e-Macao Course Assessment

Course: XML Technology and Java, Team E

Name (CAPITALS, English): _____

Agency (English): _____

Please select ALL correct answers for the following questions.

1	What is the latest version of XML?		tick
	a	1.0	
	b	1.1	x
	c	1.4	
	d	2.0	

2	Which of the following languages apply XML syntax?		tick
	a	DTD	
	b	XSLT	x
	c	PDF	
	d	SVG	x

3	How many bytes are used in UTF-8 to represent a single character?		tick
	a	1, if the character belongs to the Latin alphabet	x
	b	2, for all characters	
	c	between 1 and 2	
	d	between 1 and 4	x

4	Which character encodings must be supported by an XML processors?		tick
	a	ASCII	
	b	UTF-32	
	c	UTF-8	x
	d	big5	

5	Which attributes must appear in an XML declaration?		tick
	a	xml	
	b	xml-stylesheet	
	c	version	x
	d	encoding	

6	Which of the following strings are legal XML names?		tick
	a	bed-breakfast	x
	b	bed&breakfast	
	c	-done	
	d	xml.class	x

7	Consider the document on the right. Which of the statements below are true?	<!DOCBOOK publisher [<!ELEMENT publisher (#PCDATA)>]> <publisher>Chapman&Hall</publisher>	tick

	a	The document is not well-formed and not valid	x
	b	The document is not well-formed but valid.	
	c	The document is well-formed but not valid.	
	d	The document is well-formed and valid.	

8	Consider the XML fragment on the right. What is wrong with it?	<e e=””/>	tick
	a	no XML declaration	
	b	missing end-tag	
	c	element and attribute have the same names	
	d	nothing	x

9	Consider the XML fragment on the right. How to fill "…" to make it well-formed?	<e1><e2 a=…	tick
	a	<e2/><e1/>	
	b	<e1/><e2/>	
	c	` `</e2></e1>	x
	d	" "</e2></e1><!- the end ->	x

10	Consider the XML fragment on the right. How to fill "…" to make it valid?	<!DOCTYPE e [<!ELEMENT e (d?)+> <!ELEMENT d EMPTY>]> <e>…</e>	tick
	a	<d>text</d>	
	b	<d/>	x
	c		x
	d	<d/><d/>	x

11	Consider the XML fragment on the right. How to fill "…" to make it valid?	<!DOCTYPE e [<!ELEMENT e (#PCDATA)> …]> <e>text</e>	tick
	a		x
	b	<!ATTLIST e a CDATA #REQUIRED>	
	c	<!ATTLIST e a CDATA #IMPLIED>	x
	d	<!ATTLIST e a CDATA "text">	x

12	What kind of markup may occur between a start-tag and an end-tag?		tick
	a	element	x
	b	attribute	
	c	entity reference	x
	d	entity declaration	

13	Which of the following are XML well-formedness constraints:		tick
	a	an entity must not contain a recursive call to itself	x
	b	an attribute name may not repeat, unless in empty-element tag	
	c	all entities referenced in a document must be declared	
	d	the name in the start-tag must match the name in the end-tag	x

14	What kind of entity is declared here? <!ENTITY e SYSTEM "f">		tick
	a	parameter	
	b	general internal	
	c	external parsed	x
	d	external unparsed	

15	Which namespace does the element f belong to?	<e xmlns:n="1"> <n:f xmlns:n="2"/> </e>	tick
	a	1	
	b	2	x
	c	default	
	d	none	

16	Which namespace does the element f belong to?	<e xmlns="1"> <f xmlns=""/> </e>	tick
	a	1	
	b	default	
	c	none	x
	d	the document is not well-formed	

17	How many times can the element e occur?	<element name="e" minOccurs="2"/>	tick
	a	2	
	b	1	
	c	0	x
	d	not well-formed schema	

18	Which statements below about XML are true?		tick
	a	Elements can have simple types.	x
	b	Attributes can have simple types.	x
	c	Elements can have complex types.	x
	d	Attributes can have complex types.	

19	What is the DTD corresponding to the following schema content model?	<all> <element ref="a"/> <element ref="b"/> </all>	tick
	a	(a,b)	
	b	(a|b)	
	c	(a?|b?)	
	d	none	x

20	How to write an Xpath expression to access all grandchildren "note" of a "book" child element, which contain the attribute "type"?		tick
	a	book/note[type]	
	b	book//note[@type]	
	c	book/*/note[@type]	x
	d	book/*/note[@type='check it']	

XML Technology and Java

Training Course

Tomasz Janowski

e-Macao Report 22

Version 1.0, October 2005

Table of Contents

1. Overview

Extensible Markup Language (XML) is a meta-language promoted and developed by the World Wide Web Consortium [1]. XML provides generic markup syntax to design structured, self-describing documents in a presentation- and device-independent way.

On the basis of XML, many vertical application-specific languages have been built in various domains. Some well-known examples include XHTML (online publishing), DOCBook (book typesetting), ebXML (business interoperability), Visa Invoice (financial exchange) and many others. Notably, XML can be used to build its own horizontal extensions. Comprising the XML Technology Family, the extensions include: XML Schema (validation), XSLT (transformation), XSL-FO (presentation), XLink (navigation) and others. In addition, XML is receiving a lot of support from software community, with Application Programming Interfaces defined and implemented for all main programming languages. Less than 10 years since its introduction, XML became the mainstream of industrial software development and the cornerstone of interoperability solutions in both private and public sectors.

This document presents XML Technology and shows how to develop XML Applications using Java. The alliance of XML and Java is certainly not accidental, bringing together the portability and platform-independence of Java solutions with open data exchange facilitated by XML and its extension technologies. The course introduces the XML language, presents selected members of the core family of XML Technologies and describes different techniques for programming XML-processing applications in Java. The duration is 42 hours and the target audience is software practitioners with some Java programming experience.

The rest of this document explains the objectives, prerequisites and methodology for teaching the course in Sections 2, 3 and 4. The content of the course is introduced in some detail in Section 5. The assessment and organization of the course are explained in the final Sections 6 and 7. Following references, Appendix A includes the complete set of slides and Appendix B contains two sets of assessment questions and answers.

2. Objectives

The course has three main objectives:

1) To provide students with a solid foundation and understanding of XML, XML-related technologies and XML-processing Java applications.
2) To introduce the syntax, semantics and pragmatics of: XML, DTD, XML Schema, XPath and XSLT languages, as well as SAX, DOM and XSLT Application Programming Interfaces.
3) To equip students with skills in:
   a) writing and parsing well-formed XML documents,
   b) validating XML documents with respect to DTD or XML Schema descriptions,
   c) transforming XML documents to XML or other formats using the XSLT language and
   d) writing XML-processing applications with SAX (Simple API for XML), DOM (Document Object Model) or XSLT (XSL Transformations) Application Programming Interfaces.

To practice the skills learnt, the course relies on a concrete set of tools and APIs - Apache XML tools (Xerces and Xalan) and Java API for XML Processing (JAXP).

3. Prerequisites

Prerequisites are different for the first and second parts of the course:

- The first part – Introduction, XML Language and XML Technologies is self-contained. It explains XML from more or less the first principles. This part is accessible to any technically-inclined student with general background in computing and applications.

- The second part – XML Programming and Java assumes knowledge of the principles of object-oriented programming, some experience with Java programming and familiarity with Java API Specification and its basic packages.

4. Methodology

Four principles have been applied to deliver the material of this course:

- *Depth versus Breadth* - As foundation, the syntax of XML is introduced in detail. In contrast, selected XML horizontal technologies and APIs are explained in less detail, while the presentation of selected vertical technologies is cursory.
- *Academic Orientation* – A body of concepts is defined rigorously and incrementally to establish a proper foundation for understanding and use of technology.
- *From Definitions to Demonstrations* – All major concepts introduced during the course are illustrated with small-size examples and run-time demonstrations, often introducing skeletal code to build own custom-made solutions.
- *From Demonstrations to Assignments* – On the basis of demonstrations, students are asked to perform different tasks with increasing level of difficulty and independence.

5. Content

The course consists of 720 slides divided into four sections: Introduction, XML Language, XML Technologies and XML Processing and Java. They are described in the sequel.

5.1. Introduction

This section, comprising the slides from 1 through 119 explains the motivation, features, origin and development of XML and its surrounding technologies as follows:

- *Motivation* - limitations of the World Wide Web are explained, such as: browser-specific extensions to the HTML language, dependence on computer screen as the presentation device, and lack of structure-style separation in HTML documents, among others.
- *Features* - As a possible solution, XML is introduced through a number of features. For instance: documents are self-describing, structure and style are kept separate, multi-lingual documents can be written based on Unicode, syntactic rules are strictly enforced, validation rules can be specified and enforced, the whole family of XML technologies is self-describing in the sense of only relying on XML syntax, and many others.
- *Origin* - The origins of XML refer to the GenCode project, then GML, SGML and HTML. HTML is a particular SGML instance language and XML is a simplification of SGML, to be used online in the ways it is now possible with HTML.
- *Development* - The World Wide Web Consortium and its activities to develop and promote XML Technologies, particularly through W3C Recommendations, are introduced.

5.2. XML Language

This section, comprising the slides 120 through 407 presents the core components of the XML Language: Unicode, XML, DTD and namespaces. Each is introduced as follows:

- *Unicode* – We introduce the Universal Character Set (UCS), UTF-8, UTF-16 and other encodings of UCS, and how XML represents characters through UCS and its encodings.
- *XML* – W3C XML Recommendation is introduced, followed by the principles of XML design and the main elements of the language – parsed data, unparsed data and markup. The details of the language are presented through EBNF rules introducing various language constructs: document, prolog, element, comment, processing instruction, character, name, XML declaration, document type declaration, start tag, end tag, empty element tag, content, attribute, attribute value, reference, entity, CDATA section and character data, etc.. The rules defining well-formed XML documents are introduced as well.
- *DTD* – The difference between well-formed and valid XML documents is explained, followed by the explanation of DTD as the language to define validation rules. DTD is introduced in detail through presentation of EBNF rules for markup declarations of: elements - empty, any, mixed and children content models; attribute lists - types and defaults; entities – general and parameter, pre-defined and user-defined, internal and external, parsed and unparsed; and notations. Conditional sections and behaviour or validating versus non-validating XML processors are explained as well.
- *Namespaces* – The rationale for namespaces is explained, followed by regular and default namespace declarations, and the use of namespaces in XML documents.

5.3. XML Technologies

This section, comprising the slides 408 through 595 presents three major XML Technologies: XML Schema, XPath and XSL Transformations. They are explained as follows:

- *XML Schema* – The limitations of DTD are explained, followed by W3C XML Schema Recommendation and comparison between DTD- and Schema-based validations. The details of the language are introduced: comments; elements declarations; sequence, choice and all content models; element repetition; elements with mixed and non-mixed content; attribute declarations, defaults and occurrence; named versus anonymous types; simple versus complex types; pre-defined versus user-defined types; defining simple types through restriction, list and union; defining facets for simple type restriction; element and attribute groups; deriving new types from existing types; limiting type derivation. We also explain how to build schema documents by reuse, and how to schema-validate XML with no namespace, single namespace or multiple namespaces.
- *XPath* – The role of XPath to address parts of an XML document as well as carry out manipulation of text, numbers and strings is explained. Selected elements of the XPath language are introduced: single, multiple and wildcard steps; relative and absolute paths; descendant selection; self, parent and grandparent reference; steps with predicates; position, presence and value tests; Boolean operators and multiple predicates.
- *XSLT* – XSLT is introduced as a fully-fledged declarative programming language specialising in XML transformations. The process of matching input XML elements against XSLT templates is explained. The details of the XSLT language are introduced: templates with XPath expressions, templates with names, calling of matched and named templates, default templates and when they are called, templates with mode, templates with parameters, local and global parameters, variables, conditional execution, execution by choice, iterative execution, sorting of elements and creating new document nodes.

5.4. XML Processing and Java

This section, comprising the slides 596 through 719 explains three well-established models for programming XML-processing applications - event-based programming with SAX, tree-based programming with DOM and rule-based programming with XSLT, and how to use them when writing applications in Java. The three models are explained as follows:

- *SAX* – The architecture of SAX - Simple API for XML Processing and its use as part of an XML Java application is explained. SAX comprises the SAX parser factory, SAX parser and four kinds of event handlers: content handler, error handler, DTD handler and entity resolver. The setup and invocation of a SAX parser is described, followed by the detailed explanation of the four event-handling interfaces. In particular, this includes the methods of the content handler, called by the SAX parser to inform about various kinds of XML content and implemented by the application, such as: begin and end of a document, begin and end of an element, begin and end of a namespace, character data and processing instruction. The skeleton Java code of a SAX application is presented as well.
- *DOM* – The DOM – Document Object Model standard is explained, comprising a tree-like model of a document, a language-independent API operating on this model and the W3C Recommendation to explain the use of this API. Comparisons between DOM and SAX are drawn, with their relative strengths and weaknesses. The hierarchy of DOM data types is explained in some detail: node, node-list, named node map, string, implementation, exception and timestamp, and how they relate to one another. In particular, the node interface is elaborated to represent all kinds of nodes: elements, attributes, entity references, entities, documents, document fragments, text, CDATA section, processing instruction, comment, document type and notation nodes. Methods and attributes of the node interface are introduced, as well as methods and attributes of the different interface derived from node: element, attribute, character data and document. Skeleton Java code is presented for DOM applications for reading an XML document, and for reading and writing (modifying) an XML document. It is also explained how a Java DOM application can validate an input XML document, both using DTD and XML Schema languages.
- *XSLT* – We explain how XSLT transformations can be used as part of Java applications for processing XML. The technique combines writing standalone XSLT stylesheets and using DOM for writing read-write XML applications. A skeleton Java code is presented.

6. Assessment

The course ends with assessment. This comprises 20 multiple-choice questions to cover all sections and major concepts introduced.

Two sets of 20 assessment questions and answers, obtained from each other largely by permutation, are presented in Appendices B.1 and B.2.

7. Organization

The course consists of lectures, demonstrations and assignments:

- *lectures* – The lectures aim to incrementally build a body of concepts to establish the foundation for proper understanding and the use of technology.
- *demonstrations* - Demonstrations illustrate the concepts introduced during the lectures with running code and small-size examples.
- *assignments* - During assignments, students are asked to perform a range of tasks with increasing level of difficulty and independence. They are encouraged to reuse the demonstrated code and examples in their assignments.

