data types for elements and attributes, defines default and fixed values for elements and attributes.

XML PROCESSING:

The JavaTM API for XML Processing (JAXP) includes the basic facilities for working with XML documents through the following standardized set of Java Platform APIs. There are two types of XML Parsers namely Document Object Model (DOM), Simple API For XML Parsing (SAX).

DOM:

The XML DOM views an XML document as a tree-structure. The tree structure is called a node-tree. All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created. The nodes in the node tree have a hierarchical relationship to each other. The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters). In a node tree, the top node is called the root. Every node, except the root, has exactly one parent node. A node can have any number of children. A leaf is a node with no children. Siblings are nodes with the same parent.



SAX:

SAX (Simple API for XML) is an event-driven model for processing XML. Most XML processing models (for example: DOM and XPath) build an internal, tree-shaped

representation of the XML document. The developer then uses that model's API (getElementsByTagName in the case of the DOM or findnodes using XPath, for example) to access the contents of the document tree. The SAX model is quite different. Rather than building a complete representation of the document, a SAX parser fires off a series of events as it reads the document from beginning to end. Those events are passed to event handlers, which provide access to the contents of the document.

Event Handlers:

There are three classes of event handlers: DTDHandlers, for accessing the contents of XML Document-Type Definitions; ErrorHandlers, for low-level access to parsing errors; and, by far the most often used, DocumentHandlers, for accessing the contents of the document. A SAX processor will pass the following events to a DocumentHandler:

1) The start of the document.
2) A processing instruction element.
3) A comment element.
4) The beginning of an element, including that element's attributes.
5) The text contained within an element.
6) The end of an element.
7) The end of the document.

EXAMPLE FOR SAX:

```
<html><body>
<script type="text/javascript">
 try //Internet Explorer
 {
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
 } catch(e)
{ try
//Firefox, Mozilla, Opera, etc.
 {
xmlDoc=document.implementation.createDocument("","",null);
} catch(e)
 {
alert(e.message)
}
} try
{ xmlDoc.async=false;
 xmlDoc.load("books.xml");
document.write("xmlDoc is loaded, ready for use");
} catch(e)
{alert(e.message)}
</script>
</body>
</html>
```

SAX vs DOM

- SAX is straightforward and clear
  - it can process documents of any size
  - the data becomes available to the handlers as soon as the parser starts to read
- But for some applications the application may have to build an intermediate data structure to hold the elements of the input document
  - this can involve a lot of complicated programming
- Alternatively, the parser itself could build a data structure holding the whole document (if it is not too big!)
  - and we could navigate the structure using supplied methods
  - this saves us a lot of work
- DOM (Document Object Model) parsers do just that
  - JAXP contains classes to implement DOM as well as SAX

PRESENTATION TECHNOLOGIES:

1) XSL
2) XFORMS
3) XHTML

XSL & XSLT:

XSL stands for EXtensible Stylesheet Language.

What is XSLT?

XSLT stands for XSL Transformations. XSLT is the most important part of XSL. XSLT transforms an XML document into another XML document. XSLT uses XPath to navigate in XML documents. XSLT is a W3C Recommendation. We want to **transform** the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
        <catalog>
        <cd>
        <title>Empire Burlesque</title>
        <artist>Bob
Dylan</artist><country>USA</country><company>Columbia</company><price>
10.90</price>
        <year>1985</year>
        </cd> . . .

</catalog>
```

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
```

```
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th align="left">Title</th>
<th align="left">Artist</th>
</tr>
<xsl:for-each select="catalog/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

The result is:

XFORMS:

XForms is the next generation of HTML forms. XForms is richer and more flexible than HTML forms. XForms will be the forms standard in XHTML 2.0. XForms is platform and device independent. XForms separates data and logic from presentation. XForms uses XML to define form data. XForms stores and transports data in XML documents. XForms contains features like calculations and validations of forms. XForms reduces or eliminates the need for scripting. XForms is a W3C Recommendation. The XForms Model. The XForms model is used to describe the data. The data model is an instance (a template) of an XML document. The XForms model defines a data model inside a <model> element:

```
<model>
<instance>
<person>
<fname/>
<lname/>
</person>
```

```
</instance>
<submission id="form1" action="submit.asp" method="get"/>
</model>
```

The XForms Model

The XForms **model** is used to **describe** the data. The data model is an instance (a template) of an XML document. The XForms model defines a data model inside a <model> element:

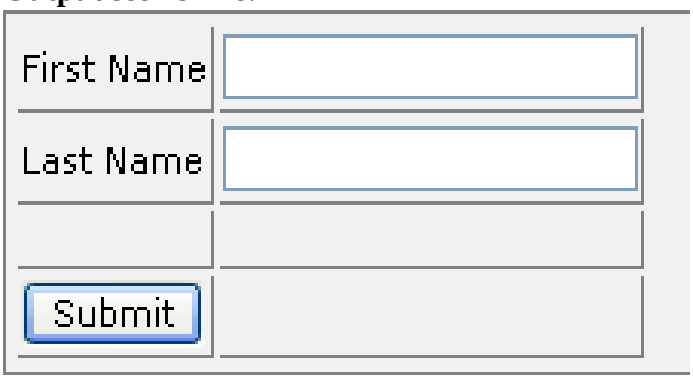
```
<model>
<instance>
<person>
<fname/>
<lname/>
</person>
</instance>
<submission id="form1" action="submit.asp" method="get"/>
</model>
```

All together it looks as below

```
<xforms>
<model>
<instance>
<person><fname/><lname/></person></instance>
<submission id="form1" action="submit.asp" method="get"/>
</model>
<input ref="fname"><label>First Name</label></input><input


ref="lname"><label>Last Name</label></input><submit submission="form1">
<label>Submit</label>
</submit>
</xforms>
```

Output seems like:



XHTML:

XHTML stands for EXtensible HyperText Markup Language. XHTML is aimed to replace HTML. XHTML is almost identical to HTML 4.01. XHTML is a stricter and cleaner version of HTML. XHTML is HTML defined as an XML application. XHTML is a W3C Recommendation. XHTML elements must be properly nested. XHTML elements must always be closed. XHTML elements must be in lowercase. XHTML documents must have one root element.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
    <head>
    <title>simple document</title>
    </head>
    <body><p>a simple paragraph</p></body>
</html>
```

The 3 Document Type Definitions :

1) DTD specifies the syntax of a web page in SGML.
2) DTD is used by SGML applications, such as HTML, to specify rules that apply to the markup of documents of a particular type, including a set of element and entity declarations.
3) XHTML is specified in an SGML document type definition or 'DTD'.

An XHTML DTD describes in precise, computer-readable language, the allowed syntax and grammar of XHTML markup. There are currently 3 XHTML document types:

i. STRICT
ii. TRANSITIONAL
iii. FRAMESET

XHTML 1.0 specifies three XML document types that correspond to three DTDs:

i. Strict
ii. Transitional
iii. Frameset

XHTML 1.0 Strict:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

We can use this when you want really clean markup, free of presentational clutter. We can use this together with Cascading Style Sheets.

XHTML 1.0 Transitional:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

We can use this when you need to take advantage of HTML's presentational features and when you want to support browsers that don't understand Cascading Style Sheets.

XHTML 1.0 Frameset:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

We can use this when you want to use HTML Frames to partition the browser window into two or more frames.

Why XHTML Modularization?

By splitting XHTML into modules, the W3C (World Wide web Consortium) has created small and well-defined sets of XHTML elements that can be used separately for small devices, or combined with other XML standards into larger and more complex applications.

Some of the modules are as below:

Module name	Description
Applet Module	Defines the deprecated* applet element
Base Module	Defines the base element
Basic Forms Module	Defines the basic forms elements
Basic Tables Module	Defines the basic table elements
Bi-directional Text Module	Defines the bdo element
Client Image Map Module	Defines browser side image map elements
Edit Module	Defines the editing elements del and ins
Forms Module	Defines all elements used in forms
Frames Module	Defines the frameset elements

TRANSFORMATION:

- XSLT
- XLINK
- XPATH
- XQuery

XLINK:

XLink Syntax:

In HTML, we know (and all the browsers know!) that the <a> element defines a hyperlink. However, this is not how it works with XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what hyperlink elements will be called in XML documents.The solution for creating links in XML documents was to put a marker on elements that should act as hyperlinks.