The full course can be taught during seven days, 6 hours every day, or spread over the whole semester. The example schedule for consecutive days is: (1) introduction, (2) XML, (3) DTD, (4) namespaces and XML Schema, (5) XPath and XSLT, (6) Java XML with SAX and (7) Java XML with DOM and XSLT.

A shorter version can be also taught over four days, roughly divided into four major sections: (1) Introduction, (2) XML Language, (3) XML Technologies and (4) Java XML Programming.

References

1. World Wide Web Consortium. Extensible Markup Language (XML) 1.0, W3C Recommendation. 2000, http://www.w3.org/TR/REC-xml.
2. Erik T. Ray, Learning XML, O'Reilly, 2001.
3. Kenneth B. Stall, XML Family of Specifications, Addison Wesley, 2003.
4. Processing XML with Java, E. R. Harold, Addison Wesley.
5. David Brownell. SAX2. O'Reilly, 2002.
6. James Clark, editor. XSL Transformations (XSLT), W3C Recommendation. 1999 http://www.w3.org/TR/xslt.
7. James Clark and Steve DeRose, editors. XML Path Language (XPath), W3C Recommendation. 1999. http://www.w3.org/TR/xpath.
8. Arnaud Le Hors et. al., editor. Document Object Model (DOM) Level 2 Core Specification, W3C Recommendation. 2000. http://www.w3.org/TR/DOM-Level-2-Core.
9. Henry S. Thompson et. al., editor. XML Scheme Part 1 (Structures), W3C Recommendation.
10. XML Internationalisation and Localization, Yves Savourel, SAMS.
11. XML Topic Maps, Jack Park (Ed.), Addison Wesley.
12. Secure XML, D. E. Eastlake III and Kitty Niles, Addison Wesley.
13. XML Data Management, A. Chaudhri et. al., Addison Wesley.
14. ebXML, A. Walsh, Prentice Hall.
15. XML Distributed Systems Design, A.M. Rambhia, SAMS.
16. Modelling XML Applications with UML, D. Carlson, Addison Wesley.
17. The Apache Software Foundation. Xalan Java 2.3.1. http://xml.apache.org/xalan-j.
18. The Apache Software Foundation. Xerces Java Parser 2.0.0. http://xml.apache.org/xerces2-j

Appendix

A. Slides

A.1. Introduction

XML Technology and Java

Tomasz Janowski

The United Nations University IIST, Macau
University of Gdańsk, Poland

Objective

The course has two main objectives:

1) To provide students with the solid foundation and understanding of XML and XML-related technologies.
2) To develop skills in writing XML-processing Java applications using:
   a) SAX (Simple API for XML),
   b) DOM (Document Object Model) and
   c) XSLT (Extensible Stylesheet Language Transformations).

e-Macao-16-4-3

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-4

Timetable

7 days timetable:

- Introduction, Unicode
- XML
- DTD
- Namespaces, XML Schema
- XPath, XSLT
- Java XML with DOM
- Java XML with SAX and XSLT

e-Macao-16-4-5

Organization

The course consists of:

- lectures
- demonstrations
- assignments (tasks to perform)
- project work

e-Macao-16-4-6

Literature

The basis for the course are official technical documents of W3C:

- World Wide Web Consortium, Technical Reports, http://www.w3c.org/TR/

Recommended general books:

- Erik T. Ray, Learning XML, O'Reilly, 2001
- Kenneth B. Stall, XML Family of Specifications, Addison Wesley, 2003

e-Macao-16-4-7

More Literature

Recommended specialised books:

- Processing XML with Java, E. R. Harold, Addison Wesley
- XML Internationalisation and Localization, Yves Savourel, SAMS
- XML Topic Maps, Jack Park (Ed.), Addison Wesley
- Secure XML, D. E. Eastlake III and Kitty Niles, Addison Wesley
- XML Data Management, A. Chaudhri et. al., Addison Wesley
- ebXML, A. Walsh, Prentice Hall
- XML Distributed Systems Design, A.M. Rambhia, SAMS
- Modelling XML Applications with UML, D. Carlson, Addison Wesley
- etc.

e-Macao-16-4-8

History

This course was delivered before:

1) September 2003, UNU-IIST, Macau, 40 hours course for Macau IT staff from government, academia and industry.
2) October 2003 – January 2004, University of Gdańsk, Poland, 120 hours (60 hours of lectures and 60 of exercises) monograph elective course for Master degree students.

A.1.1. Motivation

Motivation

e-Macao-16-4-10

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-11

World Wide Web

If it ain't broke, don't fix it.
Millions of people surf the Web every day:

- home-makers searching for do-it-yourself recipes
- students looking for help in their mid-term projects
- investors seeking the latest stock quotes
- tourists exploring the best holiday packages
- readers purchasing books from on-line bookshops
- researchers learning the latest results by their peers

World Wide Web works well for them. Or does it?

e-Macao-16-4-12

Problems with the Web

1) Browser-specific extensions
2) Explicit browser support
3) Browser orientation
4) Structure and style intermixed
5) Data exchange is problematic
6) Monitor orientation
7) Unfocused searches
8) Static content
9) One-page limitation
10) One-way linking only
11) etc.

e-Macao-16-4-13

Problems: Browser-Specific Extensions

HTML standards take a long time to agree. Not willing to wait, vendors decide to introduce browser-specific extensions.

This HTML displays differently in every browser we tested:

```
<html>
  <head>
    <title>Welcome Message</title>
  </head>
  <body>
    <marquee>Welcome</marquee>
    to the
    <blink>XML Technology</blink>
    course!
    <img src="smiley.gif"/>
  </body>
</html>
```

e-Macao-16-4-14

Demo: Browser-Specific Extensions

```
> cd "demos/browser-specific extensions"
> dir
smiley.gif welcome.html
> opera welcome.html
> netscape welcome.html
> iexplorer welcome.html
> amaya welcome.html
```

e-Macao-16-4-15

Problems: Explicit Browser Support

Browser-specific features cause a dilemma for content providers:

- use old features only
- support one browser

- support multiple browsers:

```
<script language="javascript">
  if (version < 4.0)
    location.href='index1.html';
  if (vendor == 'Netscape')
    location.href='index2.html';
  if (vendor == 'Microsoft')
    location.href='index3.html';
</script>
<noscript>
  <a href="index4.html">
    No scripting.
  </a>
</noscript>
```

e-Macao-16-4-16

Problems: Browser Orientation

A Web browser became a launcher for applications, not just a tool for surfing the Web.

This paradigm is too restrictive. Our everyday applications

1. editors
2. spreadsheets
3. media players, . . .

should access the Web directly.

HTML calling a Java applet:

```
<html>
<body>
<h1>XML Technology Course</h1>
<applet code="menuscroll2.class">
<param name="text1" value="XML"/>
<param name="text2" value="DTD"/>
<param name="text3" value="DOM"/>
...
</applet>
</body>
</html>
```

e-Macao-16-4-17

Demo: Browser Orientation

```
> cd "demos/browser orientation"
> dir
menuScroll.html menuscroll2.class
> opera menuScroll.html
```

e-Macao-16-4-18

Problems: Structure and Style Intermixed

- Structural elements (title) and style elements (i) are freely intermixed in HTML.
- Such documents are tied to a single style vocabulary, and difficult to convert.

```
<html> ...
XML is <i>fun</i>.
In Polish we say
<i>fajny</i>
</html>
```

```
<html> ...
XML is
<emphasis>fun</emphasis>.
In Polish we say
<foreign>fajny</foreign>.
</html>
```

e-Macao-16-4-19

Problems: Monitor Orientation

With the growing number of Internet-connected devices:

1) computers
2) phones
3) hand-held devices
4) TV, etc.

data description must be independent of the device on which it will be displayed.

```
<html>
<head>
<title>Framed Page</title>
</head>
<frameset cols="100,*">
<frame name="navigation" .../>
<frame name="main" .../>
<noframes>
<p> your browser does
not support frames</p>
</noframes>
</frameset>
</html>
```

e-Macao-16-4-20

Problems: Unfocused Search Engines

Most search engines can only index the frequency of words in a document, producing thousands of hits, most missed. We need to convey semantic information about the document content.

```
<html>
  <head>
    <meta name="keywords" content="XML course macao"/>
    <meta name="description" content="This site ..."/>
    <title>XML Technology Course</title>
  </head>
  <body>
    <p> We invite applications to attend ... </p>
  </body>
</html>
```

e-Macao-16-4-21

More Problems

- Data exchange is problematic

  HTML is not suited for data exchange: filtering, integrating from different sources, verifying that data has required types.

- Static content

  On data-intensive web sites, content changes frequently. Presentation should be generated for a given content, and re-generated every time the content changes.

e-Macao-16-4-22

More Problems

- One-page limitation

  The Web does not support handling collections of inter-related documents, except one at a time. We needs ways to express relations within a document and between different documents.

- One-way linking

  The current one-way linking mechanism is too restrictive. We need more powerful mechanisms: links with multiple targets, multi-directional links and external link-bases.

A.1.2. Overview

Overview

e-Macao-16-4-24

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-25

XML

XML is called upon to solve such problems.

- XML is not to replace HTML.
- HTML will be (is currently) absorbed into XML as a pure version of itself: XHTML.
- XML is to create a more solid and flexible foundation for the next generation Internet technology.

e-Macao-16-4-26

What is XML?

1) a protocol for containing and managing information
2) a family of technologies that can do anything from writing, validating, presenting, to processing documents
3) a philosophy promoting the maximum usefulness for data by refining it to its purest and most structured form

e-Macao-16-4-27

What is not XML?

- It is not a programming language:

  XML does not prescribe an execution to be carried out by a machine, unlike e.g. Java

- It is not a presentation language:

  XML does not contain instructions to render a document, unlike e.g. Postscript

e-Macao-16-4-28

XML as a Language

"L" in XML stands for *Language*.

XML provides a notation to write self-describing data. It does so by capturing the structure of data separately from its style.

- **syntax:** XML documents have well-defined syntax.
- **semantics:** XML does not assign semantics to its documents, but permits external applications to do so.

e-Macao-16-4-29

XML as a Meta-Language

"X" in XML stands for *eXtensible*.

XML is a *meta-language* - a syntax to describe other languages.

Those languages can span diverse industrial domains.
Notably, XML can be applied to describe its own extensions.

e-Macao-16-4-30

XML Domains

vertical domains		horizontal domains	
law	LegalXML	validation	XML Schema
news	NewsML	transformation	XSLT
finances	Visa Invoice	presentation	XSL-FO
business	ebXML	navigation	XLink
telephony	VoiceXML	retrieval XML	Query
publishing	XHTML	distribution	SOAP
governance	GovML	security	XML Encryption
…		…	

e-Macao-16-4-31

XML as a Meta-Markup-Language

"M" in XML stands for *Markup*.

XML instance languages are *markup languages*: they annotate their expressions (documents) with *markup* to highlight their structure.

What is markup then?

e-Macao-16-4-32

Markup

- Markup is information added to a document to enhance its meaning in certain ways, by identifying parts of a document and how they relate to one another.
- Markup language is a set of symbols that can be placed in the text of a document to demarcate and label its parts.

e-Macao-16-4-33

Example: XML Markup

```
<?xml version="1.0"?>
<message date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

- boundaries – beginning and end tags
- roles – tag name to identify the element's role
- meta-data – attributes to inform about content
- position – one element comes before another
- containment – one element is inside another
- relationships – linking to external resources

e-Macao-16-4-34

Demo: XML in Browsers

```
> cd "demos/xml in browsers"
> ls
welcome.xml
> opera welcome.xml
> iexplore welcome.xml
```

e-Macao-16-4-35

Overview of XML

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up

e-Macao-16-4-36

Overview of XML: Self-Describing Data

The name of an element describes the meaning or function of the data it contains:

- from – sender
- to – receiver
- emphasis – important text

An application can then process the document taking into account this semantic information

```
<?xml version="1.0"?>
<message
   date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

e-Macao-16-4-37

Overview of XML: Flexible Structuring

Two approaches to data modeling:

1. Data-centric XML stores all data inside elements.
2. Document-centric XML interleaves elements with text.

XML permits both approaches.

e-Macao-16-4-38

Example: Data- versus Document-Centric

```
<?xml version="1.0"?>
<message date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

```
<?xml version="1.0"?>
<message>
<from>Tomasz</from> sends
a welcome message to
<to>Participants</to>:
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</message>
```

e-Macao-16-4-39

Overview of XML: Structure-Style Separation

XML versus HTML:

1. XML describes the structure of data, regardless of how it will be formatted for presentation.
2. HTML contains a mixture of the structural (title) and presentation (table) markup.

e-Macao-16-4-40

Example: Structure versus Style

```
<?xml version="1.0"?>
<message date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

```
<html>
<table>
<tr>
<td><b>from</b></td>
<td>Tomasz</td>
...
</tr>
</table>
Welcome to the
<i>XML Technology</i> course!
<img src="smiley.gif"/>
</html>
```

e-Macao-16-4-41

Overview of XML: Style Kept Externally

XML versus HTML:

- HTML is restricted to particular presentation.
- One XML document can be formatted in different ways.
- The formatting information is kept in a stylesheet document.
- The stylesheet document is external to the instance document.
- The stylesheet may be referred from the instance document.

e-Macao-16-4-42

Example: External Stylesheet

Given this CSS (Cascading Stylesheets) document (welcome.css)

```
to:before {content: "to:"}
from:before {content: "from:"}
subject:before {content: "subject:"}
body {display:block}
```

and this XML document:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="welcome.css"?>
<message>...</message>
```

a CSS-aware browser produces this result:

```
from: Tomasz to: Participants subject: Welcome
Welcome to the XML Technology course!
```

e-Macao-16-4-43

Example: Another External Stylesheet

while applying this stylesheet:

```
to:before {content: "to:"}
from:before {content: "from:"}
to, from {display:block; font-weight: bold}
subject {display:none}
emphasis {font-style: italic}
```

the result is as follows:

from: Tomasz
to: Participants
Welcome to the *XML Technology* course!

e-Macao-16-4-44

Demo: Different Presentations

```
> cd "demo/different presentations"
> dir
welcome1.css welcome2.css welcome1.xml welcome2.xml
> opera welcome1.xml
> opera welcome2.xml
```

e-Macao-16-4-45

Overview of XML: Document Transformations

More radical results can be obtained through XSLT:

- eXtensible Stylesheet Language Transformations
- XSLT is a fully-fledged declarative programming language specializing in XML transformations
- XSLT programs are themselves written in XML

e-Macao-16-4-46

Example: XML Transformations with XSLT 1

This program generates HTML to display the welcome message.

Elements prefixed by xsl are formatting instructions of XSLT.

```
<xsl:stylesheet
   xmlns:xsl="www.w3.org...">
  <xsl:template match="message">
    <html>
      <table> ...
        <td><b>from</b></td>
        <td>
          <xsl:value-of select="from"/>
        </td>
      </table>
      <xsl:value-of select="body"/>
      <img src="smiley.gif"/>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

e-Macao-16-4-47

Example: XML Transformations with XSLT 2

Here is the generated HTML:

```
<html>
   <table>
    <tr><td><b>from</b></td><td>Tomasz</td></tr>
    <tr><td><b>to</b></td><td>Participants</td></tr>
   </table>
   Welcome to the <i>XML Technology</i> course!
   <img src="smiley.gif"/>
</html>
```

and the output rendered by a browser:

from Tomasz
to Participants
subject Hello
Welcome to the *XML Technology* course!

e-Macao-16-4-48

Demo: HTML Generated by XSLT

```
> cd "demos/html generated by xslt"
> dir
welcome.xml welcome.xsl smiley.gif
> xalan welcome.xml welcome.xsl welcome.html
> ls
welcome.xml welcome.xsl welcome.html smiley.gif
> opera welcome.html
```

e-Macao-16-4-49

Demo: Built-in XSLT Processing

```
> cd "demos/built-in xslt processing"
> dir
welcome.xml welcome.xsl smiley.gif
> iexplore welcome.xml
```

e-Macao-16-4-50

Overview of XML: Where are We?

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations HERE!
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up

e-Macao-16-4-51

Overview of XML: Document Processing

Transformations from XML to HTML is just one kind of processing.

Many other transformations are possible:

- XML to XML
- XML to text
- XML to PDF
- XML to Latex
- XML to troff
- etc.

They may have nothing to do with the Web.

e-Macao-16-4-52

Example: Generating LaTeX from XML

- XSLT to build the welcome message
- the input in XML
- the output in LaTeX
- notice the interleaving of different kinds of markup

```
<xsl:stylesheet
   xmlns:xsl="www.w3.org...">

<xsl:template match="message">
   \documentclass{article}
   \title{<xsl:value-of
    select="subject"/>}
   \begin{document}
   ...
   <xsl:apply-templates
    select="body"/>
   \end{document}
</xsl:template>

<xsl:template match="emphasis">
   {\it <xsl:value-of select="."/>}
</xsl:template>

</xsl:stylesheet>
```

e-Macao-16-4-53

Demo: LaTeX Generated by XSLT

```
> cd "demos/latex generated by xslt"
> dir
welcome.xml welcome.xsl smiley.eps
> xalan welcome.xml welcome.xsl welcome.tex
> dir
welcome.xml welcome.xsl welcome.tex smiley.eps
> latex welcome.tex
> dir
welcome.xml welcome.xsl welcome.tex welcome.aux
welcome.dvi welcome.log smiley.eps
> yap welcome.dvi
```

e-Macao-16-4-54

Overview of XML: Programming Support

Major programming languages all provide support for XML.

Notably, these include C++, Java and Perl.

Programming support falls into:

- event-based: a program responds to the events generated by an XML parser, as it is processing an XML document (SAX)
- tree-based: a program traverses and modifies locally a parser-generated document tree (DOM)
- rule-based: a program executes recursive transformation rules, causing global modifications to the document tree (XSLT)

e-Macao-16-4-55

Example: Document Object Model API

Java code to process an XML document.

A parser is invoked on the input document.

DOM API is used to traverse recursively the parser-generated tree.

```
package dom;
public class Counter {
  public void count(Node node) {
    switch (node.getNodeType()){
      case Node.TEXT_NODE: { ... }
      case Node.ELEMENT_NODE: {
      Node child = node.getFirstChild();
      while (child != null) {
        count(child);
        child = child.getNextSibling();
  }}}}
  public static void main(String argv[]) {
    Counter counter = new Counter();
    document = parser.parse(argv[2]);
    counter.count(document);
  }
}
```

e-Macao-16-4-56

Demo: XML Processing with Java DOM API

```
> cd "demos/xml processing with java dom api"
> dir
welcome.xml
> java dom.Writer welcome.xml
> java dom.Counter welcome.xml
> java dom.GetElementsByTagName -e message welcome.xml
> java dom.GetElementsByTagName -a img welcome.xml
```

e-Macao-16-4-57

Overview of XML: International Support

XML supports Unicode:

- UTF8 – 8-bit Unicode – is the default encoding
- every XML processor must support both UTF8 and UTF16
- they may, and usually do, support alternative character sets
- text, element and attribute names can all be international
- several languages can co-exist in one document

e-Macao-16-4-58

Example: XML Document in Polish

Here is a welcome message in Polish.

iso-8859-2 is the Central-European encoding.

Text and tag/attribute names can all appear in Polish.

```
<?xml version="1.0"
   encoding="iso-8859-2"?>
<wiadomość data="15.09.2003">
   <od> Tomasz </od>
   <do> Uczestnicy </do>
   <tytuł> Powitanie</tytuł>
   <treść>
    Witam na kursie
    <ważne>Technologii
    XML</ważne>
    <czołem obraz="smiley.jpg"/>
    This is a message in Polish:
    ąćęłńóśźĄĆĘŁŃÓŚŹŻ
   </treść>
</wiadomość>
```

e-Macao-16-4-59

Demo: XML in Polish

```
> cd "demos/xml in polish"
> ls
polish.xml
> opera polish.xml
> iexplore polish.xml
```

e-Macao-16-4-60

Overview of XML: Strict Rules of Syntax

An XML document must be well-formed to be process-able by XML-compliant applications:

- XML parsers are explicitly required not to process ill-formed XML, but to exit with a suitable error message
- Browsers accept ill-formed HTML, trying to guess the intentions of the document's author.
- A lot of browser code goes to processing ill-formed HTML, increasing complexity and decreasing predictability.

e-Macao-16-4-61

Example: Ill-Formed XML

1. date lacks quotes
2. emphasis is misspelled
3. cheers does not end
4. message and body overlap

```
<?xml version="1.0"?>
<message date=15.09.2003>
   <from>Tomasz</from>
   <to>Participants</to>
   <subject>Welcome</subject>
   <body>
    Welcome to the
    <emphasis>XML Technology
    </emphazis> course!
    <cheers img="smiley.jpg">
   </message>
</body>
```

e-Macao-16-4-62

Demo: Ill-Formed XML

```
> cd "demos/ill-formed xml"
> dir
welcome.xml welcomeIll.xml
> cp welcomeIll.xml welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
```

e-Macao-16-4-63

Overview of XML: Enforcing Validation Rules

Well-formedness contains the most basic rules of syntax checking, common to all kinds of XML documents.

Validation permits the expression and checking of the properties common to particular XML instance languages:

- A document is valid if it complies with certain rules of correctness defined for a language.
- One way to express such rules is Document Type Definition.
- A ill-formed document is invalid, but an invalid document need not be ill-formed.

e-Macao-16-4-64

Example: Invalid Well-Formed Document

This message is well-formed, but who is the receiver?

```
<?xml version="1.0"?>
<message date="15.09.2003">
   <from>Tomasz</from>
   <subject>Welcome</subject>
   <body>
    Welcome to the
    <emphasis>XML Technology
    </emphasis> course!
    <cheers img="smiley.jpg"/>
   </body>
</message>
```

e-Macao-16-4-65

Example: Validating Welcome Documents

Validation rules:

1. message is the root element
2. it contains elements from, to, subject and body
3. also the optional date attribute that contains character data
4. etc.

```
<?xml version="1.0"?>
<!DOCTYPE message welcome.dtd>
<message date="15.09.2003">
...
</message>
```

welcome.dtd

```
<!ELEMENT message (from,to,subject,body)>
<!ATTLIST message date CDATA #IMPLIED>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA | emphasis |
cheers)*>
<!ELEMENT emphasis (#PCDATA)>
<!ELEMENT cheers EMPTY>
<!ATTLIST cheers img CDATA #REQUIRED>
```

e-Macao-16-4-66

Demo: Invalid XML

```
> cd "demos/invalid xml"
> dir
welcome.xml welcomeInvalid.xml
> cp welcomeInvalid.xml welcome.xml
> xerces welcome.xml
> emacs welcome.xml
> xerces welcome.xml
```

e-Macao-16-4-67

Overview of XML: Where are We?

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
    HERE!
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up

e-Macao-16-4-68

Overview of XML: Creating Languages

DTD provides a mechanism for language definition:

- `welcome.dtd` file leaves:
  - some freedom as to structure of valid XML documents
  - a lot of freedom as the textual content of such documents
- The file can be attached to various XML documents to check their validity with respect to its definitions.
- All XML documents valid with respect to `welcome.dtd` constitute a language of “welcome messages”.

e-Macao-16-4-69

Example: Another ValidWelcome Document

Welcome for applications for the Software Project Management course.

```
<?xml version="1.0"?>
<!DOCTYPE message welcome.dtd>
<message>
   <from>UNU/IIST</from>
   <to>Students</to>
   <subject>Welcome</subject>
   <body>
    We invite applications to
    attend the Software Project
    Management course. The    deadline is
    <emphasis>15.09.2003 </emphasis>.
    Please apply to
    <emphasis>spm@iist.unu.edu
    </emphasis>.
   </body>
</message>
```

e-Macao-16-4-70

Demo: Valid XML

```
> cd "demos/valid xml"
> dir
projectManagement.xml welcome.dtd
> xerces projectManagment.xml
```

e-Macao-16-4-71

Overview of XML: Adopting Languages

Instead of defining our own language, chances are that a public XML language exists to suit our needs.

There are hundreds of XML languages defined:

1. MathML – mathematics in XML
2. SVG – vector graphics in XML
3. DocBook – authoring books in XML
4. CML – describing chemical molecules
5. Visa Invoice – writing invoices in XML
6. XHTML – authoring hypertext documents in XML
7. and many others

e-Macao-16-4-72

Example: MathML

Here is a fragment of MathML – an XML language to describe mathematical notation.

`mathml.dtd` is a public DTD for this language.

```
<?xml version="1.0"?>
<!DOCTYPE math
"http://www.w3.org/mathml.dtd">
<math>
<mi>x</mi><mo>=</mo>
<mfrac>
   <mrow>
    <mrow>
       <mo>-</mo><mi>b</mi>
    </mrow>
    <mo>&PlusMinus;</mo>
    <msqrt> ... </msqrt>
   </mrow>
   <mrow> ... </mrow>
</mfrac>
</math>
```

e-Macao-16-4-73

Demo: MathML

```
> cd "demos/mathml"
> dir
math.xml
> xerces math.xml
> opera math.xml
> amaya math.xml
```

e-Macao-16-4-74

Example: Scalable Vector Graphics

SVG is another example: a language to define scalable vector graphics.

Here the SVG markup to render a welcome text along a sinusoidal path.

`svg.dtd` is a public DTD for this language.

```
<?xml version="1.0"?>
<!DOCTYPE svg "http://www.w3.org/svg.dtd">
<svg width="12cm" height="3.6cm">
   <text fill="blue">
    <textPath xlink:href="#MyPath">
        Welcome to the
        <tspan dy="50" fill="red">
            XML Technology
        </tspan>
        course!
    </textPath>
   </text>
   <rect width="998" height="298"/>
</svg>
```

e-Macao-16-4-75

Demo: Scalable Vector Graphics

```
> cd "demos/scalable vector graphics"
> dir
welcome.svg
> xerces welcome.svg
> opera welcome.svg
> amaya welcome.svg
> iexplore welcome.svg
```

e-Macao-16-4-76

Overview of XML: Integrating Languages

Several vocabularies can be used in the same document:

- A prefix indicates which language an element comes from:

svg:title	title from SVG
xhtml:title	title from XHTML
math:title	title from MathML

- A prefix is declared referring to the language's official URL
- It tells XML processor how to process an element, or which external application should be invoked.

e-Macao-16-4-77

Example: XHTML and MathML and SVG

Here is a document using three languages:

1. XHTML,
2. MathML and
3. SVG.

```
<?xml version="1.0"?>
<html xmlns="www.w3.org/xhtml">
   <body>
    <p>Here is XHTML ... </p>
    <p>Here is MathML:
    <math xmlns="www.w3.org/MathML">
        ...
    </math>
    </p>
    <p>Here is SVG:
    <svg xmlns="www.w3.org/svg">
        ...
    </svg>
    </p>
   </body>
</html>
```

e-Macao-16-4-78

Demo: XHTML and MathML and SVG

```
> cd "demos/xhtml and mathml and svg"
> dir
mixed.xml
> amaya mixed.xml
```

e-Macao-16-4-79

Overview of XML: Self-Describing Technology

CSS/DTD predate XML and apply their own native syntax.

The general trend is to replace such legacy syntax with XML:

- XSLT – XML transformation language
- XML Schema – XML class description language

e-Macao-16-4-80

Example: XML Transformations in XML

XSLT is entirely expressed with XML.

An XML-transforming program is itself an XML document.

Here is an earlier transformation of the welcome message.

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="www.w3.org...">
<xsl:template match="message">
<html>
   <table> ...
    <td><b>from</b></td>
    <td>
    <xsl:value-of select="from"/>
    </td>
   </table>
   <xsl:value-of select="body"/>
</html>
</xsl:template>
</xsl:stylesheet>
```

e-Macao-16-4-81

Example: XML Classes in XML

Also XML Schema – a language to define XML classes of XML documents – is entirely described in XML.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="www.w3.org/XMLSchema">
   <xsd:element name="message" type="Message"/>
   <xsd:complexType name="Message">
    <xsd:sequence>
        <xsd:element name="from" type="xsd:string"/>
        <xsd:element name="to" type="xsd:string"/>
        <xsd:element name="subject" type="xsd:string"/>
        <xsd:element name="body" type="Body"/>
    </xsd:sequence>
    <xsd:attribute name="date" type="xsd:date" use="optional"/>
   </xsd:complexType> ...
</xsd:schema>
```

e-Macao-16-4-82

Demo: XML Schema Validation

```
> cd "demos/xml schema validation"
> dir
welcome.xml welcome.xsd welcomeInvalid.xml
> cp welcomeInvalid.xml welcome.xml
> xercesSchema welcome.xml
> emacs welcome.xml
> xercesSchema welcome.xml
> emacs welcome.xml
> xercesSchema welcome.xml
> emacs welcome.xml
> xercesSchema welcome.xml
```

e-Macao-16-4-83

Overview of XML: Technology Clean-Up

Another trend is to reformulate pre-XML technologies in XML.