Example:

```
<?xml version="1.0"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
<homepage xlink:type="simple" xlink:href="http://www.w3schools.com">Visit W3Schools</homepage><homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

XLink Attribute Reference

Attribute	Value	Description
xlink:actuate	onLoad onRequest other none	Defines when the linked resource is read and shown
xlink:href	*URL*	The URL to link to
xlink:show	embed new replace other none	Where to open the link. Replace is default
xlink:type	simple extended locator arc resource title none	The type of link

XPATH:

XPath is a syntax for defining parts of an XML document. XPath uses path expressions to navigate in XML documents. XPath contains a library of standard functions
XPath is a major element in XSLT. XPath is a W3C Standard.

XPath Terminology

Nodes:

In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes. XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node).

Relationship of Nodes

i. Parent
ii. Children
iii. Siblings
iv. Ancestors
v. Descendants

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Expression	Description
nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Path Expression	Result
bookstore	Selects all the child nodes of the bookstore element
/bookstore	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Predicates:

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** IE5 and later has implemented that [0] should be the first node, but according to the W3C standard it should have been [1]!!
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00

Selecting Unknown Nodes:

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

Path Expression	Result
/bookstore/*	Selects all the child nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have any attribute

Selecting several paths:

Path Expression	Result
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

XQuery:

XQuery is ***the*** language for querying XML data. XQuery for XML is like SQL for databases. XQuery is built on XPath expressions. XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.). XQuery is a W3C Recommendation .

<title lang="**en**">**XQuery Kick Start**</title>
<author>**James McGovern**</author>
<author>**Per Bothner**</author>
<author>**Kurt Cagle**</author>
<author>**James Linn**</author>
<author>**Vaidyanathan Nagarajan**</author>
<year>**2003**</year>
<price>**49.99**</price>
</book>
- <book category="**WEB**">
<title lang="**en**">**Learning XML**</title>
<author>**Erik T. Ray**</author>
<year>**2003**</year>
<price>**39.95**</price>
</book>
</bookstore>

Functions:

XQuery uses functions to extract data from XML documents. The doc() function is used to open the "books.xml" file:

doc("books.xml"), Path Expressions

XQuery uses path expressions to navigate through elements in an XML document.The following path expression is used to select all the title elements in the "books.xml" file: doc("books.xml")/bookstore/book/title(/bookstore selects the bookstore element, /book selects all

the book elements under the bookstore element, and /title selects all the title elements under each book element), The XQuery above will extract the following:

<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

Predicates:

XQuery uses predicates to limit the extracted data from XML documents. The following predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30: doc("books.xml")/bookstore/book[price<30]The XQuery above will extract the following:

<book category="CHILDREN">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>

With FLWOR:

FLWOR is an acronym for **"For, Let, Where, Order by, Return"**. The **for** clause selects all book elements under the bookstore element into a variable called $x.The **where** clause selects only book elements with a price element with a value greater than 30.The **order by** clause defines the sort-order. Will be sort by the title element.The **return** clause specifies what should be returned. Here it returns the title elements.

Example: doc("books.xml")/bookstore/book[price>30]/title

The following FLWOR expression will select exactly the same as the path expression above:

for $x in doc("books.xml")/bookstore/book where $x/price>30 return $x/title

The result will be:

<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

With FLWOR you can sort the result:

for $x in doc("books.xml")/bookstore/book where $x/price>30 order by $x/title return $x/title

QUESTION BANK

PART – A(2 MARKS)

1. What are the three major aspects to extend the enterprise from a constrained network

to broad reach of web?

2. What are the three key design elements that by omission contribute XML‘s success?

3. XML History

4. What are the different revolution in which XML is playing a major role?

5. What are the advantages of xml?

6. What is Electronic Data Interchange (EDI)?

7. What is W3c (World Wide Web) Consortium?

8. What is XML?

9. List out the reasons for not using attributes to store data.

10. What is SOAP?

11. What is a web service?

12. What are all the xml language basics?

13. What is an entity? Give Example.

14. Explain briefly about .NET and J2EE?

15. Explain briefly about data revolution?

16. What is the role of xml?

17. What are XForm?

18. What is VoiceXML?

19. What is XPath?

20. What are the Element Naming Rules used in XML?

PART – B (16 MARKS)

1. Roles and Advantages of XML (16)

2. Explain briefly XML: The Three Revolutions (16)

3. Explain Web Services. (16)

4. Explain XML & DTD. (16)

5. XML Language Basic (16)

UNIT – II

ARCHITECTING WEB SERVICES 9

Business motivations for web services – B2B – B2C – Technical motivations – Limitations of CORBA and DCOM – Service Oriented Architecture (SOA) – Architecting web services – Implementation view – Web services technology stack – Logical view – Composition of web services – Deployment view – From application server to peer to peer – Process view – Life in the runtime.

What are Web Services?

Simply put, Web Services are loosely coupled, contracted components that communicate via XML-based interfaces. Let‘s take a closer look at this definition:

- Loosely coupled means that Web Services and the programs that invoke them can be changed independently of each other. Loose coupling also implies that Web Services are platform independent.
- Contracted means that a Web Service‘s behavior, its input and output parameters, and how to bind to it are publicly available.
- A component is encapsulated code, which means that the implementation of each component is hidden from outside the component. Each component‘s functionality is only known by the interface it exposes.
- Because all Web Services‘ interfaces are built with XML, they all share the advan- tages of XML: They have a human readable, text-based format that is firewall friendly and self-describing. All Web Services are described using a standard XML notation called its service description.

Put another way, Web Services are self-contained applications that can be described, published, located, and invoked over the Internet (or any network, for that matter).

Business Motivations for Web Services;

The vision of global e-business largely remains unrealized. Executives dream about seamless interactions both with other companies as well as e-marketplaces, but the technology lags behind the vision. Today‘s information technology is still extraordinarily complex and expensive. Even with standards such as Electronic Data Interchange (EDI), Java 2 Enterprise Edition (J2EE), Common Object Request Broker Architecture (CORBA), and Windows Distributed interNet Application (Windows DNA), communicating between different corporate systems is still filled with hair-pulling detail work.

The business world needs more powerful techniques to scale business solutions without increasing complexity to unmanageable levels. In addition, there is a clear need for open, flexible, and dynamic solutions for enabling global e-business interactions among systems. The Web Services model promises to deliver these solutions by addressing complexity and costs, providing a common language for B2B e-commerce, and enabling the vision of a global e-marketplace.

B2B E-Commerce:

Business to Business (B2B) e-commerce has been around for more than a decade in the form of the Electronic Data Interchange (EDI). EDI is quite powerful and has gained widespread acceptance but is limited by its semantic ambiguity. For example, a ―quantity‖ field in a given

form may stand for number of boxes for one company but the number of pallets for another. People have to resolve each ambiguity manually, making EDI useful primarily in a hub-and-spoke arrangement, where one large company can dictate the meaning of each field to its suppliers.

When the Internet opened up the prospect of many-to-many e-commerce, it soon became clear that there needed to be a way to agree upon a single business vocabulary for all participants in each trading group. XML provided the basis for building such vocabularies because of its inherent extensibility. However, XML‘s greatest strength also proved to be a weakness, because its extensibility led to hundreds of different business vocabularies, often with overlapping applicability.

The Web Services model addresses this Tower of Babel problem by providing for dynamic service descriptions. Individual Web Services can describe their interfaces at runtime, allowing for dynamic interpretation of the semantics of the XML that underlies the messages Web Services send and receive.

Technical Motivations for Web Services:

The technical motivations for Web Services are far more complex than the business motivations. Fundamentally, technologists are looking for the simplicity and flexibility promised, but never delivered, by RPC architectures and object-oriented technologies.

Limitations of CORBA and DCOM:

Programming has been performed on a computer-by-computer basis for much of the history of computing. Programs were discrete chunks of computer code that ran on individual computers. Even object-oriented programming originated in a single-computer environment. This isolated computer mind set has been around so long that it pervades all thinking about software.

Then along came networks, and technologists looked for ways to break up program functionality onto multiple computers. Early communication protocols, such as the Network

File System for Unix and Microsoft‘s Distributed Computing Environment, focused on the network layer. These protocols, in turn, led to the development of wire protocols for distributed computing—in particular, the Object Remote Procedure Call (ORPC) proto- col for Microsoft‘s DCOM and the Object Management Group‘s Internet Inter-ORB Protocol (IIOP) that underlies CORBA.

RPC architectures such as DCOM and CORBA enabled programs to be broken into different pieces running on different computers. Object-oriented techniques were particularly suited to this distributed environment for a few reasons. First, objects maintained their own discrete identities. Second, the code that handles the communication between objects could be encapsulated into its own set of classes so that programmers working in a distributed environment needn‘t worry about how this communication worked.

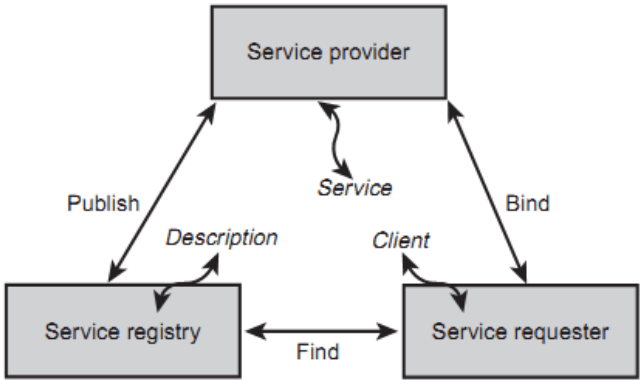
However, programmers still had that isolated computer mindset, which colored both DCOM‘s and CORBA‘s approach: Write your programs so that the remote computer appears to be a part of your own computer. RPC architectures all involved marshalling a piece of a program on one computer and shipping it to another system.

The Service-Oriented Architecture (SOA)

In order for Web Services to be able to work well together, they must participate in a set of shared organizing principles we call a service-oriented architecture (SOA). The term service-

oriented means that the architecture is described and organized to support Web Services‘ dynamic, automated description, publication, discovery, and use.

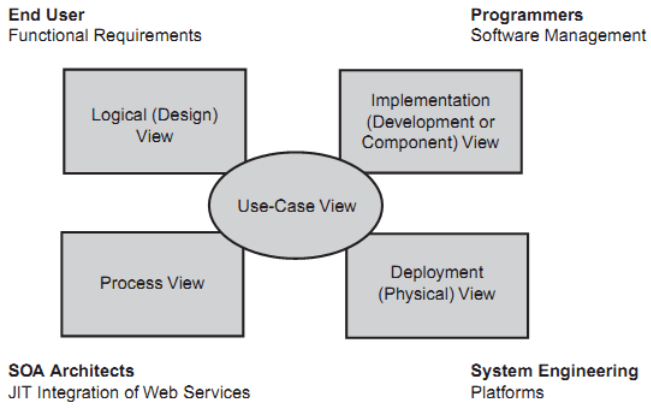
The SOA organizes Web Services into three basic roles: the service provider, the service requester, and the service registry. The relationships among these three roles are shown in Figure.



Architecting Web Services:

Oneestablishedmodelforhowarchitectsvisualizethesystemsbeforethemisthe*4+1ViewModelofSoftwareArchitecture*,popularizedbyPhilippeKruchtenofRational Software.Whereasthefourblindmeneachtouchtheelephantinadifferentplaceand thereforecometodifferentunderstandingsofit,thearchitecthasclearvision,seeingthe elephantfromallfourviews.Asaresult,thearchitecthasacomprehensivepicture oftheelephant.

Thisisthesamewiththe4+1ViewModel.Thismodeldescribesfourdistinctwaysof lookingatthearchitectureforasystem,plusafifthviewthatoverlapstheothers,as showninFigure.

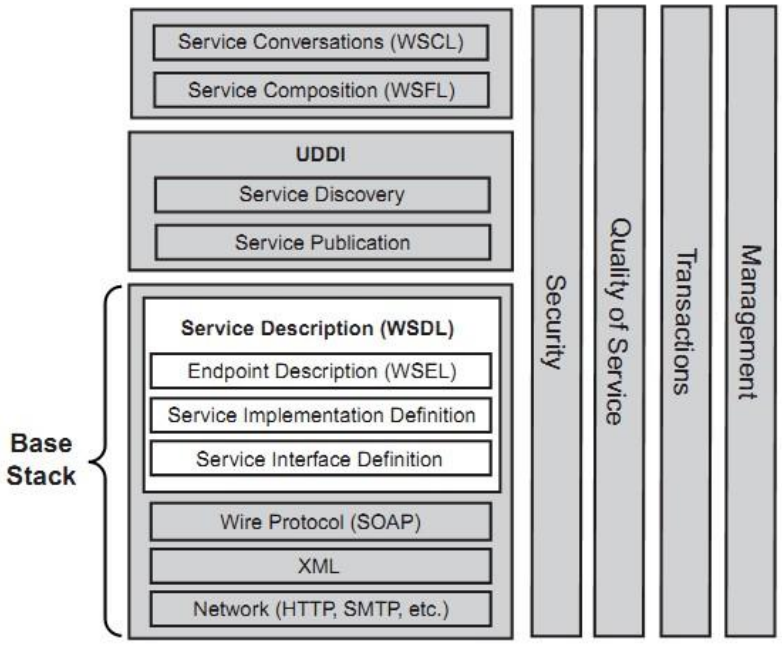


The Web Services Technology Stack:

Thebasestackincludesthosetechnologiesnecessarytocreateandinvoke WebServices. Atthebottomisthenetworklayer,whichfundamentallyallowsWeb

Servicestobe availabletoservicerequesters.AlthoughHTTPisthedefactostandardnetworkprotocol, thearchitectmayconsideranyofanumberofotheroptions,includingSMTP(fore-mail),FTP,IIOP,ormessagingtechnologiessuchasMQ.Someofthesechoicesare

request/responsebased,whereasothersaremessagebased;furthermore,somearesynchromos,whereas othersareasynchronous.Thearchitectmayfindthatinalargesys-
tem,acombinationofdifferentnetworkprotocolsisappropriate.



On top of the SOAP layer comes three layers that together form the service description. WSDL is the de facto standard for service descriptions, with the addition of the still- tentative WSEL for endpoint descriptions. The service interface definition contains the binding, portType, message, and type elements, which form the portion of the service description that is reusable from one implementation to another.

The service implementation definition, however, contains those elements that are specific to each implementation: the service and port elements. A third party (say, a standards body) might specify the service interface definition for a particular type of Web Service, leaving the service implementation definition up to each implementation team.

Next comes the endpoint description, which introduces semantics to the service descriptions that apply to a particular implementation. Endpoint descriptions can contain security, QoS, and management attributes that help to define the policies for each of these vertical columns.

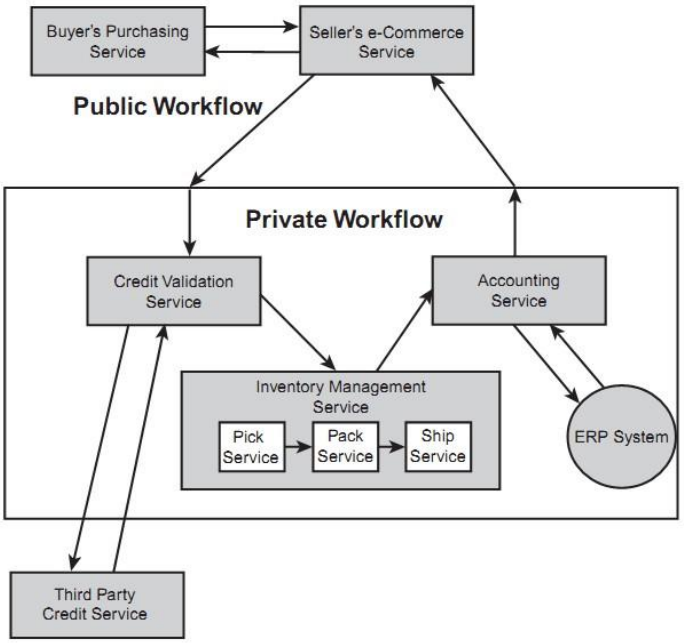
Once the architect has dealt with all the issues in the base stack, the Web Services are essentially fully constructed. Next, the development team uses UDDI to publish the services to a registry or another repository of information about available Web Services. Once Web Services are published, UDDI can then be used to discover them in the registries.

The Logical Architectural View: Composition of Web Services:

The Logical (or Design) Architectural View starts with the end user‘s functional requirements and provides a top-down abstraction of the overall design of the system. In the case of B2B functionality (say, in the case of processing a purchase order), the user interface may be handled separately from the Web Services; therefore, the ―end users‖ in this case are the businesses themselves. In other cases, Web Services may provide functionality to the user interface more directly.

In the B2B case, the functional requirements of a Web Services–based system will typically involve complex conversations among Web Services that participate in multi-step business processes. In addition, the individual Web Services involved are likely to be composed of component Web Services. As a result, an architect working from the Logical View will likely be concerned with workflows of Web Services.

For example, let‘s take the case of a buyer‘s Web Service contacting a seller‘s Web Service to make a purchase. Figure shows a possible (simplified) workflow for this interaction.



This workflow consists of two separate workflows: a public workflow as well as one private to the seller. From the buyer‘s point of view, the seller is exposing a single public Web Service that is composed of separate Web Services in succession.

The interfaces to the two public services are both written in WSDL. The buyer has obtained the seller‘s service description beforehand—either by looking it up in a registry or through a prearranged relationship between the buyer and the seller. The buyer uses the service description to build the SOAP messages it exchanges with the seller.

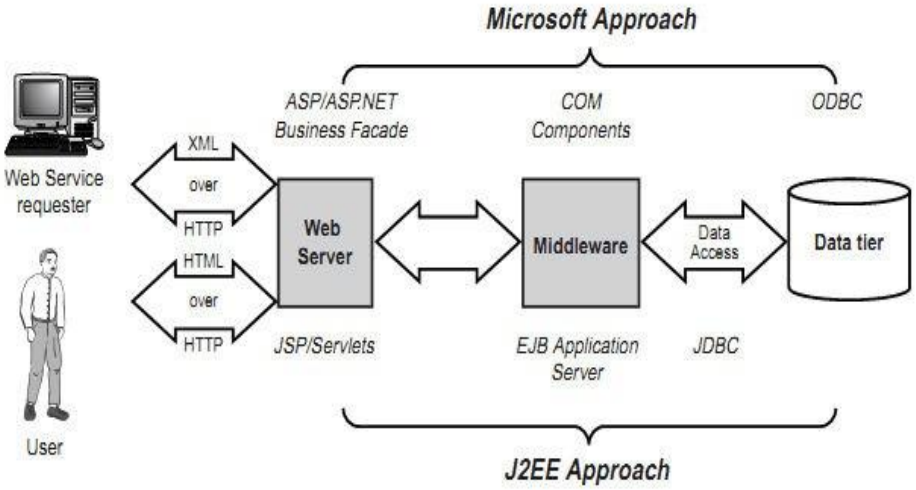
Once the seller receives a request from the buyer, a sequence of business processes within the private workflow takes place. First, a credit-validation service sends a request to a third-party, credit-checking Web Service, which it may have established a preexisting relationship with. This third-party service is an example of an enabling service. Depending on the response from the third-party service, the seller continues with the e-commerce workflow or possibly sends a ―credit rejected‖ response back to the buyer. (The architect must consider both the ―rejected‖ special case as well as how to handle the situation where the third-party credit service is unavailable.) In a more general case, it will likely not be necessary to query this service if the seller has an

established relationship with the buyer.

Once the buyer‘s credit is approved, the internal credit-validation service sends a request to the inventory-management service. This service is recursively constructed from individual component services (three of which are shown for illustration purposes, but in reality such services would be more complex). The architect must determine the interface for the inventory-management service as well as detail the workflow that takes place within the service.

The Deployment Architectural View: From Application Servers to Peer-to-Peer:

The Deployment (or Physical) Architectural View maps the software to its underlying platforms, including the hardware, the network, and the supporting software platforms. Today, Web Services are hosted on application server platforms such as IBM‘s WebSphere, BEA‘s WebLogic, and Microsoft‘s Windows 2000. There are many benefits to building Web Services on top of platforms like these: They handle database access, load balancing, scalability, and interface support as well as provide a familiar environment for dealing with hardware and network issues.



This model follows a traditional n-tier architecture, except that the Web server is also responsible for sending and receiving the XML messages that form the Web Services interface. The technology that supports Web Services is therefore already well under- stood; the fundamental difference between Web Services and Web pages is that pages are intended for humans to read, whereas Web Services expose an interface intended for machines.

Running Web Services off of Web servers is not the only way to support the services, however. It is also possible to build Web Services on a peer-to-peer (P2P) developer model. P2P, popularized by the Napster music service, is a distributed

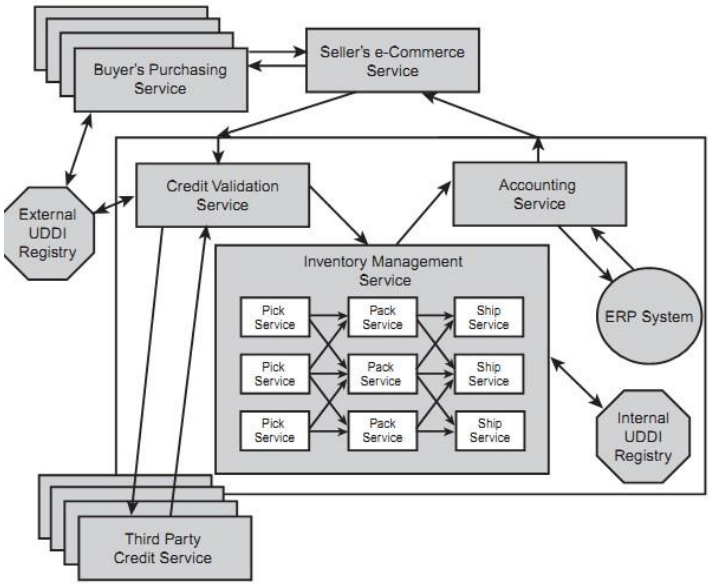
architecture that does not rely on central servers but rather distributes responsibility to systems (called peers) in the network. Unfortunately, P2P technologies are every bit as new and bleeding edge as Web Services, so only time will tell which P2P models will become established. The self-organizing promise of Web Services does lend itself to P2P, but a lot of work remains before we will see how this fascinating area will develop.

The Process Architectural View: Life in the Runtime

The Process Architectural View addresses all runtime issues, including processes, concurrency, and scalability. As the applications of Web Services move up the hierarchy of Web Service integration options to JIT integration the Process Architectural View will take on increasing importance. In fact, the Process Architectural View will be where the bulk of the SOA architect‘s work will take place.

For example, let‘s take another look at the simple e-commerce workflow. If you just look at the figure, you might think that there‘s nothing much new here; this diagram could represent an e-commerce system based on a simple n-tier architecture.

The reason that the diagram doesn‘t immediately demonstrate the power of the Web Services model is that in the diagram, the buyer has already identified the seller, the seller has already identified its third-party credit service, and the seller‘s private work- flow is already put in place. If all these statements are in fact true, then, yes, Web Services has little to offer over traditional n-tier architectures. On the other hand, let‘s take a JIT approach, as shown in Figure.



QUESTION BANK

PART – A(2 MARKS)

1. What are the advantages of schema over DTD?

2. What are the data types in an xml schema?

3. What is DOM? What are the different levels of DOM?

4. What are the drawbacks of CSS?

5. Write any two differences between XSLT and CSS?

6. What are the different XSLT elements?

7. What is VoiceXML?

8. What is XQuery?

9. What is XForm?

10. What is XPath?

11. What are complex types?

12. What all are the presentation technologies?

13. What are all the Transformation techniques?

14. Explain any two XForm implementations?

15. Important of SAX?

16. What is Info Set?

17. What is RDF (Resource Description Framework)?

18. What is metadata?

19. What are the components of RDF?

20. What are RDF vocabularies?

PART – B (16 Marks)

1. Explain briefly XML Transformation? (16)

2. Explain briefly XML Schema: (16)

3. Explain about Presentation Technique? (16)

4. Short notes on XML Namespaces?(16)

5. Explain briefly DTD? (16)

UNIT – III

WEB SERVICES BUILDING BLOCKS 9

Transport protocols for web services – Messaging with web services – Protocols – SOAP – Describing web services – WSDL – Anatomy of WSDL – Manipulating WSDL – Web service policy – Discovering web services – UDDI – Anatomy of UDDI – Web service inspection – Ad hoc discovery – Securing web services.

Transport protocols for web services:

Without standards for how to send messages, Web services wouldn't exist. Transport protocols are key to the success of any network, including the World Wide Web and the Internet in general. A thorough understanding of transports and how they work (and can break) is key to understanding Web services.

One of the most powerful design decisions with SOAP was to make it transport independent, which means that you can send a message over any transport you choose. SOAP merely specifies how to wrap the envelope in a standard manner. Transports such as HTTP and TCP are becoming common for SOAP traffic; however, UDP (User Datagram Protocol) and SMTP (Simple Mail Transport Protocol—the transport for e-mail) are also being used to some effect with SOAP messaging.

SOAP:

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built. This XML based protocol consists of three parts: an envelope, which defines what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing procedure calls and responses. SOAP has three major characteristics: Extensibility (security and WS-routing are among the extensions under development), Neutrality (SOAP can be used over any transport protocol such as HTTP, SMTP or even TCP), and Independence (SOAP allows for any programming

model).

As an example of how SOAP procedures can be used, a SOAP message could be sent to a web-service-enabled web site such as a real-estate price database, with the parameters needed for a search. The site would then return an XML-formatted document with the resulting data, e.g., prices, location, features. With the data being returned in a standardized machine-parseable format, it can then be integrated directly into a third-party web site or application.

The SOAP architecture consists of several layers of specifications: for message format, Message Exchange Patterns (MEP), underlying transport protocol bindings, message processing models, and protocol extensibility. SOAP is the successor of XML-RPC, though it borrows its transport and interaction neutrality and the envelope/header/body from elsewhere (probably from

WDDX)

The SOAP specification defines the messaging framework which consists of:

- The SOAP processing model defining the rules for processing a SOAP message
- The SOAP extensibility model defining the concepts of SOAP features and SOAP modules
- The SOAP underlying protocol binding framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes
- The SOAP message construct defining the structure of a SOAP message

Processing model

The SOAP processing model describes a distributed processing model, its participants, the **SOAP nodes** and how a SOAP receiver processes a SOAP message. The following SOAP nodes are defined:

- **SOAP sender**

A SOAP node that transmits a SOAP message.

- **SOAP receiver**

A SOAP node that accepts a SOAP message.

- **SOAP message path**

The set of SOAP nodes through which a single SOAP message passes.

- **Initial SOAP sender (Originator)**

The SOAP sender that originates a SOAP message at the starting point of a SOAP message path.

- **SOAP intermediary**

A SOAP intermediary is both a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.

- **Ultimate SOAP receiver**

The SOAP receiver that is a final destination of a SOAP message. It is responsible for processing the contents of the SOAP body and any SOAP header blocks targeted at it. In some circumstances, a SOAP message might not reach an ultimate SOAP receiver, for example because of a problem at a SOAP intermediary. An ultimate SOAP receiver cannot also be a SOAP intermediary for the same SOAP message.

Message format:

XML was chosen as the standard message format because of its widespread use by major corporations and open source development efforts. Additionally, a wide variety of freely available tools significantly eases the transition to a SOAP-based implementation. The somewhat lengthy syntax of XML can be both a benefit and a drawback. While it promotes readability for humans, facilitates error detection, and avoids interoperability problems such as byte-order (Endianness), it

can slow processing speed and can be cumbersome. For example, CORBA, GIOP, ICE, and DCOM use much shorter, binary message formats. On the other hand, hardware appliances are available to accelerate processing of XML messages. Binary XML is also being explored as a means for streamlining the throughput requirements of XML.

Example:
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8 Content-
Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<soap:Header>
</soap:Header>

<soap:Body>
<m:GetStockPrice xmlns:m="http://www.example.org/stock">
<m:StockName>IBM</m:StockName>
</m:GetStockPrice>
</soap:Body>
</soap:Envelope>

Advantages:

SOAP is versatile enough to allow for the use of different transport protocols. The standard stacks use HTTP as a transport protocol, but other protocols such as JMS and SMTP are also usable.

Since the SOAP model tunnels fine in the HTTP get/response model, it can tunnel easily over existing firewalls and proxies, without modifications to the SOAP protocol, and can use the existing infrastructure.

Disadvantages :

Because of the verbose XML format, SOAP can be considerably slower than competing middleware technologies such as CORBA. This may not be an issue when only small messages are sent. To improve performance for the special case of XML with embedded binary objects, the Message Transmission Optimization Mechanism was introduced. When relying on HTTP as a transport protocol and not using WS-Addressing or an ESB, the roles of the interacting parties are fixed. Only one party (the client) can use the services of the other. Developers must use polling instead of notification in these common cases.

SOAP:

SOAP is a standard way of serializing the information needed to invoke services located on remote systems so that the information can be sent over a network (or ―wire‖) to the remote system, in a format the remote system can understand, regardless of what platform the remote service runs on or what language it‘s written in. If you‘re familiar with RPC architectures such as

CORBA and DCOM, this description of SOAP should sound familiar, because SOAP resembles the wire protocols underlying both architectures: the Internet Inter-ORB Protocol (IIOP) that underlies CORBA and Microsoft‘s Distributed Component Object Model (DCOM) protocol, respectively. In fact, SOAP can be thought of as a simplified XML-based replacement for these protocols.

Key Building Block for Web Services:

SOAP provides an additional advantage over traditional ORPC architectures: Because SOAP messages are self-describing, the method calls contained in a SOAP message can vary each time the message is sent. In addition, it is possible to marshal several method calls in a single SOAP message. With a traditional ORPC, each call to a remote method must be handled as a separate roundtrip. A SOAP message, however, can be constructed on-the-fly to send data to multiple methods. Used judiciously, this capability can more than compensate for the slowness of SOAP‘s text-based messages as compared to the binary messages of CORBA and DCOM.

Basic SOAP Syntax:

- SOAP is a messaging framework, consisting of an outer Envelope element that contains an optional Header element and a mandatory Body element.
- SOAP is an encoding format that describes how objects are encoded, serialized, and then decoded when received.
- SOAP is an RPC mechanism that enables objects to call methods of remote objects.

SOAP Message Structure and Namespaces:

Let‘s start with a simple example of a message we might want to send—a request to the server for a person‘s phone number. We might have an interface (here, written in Java) that would expose a method we might call to request the phone number:

```
public  interface  PhoneNumber
{
public  String  getPhoneNumber(String  name);
}
```

Let‘s say, then, that instead of using CORBA or RMI, our client sends an XML-formatted request to the server. This XML might look like the following:

```
<?xml   version=‖1.0‖?>
<PhoneNumber>
<getPhoneNumber>
<name>John  Doe</name>
</getPhoneNumber>
</PhoneNumber>
```

Notice that the root node corresponds to the Java interface, and the method as well as its parameter are nodes, too. We then use our client to create an HTTP request,

and we put the preceding XML in the body of an HTTP POST. We might expect a response from the

server that looks something like the following:

```
<?xml version=‖1.0‖?>
<PhoneNumber>
<getPhoneNumberResponse>
<thenumber>
<areacode>617</areacode>
<numberbody>555-1234</numberbody>
</thenumber>
</getPhoneNumberResponse>
</PhoneNumber>
```

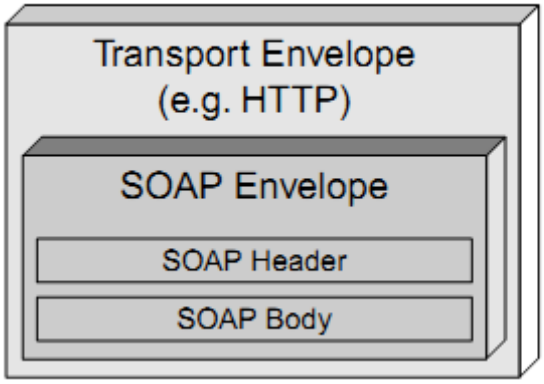
The root node retains the name of the interface, but the method name has the word ―Response‖ appended to it, so the client can identify the correct response by appending ―Response‖ to the calling method name.

In general, constructing request and response messages like the preceding ones is a simple but limited approach. The biggest limitation is that the vocabulary that the client and server use to exchange messages must be agreed upon beforehand. If there is a new method or a new parameter, both the client and the server must reprogram their interfaces. In addition, in a complex message, there could easily be confusion if two methods have parameters with the same name.

In order to resolve these limitations with such simple message formats, SOAP takes advantage of XML namespaces. Let‘s take a look at the same request message recast in SOAP:

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:xsi=”http://www.w3.org/1999/XMLSchema-instance”
xmlns:xsd=”http://www.w3.org/1999/XMLSchema”
SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
<SOAP-ENV:Header>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns:getPhoneNumber xmlns:ns=”PhoneNumber”>
<name xsi:type=”xsd:string”>John Doe</name>
</ns:getPhoneNumber>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Let‘s break down this request and take a closer look. First of all, its root node isEnvelope, which has an optional Header section and a mandatory Body section. The SOAP Envelope is then enclosed in the outer transport envelope, which might be HTTP, SMTP, and so on. All SOAP messages are structured like this, as shown in Figure.



You declare namespaces with the xmlns keyword. There are two forms of namespace declarations: default declarations and explicit declarations. In our sample request, all the declarations are explicit. Explicit declarations take the following form:

xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/

The default declarations look like this:

xmlns=”SomeURI”

SOAP Envelope Element:

The SOAP Envelope element is the mandatory top element of the XML document that represents the SOAP message being sent. It may contain namespace declarations as well as other attributes, which must be ―namespace qualified.‖ The Envelope element may also contain additional subelements, which must also be namespace qualified and follow the Body element.

SOAP Header Element:

The SOAP Header element is optional and is used for extending messages without any sort of prior agreement between the two communicating parties. You might use the Header element for authentication, transaction support, payment information, or other capabilities that the SOAP specification doesn‘t provide.

Let‘s take a look at a typical Header element:

<SOAP-ENV:Header>
<t:Transaction xmlns:t=―myURI‖ SOAP-ENV:mustUnderstand=―1‖></t:Transaction>
</SOAP-ENV:Header>

SOAP Body Element:

The mandatory Body element is an immediate child of the Envelope element and must immediately follow the Header element if a header is present. Each immediate child ofthe Body element is called a body entry. The Body element is used to carry the payload of the SOAP message, and there is a great deal of latitude in what you can place in the Body element. Body entries are identified by their fully qualified element names. Typically, the SOAP encodingStyle attribute is used to indicate the serialization rules for body entities, but this encoding style is not required.

Data Types:

The SOAP specification allows for the use of custom encodings, but typically you are likely to use the default encoding defined in http://schemas.xmlsoap.org/soap/ encoding/. If you use this encoding, you get to take advantage of its data model, which is structured to be consistent with the data models of today's popular programming languages, including Java, Visual Basic, andC++.

TABLE SOAPTerminology

Term	*Meaning*
Value	Astring,thenameofameasurement(includingnumbers, dates,andsoon),oracombinationofsuchvalues.
Simplevalue	Avaluethatdoesn'thavenamedparts.
Compoundvalue	Anaggregateofvalues.Forexample,acompletestreet address(number,street,city,state,andzipcode)wouldbe acompoundvalue.Arraysarealsocompoundvalues. Accessor
	Arolenameorordinalthatdistinguishesavaluewithina compoundvalue.—Zipcode‖wouldbeanaccessortothe streetaddressvalue.
Array	Acompoundvaluewherethemembervaluesaredistin-guishedsolelybytheirordinalposition.
Struct	Acompoundvaluewherethenecessarilyuniqueaccessor nameistheonlydistinctionamongmembervalues.
Simpletype	Aclassofsimplevalues.Forexample,stringisasimple type,whereas—thisstring‖isasimplevaluethatisan instanceofthesimpletype.
Compoundtype	Aclassofcompoundvalues.Eachinstanceofaparticular compoundtypewouldhavetosharethesameaccessors.
Locallyscoped	Anaccessorwhosenameisuniquewithinaparticulartype butnotacrossalltypes.Locallyscopedaccessorsmustbe combinedwiththetypenametobeuniquelyidentified.
Universallyscoped	Anaccessorwhosenameisbased(directlyorindirectly) onaURIandisthereforeuniqueacrossalltypes.
Single-reference	Avaluethatcanbereferencedbyonlyasingleinstanceof anaccessor,asdeterminedbytheschema.
Multi-reference	Avaluethatcouldpotentiallybereferencedbymorethan oneinstanceofanaccessor,asdeterminedbytheschema.
Independent	Anelementthatappearsatthetoplevelofaserialization.
Embedded	Anyelementthatisn'tindependent.

TABLE:SOAPPrimitiveDataTypes

DataType	*Meaning*
SOAP-ENC:string	AnystringofUnicodecharactersthatareallowedina SOAPmessage.
SOAP-ENC:boolean	true,false,1,or0.
SOAP-ENC:decimal	Anumbersuchas44.145629or-0.32,withanarbitrary sizeandprecision.
SOAP-ENC:float	The4-byteIEEE-754floating-pointnumberthatisclosest tothespecifieddecimalstring.
SOAP-ENC:double	The8-byteIEEE-754floating-pointnumberthatisclosest tothespecifieddecimalstring.

SOAP-ENC:integer	Anyinteger.
SOAP-ENC:positiveInteger	Anintegerthatisstrictlygreaterthanzero.
SOAP-ENC:nonPositiveInteger	Anintegerthatislessthanorequaltozero.
SOAP-ENC:negativeInteger	Anintegerthatisstrictlylessthanzero.
SOAP-ENC:nonNegativeInteger	Anintegerthatisgreaterthanorequaltozero.
SOAP-ENC:long	Anintegerbetween-9,223,372,036,854,775,808and +9,223,372,036,854,775,807.
SOAP-ENC:int	Anintegerbetween-2,147,483,648and2,147,483,647.
SOAP-ENC:short	Anintegerbetween-32,768and32,767.
SOAP-ENC:byte	Anintegerbetween-128and127.
SOAP-ENC:unsignedLong	Anintegerbetween0and18,446,744,073,709,551,615.
SOAP-ENC:unsignedInt	Anintegerbetween0and429,496,729.
SOAP-ENC:unsignedShort	Anintegerbetween0and65,535.
SOAP-ENC:unsignedByte	Anintegerbetween0and255.
SOAP-ENC:duration	AlengthoftimegivenintheISO8601extendedformat, representedbyPnYnMnDTnHnMnS(forexample, 19951231T235959).Thenumberofsecondscanbeadec-imaloraninteger.Alltheothervaluesmustbenon-nega-tiveintegers.
SOAP-ENC:dateTime	Aparticularmomentoftimeonaparticulardayuptoan arbitraryfractionofasecondintheISO8601format, whichisCCYY-MM-DDThh:mm:ss(forexample, 1995-12-31T23:59:59).PutonaZsuffixtoindicate coordinateduniversaltime(UTC)oranoffsetfromUTC.
SOAP-ENC:time	AtimeofdayintheISO8601format:hh:mm:ss.sss.A timezonespecifiedasanoffsetfromUTCmayalsobe added.

SOAP-ENC:date	AparticulardategiveninISO8601format:YYYYMMDD
SOAP-ENC:gYearMonth	Aparticularmonthinaparticularyearintheform YYYY-MM.
SOAP-ENC:gYear	AyearintheGregoriancalendarrangingfrom0001upor -0001down.(Thereisnoyearzero.)
SOAP-ENC:gMonthDay	AparticulardayofaparticularmonthintheformMM-DD.
SOAP-ENC:gDay	AparticulardayintheformDD.
SOAP-ENC:gMonth	AparticularmonthintheformMM.
SOAP-ENC:hexBinary	Encodedhexadecimalbinarydata;eachbyteofthedatais replacedbythetwohexadecimaldigitsthatrepresentits unsignedvalue.
SOAP-ENC:base64Binary	Base-64encodedbinarydata.
SOAP-ENC:anyURI	AnabsoluteorrelativeURI.
SOAP-ENC:QName	AnXMLnamesuchasSOAP-ENV:BodyorBody,which mayhaveanoptionalprefix.However,nonprefixednames mustbeinthedefaultnamespace.
SOAP-ENC:NOTATION	Thenameofanotationdeclaredinthecurrentschema.
SOAP-ENC:normalizedString	Astringthatdoesnotcontainanycarriagereturn(\r), linefeed(\n),ortab(\t)characters.Suchstringsarecalled *normalized*.
SOAP-ENC:token	Anormalizedstringwithoutanyleadingortrailingwhite-spaceandnorunsofconsecutivewhitespacecharacters.

	Whitespacecharactersincludethespaceitself,tabs,andso on(aswellasthethreecharactersdisallowedinallnormal- izedstrings).
SOAP-ENC:language	AnRFC1766languageidentifier(theRFC1766standard canbefoundathttp://www.ietf.org/rfc/rfc1766.txt).
SOAP-ENC:NMTOKEN	AnXMLnametoken.
SOAP-ENC:NMTOKENS	AwhitespaceseparatedlistofXMLnametokens.
SOAP-ENC:Name	AnXMLname.
SOAP-ENC:NCName	AnXMLnamethatdoesnotcontainanycolons.
SOAP-ENC:ID	AnNCNamethatisuniqueamongotherIDsinthesame document.
SOAP-ENC:IDREF	AnNCNameusedasanIDsomewhereinthedocument.
SOAP-ENC:IDREFS	AwhitespaceseparatedlistofIDREFelements.
SOAP-ENC:ENTITY	AnNCNamethathasbeendeclaredasanunparsedentity (notyetimplementedconsistently).
SOAP-ENC:ENTITIES	AwhitespaceseparatedlistofENTITYnames(alsoimple- mentedinconsistently).

ThesedatatypescanbeuseddirectlyinSOAPelements:

```
<SOAP-ENC:int>47</SOAP-ENC:int>
```

Inaddition,thedatatypessupporttheidandhrefattributes,allowingmultiplerefer- encestothesamevalue:

```
<SOAP-ENC:stringid=—mystrl>The string</SOAP-ENC:string>
<SOAP-ENC:stringhref=—#mystrl/>
```

Furthermore,iftheattributesaredefinedwithinaschema,youmighthavethefollowing example:

```
<bodystringid=—mystrl>The string</bodystring>
<newstringhref=l#newstringl/>
```

Inthiscase,theschemawouldincludethefollowingfragments:

```
<elementname=lbodystringltype=lSOAP-ENC:stringl>
<elementname=lnewstringl type=lSOAP-ENC:stringl>
```

Sending SOAP messages:

The primary motivation for developing the SOAP specification has been to find a way to make RPC architectures simpler and less problematic. The problems with DCOM and CORBA—vendor dependence, firewall unfriendliness, and unnecessary complexity led to the development of early XML-based RPC architectures, such as XML-RPC.

XML-RPC paved the way for SOAP. Although XML-RPC was a straightforward application of XML, it did not take advantage of XML namespaces and was therefore not fully extensible. For this reason, SOAP was originally thought of as a namespace-capable augmentation-toXML-RPC.

- A request-response operation, which is bidirectional. In this type of operation, the server receives a message from the client and replies with a response message.
- A solicit-response operation, which is also bidirectional, except that the server solicits a request from the client, who then responds, essentially putting the response before the request.
- A one-way message sent from the client to the server with no response message returned.
- A notification message sent from the server to the client.

In essence, the bidirectional messages are inverses of each other, as are the unidirectional ones. In addition to these four basic operations, SOAP also supports the forwarding by intermediaries, which can also be either unidirectional or bidirectional. Furthermore, SOAP faults are only supported by bidirectional messages.

SOAP and HTTP:

HTTP supports two request methods: GET and POST. The GET method sends its parameters in the URL and is typically used to request Web pages from a Web server. The POST method sends data to the server in a payload that comes after the HTTP header. Because POST payloads can be of indefinite length, SOAP requests transmitted via HTTP are sentas HTTP POST requests.

Here‘s the format of a simple HTTP POST request that you might send when submitting a form on a Web page:

```
POST /mypath HTTP/1.1
Host: 123.45.67.89
Content-Type: text/plain; charset=‖utf-8‖ Content-Length: 20
```

Now, let‘s take a look at an HTTP POST request that contains a simple SOAP message:

```
POST /mypath HTTP/1.1
Host: 123.45.67.89
Content-Type: text/xml Content-
Length: 300
SOAPMethodName: urn:mysite-com:PhoneNumber#getPhoneNumber
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=‖http://schemas.xmlsoap.org/soap/envelope/‖
SOAP-ENV:encodingStyle=‖http://schemas.xmlsoap.org/soap/encoding/‖>
<SOAP-ENV:Body>
<ns:getPhoneNumber xmlns:ns=‖PhoneNumber‖>
<name>John Doe</name>
</ns:getPhoneNumber>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In this request, the URI /mypath indicates the SOAP endpoint: It is up to the server to translate this URI into the location of the application charged with accepting this request. The Content-Type for all SOAP messages must be text/xml (as opposed to text/plain for Web pages).

A SOAP response looks pretty much the way you would expect:

```
HTTP/1.0 200 OK
Content-Type: text/xml
Content-Length: 374

<SOAP-ENV:Envelope
xmlns:SOAP-ENV=‖http://schemas.xmlsoap.org/soap/envelope/‖
SOAP-ENV:encodingStyle=‖http://schemas.xmlsoap.org/soap/encoding/‖>
<SOAP-ENV:Body>
```

<m:getPhoneNumberResponse xmlns:m=‖urn:mysite-com:PhoneNumber‖>
<areacode>617</areacode>
<numberbody>555-1234</numberbody>
</m:getPhoneNumberResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Note that the URN for the receiving class appears in the getPhoneNumberResponse element, but there is no SOAPMethodName HTTP header. Such headers are only required for HTTP requests and are not allowed in responses. In addition, if the server encounters an error and returns a SOAP fault, the first line of the HTTP header would be this:

500 Internal Server Error

SOAP and SMTP:

The Simple Mail Transport Protocol (SMTP) is the established standard protocol for sending e-mail messages. Because SOAP envelopes are nothing more than text messages, e-mailing them is elementary on the surface. However, there are several issues that must be dealt with when using SMTP to send a message to an application.

A SOAP message sent via SMTP goes to a mailbox and waits for the server to act upon it. The mailbox will be typically provided by a Post Office Protocol (POP3) server. Therefore, in order for the server to access the SOAP message in a mailbox, the server will typically use a POP3-to-HTTP bridge to post the incoming message to the processing application, and then take the response and use an HTTP-to-SMTP bridge to send it back to the client. The client must then poll its own POP3 mailbox in order to accept the message.