A case in point is XHTML, which comes in three flavors:

- strict – most XML-like and forward-moving
- transitional – retains enough HTML to use older browsers
- frameset – string XHTML plus frames

Legacy software is a concern when moving from non-XML to XML.

e-Macao-16-4-84

Example: XHTML

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <body>
    <table>
        <tr><td><b>from</b></td><td>Tomasz</td></tr>
        <tr><td><b>to</b></td><td>Participants</td></tr>
        <tr><td><b>subject</b></td><td>Welcome</td></tr>
    </table>
    <p>Welcome to the <i>XML Technology</i> course!
    <img src="smiley.gif" alt="Cheers"/></p>
   </body>
</html>
```

e-Macao-16-4-85

Demo: XHTML Validation

```
> cd "demos/xhtml validation"
> dir
welcome.html welcome.xhtml smiley.gif
> opera welcome.html
> opera welcome.xhtml
> emacs welcome.xhtml
> opera welcome.xhtml
```

e-Macao-16-4-86

Overview of XML: Where are We?

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up HERE!

A.1.3. Origin

Origins

e-Macao-16-4-88

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-89

XML Timeline

1967	GenCode
1969	Generalized Markup Language
1980's	GML adopted by government and industry
1986	Standard Generalized Markup Language
1989	World Wide Web = HTML + HTTP + URL
1991	World Wide Web is online
1994	XML is envisioned at the 2nd WWW Conference
1994	World Wide Web Consortium is founded
1996	W3C XML Activity Area starts
1998	XML becomes W3C Recommendation
1998	first applications of XML emerge
1999	Internet Explorer 5.0 – the first browser to support XML

e-Macao-16-4-90

Origins of XML: GenCode

1967	GenCode project of Graphic Communication Association (GCA) proposes the use of descriptive tags to aid the separation of content and presentation in electronic documents.

e-Macao-16-4-91

Origins of XML: GML

1969	Charles Goldfarb, Ed Mosher and Ray Lorie of IBM develop GML – the first document markup language using descriptive tags. GML added DTDs to GenCode.
1980's	GML is adopted by government and industry: 1. CALS Tables by the CALS group of the US DoD (format to represent tabular data). 2. Berglund of CERN – European Particle Physics Laboratory – develops a publishing system to test SGML.

e-Macao-16-4-92

Origins of XML: SGML

1978	American National Standards Institute (ANSI) forms a committee with Goldfarb and GCA Gen-Code group to work on the standard for GML.
1986	SGML – Standard Generalized Markup Language – becomes an ISO standard (8879:1986). SGML defines a syntax for creating application-specific markup languages with grammars expressed by DTDs.

e-Macao-16-4-93

Origins of XML: HTML

1989	Tim Berners-Lee makes the first proposal for WWW – a system to access hypertext documents on the Internet: 1. HTTP – HyperText Transfer Protocol 2. URL – Uniform Resource Locator 3. HTML – HyperText Markup Language HTML is an SGML document type for hypertext documents, simple and easy to program.
1991	Tim Berners-Lee announces his work on alt.hypertext and makes the first server and browser software freely available.

e-Macao-16-4-94

Origins of XML: XML is Envisioned

1994	C. M. Sperberg-McQueen and Robert F. Goldstein, „*HTML to the Max: A Manifesto for Adding SGML Intelligence to the WWW*", 2nd WWW Conference: • HTML had to sacrifice the principles of GenCode to become truly useful. • Adapt SGML for the Web? SGML is too complex. • A new language is needed, as simple as HTML, but retaining the generality of SGML. • XML is envisioned.

e-Macao-16-4-95

Origins of XML: W3C is Formed

1994	WWW Consortium is founded by MIT, INRIA and Keio with headquarters in MIT, led by Tim Berners-Lee.
1995	HTML Working Group is formed by W3C.
1996	XML Activity Area, Phase 1, formed by W3C.
1997	First public XML specification draft presented, Tim Bray and C.M.Sperberg McQueen editors.
1998	HTML 4.0 becomes W3C Recommendation. HTML Working Group reforms to work on XHTML.

e-Macao-16-4-96

Origins of XML: XML is Formalized by W3C

1998	XML 1.0 becomes W3C Recommendation. From this moment on, 80% of W3C activities are related to XML.
1998	The first applications of XML emerge: Mathematical Markup Language (MathML) and Chemical Markup Language (1997).
1998	XML media types text/xml and application/xml proposed by Internet Engineering Task Force (IEFT).

e-Macao-16-4-97

Origins of XML: XML is Implemented

1998	DOM becomes W3C Recommendation.
1999	Namespaces in XML become W3C Recommendation.
1999	RDF becomes W3C Recommendation.
1999	Internet Explorer 5.0 is released – the first mainstream browser to support XML.
1999	"XML and the Second-Generation Web" by Jon Bosak and Tim Bray published in Scientific American.

e-Macao-16-4-98

Origins of XML: Open-Source XML Initiatives

1999	Apache XML project starts, soon producing a host of XML tools: xerces, xalan, cocoon, batik, etc.
1999	ebXML – a worldwide project to standardize XML business specifications – is initiated by: • UN/CEFACT (Trade Facilitation and Electronic Business organization) and • OASIS (Organization for the Advancement of Structured Information Standards).

e-Macao-16-4-99

Origins of XML: More Implementations

1999	XSL and XPath become W3C Recommendations.
2000	XHTML becomes W3C Recommendation.
2000	Opera 4.0 is released with XML support.
2000	Netscape 6.0 is released with XML support.
2000	Amaya 4.0 is released with support for HTML, XHTML, MathML, SVG and more.

e-Macao-16-4-100

Origins of XML: More W3C Recommendations

2001	XML Schema
2001	XLink and XBase
2001	Scalable Vector Graphics
2001	XSL Formatting Objects
2001	XML Information Sets

It is not history anymore. It is today.

A.1.4. W3C

W3C

e-Macao-16-4-102

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-103

W3C Founding

- Founded in 1994 by Tim Berners-Lee at the Massachusetts Institute of Technology, Laboratory for Computer Science.
- INRIA (Institut National de Recherche en Informatique et Automatique) joined as the first European W3C host in 1995.
- Keio University of Japan became the first Asian host in 1996.
- In 2003, European Research Consortium in Informatics and Mathematics took over the role of W3C Host from INRIA.

e-Macao-16-4-104

W3C Members

W3C consists of:

- a small full-time staff and
- hundreds of members: corporations, government agencies, universities, etc.

W3C issues Recommendations that explain how certain Web technologies should be used and integrated with existing ones.

e-Macao-16-4-105

Demo: Exploring W3C Site

```
> opera http://www.w3.org/
> opera http://www.w3.org/Consortium/
> opera http://www.w3.org/Consortium/Member/List
> opera http://www.w3.org/People/
```

e-Macao-16-4-106

W3C Recommendations

- W3C has no official jurisdiction to enforce its recommendations.
- The hope is that:

  once a consensus is reached on a particular specification, there will be sufficient vendor and developer support so that compliance results from “peer pressure”

- In the past, this hope now always materialized.

e-Macao-16-4-107

About W3C Recommendations

W3C Recommendations are written in a language that is precise and rigorous, albeit informal.

- They are not an easy reading: W3C Recommendations are aimed at XML developers, not XML users.
- There are 400+ XML books that explain and interpret W3C Recommendations at various levels, and courses like this one.

e-Macao-16-4-108

Demo: XML Resources

```
> opera http://www.w3.org/TR/
> opera www.amazon.com
> opera www.xml.org
```

e-Macao-16-4-109

W3C Recommendation Process

The emerge of a W3C Recommendation is the result of a formalized, often lengthy process:

1. An idea for a new technology arrives from:
   - a member,
   - industry or
   - the W3C team,

   perhaps published as a note on the W3C site.
2. The idea is added to existing Activity Area or, if none is appropriate, cause the creation a new Activity Area.

e-Macao-16-4-110

W3C Recommendation Process

3. The activity is assigned to:
   - one of Working Groups,
   - a special Interest Group or
   - a Coordination Group.
4. If sufficient interest is generated by the proposal, the Working Group eventually produced the first Working Draft.
5. W3C announces the Working Draft to solicit feedback from its members, the industry and the developer community.

e-Macao-16-4-111

W3C Recommendation Process

6. Subsequent revisions are published:
   - every three months,
   - until the Working Group decides to publish the Last Working Draft.

7. The Last Working Draft can be:
   - rejected at this point – sending back for more revisions, or
   - become a Candidate Recommendation.

e-Macao-16-4-112

W3C Recommendation Process

8. The Candidate Recommendation is a call to industry for:
   - implementations and
   - technical feedback

   This stage can be skipped if implementations already exist.

9. The outside experience may result in:
   - finalizing the Candidate Recommendation into Proposed Recommendation, or
   - sending it back into the Working Draft stage.

e-Macao-16-4-113

W3C Recommendation Process

10. Proposed Recommendation lasts from one to three months and is the last chance for members to suggest changes.

11. It is voted by members to become a W3C Recommendation:
    - as it is,
    - with changes,
    - return to the working draft, or
    - reject (drop from W3C activities).

e-Macao-16-4-114

Demo: W3C Recommendations

```
> opera http://www.w3.org/TR/
> opera http://www.w3.org/TR/#Recommendations
> opera http://www.w3.org/TR/#PR
> opera http://www.w3.org/TR/#PER
> opera http://www.w3.org/TR/#CR
> opera http://www.w3.org/TR/#WD
```

e-Macao-16-4-115

W3C Domains

The resources of W3C are spread across five domains:

1. Architecture Domain: to develop the underlying technologies of the Web (e.g. HTTP, XML, DOM, Jigsaw).
2. Document Formats Domain: to develop languages that Web content developers can use (e.g. SVG, XHTML, CSS).

e-Macao-16-4-116

W3C Domains

3. Interaction Domain: to improve user interaction with the Web(mobile access, multimedia, voice browsers)
4. Technology and Society Domain**:** to build infrastructure to address social, legal and public policy concerns (encryption).
5. Web Accessibility Initiative: to promote usability of the Web for people with disabilities (web content guidelines).

e-Macao-16-4-117

W3C XML Working Groups

There are five XML Working Groups:

1. XML CoreWorking Group: XML, namespaces
2. XML SchemaWorking Group: schema languages
3. XML Linking Working Group: XPointer, XLink, XML Base
4. XML Query Working Group: developing query data models
5. XML Coordination Group: coordination of all XML activities

e-Macao-16-4-118

W3C XML Working Groups

and three related ones:

1. XSL Working Group: XML transformation and display
2. DOM Working Group: models for manipulating documents
3. Web Services Working Group: XML-based protocols

e-Macao-16-4-119

Demo: W3C Areas and Working Groups

```
opera http://www.w3.org/Consortium/Activities
opera http://www.w3.org/XML/
```

A.2. XML Language

A.2.1. Unicode

Unicode

e-Macao-16-4-121

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-122

XML and Character Encoding

Coding matters:

- XML requires that after parsing, the XML document belongs to the Universal Character Set (UCS).
- XML processors must support UTF-8 and UTF-16 encodings of UCS.
- They may support other encodings of UCS and other character sets, but this is not required.

What does it all mean?

e-Macao-16-4-123

Coding Terminology

character: smallest component of a written language, such as: letters, digits, ideographs, punctuation and diacritical marks are all characters.

character set: a group of characters without associated numerical values, such as: Latin alphabet, Chinese phonetic symbols and Cyrillic alphabet.

coded character set: a character set where each character is associated with a code, such as ASCII or Unicode.

e-Macao-16-4-124

Universal Character Set

Character set developed by:

- ISO and
- the Unicode Consortium.

Supports most of the existing written languages.

e-Macao-16-4-125

Scripts in UCS

basic latin	0000-007F
latin supplement	0080-00FF
latin extended A	0100-017F
latin extended B	0180-024F
IPA extensions	0250-02AF
spacing modifier	02B0-02FF
diactritical	0300-036F
greek	0370-03FF
cyrillic	0400-04FF
armenian	0530-058F

hebrew	0590-05FF
arabic	0600-06FF
syriac	0700-074F
thanaa	0780-07BF
devanagari	0900-097F
bengali	0980-09FF
gurmukhi	0A00-0A7F
gujarati	0A80-0AFF
oriya	0B00-0B7F
tamil F	0B80-0BFF
...	

e-Macao-16-4-126

Scripts and Languages

One block is one script.

One script can support several languages.

For instance, Cyrillic is used to write:
1. Russian,
2. Bulgarian,
3. Ukrainian.

Some languages use several scripts.

Japanese uses:
1. Kanji
2. Hiragana
3. Katakana and
4. Romaji scripts.

e-Macao-16-4-127

Demo: Greek Letters in Unicode

```
> cd "demos/greek letters in unicode"
> acroread U0370.pdf
> opera polish-greek.xml
> iexplore polish-greek.xml
```

e-Macao-16-4-128

UCS Transformation Formats: UTF-8

UTF-8 – 8 bit encoding

- Each character is mapped to a sequence of 1-4 bytes.
- For documents in Latin script, UTF-8 is the same as ASCII.
- Default for XML.

e-Macao-16-4-129

UCS Transformation Formats: UTF-16

UTF-16 – 16 bit encoding

- Each character upto `FFFF` is encoded as a single 16-bit value.
- Characters above `FFFF` are represented as pairs of 16-bit values: high- and low-surrogates.
- Starts with a single character (Byte Order Mark):

  `FFFE` – most significant byte first
  `FEFF` – most significant byte second

e-Macao-16-4-130

Surrogates

D800-DFFF is a surrogate block:

D800-DBFF low surrogate
DC00-DFFF high surrogate

A pair of surrogate characters (L,H) represents a character:

```
(H – D800) * 400 + (L – DC00) + 10000
```

in the range 10000-10FFFF.

e-Macao-16-4-131

Byte Order Mark

BOM – Byte Order Mark

The first character of a document:

- FFFE – the document is coded in UTF-16, big-endian
- FEFF – the document is coded in UTF-16, little-endian
- any other character – the document is coded in UTF-8

e-Macao-16-4-132

Example: UTF-16 and UTF-8

“Gulliver” in UTF-16 and UTF-8:

	G	u	l	l	i	v	e	r
FE FF	00 45	00 75	00 6C	00 6C	00 69	00 76	00 65	00 72
FF FE	45 00	75 00	6C 00	6C 00	69 00	76 00	65 00	72 00
	45	75	6C	6C	69	76	65	72

e-Macao-16-4-133

Demo: UTF-16 Encoding

```
> cd "demos/utf16 encoding"
> ls
doc.xml
> xvi32 doc.xml
> opera doc.xml
> iexplore doc.xml
```

e-Macao-16-4-134

Characters in XML 1.0

0001-0008	forbidden
0009	allowed – TAB
000A	allowed – NEW LINE
000B-000C	forbidden
000D	allowed – CARRIAGE RETURN
000E-001F	forbidden
0020	allowed – SPACE
0021-D7FF	allowed
D800-DBFF	forbidden – low surrogate
DC00-DFFF	forbidden – high surrogate
E000-FFFD	allowed
10000-10FFFF	allowed – encoded as pairs of surrogates

XML 1.1 permits representation of arbitrary Unicode characters.

e-Macao-16-4-135

XML and UCS

XML requires that:

- XML processors support the UTF-8 and UTF-16 encodings.

XML permits that:

- XML processors support alternative character sets/encodings.
- They are specified by the `encoding` attribute.

e-Macao-16-4-136

Alternative Encodings of UCS

iso-8859-1	Western Europe
iso-8859-2	Central Europe
iso-8859-3	Southern Europe
iso-8859-4	Northern Europe
iso-8859-5	Cyrillic
iso-8859-6	Arabic
iso-8859-7	Greek
iso-8859-8	Hebrew
iso-8859-9	Turkish
iso-8859-10	Nordic
...	...

big5	traditional Chinese
gb2312	simplified Chinese
euc-jp	Japanese (unix)
euc-kr	Korean (unix)
koi8-r	Russian
koi8-u	Ukrainian
tis-620	Thai
windows-*	Windows
cp-*	IBM
us-ascii	basic ASCII
…	...

A.2.2. XML

XML

e-Macao-16-4-138

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-139

XML 1.0 – W3C Recommendation

- history:
  - first published in February 1998
  - revised October 2000
- editors:
  1. Tim Bray (Textuality and Netscape),
  2. Jean Paoli (Microsoft),
  3. C. M. Sperberg-McQueen (WorldWideWeb Consortium),
  4. Eve Maler (Sun Microsystems).

e-Macao-16-4-140

XML 1.0 – W3C Recommendation

- abstract:

  The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

- publication:

  `http://www.w3.org/TR/REC-xml`

e-Macao-16-4-141

XML 1.0 – W3C Recommendation

The core of the document is presentation of EBNF production rules to define the legal syntax of XML documents:

```
document ::= prolog element Misc*
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
XMLDecl ::= '<?xml' VerInfo EncodingDecl? SDDecl? S? '?>'
VerInfo ::= S 'version' Eq ("'" VerNum "'" | '"' VerNum '"')
VerNum ::= ([a-zA-Z0-9_.:] | '-')+
Eq ::= S? '=' S?
S ::= (#x20 | #x9 | #xD | #xA)+
```

There are also references to the behavior of XML processors:

Processors may signal an error if they receive documents labeled with versions they do not support.

e-Macao-16-4-142

XML 1.1 – W3C Candidate Recommendation

Reasons for the new version:

- to keep up with the changing Unicode standard
- to add two more line-end characters
- to permit representation of arbitrary Unicode characters

e-Macao-16-4-143

XML 1.1 – W3C Candidate Recommendation

An excerpt from the 1.1 Candidate Recommendation:

> *Whereas XML 1.0 provided a rigid definition of names, wherein everything that was not permitted was forbidden, XML 1.1 names are designed so that everything that is not forbidden (for a specific reason) is permitted.*

e-Macao-16-4-144

Demo: XML W3C Recommendation

```
> opera http://www.w3.org/TR/REC-xml
> opera http://www.w3.org/TR/xml11/
```

e-Macao-16-4-145

XML Design Goals 1

Design goals for XML (XML 1.0 W3C Recommendation):

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs to process XML documents.

e-Macao-16-4-146

XML Design Goals 2

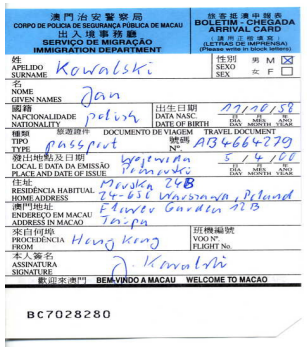
5. The number of optional features is to be kept to the minimum.
6. XML documents shall be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

e-Macao-16-4-147

Example: Macao Arrival Card

A case study for this section:

- Macao arrival card
- here filled by Jan Kowalski
- to be converted into XML



e-Macao-16-4-148

Markup versus Character Data

XML document contains text that falls into two categories:

- markup – there are 12 different kinds of XML markup
- character data – parsed or unparsed

e-Macao-16-4-149

Example: Macao Arrival Card in XML

Character data:

- Kowalski,
- 24-630 Gdask, Poland

The rest is markup.

```
<?xml version="1.0"?>
<!DOCTYPE card SYSTEM "card.dtd">
<!-- arrival card for Jan Kowalski -->
<card type="arrival">
  <visitor>
    <name type="surname">
      Kowalski
    </name>
    ...
  </visitor>
  <address where="home">
    24-630 Gda&#x0144;sk, Poland
  </address>
  ...
  <signature sigfile="mysig"/>
</card>
```

e-Macao-16-4-150

XML Markup: 1-8

no	name	example
1	start tags	`<visitor>`
2	end tags	`</visitor>`
3	empty-element tags	`<signature/>`
4	entity references	`&copyright;`
5	character references	`ń`
6	comments	`<!-- whatever -->`
7	CDATA sections	`<!CDATA[ whatever ]]>`
8	document type declarations	`<!DOCTYPE ... >`

e-Macao-16-4-151

XML Markup: 9-12

no	name	example
9	processing instructions	`<?myApp ... ?>`
10	XML declarations	`<?xml version= ... ?>`
11	text declarations	`<?xml encoding= ... ?>`
12	white space at the top level	`<?xml version="1.0"?>` `<card>...</card>`

e-Macao-16-4-152

Unparsed Character Data

Character data falls into two kinds:

- unparsed (CDATA) – data that include entity/character references:
  - entity references – `&copyright;`
  - character references – `ń`

For example:

```
Morska 24B, 24-630 Gda&#x0144;sk, Poland
```

e-Macao-16-4-153

Parsed Character Data

- parsed (PCDATA) – character data where entity references have been replaced by their definitions

For example:

```
Morska 24B, 24-630 Gdańsk, Poland
```

`0144` (hexadecimal) is the Unicode value for the character ń.

e-Macao-16-4-154

Demo: Exploring Unicode Character Charts

```
> cd "../cdrom/unicode/Code Charts/"
> acroRd32 U0100.pdf
```

e-Macao-16-4-155

Document

A document is a sequence of:

- prolog (obligatory)
- element (obligatory)
- miscellenous markup (repetitive)

```
document ::= prolog element Misc*
```

e-Macao-16-4-156

Example: XML Document Structure

```
prolog ->   <?xml version="1.0"?>
            <!DOCTYPE card SYSTEM "card.dtd">
element ->  <card type="arrival">
              <visitor>
               <name type="surname">
                   Kowalski
               </name>
               ...
              </visitor>
misc ->       ...
            </card>
            <!-- end of document -->
```

e-Macao-16-4-157

Task: Prepare a Place on Your Hard Disk

```
> cd ''a convenient location''
> mkdir course course/tasks course/tasks/card
> cd course/tasks/card
```

e-Macao-16-4-158

Task: Write, Save and Open a Document

- Use your favorite text editor to create this document:

```
<?xml version="1.0"?>
<card type="arrival">
 <visitor>
    <name type="surname">your surname</name>
    <name type="given">your given names</name>
 </visitor>
</card>
```

- Save the file as card.xml
- Open the file with your favorite browser.

e-Macao-16-4-159

Task: Damage and Repair the Document

- Like this:

```
<xml version=2.0>
<card type=arrival>
 <visitor>
     <name type="surname">your surname
     <name type="given">your given names</name>
 </guest>
</card>
```

- Save and re-open.
- Follow error messages to repair the document.

e-Macao-16-4-160

Task: Modify Document Structure

- Add a comment:

```
<?xml version="1.0"?>
<card type="arrival">...</card>
<!-- the end -->
```

- Remove the declaration:

```
<card type="arrival">...</card>
<!-- the end -->
```

- Change the order:

```
<!-- the end -->
<card type="arrival">...</card>
```

e-Macao-16-4-161

Miscellaneous Markup

Miscellaneous markup is one of:

- a comment,
- a processing instruction
- or a whitespace

```
Misc ::= Comment | PI | S
```

e-Macao-16-4-162

Example: Miscellaneous Markup

```
                    comment ->  <?xml version="1.0"?>
                                <!-- arrival card -->
                                <!DOCTYPE card SYSTEM "card.dtd">
processing instruction ->       <?myApplication date="12-03-2003"?>
                    whitespace ->   <card type="arrival">
                                    ...
                                </card>
```

e-Macao-16-4-163

Characters

A character is any Unicode character, excluding the surrogate blocks D800-DBFF and the characters FFFE and FFFF:

```
Char ::=    #x9 |
      #xA |
      #xD |
      [#x20-#xD7FF] |
      [#xE000-#xFFFD] |
      [#x10000-#x10FFFF]
```

e-Macao-16-4-164

Whitespace

A whitespace consists of one or more space characters, carriage returns, line feeds, or tabs:

```
S ::= (#x20 | #x9 | #xD | #xA)+
```

Characters are further divided into. . .

e-Macao-16-4-165

Characters: Letters and Digits

1. letters

```
Letter ::= BaseChar | Ideographic
BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] |...
Ideographic ::= [#x4E00-#x9FA5] | #x3007 | ...
```

2. digits

```
Digit ::= [#x0030-#x0039] | ...
```

e-Macao-16-4-166

Combining Characters

3. combining characters – characters that graphically combine with a preceding base character (e.g. diactrical marks)

```
CombiningChar ::= [#x0300-#x0345] | ...
```

e-Macao-16-4-167

Demo: Exploring Unicode Character Charts

```
> cd "../cdrom/unicode/Code Charts/"
> acroRd32 U0000.pdf
> acroRd32 U4E00.pdf
> acroRd32 U0300.pdf
```

e-Macao-16-4-168

XML Names

A Name begins with a letter, underscore or colon, and continues with letters, digits, hyphens, underscores, colons, or full stops:

```
NameChar ::=
   Letter | Digit |
   '.' | '-' | '_' | ':' |
   CombiningChar

Name ::= (Letter | '_' | ':') (NameChar)*

Nmtoken ::= (NameChar)+
```

e-Macao-16-4-169

Example: XML Names

```
            wrong beginning -> 1class
                               firstclass
          illegal character -> first&class
                               first_class
reserved for namespaces     -> first:class
                               first.class
           reserved for XML -> xml.class
                            -> Xml.class
                            -> XML.class
                               class.xml
                               class-Xml
                               class_XML
       strange but correct  -> _
```

Names beginning with `xml` in any combination of upper- and lower-case letters are reserved.

e-Macao-16-4-170

Task: Modify the Names

- Change all start tags into uppercase, re-open:

```
<?xml version="1.0"?>
<CARD type="arrival">
  ...
</card>
```

- Add the `my` prefix to card, re-open:

```
<?xml version="1.0"?>
<my:card type="arrival">
  ...
</my:card>
```

e-Macao-16-4-171

Comments

Reminder:

```
document ::= prolog element Misc*
Misc ::= Comment | PI | S
```

A comment is any sequence of characters surrounded by the literal strings `<!--` and `-->` that does not contain `--`:

```
Comment ::= '<!--'
((Char - '-') | ('-' (Char - '-')))* '-->'
```

Comments may appear anywhere outside markup. They are not part of the character data and need not be passed to applications.

e-Macao-16-4-172

Example: Comments

```
                          <?xml version="1.0"?>
correct comment ->        <!-- arrival card for Jan Kowalski -->
                          <card type="arrival">
                             ...
                          </card>
incorrect comment ->      <!-- end of document --->
```

e-Macao-16-4-173

Task: Add Comments to Your Document

- Add comments to your document:

```
<?xml version="1.0"?>
HERE
<card type="arrival">
HERE
   <visitor>
     <name HERE type="surname">your surname</name>
     <name type="given">your given names</name>
   </visitor>
</card>
HERE
```

- Save, reopen and correct.

e-Macao-16-4-174

Processing Instruction

Reminder:

```
document ::= prolog element Misc*
Misc ::= Comment | PI | S
```

Processing instructions contain instructions for applications.

e-Macao-16-4-175

Processing Instruction

PI consists of a target – the application to which this instruction is directed – and the data to be passed, all inside <? and ?>:

```
PI ::=
  '<?' PITarget
  (S (Char* - (Char* '?>' Char*)))? '?>'
PITarget ::=
  Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but must be passed through to the application.

XML Notation may be used for formal declaration of PI targets.

e-Macao-16-4-176

Example: Processing Instruction

```
processing instruction ->   <?xml version="1.0"?>
processing instruction ->   <?oneApp date="12-03-2003"?>
                            <?anotherApp whatever?>
                            <card type="arrival">
                               ...
                            </card>
```

e-Macao-16-4-177

Task: Add Processing Instructions

Add two processing instructions to your document:

- to the mail application

```
<?mail message="coming back" date="30.09.2003"?>
```

- to the stylesheet processor

```
<?xml-stylesheet type="text/css"?>
```

Save and reopen in both cases. What happens?

e-Macao-16-4-178

Prolog

Reminder:

```
document ::= prolog element Misc*
```

A prolog contains optional XML declaration, `Misc*`, and optional document type declaration followed again by `Misc*`:

```
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

e-Macao-16-4-179

Example: Prolog

```
          XML declaration -> <?xml version="1.0"?>
document type declaration -> <!DOCTYPE card SYSTEM "card.dtd">
   processing instruction -> <?MyApp whatever ?>
                  comment -> <!-- silly comment -->
                             <element>
                                ...
                             </element>
```

e-Macao-16-4-180

XML Declaration

Indicates that the document is written in XML.

Its syntax resembles a processing instruction for the “xml” target:

```
XMLDecl ::=
  '<?xml' VersionInfo EncodingDecl?
  SDDecl? S? '?>'
```

If present, it must occur at the very beginning of a document.

Three attributes are allowed.

e-Macao-16-4-181

XML Version Declaration

1. The version of XML being used:

```
VersionInfo ::=
   S 'version' Eq
   ("'" VersionNum "'" | '"' VersionNum '"')
Eq ::= S? '=' S?
VersionNum ::= ([a-zA-Z0-9_.:] | '-')+
```

So far, the only allowed value is 1.0. In future, this feature should permit automatic version recognition.

e-Macao-16-4-182

XML Standalone Declaration

2. Is it possible to parse the document alone? Are there any external declarations that have to be consulted?