The Future of SOAP:

It should be clear at this point that SOAP is a work in progress. On the one hand, current implementations are inconsistent in their support of the SOAP 1.1 specification. On the other hand, the current spec leaves much to be desired, as well. This section covers some of the most critical features either missing in the 1.1 spec or poorly supported by the cur- rent implementations.

SOAP with Attachments:

At its core, SOAP consists of self-defining serialization rules that allow for the marshaling and unmarshaling of objects into a simple text stream. SOAP‘s focus on objects is quite understandable—after all, it is the Simple Object Access Protocol. However, for SOAP to be a truly useful wire protocol, it must be able to handle large binary objects that don‘t lend themselves to marshaling.

The SOAP Messages with Attachments specification (found at http://www.w3.org/TR/SOAP-attachments) uses the MIME Multipart/Related mechanism for handling attach- ments. This mechanism is the established protocol for handling e-mail attachments and is therefore well accepted in the technical community. When the technical community turned to the discussion of SOAP attachments using MIME, however, it had a problem: How to handle such attachments without burdening the SOAP specification with addi- tional elements? The answer was to construct the SOAP message package as a Multipart/Related media type. In other words, a ―SOAP message with attachments‖ package is actually a MIME Multipart/Related message, where the

SOAP Envelope is one of the parts, instead of the MIME message being included in the SOAP Envelope.

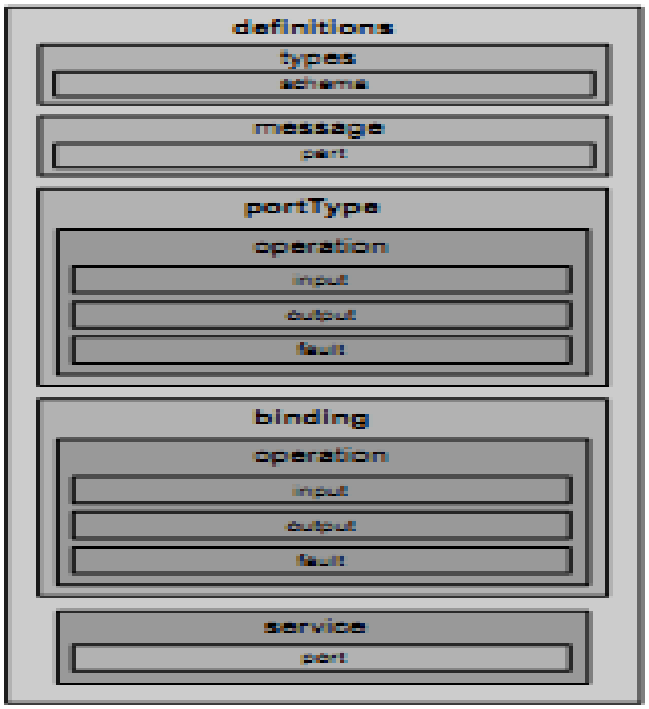
WSDL:

The Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI), along with SOAP, form the essential building blocks for Web Services. Each one, taken separately, serves its own particular purposes.

Basic WSDL Syntax:

WSDL documents use the following elements:

- definitions. Associates the Web Service with its namespaces.
- types. A container for data type definitions, typically using a XML Schema Definition (XSD) or possibly some other type system.
- message. An abstract, typed definition of the data contained in the message.
- operation. An abstract description of an action that the Web Service supports.
- portType. The set of operations supported by one or more endpoints.
- binding. A specification of the protocol and data format for a particular portType.
- port. An endpoint, defined in terms of a binding and its network address (typically a URL). This is not a TCP/IP port, which is represented by a number.
- service. A collection of related endpoints.



Example:

```
<?xml  version=‖1.0‖?>

<definitions  name=‖MyService‖  targetNamespace=‖http://mySite.com/myService.wsdl‖
xmlns:tns=‖http://mySite.com/myService.wsdl‖  xmlns:xsd1=‖http://mySite.com/myService.xsd‖
xmlns:soap=‖http://schemas.xmlsoap.org/wsdl/soap/‖
xmlns=‖http://schemas.xmlsoap.org/wsdl/‖>

<import  namespace=‖http:/mySite.com/myService/schemas‖
location=‖http://  mySite.com/myService/myNameSpace.xsd‖/>
```

```
<types>
<schema targetNamespace="http://mySite.com/myService.xsd"
xmlns="http://www.w3.org/2000/10/XMLSchema">
<element name="MyRequest">
...
</element>
</schema>
</types>

<message name="GetMyInput">
<part name="body" element="xsd1:MyRequest"/>
</message>

<message name="GetMyOutput">
<part name="body" element="xsd1:myParameter"/>
</message>

<portType name="MyServicePortType">
<operation name="MyMethod">
<input message="tns:GetMyInput"/>
<output message="tns:GetMyOutput"/>
</operation>
</portType>

<binding name="MyServiceSoapBinding" type="tns:MyServicePortType">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="MyMethod">
<soap:operation soapAction="http://mySite.com/MyMethod"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>

<service name="MyService">
<documentation>My first service</documentation>
<port name="MyServicePort" binding="tns:MyServiceBinding">
<soap:address location="http://mySite.com/myService"/>
</port>
</service>

</definitions>
```

The types Element:

Thetypeselementcontainsdatatypedefinitionsthatarerequiredbythemessages describedintheWSDLdocument:

```
<types>
  <schematargetNamespace=‖http://mySite.com/myService.xsd‖
      xmlns=‖http://www.w3.org/2000/10/XMLSchema‖>
    <elementname=‖MyRequest‖>
      ...
    </element>
  </schema>
</types>
```

The message and portType Elements:

Within the definitions element is one or more message elements:

```
<message name=‖GetMyInput‖>
<part name=‖body‖ element=‖xsd1:MyRequest‖/>
</message>

<message name=‖GetMyOutput‖>
<part name=‖body‖ element=‖xsd1:myParameter‖/>
</message>
```

SOAP Binding:

When using WSDL documents to describe Web Services that will exchange SOAP messages (that is, SOAP endpoints), you need to have a way to indicate within the WSDL document all the necessary information about the SOAP messages that will be exchanged. WSDL uses extensibility elements to provide this information. The SOAP binding that is provided with WSDL supplies the following information:

- An indication that the WSDL binding is bound to the SOAP protocol.
- How to specify the address for the SOAP endpoints.
- For the HTTP binding of SOAP, the URI for the SOAPAction HTTP header
- A list of definitions for all Header elements in the SOAP Envelope.
- A way of specifying SOAP roots in XSD.

The service Element:

The service element represents a collection of port elements, where each port repre- sents the availability of a binding at a particular endpoint:

```
<service name=‖MyService‖>
<documentation>My first service</documentation>
<port name=‖MyServicePort‖ binding=‖tns:MyServiceBinding‖>
<soap:address location=‖http://mySite.com/myService‖/>
</port>
</service>
```

The binding attribute of the port element ties it to the corresponding binding element defined previously.

The documentation Element:

You should also notice the documentation child element of the preceding service ele- ment. This element essentially allows you to provide a human-readable comment and is allowed in every

other element as well.

The import Element

The import element is an optional element that allows you to break up a WSDL document into multiple documents. When present, it must immediately follow the definitions element. The following example imports a schema, but it is possible to import any WSDL elements, including the definitions element, essentially allowing you to import
an entire WSDL document:

```
<import namespace="http:/mySite.com/myService/schemas"
location="http:// mySite.com/myService/mySchema.xsd"/>
```

The import element is particularly useful for breaking up a WSDL document into inter- face and implementation documents.

soap:binding, soap:operation, soap:header, and soap:body

The following example shows a SOAP binding of a request/response operation over HTTP:

```
<binding name="MyServiceSoapBinding" type="tns:MyServicePortType">
<soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="MyMethod">
<soap:operation SOAPAction="http://mySite.com/MyMethod" style="rpc" />
<input>
<soap:body use="encoded" namespace="http://mySite.com/myService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
<soap:header message="tns:MyMethod" part="MyHeader" use="literal"/>
</input>
<output>
<soap:body use="encoded" namespace="http://mySite.com/myService"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
</binding>
```

Note that the style attribute of the soap:binding element is set to rpc. In the case of a one-way operation over SMTP, for example, the style attribute would have a value of document (and document is the default if the attribute is omitted). The transport attribute (here, set to the URI of the HTTP binding in the SOAP specification) indicates to which transport of SOAP this binding corresponds.

soap:address, soap:fault, and
soap:headerfault

There are a few additional elements in the SOAP binding worth mentioning. First, the soap:address element simply assigns a URI to a port:

```
<service name=‖MyService‖>
<port name=‖MyServicePort‖ binding=‖tns:MyServiceBinding‖>
<soap:address location=‖http://www.mySite.com/MyServiceURL/‖ />
</port>
</service>
```

The soap:fault element specifies the contents of the SOAP Fault element:

```
<fault>
<soap:fault name=‖MyFault‖ use=‖encoded‖
encodingStyle=‖http://schemas.xmlsoap.org/soap/encoding/‖
<http://schemas.xmlsoap.org/soap/encoding/>/>
</fault>
```

Finally, the soap:headerfault element follows the syntax of the soap:fault element but refers to SOAP Fault in Header elements.

WSDL Implementations:

Because WSDL is a bridge technology in the sense that it bridges SOAP and UDDI, you‘re unlikely to find a WSDL toolkit that stands by itself. The two most popular WSDL implementations, therefore, are parts of other toolkits:

- The Microsoft SOAP Toolkit. This toolkit, covered in depth in Chapter 15, is primarily aimed at developers who want to work with SOAP in a Microsoft environment, although it does support Microsoft‘s UDDI implementation.
- The IBM Web Services Toolkit (WSTK). This toolkit provides WSDL support, several security enhancements, UDDI integration, and support for the IBM WebSphere application server. The WSTK also includes the open-source Web Services Description Language for Java Toolkit (WSDL4J)

Demo Client Application MyClass.java

```
public class MyClass
{
public int MyMethod (String arg)
{
return 47;
}
public static void main ( String[] args )
{
System.out.println( ―output‖ );
}
}
```

Introduction to UDDI:

If WSDL is The Empire Strikes Back, then UDDI is The Return of the Jedi, concluding the chapters on Web Services. Universal Description, Discovery, and Integration (UDDI) is a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet as well as public registries of Web Services designed to store information about businesses and the services they offer. UDDI is also a specification for building such registries as well as an application programming interface (API) that exposes the functionality of the registries. Fundamentally, UDDI provides for the publication and discovery of Web Services, which are the key functional components of the Service-Oriented Architecture explained in Chapter 14, ―Architecting Web Services.‖

The UDDI Consortium, established by hundreds of companies, emerged in response to a series of challenges posed by the new Web Services model. These challenges included the following:

- How do you discover Web Services?
- How should information about Web Services be categorized?
- How do you handle the global nature of Web Services? How do you provide for localization?
- How can interoperability be provided, both in the discovery and invocation mechanisms?
- How can you interact with the discovery and invocation mechanisms at runtime?

For UDDI to provide the foundation for Web Services registries, therefore, it had to serve two primary roles within the Web Services model: Service publication and service discovery). The rest of this chapter addresses how UDDI‘s publication and discovery capabilities operate as well as how UDDI addresses the listed challenges.

UDDI Basics:

Public UDDI registries are hosted by operator nodes, which are companies such as Microsoft and IBM that have committed to running public UDDI nodes. The public reg- istry system is loosely modeled after the Domain Name Service (DNS) system, in that there are multiple registries responsible for synchronizing their data with each other. Currently, each public UDDI node synchronizes with the others daily, but a more fre- quent schedule is in the works.

Likewise, as is the case with the DNS system, UDDI registries are not repositories of data; rather, in their roles as registries, they simply provide information on how to find and invoke Web Services. Just as the DNS system provides for unique domain names,
UDDI relies on globally unique identifiers (GUIDs), which are URNs that uniquely iden- tify the resources in each registry.

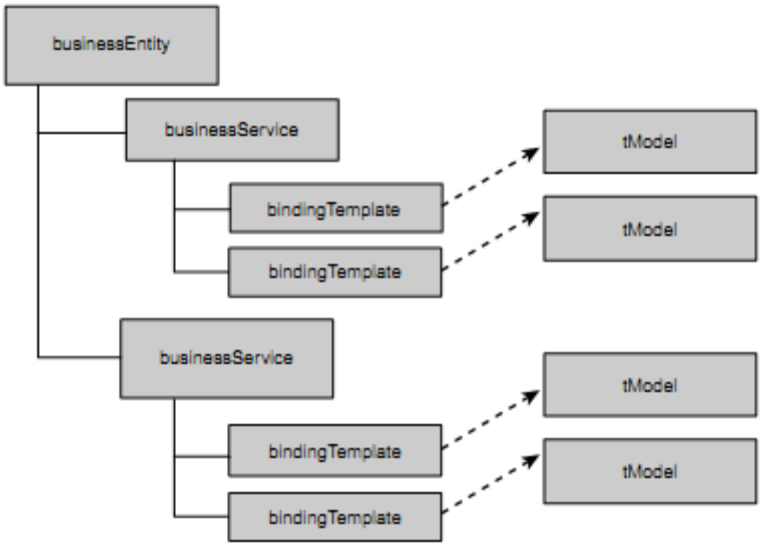
There are three levels of information available in each UDDI registry, which correspond

roughly to the features of a phone book:

• White pages. These pages provide listings of companies that can be queried by name, text description, contact information, and known identifiers (like Dun and Bradstreet‘s DUNS numbers).
• Yellow pages. These pages allow for the looking up of companies by the kind of services they offer, organized into established business categories (industry codes, location, and products and services). These categories are organized into taxonomies.
• Green pages. These pages provide information about how to interact with compa- nies‘ Web Services by exposing service descriptions and binding information.

The Structure of UDDI:

The XML schema that provides the structure of UDDI defines four core types of information in a UDDI registry, as shown in Figure.



• Business information. Provided by the businessEntity element. The

businessEntity element supports the ―white pages‖ and ―yellow pages‖ taxonomies, allowing for structured information about companies. This element is the top-level information manager for the information in the registry about a particular business.

• Service information. Provided by the businessService element. This element supports the ―green pages‖ functionality. The business Service structure contains a group of Web Services related to a category of services or possibly a business process. Each businessService element contains one or more technical Web Services descriptions, which describe how to find and bind to each of the Web Services.

• Binding information. Provided by the bindingTemplate element, which is the element contained within the businessService element that provides the information needed to bind to and invoke a Web Service.‖

• Specifications for services. Enclosed within each bindingTemplate element are special elements that list references to information about specifications for services. These elements, called tModel elements (from ―technical models‖), are metadata about each specification, providing information on a specification‘s name, publishing organizations, and URLs to the specifications themselves. T Model elements have several uses within a UDDI registry, in particular, representing technical specifications for wire protocols (such as SOAP), interchange formats (WSDL), and sequencing rules. Each specification registered as a tModel in a UDDI

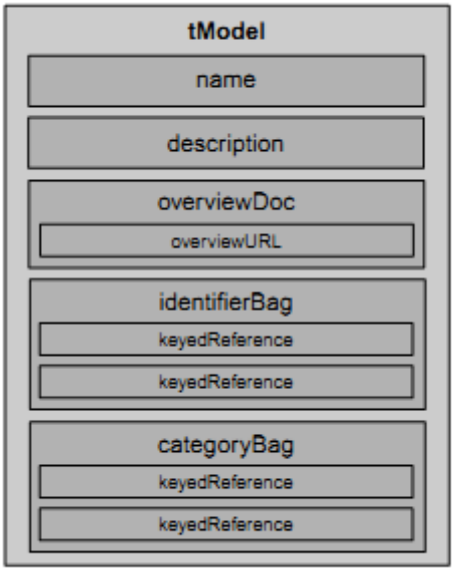
registry receives a unique GUID.

T Model Structure;

Understanding tModel elements is critical to understanding how UDDI works, because they form the basis of how UDDI deals with the meaning of the various specifications it deals with. The concept of a tModel is necessarily somewhat nebulous, because tModel elements consist of metadata (data about data). tModel elements provide a reference sys- tem for UDDI registries that is based on abstraction (in other words, a tModel can define just about anything).

The primary use for tModel elements within UDDI is to represent a technical specification for example, wire protocols (such as SOAP), interchange formats, and the like. When two parties wish to communicate using a particular specification, they must share a mutually agreed on technical identity for the specification they share. This technical identity can be registered in a tModel. Once such a specification is uniquely defined in this way, other parties can refer to it by referring to its unique tModel identifier, which is called a tModelKey. tModelKey elements act as ―technical fingerprints‖ that uniquely designate individual specifications.‖.

The other main use for tModel elements supports how UDDI handles its search capability. Searching, of course, is an essential part of UDDI‘s ―find‖ capability. Searching is provided for in UDDI with the use of two structures: identifierBag and categoryBag. The identifierBag structure defines organizational identity. It consists of name/value pairs that record and define identification numbers—for example, ―DUNS number 12345‖ or ―SS# 987-65-4321.‖ The categoryBag elements, on the other hand, are name/value pairs that correlate specific taxonomy information—for example, ―Florists 45311‖ (from the NAICS taxonomy), ―Massachusetts US-MA‖ (from the ISO 3166 Geographic Taxonomy), or ―Boston 516499‖ (from the GeoWeb taxonomy). A particular florist in Boston, Massachusetts would possess all three of these categoryBag elements, as well as each identifier‘s supercategories. The tModel, then, is used to correlate different levels of these hierarchies—for example, to express the relationship that Boston is within Massachusetts, or that florists are retailers.