```
SDDecl ::=
   S 'standalone' Eq
   (("'" ('yes' | 'no') "'") |
   ('"' ('yes' | 'no') '"'))
```

Two possibilities:

- `yes` – there are no external markup declarations
- `no` – there are or may be such external declarations

The default is `no`.

e-Macao-16-4-183

XML Encoding Declaration

3. Character encoding used in the document:

```
EncodingDecl ::=
   S 'encoding' Eq
   ('"' EncName '"' | "'" EncName "'" )
EncName ::=
   [A-Za-z] ([A-Za-z0-9._] | '-')*
```

- XML processors have to support UTF-8 and UTF-16 only.
- Encoding name should be registered with IANA, or otherwise it should start with `x-`.
- If a document does not start with BOM, nor an encoding declaration, then it must use UTF-8.

e-Macao-16-4-184

Example: XML Declaration

```
    minimum declaration -> <?xml version="1.0"?>
     predict new version -> <?xml version="1.1"?>
  incorrect declaration -> <?xml standalone="yes"?>
     standalone document -> <?xml version="1.0" standalone="yes"?>
             unnecessary -> <?xml version="1.0" encoding="UTF-8"?>
 east european encoding -> <?xml version="1.0" encoding="iso-8859-2"?>
standalone, EE encoding -> <?xml version="1.0" encoding="iso-8859-2"
                              standalone="yes"?>
```

e-Macao-16-4-185

Task: Version and Encoding Declarations

1. change version into `1.1`:

```
<?xml version="1.1"?>
```

2. remove version attribute:

```
<?xml?>
```

3. add `UTF-8` encoding:

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

4. change to `UTF-16` encoding

e-Macao-16-4-186

Task: Standalone Declaration

1. add to `card.xml`:
   - declaration for entity `text`
   - entity reference `&text;`
   - declaration `standalone="yes"`

```
<?xml version="1.0" standalone="yes">
<!DOCTYPE card [<!ENTITY text "silly text">]>
<card type="arrival">
...
&text;
</card>
```

2. re-open the file

e-Macao-16-4-187

Task: Play with Standalone Declaration

3. create the external file `card.dtd`:

```
<!ENTITY text "silly text">
```

4. modify `card.xml` to refer to this file:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">
...
&text;
</card>
```

5. re-open and repair `card.xml`

e-Macao-16-4-188

Document Type Declaration

Reminder:

```
document ::= prolog element Misc*
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

Document type declaration:

- names the document's root element
- contains or points (or both) to markup declarations that provide a grammar for a class of documents

e-Macao-16-4-189

Document Type Declaration

Declaration rule:

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

- `Name` is the document's root element
- `ExternalId` points to the document's external DTD
- `[...]` contains the document's internal DTD

e-Macao-16-4-190

Document Type Declaration versus Definition

Document Type:

- declaration – markup `doctypedecl` that links/contains the document's type definition
- definition – document's class description rules (DTD): inside `markupdecl` or pointed by `ExternalID`

DTD = Document Type Definition

DTD is the native grammar description language used by XML.
It is described in detail in the next section.

e-Macao-16-4-191

External Document Type Declaration

External DTD is pointed by `ExternalId`, which starts with the keyword `SYSTEM` or `PUBLIC`:

```
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
```

e-Macao-16-4-192

SYSTEM Document Type Declaration

XML does not constrain the syntax much

```
SystemLiteral ::=
  ('"' [^"]* '"') | ("'" [^']* "'")
```

It is usually a file name, path, or URL.

e-Macao-16-4-193

Example: SYSTEM Declaration

```
<!DOCTYPE card SYSTEM "card.dtd">
<!DOCTYPE card SYSTEM "/usr/local/dtds/card.dtd">
<!DOCTYPE card SYSTEM "http://www.w3c.org/card.dtd">
```

e-Macao-16-4-194

PUBLIC Document Type Declaration

`PubidLiteral` is often called the Formal Public Identifier.

```
PubidLiteral ::=
  '"' PubidChar* '"' |
  "'" (PubidChar - "'")* "'"
PubidChar ::= #x20 | #xD | #xA |
  [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

FPI provides a formalized description of the content of the DTD, independent from its physical location.

e-Macao-16-4-195

PUBLIC Document Type Declaration

Here is the typical syntax:

```
<!DOCTYPE card PUBLIC
"-//UNUIIST//Immigration Card Language 1.0//EN"
"http://www.iist.unu.edu/dtds/card.dtd">
```

where:

- `UNUIIST`: the institution that maintains this DTD
- `-`: the institution is not a standards body (otherwise +)
- `Immigration Card Language 1.0`: short description
- `EN`: the language used

e-Macao-16-4-196

Example: XHTML DOCTYPE

DOCTYPE for an XHTML document:

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

e-Macao-16-4-197

Example: SVG DOCTYPE

DOCTYPE for an SVG document:

```
<!DOCTYPE svg
  PUBLIC
  "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-
  20010904/DTD/svg10.dtd">
```

e-Macao-16-4-198

Example: MathML DOCTYPE

DOCTYPE for a MathML document:

```
<!DOCTYPE math
  PUBLIC
  "-//W3C//DTD MathML 2.0//EN"
  "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
```

e-Macao-16-4-199

Task: External DTD

1. add another line to `card.dtd`:

```
<!ENTITY text "silly text">
<!ENTITY moreText "more silly text">
```

2. include both entity references:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">
... &text; &moreText;
</card>
```

3. re-open

e-Macao-16-4-200

Task: Internal DTD

4. include both declarations internally:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card [
   <!ENTITY text "silly text">
   <!ENTITY moreText "more silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

5. re-open, change standalone, re-open

e-Macao-16-4-201

Task: Internal and External DTD

6. make one declaration internal, another external:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM "card.dtd" [
   <!ENTITY text "silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

7. re-open

e-Macao-16-4-202

Task: Overlapping Internal and External DTD

8. modify `card.xml`:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM "card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

9. re-open

10. Which declaration takes precedence: internal or external?

e-Macao-16-4-203

Task: Local External DTD

11. copy `card.dtd` to your temp directory:

```
> copy card.dtd C:/temp
```

12. modify `card.xml`, re-open:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM "C:/temp/card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

e-Macao-16-4-204

Task: Remote External DTD, SYSTEM

13. copy `card.dtd` to remote server

```
> ftp ftp.iist.unu.edu
> put card.dtd
```

14. modify `card.xml`, re-open:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM
   "http:://www.iist.unu.edu/~tj/course/card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

e-Macao-16-4-205

Task: Remote External DTD, PUBLIC

15. modify `card.xml`, re-open:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   PUBLIC
   "-//Small Steps Ltd.//Silly DTD Version 1.0//EN"
   "http:://www.iist.unu.edu/~tj/course/card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

e-Macao-16-4-206

Elements

Reminder:

```
document ::= prolog element Misc*
```

Element is an empty element tag or a non-empty element:

```
element ::= EmptyElemTag | STag content ETag
```

A non-empty element consists of content surrounded by the opening and closing tags, with matching names.

e-Macao-16-4-207

Start and End Tags

There are three kinds of tags:

1. start tag – a name and a list of attribute declarations separated by whitespaces, all inside < and >:

```
STag ::= '<' Name (S Attribute)* S? '>'
```

2. end tag – a name inside </ and >:

```
ETag ::= '</' Name S? '>'
```

e-Macao-16-4-208

Empty Element Tag

3. empty element tag – a name and a list of attribute declarations separated by whitespaces, all inside < and />:

```
EmptyElemTag ::=
'<' Name (S Attribute)* S? '/>'
```

e-Macao-16-4-209

Example: Tags

```
                        <card type="arrival">
        start tag ->       <address where="home">
                            24-630 Gda&#x0144;sk, Poland
          end tag ->       </address>
                           ...
empty element tag ->       <signature sigfile="mysig"/>
                           ...
                        </card>
```

e-Macao-16-4-210

Attributes

Attribute declarations are used in start- and empty-element tags.

A declaration consists of:

- a name,
- the equal sign and
- a value

```
Attribute ::= Name Eq AttValue
```

e-Macao-16-4-211

Attribute Values

The value is any sequence of characters and references, except the characters used in markup (<, &, and " or '):

```
AttValue ::=
  '"' ([^<&"] | Reference)* '"' |
  "'" ([^<&'] | Reference)* "'"
```

It is surrounded by:

- double quotes (allowing single quotes inside) or
- single quotes (allowing double quotes inside)

e-Macao-16-4-212

Example: Attributes

attribute in start tag ->
attribute with references ->
attribute in empty element tag ->

```
<card type="arrival">
   <address
   where="home"
   city="Gda&#x0144;sk">
    ...
   </address>
   ...
   <signature sigfile="mysig"/>
</card>
```

e-Macao-16-4-213

Task: Arrival Card: Return to Simple Form

1. return `card.xml` to the simple form:

```
<?xml version="1.0"?>
<card type="arrival">
   <visitor>
       <name type="surname">your surname</name>
       <name type="given">your given names</name>
   </visitor>
</card>
```

e-Macao-16-4-214

Task: Add Visitor's Elements

2. add the elements to describe the visitor's: `sex` (empty element) and `nationality` (element with text)

```
<?xml version="1.0"?>
<card type="arrival">
    <visitor>
        <name type="surname">your surname</name>
        <name type="given">your given names</name>
        <sex type="..."/>
        <nationality>...</nationality>
    </visitor>
</card>
```

e-Macao-16-4-215

Task: Add More Visitor's Elements

3. add the `date` of birth (text)

```
<date>...</date>
```

4. expand `date` into `day`, `month` and `year` elements:

```
<date type="birth">
   <day>...</day>
   <month>...</month>
   <year>...</year>
</date>
```

e-Macao-16-4-216

Task: Add Travel Document Information

5. add travel document information:

   - document type
   - document number
   - issuing authority
   - date of issue

```
<document>
   <type>...</type>
   <number>...</number>
   <authority>...</authority>
   <date>...</date>
</document>
```

e-Macao-16-4-217

Task: Modify Travel Document Information

6. make document `type` an attribute:

```
<document type="...">
   <number>...</number>
   <authority>...</authority>
   <date>...</date>
</document>
```

7. make document `type` and `number` into attributes:

```
<document type="..." number="...">
   <authority>...</authority>
   <date>...</date>
</document>
```

e-Macao-16-4-218

Task: Modify Document Issue Date

8. differentiate dates of birth and document issue:

```
<card type="arrival">
    ...
    <visitor>
        <date type="birth">...</date>
    </visitor>
    <document type="..." number="...">
        <authority>...</authority>
        <date type="issue">...</date>
    </document>
    ...
</card>
```

e-Macao-16-4-219

Task: Add Visitor's Addresses

9. add the visitor's home and Macao address:

```
<card type="arrival">
   ...
   <address>...</address>
   <address>...</address>
</card>
```

10. differentiate between the addresses using the attribute `where`:

```
<card type="arrival">
   ...
   <address where="home">...</address>
   <address where="Macao">...</address>
</card>
```

e-Macao-16-4-220

Task: Expand Visitor's Address Information

11. expand address information into: `street`, `city`, `zip`, `state` and `country`.

```
<address where="...">
   <street>...</street>
   <city>...</city>
   <zip>...</zip>
   <state>...</state>
   <country>...</country>
</address>
```

e-Macao-16-4-221

Task: Expand Visitor's Street Address

12. expand `street` address into: `street`, `house`, `floor` (if any) and `flat` (if any).

```
<street>
   <street>...</street>
   <house>...</house>
   <floor>...</floor>
   <flat>...</flat>
</street>
```

e-Macao-16-4-222

Task: Add Travel Information

13. add `travel` information: origin (`place`), `mode` and `flight` (if any).

```
<card type="arrival">
    ...
    <travel>
       <place>...</place>
       <mode>...</mode>
       <flight>...</flight>
    </travel>
</card>
```

e-Macao-16-4-223

Task: Add Digital Signature

14. add the traveller's digital `signature`: empty element referring to the `my.sig` file.

```
<card type="arrival">
   ...
   <signature sigfile="my.sig"/>
</card>
```

e-Macao-16-4-224

References

A reference is one of:

- character reference or
- entity reference

```
Reference ::= CharRef | EntityRef
```

e-Macao-16-4-225

Character References

A character reference refers to a specific character in UCS.