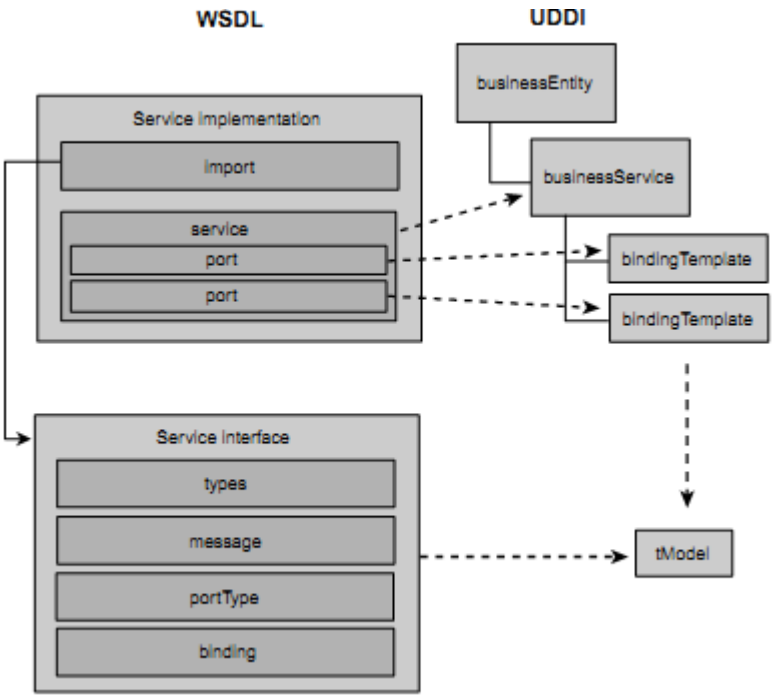
Publishing and Finding WSDL Descriptions in a UDDI Registry:

In order to understand how UDDI uses WSDL as a description language, you must be clear

on how WSDL documents are mapped to the UDDI structure. As discussed earlier, WSDL documents can be organized as service implementation and service interface documents. The service implementation document maps to the UDDI businessService element, whereas the service interface document maps to the tModel elements, as shown in Figure‖.

The first step in publishing a WSDL description in a UDDI registry is publishing the ser vice interface as a tModel in the registry. Here are the steps to follow to create the appropriate tModel:

1. Set the name field of the tModel to the targetNamespace attribute of the definitions element in the interface document. This field is used to locate the appropriate tModel.
2. The description field of the tModel corresponds to the documentation elementof the interface document. This field can have a maximum of 256 characters.
3. Set the overviewURL field of the tModel to the URL and binding specification in the interface document.
4. Set the categoryBag field of the tModel so that its keyed reference is uddi org:types and its keyValue is wsdlSpec. This defines the UDDI entry as a WSDL service interface definition.



tModel.xml—A tModel Created from a WSDL Service Interface

```
<?xml version=‖1.0‖?>
<tModel tModelKey=‖‖>
<name>http://www.myclassservice.com/MyClass-interface</name>

<description xml:lang=‖en‖>
Service interface definition for our demo Service.
</description>

<overviewDoc>
<description xml:lang=‖en‖>
WSDL Service Interface Document
</description>
```

UNIT – IV

IMPLEMENTING XML IN E–BUSINESS 9

B2B – B2C applications – Different types of B2B interaction – Components of E -Business XML systems – EBXML – RosettaNet – Applied XML in vertical industry – Web services for mobile devices.

B2B:
One of the main sources of customers for business is other businesses. This is comparably a large market than selling directly to end users. ƒ This may involve negotiating prices, sales terms, credit, delivery and product specifications.

B2C Applications:
Many of the early developments on the Internet were focused on helping businesses directly sell goods to their customers. This is Business to Consumer (B2C). The Promise of B2C is avoidance of any middleman Other than increased complexity and potential cost in dealing with customers on a direct basis using B2C, a company implementation of B2C techniques faces danger of channel conflict or disintermediation.

Different types of B2B interaction:
Roles that the parties play in B2B E-Commerce (4 main types)
ƒ Buyers
ƒ Suppliers
ƒ Market Places [third party organization]
ƒ Service Providers [third party organization]
ƒ Forms of Supply Chain Relationship
ƒ Direct Partnership
ƒ Multi-party Procurement
ƒ Agents and Distributors
ƒ Exchanges , Auctions and Digital Transaction Hubs

Components of e-business XML systems:
E-Business systems are not monolithic structures Components are as below Enterprise and back-end integration Various network and foundational layers Messaging in a transport, routing and packaging context Registries and / or repositories, Data Dictionaries, Process and workflow,Trading partner Agreements, Business Vocabularies.

EbXML:
EbXML means Electronic Business XML. Global Standard for electronic business ebXML enables anyone, anywhere to do business with anyone else over the Internet Specifically designed to support SME ƒ Complementary to existing B2B initiatives (UDDI, RosettaNet, TradeXchange, etc.)

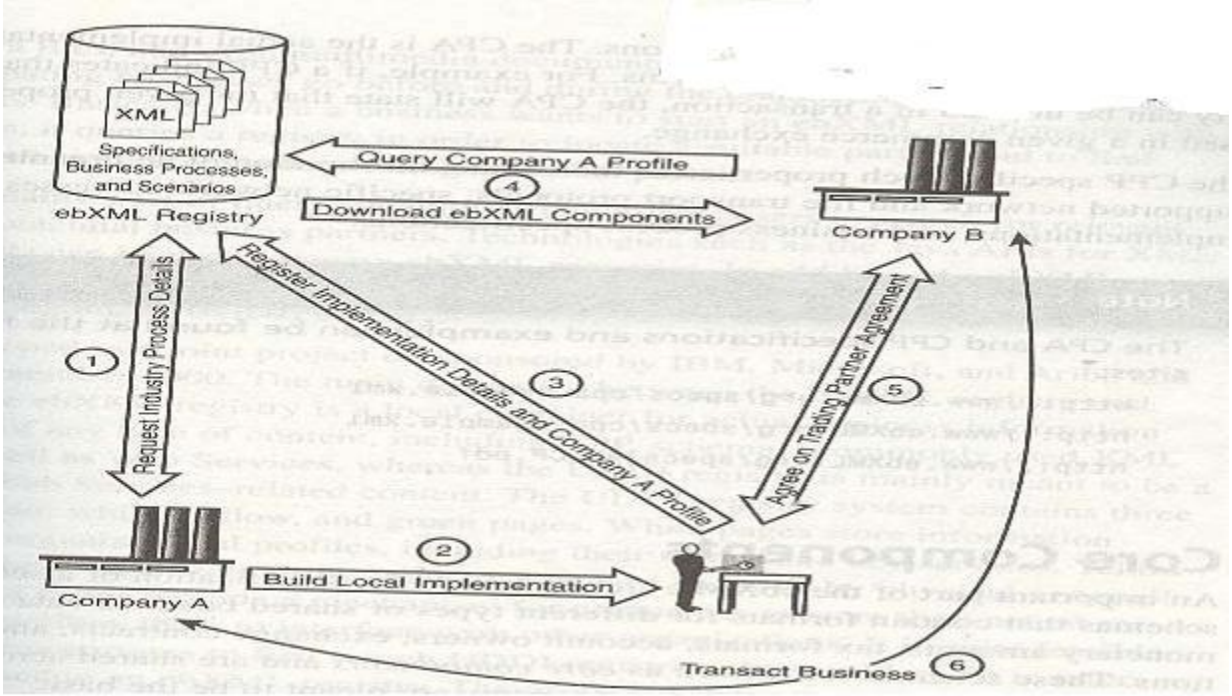
An end-to-end B2B XML Framework Components:
- Registry and Repository
- ƒ Core Components
- ƒ ebXML Specification Schema
- ƒ Business Process Model
- ƒ Information Model
- ƒ CPP/CPA
- ƒ Message Service

ebXML:

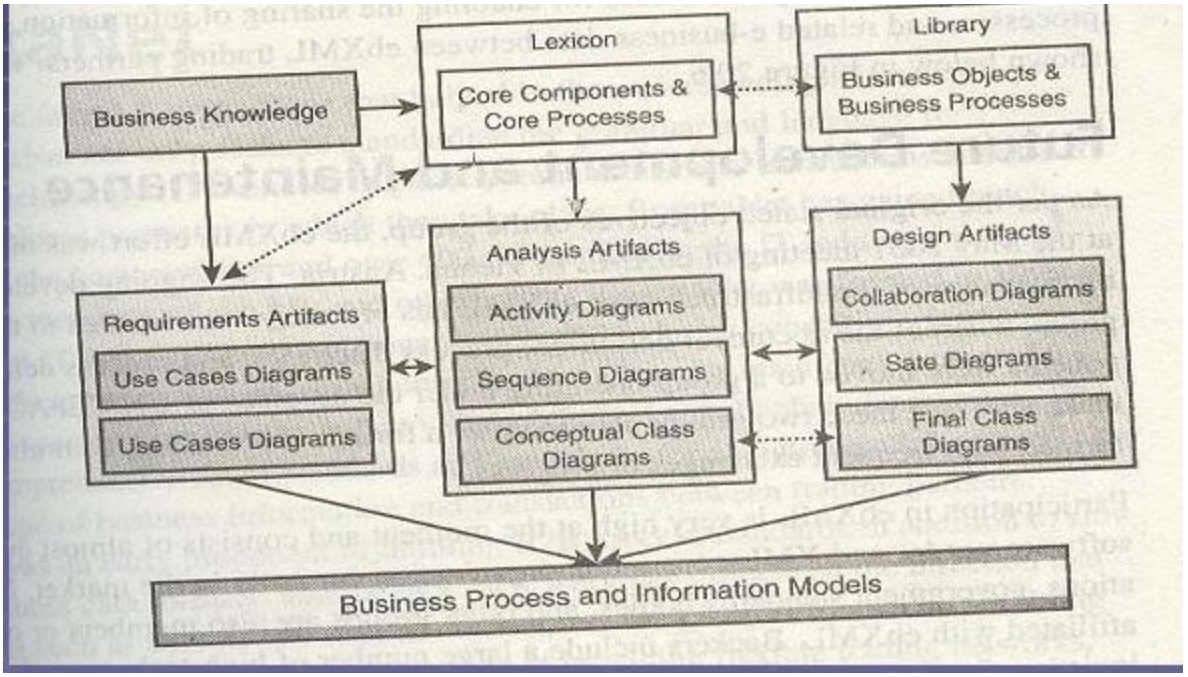
Infrastructure Components

- *f* Collaborative Protocol Profile (cpp)
- *f* Core Components
- *f* Registry and Repository
- *f* Messaging
- *f* Business Process Modelling

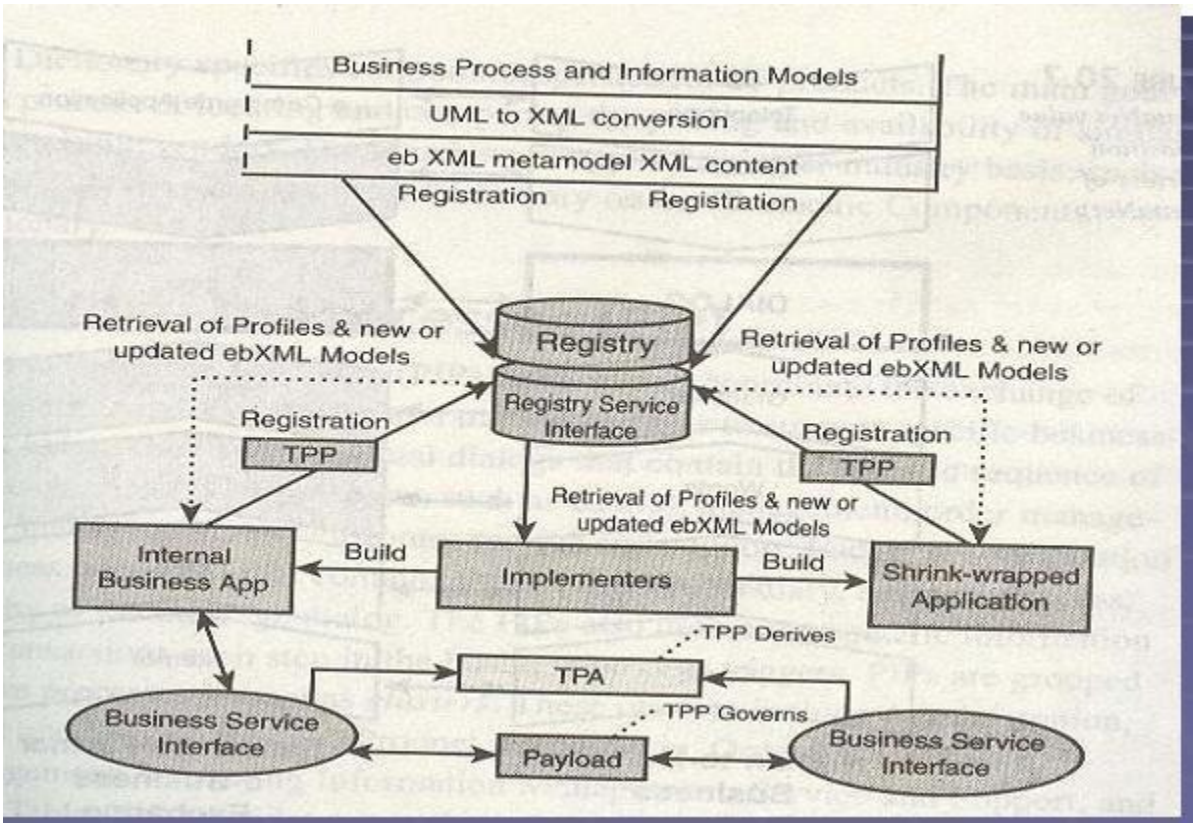
ebXML Process:



ebXML Architecture:



Functional Service View of ebXML:



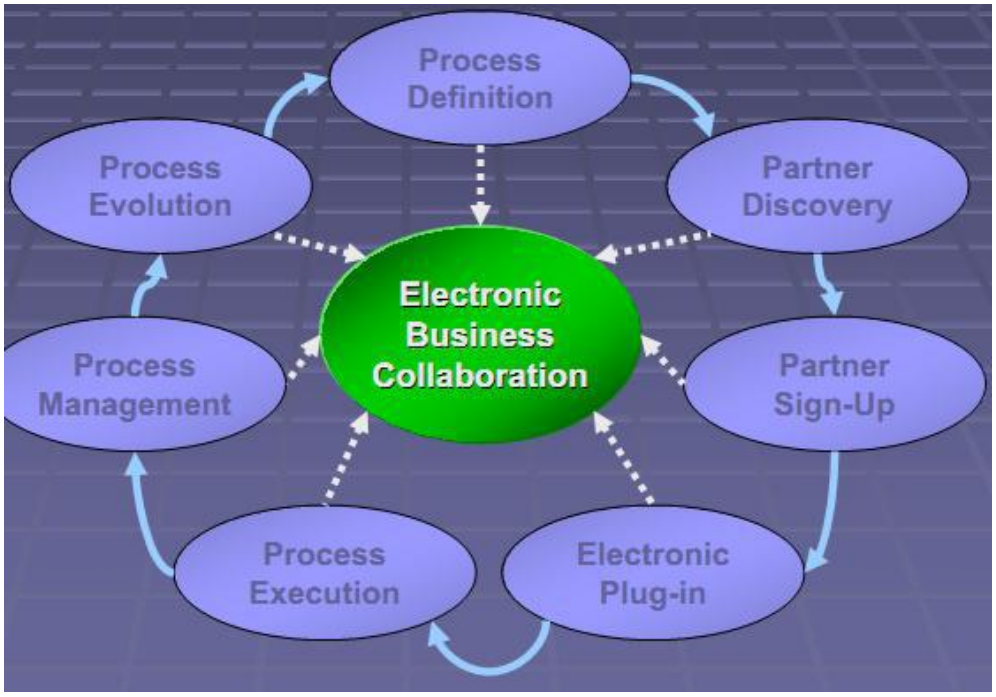
ebXML Vision:

A global electronic market place where enterprises of any size, anywhere can find each other electronically conduct business using XML messages. According to standard business process sequences with clear business semantics according to standard or mutually agreed trading partner protocol agreements using off the shelf purchased business applications.

B2B Collaboration:

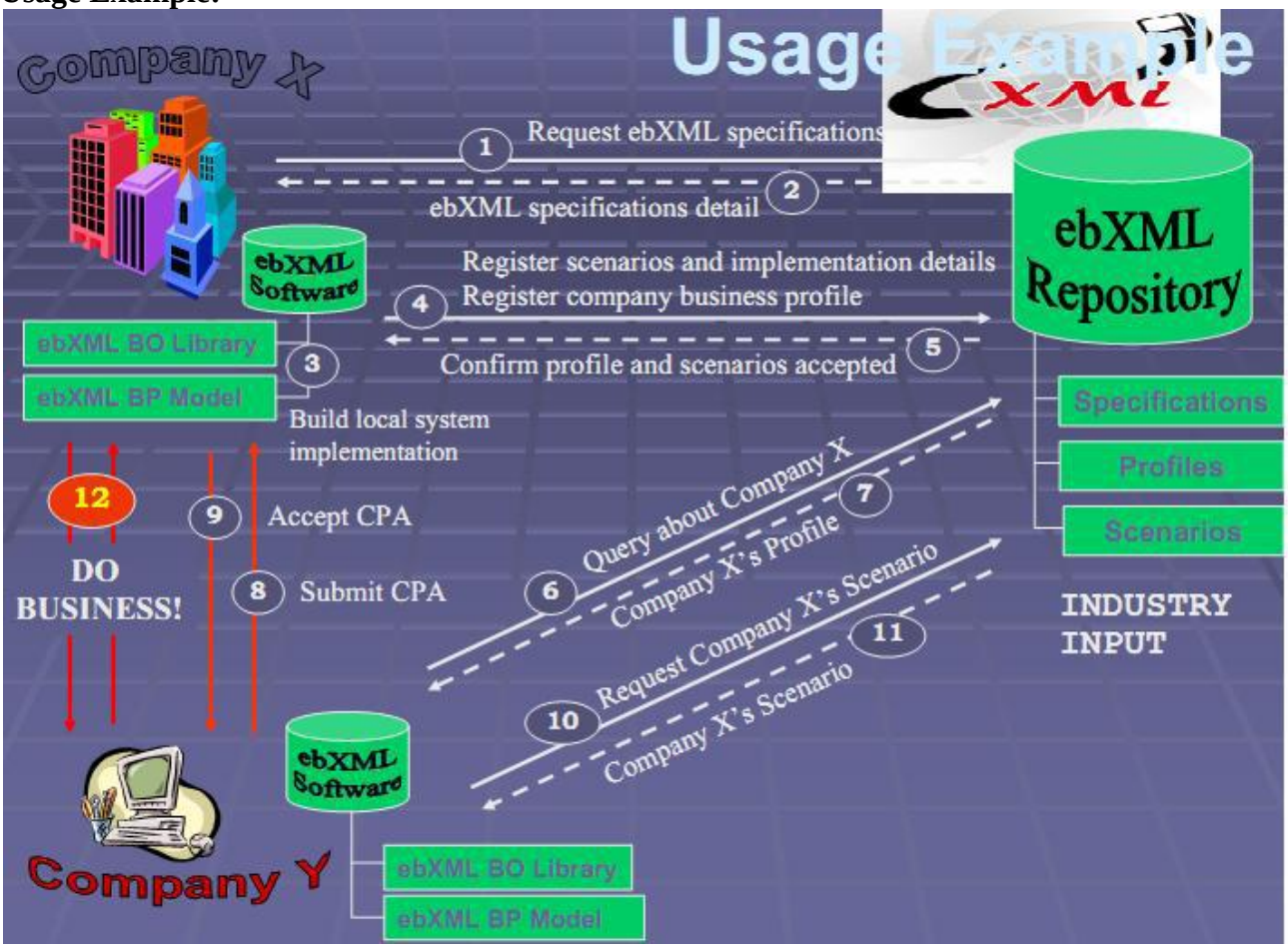
B2B collaboration requires more than just an XML protocol and a service registry. You have to deal with Business semantics, Negotiating terms and conditions, Interoperability, Security and Privacy Reliability.ebXML provides concrete specifications to enable dynamic B2Bcollaborations

B2B Collaboration Process:



ebXML Architecture:

Usage Example:



Collaboration Protocol:

Collaboration Protocol Profile is defined using ebXML Specification Schema Concrete specification of your ebusiness offerings Business scenarios you support Service interfaces you implement Document formats exchanged Technical requirements/options (protocols, security, reliability) Composed of Business process models Information models Context rules.