```
CharRef ::=
  '&#' [0-9]+ ';' |
  '&#x' [0-9a-fA-F]+ ';'
```

Two forms:

- `&#` – decimal representation of the character's code
- `&#x` – hexadecimal representation of the character's code

e-Macao-16-4-226

Entity References

An entity reference refers to the content of a named entity:

```
EntityRef ::= '&' Name ';'
```

Such a name must be declared in a DTD.

e-Macao-16-4-227

Pre-Defined Entity References

There are five pre-defined entity names:

name	value
amp	&
apos	'
quot	"
gt	>
lt	<

They are used to include markup characters literally.

e-Macao-16-4-228

Example: References

```
pre-defined
entity reference ->       <?xml version="1.0"?>
                          <hotel type="bed&amp;breakfast">
                             ...
character references:        <address>
            decimal ->          <city name="Gda&#324;sk"/>
                                <street
        hexadecimal ->             name="Okr&#x0119;&#x017D;na"/>
                                ...
user-defined                 </address>
entity reference ->          <note name="Read &legalstuff;"/>
                          </hotel>
```

e-Macao-16-4-229

Task: References for Common Places

1. define and refer to the entities for Hong Kong, Zhuhai and Macao:

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY hk "Hong Kong">
   <!ENTITY zh "Zhuhai">
   <!ENTITY mo "Macao">
]>
<card type="arrive">
   ...
   &hk; ...
   &mo; ...
   &zh; ...
</card>
```

e-Macao-16-4-230

Task: Define References for Two Sexes

2. define and refer to the entities representing two sexes:

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY male "<sex type="male"/>">
   <!ENTITY female "<sex type="female"/>">
]>
<card type="arrive">
   ... &female; ...
</card>
```

3. what is wrong? correct

e-Macao-16-4-231

Task: References for Common Origins

4. Define the entities to capture two common travel origins for visitors to Macao:

- from Zhuhai (on foot) or
- from Hong Kong (by ferry)

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY fromZhuhai "...">
   <!ENTITY fromHongKong "...">
]>
<card type="arrive">
   ... &fromHongKong; ...
</card>
```

e-Macao-16-4-232

Task: Reuse of Entities

5. Reuse the entities for common places when defining entities for common origins:

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY fromZhuhai "... &zh; ...">
   <!ENTITY fromHongKong "... &hk; ...">
]>
<card type="arrive">
   ...
</card>
```

6. Any other ideas for reuse in `card.xml`?

e-Macao-16-4-233

Content

Reminder:

```
document ::= prolog element Misc*
element ::= EmptyElemTag | STag content ETag
```

The text between start-tag and end-tag is called element content:

```
content ::=
   CharData?
   (  (element | Reference | CDSect | PI | Comment)
      CharData?
   )*
```

e-Macao-16-4-234

Types of Content

Element content may contain any number of:

- character data `CharData`
- elements `element`
- references `Reference`
- CDATA sections `CDSect`
- processing instructions `PI`
- comments `Comment`

Which are undefined yet? `CharData` and `CDSect`

e-Macao-16-4-235

CDATA Section

CDATA section is used to escape blocks of text containing characters which would otherwise be recognized as markup:

```
CDSect ::= CDStart CData CDEnd
CDStart ::= '<![CDATA['
CData ::= (Char* - (Char* ']]>' Char*))
CDEnd ::= ']]>'
```

Within CDATA, only `CDEnd` is recognized as markup.

CDATA cannot nest.

e-Macao-16-4-236

Example: CDATA Section

escaping markup without CDATA ->

```
<lecture about="XML">
   Here is a start-tag:
   &lt;message
   to=&quot;tj@iist.unu.edu&quot;&gt;
   and an end-tag: &lt;note/&gt;
</lecture>
```

escaping markup with CDATA ->

```
<lecture about="XML">
  <![CDATA[Here is a start-tag:
  <message to="tj@iist.unu.edu">
  and an end-tag: <note/>]]>
</lecture>
```

e-Macao-16-4-237

Character Data

All text that is not markup constitutes the character data of the document. More precisely:

```
CharData ::=
  [^<&]* - ([^<&]* ']]>' [^<&]*)
```

Character data is any string of characters which does not contain:

- the start-delimiter of any markup, and
- the CDATA-section-close delimiter `]]>`

e-Macao-16-4-238

Element Syntax Overview

```
document      ::= prolog element Misc*
element       ::= EmptyElemTag | STag content ETag
STag          ::= '<' Name (S Attribute)* S? '>'
Attribute     ::= Name Eq AttValue
AttValue      ::= '"' ([^<&"] | Ref)* '"' | ...
Ref           ::= EntityRef | CharRef
EntityRef     ::= '&' Name ';'
CharRef       ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'
ETag          ::= '</' Name S? '>'
EmptyElemTag  ::= '<' Name (S Attribute)* S? '/>'
Content       ::= CharData? ((element|Ref|CDSect|PI|Comment)...)*
CharData      ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
CDSect        ::= '<![CDATA[' CData ']]>'
CData         ::= (Char* - (Char* ']]>' Char*))
PI            ::= '<?' PITarget (S (Char*-(Char* '?>' Char*)))? '?>'
Comment       ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-- >'
```

e-Macao-16-4-239

Well-Formed XML Document

This concludes the explanation of the rules that define XML syntax.

What is a well-formed XML document?

1. Taken as a whole, it matches the rule:

   ```
   document ::= prolog element Misc*
   ```

2. It meets all well-formedness constraints (?)

3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed (?)

e-Macao-16-4-240

Entity Declarations

Entity declarations may:

- contain text as well as markup
- be parsed (XML content) or unparsed (non-XML)
- be defined within a document (internal) or outside (external)

e-Macao-16-4-241

Example: Parsed Entities

```
                        <!DOCTYPE hotel SYSTEM [
entity declaration ->     <!ENTITY legal "We reject any responsibility
                           for damages due to errors in this
   contains markup ->      <product type='software'/>.
 entity reference ->       &copyright;">
entity declaration ->     <!ENTITY copyright "&#x00A9;">
                        ]>
                        <hotel type="bed&amp;breakfast">
 entity reference ->      <note name="Read &legal;"/> ...
                        </hotel>
```

e-Macao-16-4-242

Well-Formed Parsed Entities

All parsed entities referred directly or indirectly from the body of the document, must be well-formed:

– internal parsed entity:

```
intParsedEnt ::= content
```

– external parsed entity:

```
extParsedEnt ::= TextDecl? content
```

e-Macao-16-4-243

Text Declaration

An external parsed entity may contain a text declaration:

```
TextDecl ::=
  '<?xml' VersionInfo? EncodingDecl S? '?>'
```

The text declaration allows for several different encoding methods to be used on the same document.

Note the differences with the XML declaration:

- `version` is optional,
- `encoding` is required,
- `standalone` is forbidden.

e-Macao-16-4-244

Task: Non-Well-Formed Entities

1. Modify `card.xml` so that some entities are not well-formed.

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY male "type='male'/>">
   <!ENTITY female "type='female'/>">
]>
<card type="arrival">
   <visitor>...</visitor>
   <document>...</document>
   ... <sex &male; ...
</card>
```

2. Reopen, and repair.

e-Macao-16-4-245

Well-Formedness Constraints

The third component to well-formedness is satisfaction of the side-conditions on BNF production rules:

1. The name in the start-tag must match the name in the end-tag:

```
element ::= EmptyElemTag | STag content ETag
STag ::= '<' Name (S Attribute)* S? '>'
ETag ::= '</' Name S? '>'
```

e-Macao-16-4-246

Well-Formedness Constraints

2. No attribute name may appear more than once in the same start-tag or empty-element tag:

```
STag ::= '<' Name (S Attribute)* S? '>'
EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'
Attribute ::= Name Eq AttValue
```

3. Attribute values cannot contain direct or indirect entity references to external entities:

```
AttValue ::=
   '"' ([^<&"] | Reference)* '"' |
   "'" ([^<&'] | Reference)* "'"
```

e-Macao-16-4-247

Well-Formedness Constraints

4. The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a `<`.

```
AttValue ::=
   '"' ([^<&"] | Reference)* '"' |
   "'" ([^<&'] | Reference)* "'"
```

5. The name given in the entity reference is one of `amp`, `lt`, `gt`, `apos` or `quot`, or it must be declared in the document's DTD.

```
EntityRef ::= '&' Name ';'
```

e-Macao-16-4-248

Well-Formedness Constraints

6. An entity must not contain a recursive reference to itself, either directly or indirectly.

7. An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to in attribute values.

This concludes the definition of XML Syntax.

Certain aspects of well-formedness will be re-investigated after the introduction to Document Type Definitions in the next section.

e-Macao-16-4-249

Task: Violation of Well-Formedness

For each of the following well-formedness conditions modify `card.xml` to break this condition, watch the error message, and repair:

1. matching names
2. no repeated attributes
3. no `<` in entities referenced in attribute values
4. entity names must be declared or pre-defined
5. no recursive references in entities

e-Macao-16-4-250

Example: Macau Arrival Card

```
                                  <?xml version="1.0" standalone="yes"?>
                                  <card type="arrival">
                                  <!-- visitor information -->
                                  <visitor>
                                     <name type="surname">Kowalski</name>
less elements? ->                    <name type="given">Jan</name>
                 <name>              <sex type="male"/>
          Jan Kowalski               <nationality>polish</nationality>
                 </name>             <date type="birth">
less attributes? ->                   11 October 1958
                 <date>              </date>
          11 October 1958          </visitor>
                 </date>
```

e-Macao-16-4-251

Example: Macau Arrival Card

```
more attributes? ->
          <document
    type="passport"
number="AB4664279">
                ...
        </document>
```

```
<!-- travel document information -->
<document type="passport">
  <number>AB4664279</number>
   <authority>
       Wojewoda Pomorski
   </authority>
  <date type="issue">
   5 April 2000
  </date>
</document>
```

e-Macao-16-4-252

Example: Macau Arrival Card

```
            more elements? ->
<address where="home">
  <street>Morska 24B</street>
  <city>Gda&#x0144;sk</city>
  <zip>24-650</zip>
  <country>Poland</country>
</address>

            more entities? ->
&fromHongKong;
<!ENTITY fromHongKong
  "<travel type="from">
  <place>Hong Kong</place>
  </travel>„
>
```

```
<!-- address information -->
<address where="home">
  Morska 24B,
  24-650 Gda&#x0144;sk, Poland
</address>
<address where="Macao">
  Flower Garden 12B, Taipa
</address>

<travel type="from">
<place>Hong Kong</place>
</travel>

<signature sigfile="mysig"/>
</card>
```

e-Macao-16-4-253

Demo: Macao Arrival Card

```
> cd "demos/macao arrival card"
> dir
card.xml
> java dom.Counter card.xml
> cp card.xml card.xml.old
> emacs card.xml
> java dom.Counter card.xml
> cp card.xml.old card.xml
```

A.2.3. DTD

DTD

e-Macao-16-4-255

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-256

Well-Formed or Toast

All XML documents must be well-formed:

- an ill-formed document cannot be even called XML
- XML parsers are required:
  - not to process non-well-formed XML documents
  - to report the problem to the calling application, and exit
- As a result, XML parsers are lightweight, XML processing is predictable and consistent between different applications.

e-Macao-16-4-257

Well-Formedness versus Validity

A well-formed XML document may in addition be valid.

- An XML document is valid if:
  1. it has an associated Document Type Definition
  2. it complies with the constraints expressed in it
- Unlike for SGML, DTDs are optional for XML, so is validity.

In this section we explain the syntax of Document Type Definitions, and the notion of validity associated with them.

e-Macao-16-4-258

Demo: Validity of Macao Arrival Card

```
> cd "demos/validity of macao arrival card"
> ls
card.dtd cardNoDTD.xml cardYesDTD.xml
> java dom.Counter cardNoDTD.xml
> java dom.Counter -v cardYesDTD.xml
```

e-Macao-16-4-259

DTD Scope

With DTDs we can specify:

1. elements and their attributes
2. nesting and order of elements
3. whether elements can contain text
4. whether text and elements can be mixed
5. whether elements are optional/required
6. whether elements can be repeated
7. default or fixed values for attributes
8. limited datatypes for attributes
9. reusable sections of text called entities
10. non-XML content via notations
11. etc.

e-Macao-16-4-260

Document Type Declaration

Reminder:

```
document ::= prolog element Misc*
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

Document type declaration:

- names the document's root element (`Name`)
- contains (inside [...]) or points (`ExternalID`) or both to the document's type definition.

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

e-Macao-16-4-261

External and Internal DTD

DTD consists of two subsets:

1. external – have been explained before

```
ExternalID ::=
   'SYSTEM' S SystemLiteral |
   'PUBLIC' S PubidLiteral S SystemLiteral
```

2. internal – a sequence of markup declarations (`markupdecl`) separated by declaration separators (`DeclSep`):

```
doctypedecl ::= ...
   ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

e-Macao-16-4-262

Example: DTD with External Subset

```
standalone=no ->  <?xml version="1.0"?>
external subset ->  <!DOCTYPE card SYSTEM "card.dtd">
root ->  <card type="arrival">...</card>
```

e-Macao-16-4-263

Example: DTD with Internal Subset

```
standalone=yes ->  <?xml version="1.0" standalone="yes"?>
internal subset ->  <!DOCTYPE card [
                    <!ELEMENT card (visitor, document, ... )>
                    <!ATTLIST card type (arrival | departure) ...>
                    <!ELEMENT visitor (name, name, sex, ... )>
                    <!ELEMENT name (#PCDATA)>
                       ...
                    ]>
                    <card type="arrival">...</card>

root ->
```

e-Macao-16-4-264

Example: DTD with External/Internal Subsets

```
  standalone=no ->  <?xml version="1.0"?>
external subset ->  <!DOCTYPE card SYSTEM "card.dtd" [
internal subset ->  <!ELEMENT card (visitor, document, ... )>
                    <!ATTLIST card type (arrival | departure)...>
                    <!ELEMENT visitor (name, name, sex, ... )>
                    <!ELEMENT name (#PCDATA)>
                       ...
                    ]>
          root ->   <card type="arrival">
                       ...
                    </card>
```

e-Macao-16-4-265

Demo: Document Type Declarations

```
> cd "demos/document type declarations"
> ls
cardExternal.xml cardInternal.xml cardMixed.xml
cardBig.dtd cardSmall.dtd cardMiddle.dtd
> emacs cardInternal.xml
> java dom.Counter -v cardInternal.xml
> emacs cardExternal.xml cardBig.dtd
> java dom.Counter -v cardInternal.xml
> emacs cardMixed.xml cardSmall.dtd
> java dom.Counter -v cardMixed.xml
```

e-Macao-16-4-266

Task: Make A Letter Document

- create a new directory

```
> mkdir course/tasks/letter
> cd course/tasks/letter
```

- create the smallest document with the letter root

```
<letter></letter>
```

- save it as `letter.xml`, parse

e-Macao-16-4-267

Task: Expand the Letter

- add the XML declaration, parse

```
<?xml version="1.0"?>
<letter></letter>
```

- add the DOCTYPE, parse

```
<?xml version="1.0"?>
<!DOCTYPE>
<letter></letter>
```

e-Macao-16-4-268

Task: Add the Document Type

- add the root element, parse

```
<?xml version="1.0"?>
<!DOCTYPE letter>
<letter></letter>
```

- add the empty internal subset, parse

```
<?xml version="1.0"?>
<!DOCTYPE letter []>
<letter></letter>
```

- validate, what happens?

e-Macao-16-4-269

Markup Declarations

Markup declarations include: element, attribute list, entity and notation. Processing instructions and comments are also allowed.

```
markupdecl ::=
   elementdecl |
   AttlistDecl |
   EntityDecl |
   NotationDecl |
   PI |
   Comment
```

e-Macao-16-4-270

Format of Markup Declarations

All markup declarations conform to similar syntax:

```
<!TYPE ...>
```

where `TYPE` is one of:

- `ELEMENT`
- `ATTLIST`
- `ENTITY`
- `NOTATION`

e-Macao-16-4-271

Element Type Declaration

Element type declaration contains the element's name and the specification of its content:

```
elementdecl ::=
  '<!ELEMENT' S Name S contentspec S? '>'
```

Four kinds of content specifications:

```
contentspec ::=
  'EMPTY' |
  'ANY' |
  Mixed |
  children
```

e-Macao-16-4-272

Empty Element Content

`EMPTY` means an element can neither contain:

- character data
- nor children elements

But it may contain attributes.

e-Macao-16-4-273

Any Element Content

`ANY` puts no constraints. The element may contain:

- elements,
- character data,
- a mixture of both, or none.