Business Scenarios:

It is Often defined by Industry Groups as Standard business scenarios remove the need for prior agreements among trading partners,
It is a Business Process Model, Interactions between parties,Sequencing of interactions, Documents exchanged in each interaction.
It is a Information Model that is Document definition, Context definition, Context rules
Core Components

Core Components:

Reusable low Reusable low- -level data structures.

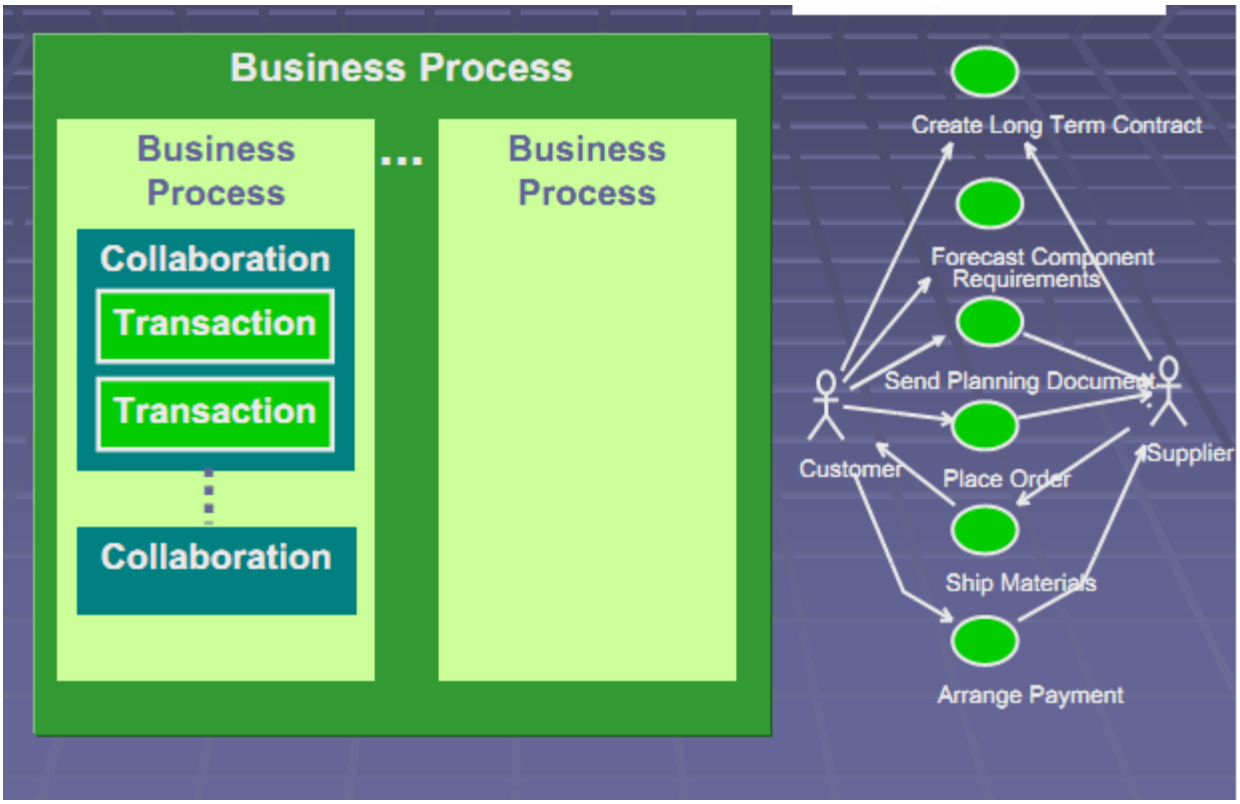
- e.g., party, address, phone, date, currency
- It is Context –sensitive

It is Single, consistent lexicon. It is Used to define business process and information models, Facilitates interoperability between disparate systems.

Context Affects Process:

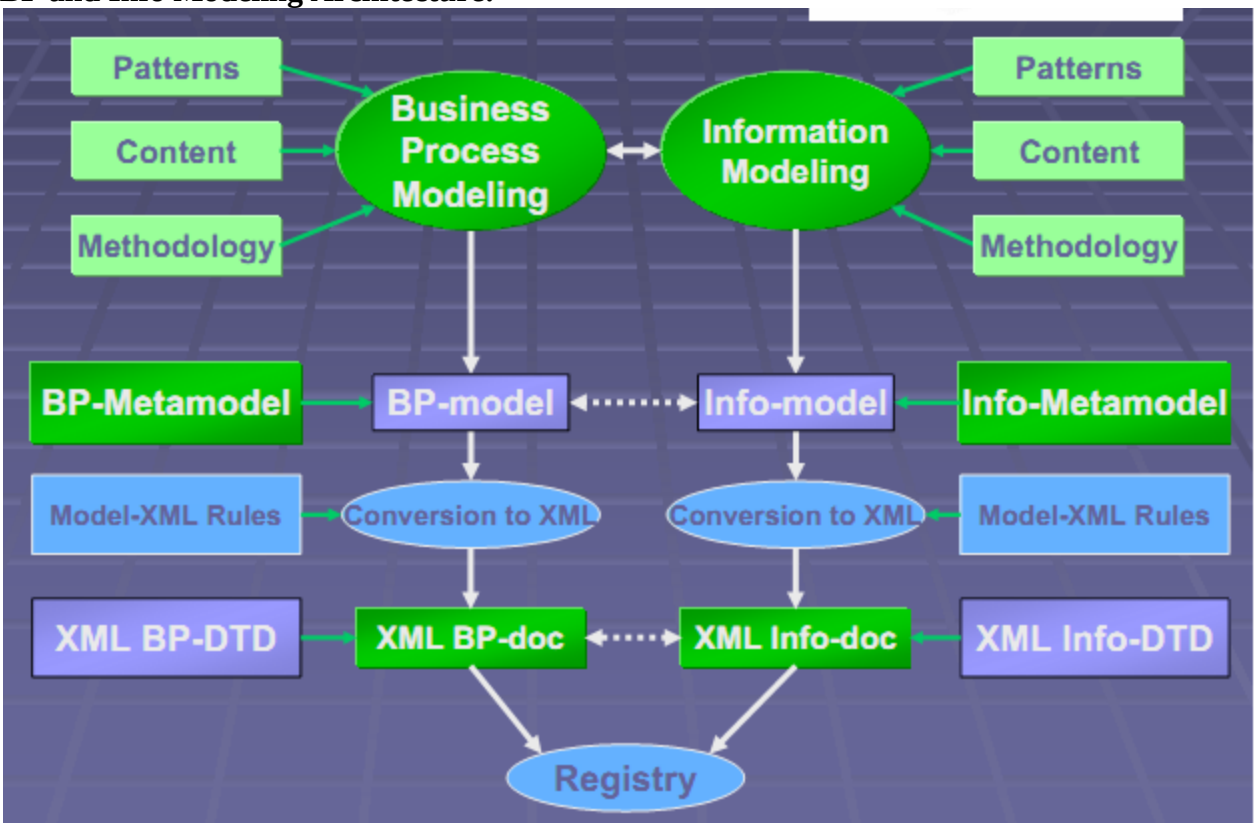
- Industry Sector Industry Sector
- Product Product
- Business process Business process
- Geo- -political region
- Official constraints
  - LegislativeStandards
  - Good practice
  - Contractual

Business Process:



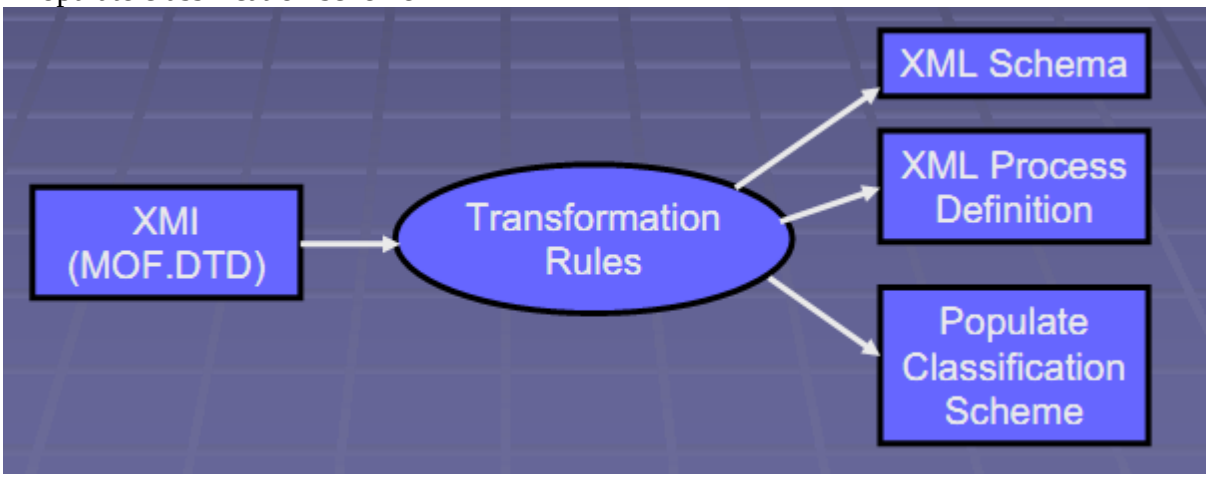
ebXML Specification Schema:

BP and Info Modeling Architecture:



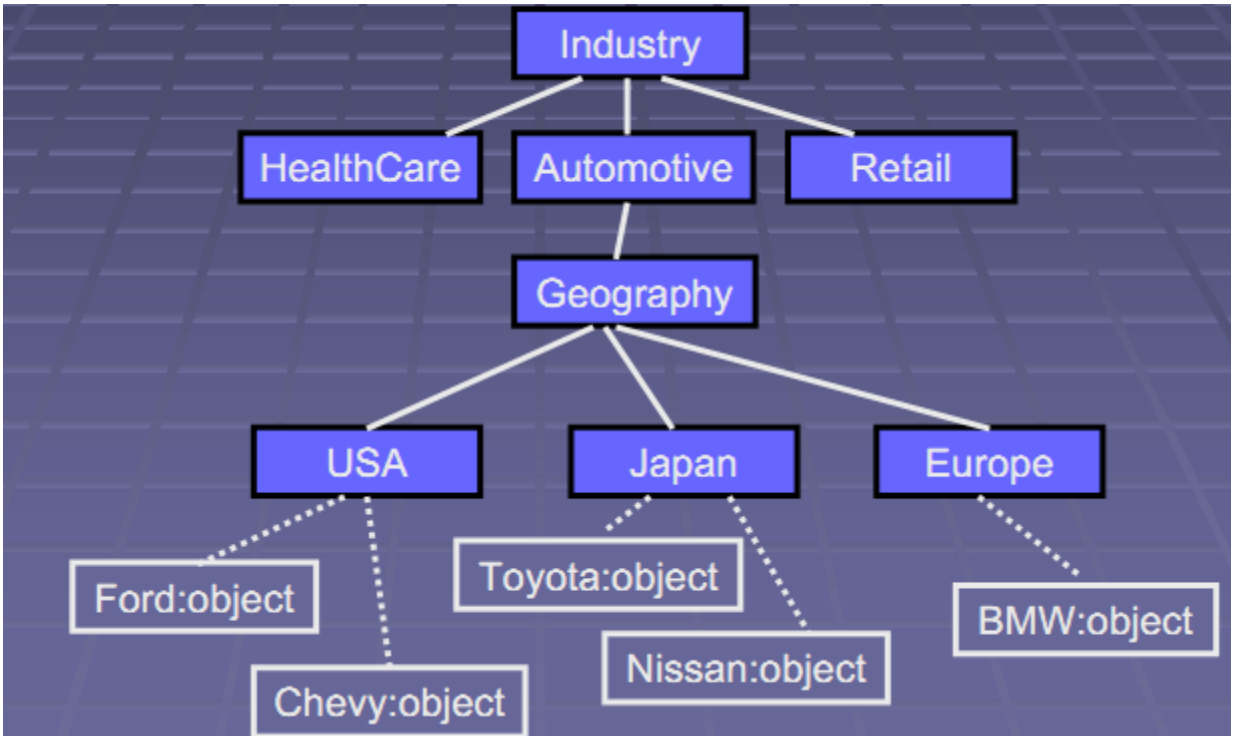
Conversion of UML Models to XML:

- Business process definitions Business process definitions
- XML Schema and DTD generation XML Schema and DTD generation
- Populate classification scheme



Traditional Classification Scheme:

Taxonomies:



Registering Your Business:

Register your business in an ebXML Registry means Index to all information in the repository and Rich query facility.It Store specifications in an ebXML Repository,

- CPP
- Schemas Schemas
- Process models Process models
- Core components Core components
- Classification and categorization schemes Classification and categorization schemes
- Arbitrary objects and code

ebXML Reg/Rep:

• ebXML Registry and Repository
– Registry = index of things
– Repository = holder of things
• Distributed model
• Nodes maintained by
– Industry groups
– Market places
– Exchanges
– Communities
– Individual companies

Negotiating an Agreement:

• Find registry and search for partners
• Examine CPP
• Ascertain compatibility of business process and technical specifications
• Stipulate your ―rules of engagement‖
• Produce Collaboration Protocol Agreement
– Conditions under which two partners will conduct business transactions together

CP Agreement Formation:

- Negotiate two Cooperative Protocol Profiles
- Party 1 queries and discovers Party 2
- Party 1 proposes rules of engagement
- Sends CPA to Party 2 for review and acceptance

Collaborative Protocol Agreement:

- Agreement for business interaction between two parties

– Technical specifications:

- Message Service requirements
- Application requirements

– References:

- CPPs
- Legal terms and conditions

Business Service Interface:

- Implements the CPA, supporting dynamic integration
- Not yet specified

– Hand-crafted for the moment

- Enables one Party to converse with the other Party using the ebXML Message Service

ebXML Message Service:

- Reliable, secure XML messaging service

– Enforces the rules of engagement in CPA

- Transport independent
- Extends SOAP Messages with Attachments (SwA)

– Reliability framework
– Security framework
– Manifest, trace, and delivery options

Delivery Options:

- Communications models

– Synchronous or asynchronous
– Request/response
– Fire and forget
– Multipart message delivery

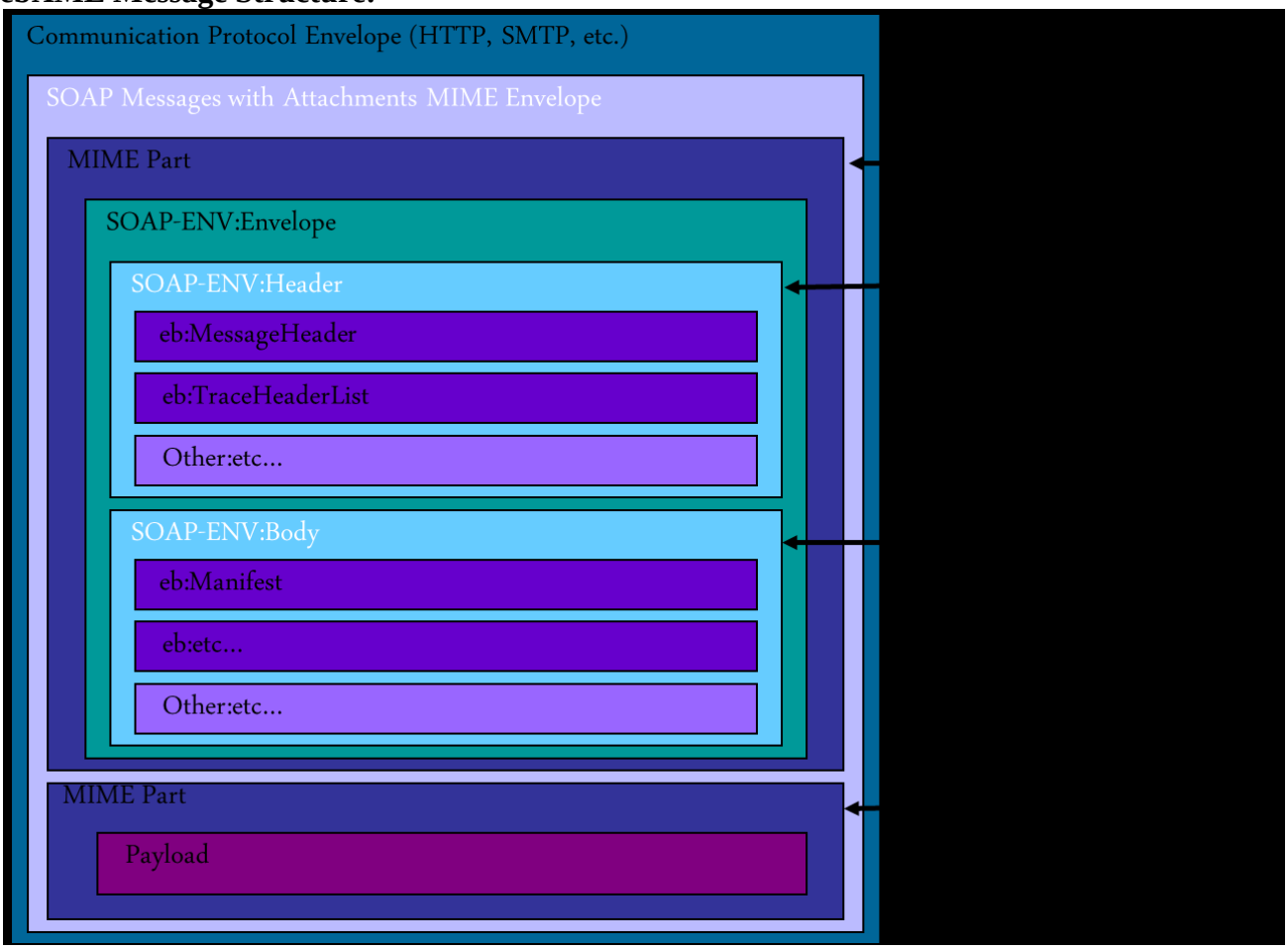
- Reliability options:

– Best effort
– Once and only once

Security:

- Identification
- Authentication
- Authorization
- Privacy
- Integrity
- Non-repudiation
- Logging

ebXML Message Structure:-



Web Services on Mobile Devices

Corporate information on handheld devices is becoming more necessary than ever. Learn about the opportunities and challenges in developing Web services applications on mobile devices. According to industry reports, existing host systems still hold 70 percent of mission-critical data and most of the pivotal business logic that runs worldwide commerce, and will continue to be a foundation for success for most organizations. An organization's ability to leverage these legacy systems in combination with newer applications for improved speed and power, therefore, will increasingly become a critical success factor in designing and implementing new business initiatives. In building mobile applications for the enterprise, integration and the corresponding challenges are as central to its design as pure application development. Because mobile applications are created simply to enable a mobile workforce to continuously access corporate business processes and information in real time, they are very rarely standalone applications in their own right. Rather, they apply the existing business logic of these host applications (to ensure data integrity, security, and so forth) as an extension and not re-create this logic in new systems. To provide a complete view of a data set to the user, many mobile enterprise applications must access multiple existing business systems. For example, to supply the sales team with a complete profile of the customer, the mobile application may need to connect to the corporate SAP system for sales information, the Siebel system for customer care issues and a mainframe for customer records. To design mobile enterprise solutions that execute such information accessand integration efficiently, the developer must consider realistically the resource limitations (limited real estate, data bandwidth, and so on) and create the applications to only provide the specific data required for the task and not try to be general purpose. Mobile integration follows the same guiding principles and tenets as do any other integration issues: service-oriented architecture (SOA) for integration and XML and Web services for the delivery of integration. With the implementation of XML and Web services, SOA can

decouple the end devices and their operating environments from the integration of mobile services with corporate applications. Industry experts today agree that the adoption of mobile middleware best deliver corporate business processes and applications to mobile devices, by enabling the combination of off-the-shelf integrated application adapters with process management capabilities to define and coordinate transactions across multiple back-end corporate applications.

Mobile Computing: Background and Key Findings

The sales of laptop, notebook, and tablet type devices have grown tremendously over the last few years as these devices have become essential tools for many different types of mobile workers, from sales people to service technicians, who need to access and enter data electronically. In recent years, the market for small handheld computers such as Personal Digital Assistants (PDAs) has been evolving even more rapidly than the market for conventional PCs and laptops, and trends are pointing toward exploiting mobile technology even more. With this rapidly increasing popularity, these modern devices are often fully equipped with extensive capabilities. Most of these devices come with more than one wireless data communications options including Bluetooth, Wi-Fi, General Packet Radio Service (GPRS) and other 3G network technologies such as EDGE. They also provide Browser capabilities, most of them beyond simple Wireless Application Protocol (WAP) to support standard HTML-based Web sites. These mobile devices allow new opportunities to develop solutions beyond mobile e-mail with development frameworks, such as Linux, J2ME, and the Microsoft .NET Compact Framework. The virtually limitless functionality of mobile devices and the availability of literally thousands of Wi-Fi hotspots can lead to the reality of delivering the mobile infrastructure needed by the enterprise, much further beyond just mobile e-mail. Mobile Internet pioneers have learned some important lessons: 1. Discretionary selection of applications that best cater to the needs of the mobile worker and match the screen space and speed constraints of today's mobile devices is crucial in creating a mobile enterprise infrastructure. This may include a combination of applications that provide on-the-spot order entry, sales force automation, customer lookups, and equipment service. 2. Mobile Internet technology is evolving very rapidly. Handsets, network speeds, interoperability standards, and protocols are all becoming developed at an increasing rate as mobile phone manufacturers, network infrastructure players, and mobile network providers are investing massive amounts of resources into creating the next-generation networks. 3. Mobile network technology is not geographically bound, GSM-specific ,or tied to any transport network or technology but blankets over local differences and enable worldwide deployment of mobile applications.

WAP-Enabling Corporate Information

A mobile phone with a specialized piece of software called a WAP browser (also called a micro-browser) used to be considered "Internet capable." Although the capabilities of the phone and its form factor have changed significantly in the past years, many cellular providers still rely on the WAP browser to offer Internet services. The Wireless Application Protocol (WAP) is a suite of standard protocols that interact to display data on a mobile phone screen, and consists of two major parts: □ □ The transport protocols define the transmission of the information into the phone, managing the over-the-air links, error detection and recovery, and session management. □ □ The presentation protocols dictate the human/machine interface—how information is displayed usefully on the small screen of a mobile phone and how the phone keypad is used to enter information and interact with applications. The WAP browser accepts a "card deck" in which each card can be thought of as a tiny Web page. However, instead of HTML (the language used to describe Web pages on the Internet), another language called WML (Wireless Markup Language) is used to describe the layout and behavior of each card in the deck. Users navigate through a WAP application by selecting from a menu of choices, and often the entire card deck that is downloaded into the phone and contains an entire menu structure, all limited to the local structure of the mobile phone and not communicating over the wireless link. Although it is called a browser and it can be useful to think of a WAP card as a "minimalist Web page," the screen and keypad of a mobile phone present very different challenges for the application designer than do the rich output and input capabilities of

a modern Web browser. And existing Web sites—with their frames and graphical navigation aids—cannot be translated easily into the menu-oriented approach of the WAP browser. At first glance, the easiest way to get corporate information deployed to mobile users is just to take existing application screens or Web pages and push them out to the WAP browser. In fact, mobile server products often offer the ability to do "on the fly" publishing—take existing text files or Web pages and boil down the text to appear on one or more WAP screens. The conversion process is mechanical and many Web pages with frames and complicated formatting do not easily translate to WAP applications. The results can range from passable to comical to awful, but almost certainly do not make the best use of the capabilities of the mobile browser. At the other end of the simplicity spectrum, well-designed and highly efficient WAP applications that replicate corporate applications can be custom built. This people-intensive
process is expensive and the ongoing cost to maintain both the standard and WAP (and possibly HTML) versions of the application is high.

The Old WAP vs. the New Web

Mobile technology has progressed to the next set of mobile
devices with much more extended capabilities than a simple micro-browser. With various types of underlying technology (Palm, Microsoft Windows Mobile, Symbian, or Java), the display capabilities are greatly enhanced from the simple monochrome WAP micro-browser with small text-only screens. Additionally, user interaction now ranges from touch screens to thumb-operated keypads, with fully capable Web browsing capabilities and support for many of the standards found on a desktop or laptop browser, such as Internet Explorer, Mozilla Firefox, or Opera. The underlying structure of Web-based support on mobile devices is very similar to that of a desktop or a laptop computer. Web-based support relies on the same standards as does the Internet: □ □ IP, the underlying communications technology □ □ the Internet, the underlying network The end-user application is a Web browser that can produce very complex, graphical, and highly interactive interfaces from data delivered in HTML and XML or their mobile equivalents, therefore bringing the mobile corporate enterprise one step closer to reality through the mobile Web. However, these mobile browsers still do not provide the full functionality of a desktop Web browser; this presents some limitations on the full Web experience. To make such mobile extensions of corporate applications, it is crucial to have products that provide a comprehensive development and deployment platform and also support the creation of new corporate applications and transactions that can be Web-enabled for PDAs and Smartphones. These products also should enable the extraction and *re-presentation* of transactions from a corporate application to create and deliver real-time mobile enterprise applications.

The Next Step: Mobile Applications

The expansion of mobile devices and their built-in capabilities, particularly toward programmable operating systems, has led to the growth of the possible range of new applications to be built and added to them. Windows Mobile devices provide full support for the Microsoft programming environment; the Compact .NET Framework is implemented on PDAs, Phone Edition PDAs, and Smartphones. Support for J2ME on Palm devices and many phones based onhe Symbian operating system (such as Nokia and Sony-Ericsson) allow Java programs to be written for and run on even generic cell phones. These applications can be delivered and installed to the mobile device automatically and wirelessly through a capability called Over-The-Air provisioning (OTA). With these new, diverse options for developing applications, organizations have new opportunities for corporate use of mobile devices and integration with corporate applications. With Microsoft .NET Compact Framework (.NET CF), it is possible to develop applications to run on the mobile device thats interact seamlessly with corporate assets over GPRS or Wi-Fi networks. For example, a GPRS wireless PDA application can be built to interact with a mobile middleware solution, which in turn interacts with the mainframe legacy system or contemporary systems such as SAP. Let's look at an example: A mobile application on a wireless PDA would allow a salesperson on the road to access customer contact information. Through an application that runs locally on the PDA, this salesperson

can tap the PDA's touch-sensitive screen and enter the name of the customer or the company to retrieve contact details. This would trigger the application running on the PDA to communicate with a mobile middleware server through the GPRS network, which then would wirelessly connect to an IBM iSeries system that runs the customer database. Then, in a reverse path, information from the iSeries system would be gathered and passed back completely transparently by the middleware server to the PDA application, seeming to the salesperson as though the whole process has run locally on the PDA—the salesperson has simply tapped the screen to access the needed information. The key standard delivering this transparency for future mobile applications will be Web services.

Looking Ahead: Mobile Web Services

Web services, lying within the broader world of SOA, will be the driver for the new mobile economy. The underlying principles of SOA are not new; existing client-server systems, such as DCE, CORBA and DCOM, have delivered distributed applications for quite some time. Unlike these client-server systems however, SOA allows information providers to be developed entirely independently from information consumers through intercommunications standards that are manifested through Web services. The information providers—legacy IBM mainframe applications, the more modern contemporary applications like SAP, or a combination thereof—that span various platforms will remain the core drivers of business in the enterprise. As an integration layer is needed between the "old" applications and the "new" Internet-enabled mobile economy for standardized information integration, the challenge then becomes Web-enabling or mobile-enabling existing corporate assets to transform the transactions lying in the disparate systems into services that can be consumed by other applications and systems. The information consumers in the mobile economy are the mobile devices. These mobile devices are no longer simple gadgets for presenting static information such as WAP card decks or Web pages, but they are fully programmable interactive systems with applications built for them by conventional development environments for desktop and laptop computers like Microsoft Visual Studio. In today's world, applications for mobile devices can consume Web services and work easily with any corporate systems.

Moving Into the Future: Mobile Middleware Platforms

Selecting a mobile middleware platform that provides real-time access to backend business systems and the tools for transforming specific business functions from those systems into Web services is critical in building enterprise mobile solutions. This platform must offer a wide array of business systems connectors out of the box and connect to legacy systems because this is a core business platform for most large organizations. Accomplishing this with the limitations of the screen size and reduced bandwidth on mobile devices requires the mobile solutions to choose and only expose the functions of backend business systems critical to the application's use. Many of the current mobile middleware solutions available today have roots in database synchronization with infrastructures based around this technology; however, unstructured data (the majority of business information) by its nature does not lend itself to a database-centric solution. As a mobile middleware platform should not require the developer to create new logic but build an extension of the existing logic, a new approach is required to deliver true real-time enterprise mobility.

QUESTION BANK

PART – A(2 MARKS)

1. What is Web Services?
2. What are the things available in Web Services?
3. What Qualifies as Web Services?
4. List out the advantage of Web services technology?
5. What are the major aspects of Web service technologies?
6. List out the key technologies.
7. What is UDDI?
8. What are Web Services Registry Directories?
9. What is WSDL?
10. What are the risks in Web Services?
11. What is ebXML?
12. What is Transaction?
13. What is Identity?
14. What is passport?
15. What is .NET?
16. What are components of .NET architecture?
17. What are the components of .NET Platform?
18. What are the key ingredients of the .NET Framework?
19. What is J2EE?
20. What is Sun ONE?
21. What is ECLIPSE?
22. What is BEA?
23. What are the components of BEA Web Logic E-Business Platform?
24. List out the different type of Adapters?
25. What is Oracle?
26. List out the Web Service Pack in J2EE?

PART – B (16 Marks)

1. Explain about Web Services Technologies? (16)
2. Explain about ebXML? (16)
3. Explain about Soap, Web Services and E-Commerce? (16)
4. Explain about .NET? (16)

UNIT V

XML CONTENT MANAGEMENT AND SECURITY 9

Semantic web – Role of meta data in web content – Resource description framework RDF schema – Architecture of semantic web – Content management workflow – XLANG – WSFL – Securing web services

Semantic Web:

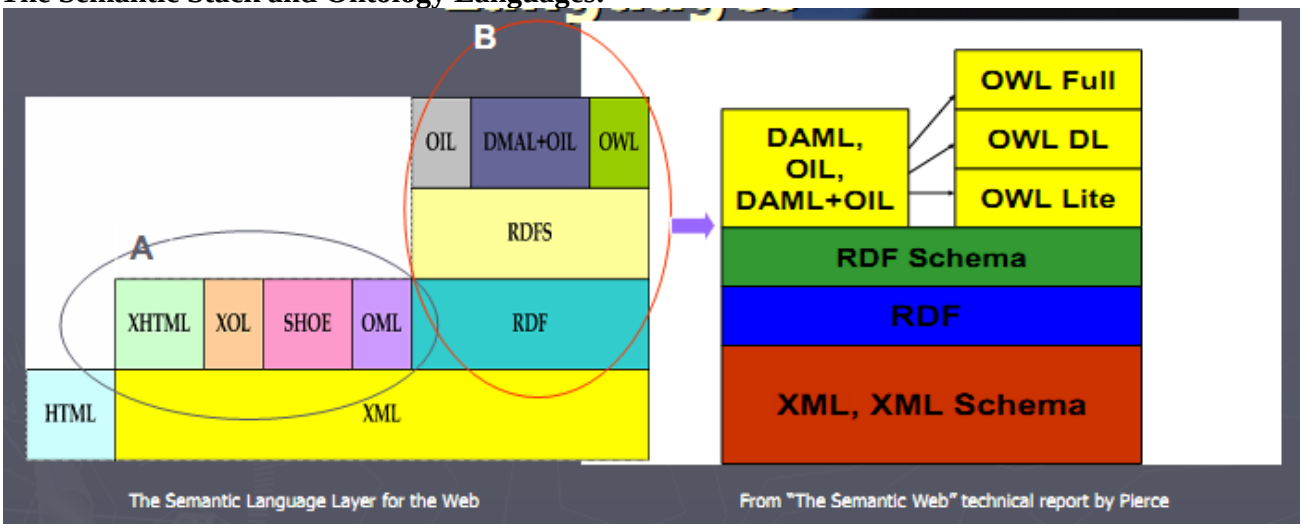
―The Semantic Web is a major research initiative of the World Wide Web Consortium (W3C) to create a metadata-rich Web of resources that can describe themselves not only by how they should be displayed (HTML) or syntactically (XML), but also by the meaning of the metadata.‖ – From W3C Semantic Web Activity Page

―The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.‖ –Tim Berners-Lee, James Hendler, Ora Lassila,The Semantic Web, Scientific American, May 2001

Motivations:

Difficulties to find, present, access, or maintain available electronic information on the web. Need for a data representation to enable software products (agents) to provide intelligent access to heterogeneous and distributed information.

The Semantic Stack and Ontology Languages:



RDF [Resource Description Framework]:

Express meaning on the web that machines can understand.

W3 Specification for Xml code:

<person name=―Jane‖>

<sells product=—books/>
</person>
Rdf code
<rdf:Description about=—[Jane] xmlns:my=—[my]>
<my:sells
rdf:resource=—[books]
/>

RDF consists of two parts
– RDF Model (a set of triples)
– RDF Syntax (different XML serialization syntaxes)

RDF Schema for definition of Vocabularies (simple Ontologies) for RDF (and in RDF)

RDF Data Model:

Resources:
– A resource is a thing you talk about (can reference)
– Resources have URI‘s
– RDF definitions are itself Resources (linkage)

Properties:

- slots, defines relationship to other resources or atomic values

Statements:

—Resource has Property with Value
– (Values can be resources or atomic XML data)

- Similar to Frame Systems

A simple Example:

Statement:
-Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila

Structure:
– Resource (subject) http://www.w3.org/Home/Lassila
– Property (predicate) http://www.schema.org/ - Creator://www.schema.org/#Creator
– Value (object) —Ora Lassila

Directed graph:

http://www.w3.org/Home/Lassila

OraLassila

Collection Containers:

Multiple occurrences of the same Property Type doesn‘t establish a relation between the values
– The Millers own a boat, a bike, and a TV set
– The Millers need (a car or a truck)
– (Sarah and Bob) bought a new car

RDF defines three special Resources:

– Bag unordered values rdf:Bag
– Sequence ordered values rdf:Seq
– Alternative single value rdf:Alt

• Core RDF does not enforce _set‘ semantics amongst values

A Formal Model of RDF:

RDF itself is mathematically straightforward:
– Basic Definitions
• Resources.
• Properties Ì Resources called
• Literals
• Statements = Resources ´ Properties

X{Resources È literals}
– Typing
• rdf:type Î Properties

• {RDF:type, sub**, obj}** Î **Statements** Þ **obj**

Î **Resources**

Formal Model of RDF II:

Reification

- rdf:Statement Resource-Properties
- □ rdf:predicate, rdf:subject, rdf:object □ □ Properties

- Reification of a triple □ pred, sub, obj □ of Statements

triple and the elements s1, s2, s3, and s4 of
Statements such that

– s1: {RDF:predicate, r, pred}

– s2: {RDF:subject, r, subj}

– s3: {RDF:object, r, obj}

– s4: {RDF:type, r, [RDF:Statement]}

– Collections

□ RDF:Seq, RDF:Bag, and RDF:Alt □ □ Resources-Properties

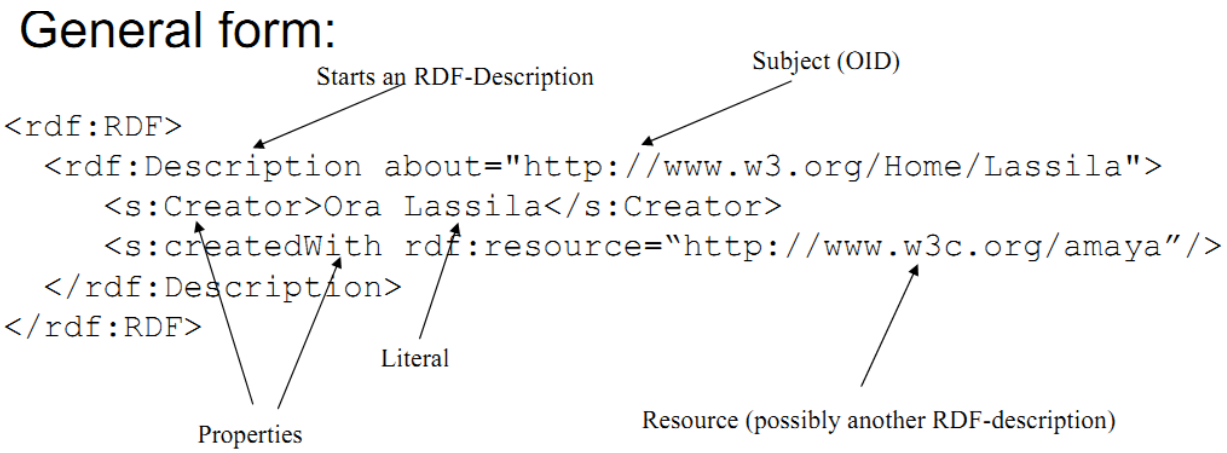
• There is a subset of Properties corresponding to the ordinals (1, 2, 3, …) called Ord. We refer to

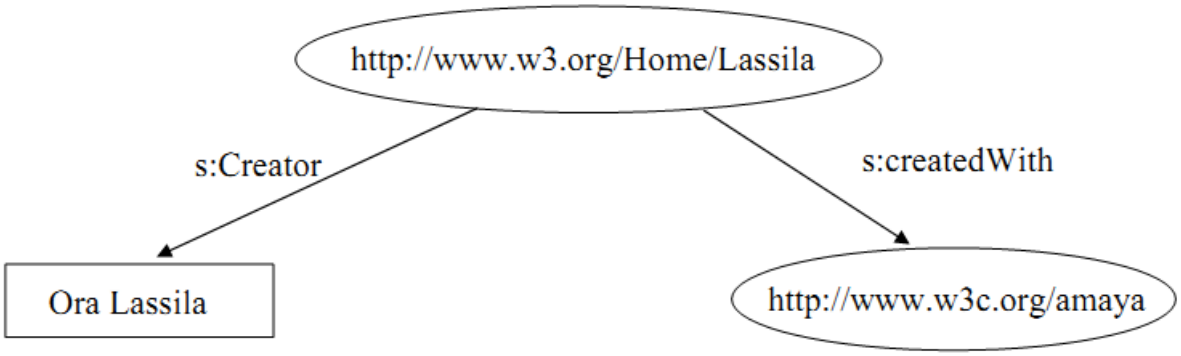
• elements of Ord as RDF:_1, RDF:_2, RDF:_3, …

RDF Syntax I:

Data model does not enforce particular syntax

- Specification suggests many different syntaxes based on XML

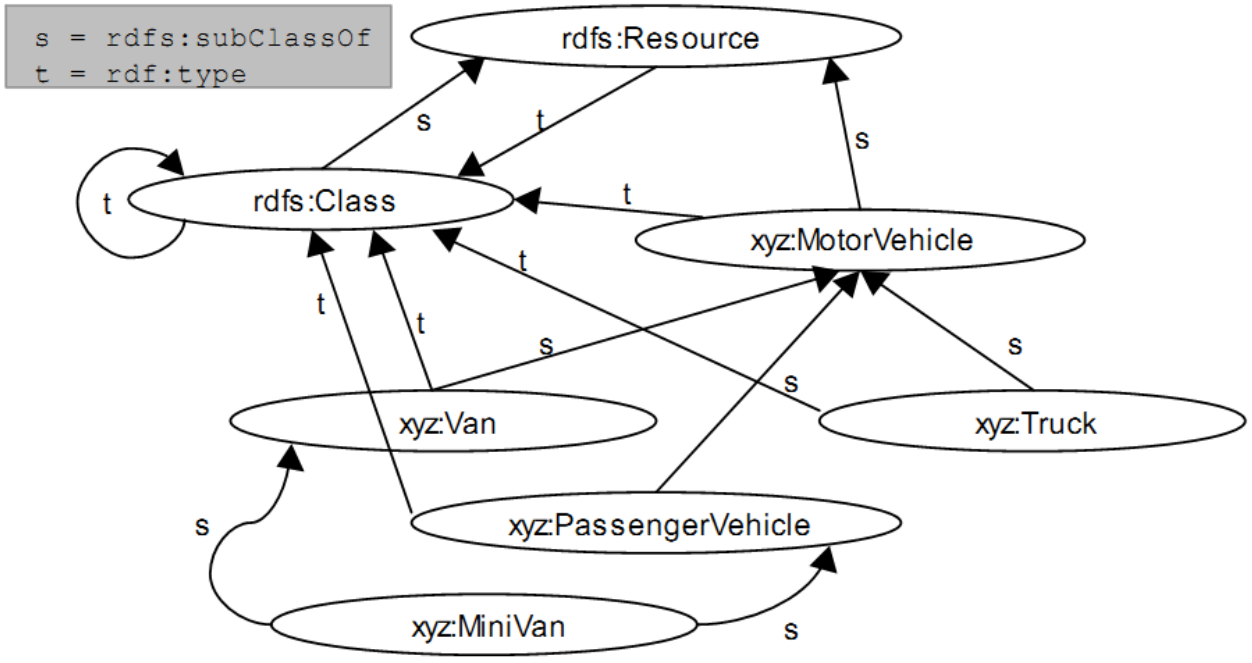




Resulting Graph:

<rdf:RDF>
<rdf:Description about=‖http://www.w3.org/Home/Lassila‖>
<s:Creator>Ora Lassila</s:Creator>
<s:createdWith rdf:resource=—http://www.w3c.org/amaya‖/>
</rdf:Description>
</rdf:RDF>

RDF Schema (RDFS):



RDF just defines the data model

- Need for definition of vocabularies for the data model – an Ontology Language!
- RDF schemas are Web resources (and have URIs) and can be described using RDF

RDF-Schema: Example:

Rdfs:subclassOf:

<rdfs:description about=„Xyz:Minivan—>
<rdfs:subclassOf about=„xyz:Van—/>
</rdfs:description>
<rdfs:description about=„myvan—>
<rdf:type about=„xyz:MiniVan—/>
</rdfs:description>

Predicate Logic Consequences:

Forall X: type(X,MiniVan) -> type(X,Van).

Forall X: subclassOf(X,MiniVan) -> subclassOf(X,Van).

Rdf:property:

<rdf:description about=„possesses—>
<rdf:type about=„….property—/>
<rdfs:domain about=„person—/>
<rdfs:range about=„vehicle—/>
</rdf:description>
<rdf:description about=„peter—>
<possesses>petersminivan</possesses>
</rdf:description>

Predicate Logic Consequences:

Forall X,Y: possesses (X,Y) -> (type(X,person) &
type(Y,vehicle)).

Simplification rules:

- People specifying text for arbitrary RDF processors can use any simplification
- Processors of arbitrary RDF therefore must accept all simplifications
- Special-purpose XML formats can be RDF- compliant while disallowing simplifications,

  requiring them, or exploiting them in specific ways

Containers vs. Multiple values:

- A property can appear more than once with different values
- What is true of a container isn‘t necessarily true of its contents and vice versa
- —aboutEach‖ lets us get to the contents when we already have a container
- —aboutEachPrefix‖ in effect manufactures a container based on URLs

Reified statements:

- We reify statements so that we can talk about them rather than asserting them
- —Charles Dickens is the author of Bleak House‖ asserts a property of Charles Dickens
- —Jack believes that Charles Dickens is the author of War and Peace‖ asserts a property of Jack, not Charles Dickens

RDF Classes:

Are groups of Web resources

- Have URLs to identify them
- The special class ―rdfs:Literal‖ consists of all possible RDF string values

XLANG:

It is aExchange Language – Web Service for Business Process Design. Microsoft – runnable on all platforms. It Written format is XML. It is a Biz Talk Application Designer.

Motivation:

It Enabling technology for new standards and descriptions around existing technologies => develope language which is able to handle.

It is a Business imperative to ease the automation of business processes& to describe a standard for reuse by other tools.

Relationship with WSDL:

- Grounding of a XLANG service description is defined by using WSDL
- XLANG describes the behavior as part of a business process
- Actions in XLANG = operations of WSDL

Business Process (I):

- Contract between two ore more parties
- XLANG service description describes the behavior
- It is used a map – describes the connections between the ports
- Each port is added to the map

Correlation:

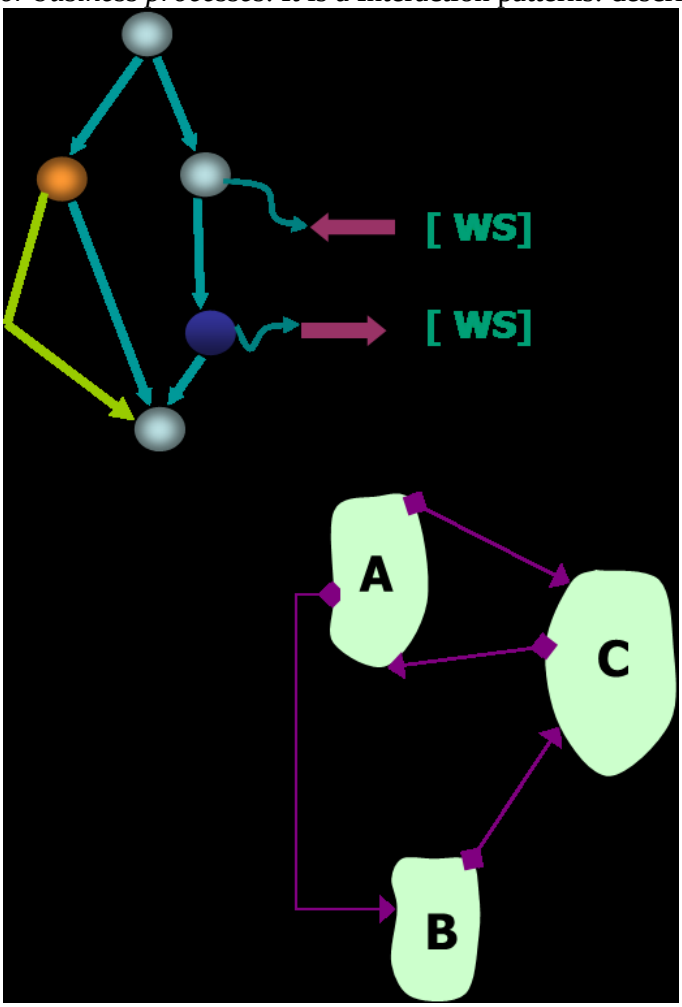
- Correlation tokens
- For automated instance routing
- Messages to the correct port and instance

☐ Instance: interaction with which the behavior of the service is represented

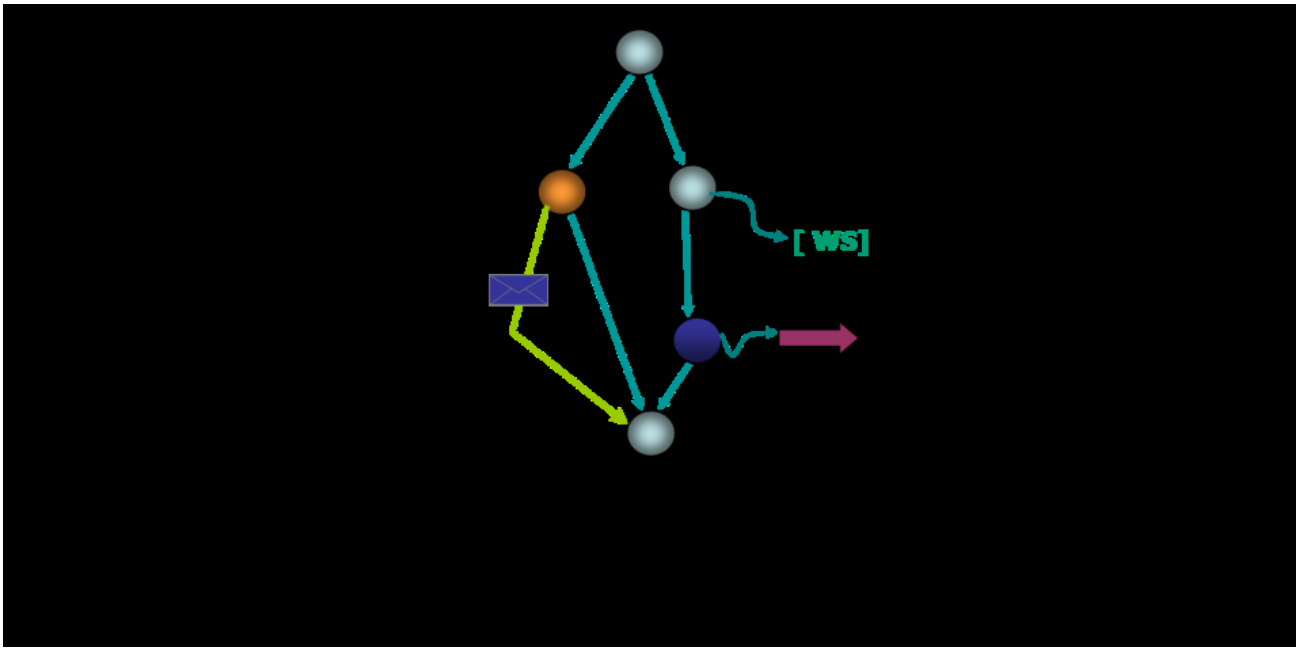
- Calculated lifetime
- Correlation set
- Correlation group

WSFL:

WSFL describes Web Service compositions. It is a Usage patterns of Web Services: describes *workflow or business processes*. It is a Interaction patterns: describes *overall partner interactions.*



WSFL Flow Models:



Using Flow Models:

• ―Public flows‖ provide a representation of the service behavior as required by its users.

– Typically, an abstraction of the actual flow begin executed

– Defines a ―behavioral contract‖ for the service.

– Internal implementation need not be flow-based.

– Flows are reusable: specify components types, but not what specific services should be used!

– ―Private flows‖ are the flows executed in practice.

– WSFL serves as a ―portable flow implementation language‖

• Same language is used in WSFL to represent both types of processes.

Global Models:

- Global models describe how the composed Web Services interact.
- RosettaNet automated.
- Like an ADL.
- Interactions are modeled as links between endpoints of two service interfaces (WSDL)
- An essentially distributed description of the interaction.

Discovery: Finding Meta-data:

• Static binding requires service ―libraries

• Dynamic binding requires runtime discovery of meta-data

Web Content Management:

• Every thing from price list to contact list, home page ,personal info are becoming a internet

• Volume of content on the internet continuous to grow in size & types of content.

• Requirements for Solid web content management solution & dynamic content distribution

Definition:

• Web Content Management is generally defined as a combination of clearly defines roles, formal process and supporting system architecture used to produce, collaborate on, maintain ,publish & distribute to the web.

Web Content:

• Any information or data on the web

• We must identify the types of content we need to manage & how content assets relate

• Role to manage the content

• Formal process to manage the workflow based on the roles.

• Web Content Management success is investing in information model & Design

Based on the Characteristics of content & Business model:

– Straight forward from data base to the web

– Automated, dynamic generation of a web site

– Managing text files

– Large volumes of multimedia based collection

– Newly created Content

– Legacy data

Components of Content Management Workflow:

• Content – Input phase

– Content Entry

– Content – Repository phase

– Store & Managed

– Content Delivery Phase

– Distributed to web

Workflow:

Creation:

• Created by the organization & imported into the content management system

• These contents can be created in any format-Word, HTML, SGML & XML

• Through directly from data base

• XML is the standard format

Database import:

• Data may come from the external database within same org or partner organization (ERP)

Legacy Inclusion

• Already Existing data. E.g.: journal

• In some other format – Conversion is required

Content Repository

• Store & Management

Storage:

• Database – index, relational (or) object oriented

• Database can be stored direction (or) pointing to external pointers

• Importing is the methodology

Revision Content – Check in/check out

– Portion of content can be changed depending on the role

– Roles are set for each and every author

– Locking

Version Control

– Complete body of the content is changed

Content Assembly

– Analysis of meta data about each contents & assemble them for final web delivery

Content – Delivery phase

• Common interface b/w the content & end user must be set

• XML – is delivery neutral form

Print rendition

– Print media

– XML – XSL – Content

Web Rendition

– Web output (XML – XSL – O/P)

WAP/Mobile Rendition

– WAP/WML

Context Syndication

– Syndication from single body of content to multiple end users

– Automate the process

– Example for web document creation – WAPDav

<?xml version=1.0>

<D:acl xmlns:D:DAV>
<D:ace>
<D:principal>
<D:href>www.xx.com</D:href>
<D:principal>
<D:grant>
<D:privilege>D:read</D:privilege>
<D:privilege>D:write</D:privilege>
</D:grant>
</D:ace>
</D:/acl>

QUESTION BANK

PART – A(2 MARKS)

1. What is XKMS?
2. Define XKMS structure?
3. Define X-KISS?
4. Define X-KRSS?
6. What are digital signatures?
7. What is single-key cryptography?
8. What is public key cryptography?
9. What are the xml security technologies?
10. Write the steps for XML Encryption?
11. What is OFX?
12. What is HR-XML?
13. What is OASIS?
14. Explain about SVG?
15. What is SMIL?
16. What are SMIL and its uses?
17. What are the advantages of SMIL?
18. List out the SMIL elements?
19. What are the three basic security requirements for e-business and explain?
20. What is data integrity?

PART – B (16 MARKS)

1. Explain briefly about the security? (16)
2. Explain briefly about i) XML Security Framework (16)
3. Explain briefly about i) XML Digital Signature (16)
4. Explain briefly about i) canonicalization (16)
5. Explain the applications of XML as occurring in three waves? (16)