Useful during initial stages in the design of a DTD.

e-Macao-16-4-274

Example: Empty and Any Element Content

```
   empty element declaration ->  <!ELEMENT sex EMPTY>
<sex type="male"/>

     any element declaration ->  <!ELEMENT test ANY>
<test>                           <!ELEMENT test1 ...>
   <test1>...</test1>            <!ELEMENT test2 ...>
   <test2>...</test2>            <!ELEMENT test3 ...>
   <test3>...</test3>            <!ELEMENT test4 ...>
   <test4>...</test4>            <!ELEMENT test5 ...>
   <test5>...</test5>
</test>
```

e-Macao-16-4-275

Task: Declare Letter as EMPTY

- declare `letter` as `EMPTY`, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter EMPTY>]>
<letter></letter>
```

- add some text, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter EMPTY>]>
<letter>this is a letter text</letter>
```

e-Macao-16-4-276

Task: Declare Letter as ANY

- declare `letter` as `ANY`, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter ANY>]>
<letter>this is a letter text</letter>
```

e-Macao-16-4-277

Mixed Element Content

The mixed model: element contains character data optionally interspersed with child elements:

```
Mixed ::=
  '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' |
  '(' S? '#PCDATA' S? ')'
```

The types of the child elements may be constrained, but not their order or their number of occurrences.

e-Macao-16-4-278

Example: Mixed Element Content

```
          text-only -> <!ELEMENT name (#PCDATA)>
<name>Kowalski</name>

 text mixed with one -> <!ELEMENT address (#PCDATA | zip)*>
<address>
  <zip>CV4 7AL</zip>
   Coventry, UK
</address>

 text mixed with two -> <!ELEMENT address (#PCDATA|zip|country)*>
<address>
  <zip>CV4 7AL</zip>
   Coventry,
  <country>UK</country>
</address>
```

e-Macao-16-4-279

Task: Declare Letter as PCDATA

- declare letter as `PCDATA`, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter (#PCDATA)>]>
<letter>this is a letter text</letter>
```

e-Macao-16-4-280

Task: Expand The Letter Text

- expand the letter text, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter (#PCDATA)>]>
<letter>
  Dear Simon White,
  We are pleased to inform you that your
  credit card application has been approved.
  Sincerely, Steven Rod
</letter>
```

e-Macao-16-4-281

Task: Add Letter Markup

- add some letter markup, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter (#PCDATA)>]>
<letter>
  Dear <customer>Simon White</customer>,
  We are pleased to inform you that your
  <product>credit card</product> application
  has been approved. Sincerely, Steven Rod
</letter>
```

e-Macao-16-4-282

Task: Declare Markup

- declare markup, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (customer | product | #PCDATA)*>
  <!ELEMENT customer (#PCDATA)>
  <!ELEMENT product (#PCDATA)>
]>
<letter>...</letter>
```

e-Macao-16-4-283

Task: Correct Markup

- correct, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (#PCDATA | customer | product)*>
  <!ELEMENT customer (#PCDATA)>
  <!ELEMENT product (#PCDATA)>
]>
<letter>...</letter>
```

e-Macao-16-4-284

Task: Modify Markup

- modify and validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT customer (#PCDATA)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT letter (#PCDATA | product | customer)*>
]>
<letter>...</letter>
```

e-Macao-16-4-285

Task: Add More Markup

- add more markup, declare, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [...]>
<letter>
  Dear <customer>Simon White</customer>,
  We are pleased to inform you that your
  <product>credit card</product> application
  has been approved. Sincerely,
  <manager>Steven Rod</manager>
</letter>
```

e-Macao-16-4-286

Element Content

An element must contain only child elements (no character data), optionally separated by white space.

Build with `choice` or `seq` and optional occurrence indicators:

```
children ::=
  (choice | seq) ('?' | '*' | '+')?
```

e-Macao-16-4-287

Occurrence Indicators

E	exactly one E
E?	zero or one E
E*	zero or more E
E+	one or more E

e-Macao-16-4-288

Choice Content

`choice` is a list of content particles `cp` separated by "|".

The list contains at least two particles, all in brackets:

```
choice ::=
  '(' S? cp ( S? '|' S? cp )+ S? ')'
```

e-Macao-16-4-289

Sequence Content

`seq` is a list of content particles `cp` separated by ",".

The list contains at least one particle, all in brackets:

```
seq ::=
  '(' S? cp ( S? ',' S? cp )* S? ')'
```

e-Macao-16-4-290

Content Particle

Content particle `cp` is one of:

- element name,
- `choice` or
- `seq`

with an optional occurrence indicator:

```
cp ::=
(Name | choice | seq) ('?' | '*' | '+')?
```

e-Macao-16-4-291

Example: Element and Mixed Content 1

(a)	a
(a?)	0,a
(a+)	a,aa,...
(a*)	0,a,aa,...
(a*)*	0,a,aa,...
(a+)?	0,a,aa,...
(a,b)	ab
(a,b)?	0,ab
(a,b?)	a,ab
(a?,b)	b,ab
(a?,b?)	0,a,b,ab
(a?,b?)?	0,a,b,ab
(a,b)*	0,ab,abab,...
(a,b*)	a,ab,abb,...
(a*,b)	b,ab,aab,...

e-Macao-16-4-292

Example: Element and Mixed Content 2

(a*,b*)	0,a,b,ab,aab,abb,...
(a*,b*)*	0,a,aa,b,bb,ab,abab,...
(a|b)	a,b
(a|b)?	0,a,b
(a|b?)	0,a,b
(a?|b?)	0,a,b
(a|b)+	a,b,aa,ab,ba,bb,...
(a?|b)+	a,b,aa,ab,ba,bb,...
(a+|b)+	a,b,aa,ab,ba,bb,...
((a,b)|c)	ab,c
(a|(b,c))	a,bc
(a,(b|c))	ab,ac
((a|b),c)	ac,bc
(#PCDATA|a)*	0,t,a,ta,at,tt,aa,...
(#PCDATA|a|b)*	0,t,a,b,ta,tb,at,bt,tt,ab,...

e-Macao-16-4-293

Demo: Element Content Models

```
> cd "demos/element content models"
> ls
test.xml
> java dom.Counter -v test.xml
```

e-Macao-16-4-294

Task: Divide the Letter

- divide the letter, declare, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (opening, body, closure)>
  <!ELEMENT opening (#PCDATA | customer)*>
  <!ELEMENT body (#PCDATA | product)*>
  <!ELEMENT closure (#PCDATA | manager)*> ...
]>
<letter>
  <opening>...</opening>
  <body>...</body>
  <closure>...</closure>
</letter>
```

e-Macao-16-4-295

Task: Make the Opening Optional

- make the opening optional, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (opening?, body, closure)>
  <!ELEMENT opening (#PCDATA | customer)*>
...
]>
<letter>
  <body>...</body>
  <closure>...</closure>
</letter>
```

e-Macao-16-4-296

Task: Add Enclosures

- add enclosures, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (opening?, body, closure, enclosure*)>
  <!ELEMENT enclosure (#PCDATA)>...
]>
<letter>
  ...
  <enclosure>...</enclosure>
  <enclosure>...</enclosure>
</letter>
```

e-Macao-16-4-297

Task: Add a Carbon Copy List

- add a carbon copy list, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (..., cc*, enclosure*)>
  <!ELEMENT enclosure (#PCDATA)>...
]>
<letter>
  ...
  <cc>...</cc>
  <cc>...</cc>
  <cc>...</cc>
  <enclosure>...</enclosure>
</letter>
```

e-Macao-16-4-298

Next Declaration Type

Reminder:

```
markupdecl ::=
   elementdecl |
   AttlistDecl |
   EntityDecl |
   NotationDecl |
   PI | Comment
```

Attribute list declaration is next. . .

e-Macao-16-4-299

Attribute List Declaration

Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element.
- To establish type constraints for these attributes.
- To provide default values for attributes.

```
AttlistDecl ::=
  '<!ATTLIST' S Name AttDef* S? '>'
```

e-Macao-16-4-300

Attribute Definition

Attribute definition contains:

- the name of the attribute,
- its type and
- default declaration.

All are necessary.

```
AttDef ::= S Name S AttType S DefaultDecl
```

e-Macao-16-4-301

Attribute Types

Attributes are of three kinds:

- a string type
- one of tokenized types
- one of enumerated types

```
AttType ::=
   StringType |
   TokenizedType |
   EnumeratedType
```

e-Macao-16-4-302

String Type of Attributes

The string type may take any literal string as a value:

```
StringType ::= 'CDATA'
```

e-Macao-16-4-303

Tokenized Types of Attributes

There are seven kinds of tokenized types:

```
TokenizedType ::=
'ID' |
'IDREF' | 'IDREFS' |
'ENTITY' | 'ENTITIES' |
'NMTOKEN' | 'NMTOKENS'
```

Such types provide limited value-checking of attributes.

e-Macao-16-4-304

Tokenized Types of Attributes

Reminder:

```
Name ::= (Letter | '_' | ':') (NameChar)*
```

1. `ID`:
   - the value of this attribute must match `Name`
   - the value is unique in the whole document
   - only one such attribute exists for any element
2. `IDREF` – the value of this attribute must match the value of some ID attribute (in `Name`).
3. `ENTITY` – the value of this attribute must be the name of an unparsed entity declared in the DTD (in `Name`).

e-Macao-16-4-305

Tokenized Types of Attributes

Reminder:

```
Names ::= Name (S Name)*
```

4. `IDREFS` – the value must be in `Names`, each name must match the value of the ID attribute on some element in the document.
5. `ENTITIES` – the value must be in `Names`, each name is the name of an unparsed entity declared in the DTD.

e-Macao-16-4-306

Tokenized Types of Attributes

Reminder:

```
Nmtoken ::= (NameChar)+
Nmtokens ::= Nmtoken (S Nmtoken)*
```

6. `NMTOKEN` – the value must be in `Nmtoken`.
7. `NMTOKENS` – the value must be in `Nmtokens`.

e-Macao-16-4-307

Enumeration Types of Attributes

There are two kinds of enumerated types:

```
EnumeratedType ::=
  NotationType |
  Enumeration
```

e-Macao-16-4-308

Enumeration Type of Attributes

Values of the attribute of the `Enumeration` type must match one of the `Nmtoken` tokens in the declaration:

```
Enumeration ::=
  '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
```

The same `Nmtoken` should not occur more than once in the enumerated attribute types of a single element type.

e-Macao-16-4-309

Notation Type of Attributes

The value of the attribute of the `Notation` type is the name of a notation declared in the DTD.

```
NotationType ::=
  'NOTATION' S
  '(' S? Name (S? '|' S? Name)* S? ')'
```

e-Macao-16-4-310

Notations

Notations are used to interpret the content (usually non-XML) of the element to which this attribute is attached.

- all notation names in the declaration must be declared
- no element may have more than one `NOTATION` attribute

e-Macao-16-4-311

Default Declarations

Reminder:

```
AttDef ::= S Name S AttType S DefaultDecl
```

Default declaration informs:

- whether the attribute's presence is required
- if not, how an XML processor should react if a declared attribute is absent in a document.

e-Macao-16-4-312

Default Declarations

Syntax:

```
DefaultDecl ::=
   '#REQUIRED' | '#IMPLIED' |
   (('#FIXED' S)? AttValue)
```

Four possibilities:

1. `#REQUIRED` – the attribute must always be provided
2. `#IMPLIED` – the attribute is optional, no default is provided

e-Macao-16-4-313

Default Declarations

3. `#FIXED` – the attribute is optional. If present, its value must equal the default value `AttValue`:

```
AttValue ::=
   '"' ([^<&"] | Reference)* '"' |
   "'" ([^<&'] | Reference)* "'"
```

4. otherwise – the attribute is optional:
   - if present, its value is the one given
   - if absent, its value is given by the default `AttValue`

e-Macao-16-4-314

Example: Attribute Declarations

```
<document number="AB45673"/>
<document number="999 999"/>

<document/>
<document number="c999999"/>

<document/>
<document type="passport"/>

<document/>
<document type="id"/>

<document number="AB45673"
children="CD32567 GH12658"/>
```

```
<!ATTLIST document
   number CDATA #REQUIRED>

<!ATTLIST document number ID #IMPLIED>

<!ATTLIST document type
   (passport | id) "id">

<!ATTLIST document type
   (passport | id) #FIXED "id">

<!ATTLIST document number
   ID #REQUIRED children IDREFS "">
```

e-Macao-16-4-315

Demo: Attribute Declarations

```
> cd "demos/attribute declarations"
> ls
documents.xml
> java dom.Counter -v documents.xml
```

e-Macao-16-4-316

Task: Add Product's Name Attribute

```
<!DOCTYPE letter [ ...
   <!ELEMENT product EMPTY>
   <!ATTLIST product name CDATA #REQUIRED>
]>
<letter>
   ...
   <body>
    We are pleased to inform you that your
    <product name="credit card"/>
    application has been approved.
   </body>
</letter>
```

e-Macao-16-4-317

Task: Add Product's ID Attribute

```
<!DOCTYPE letter [ ...
   <!ATTLIST product
    name CDATA #REQUIRED
    id ID #REQUIRED>
]>
<letter>
   ...
   <body>
    We are pleased to inform you that your
    <product name="credit card" id="ab2345"/>
    application has been approved.
   </body>
</letter>
```

e-Macao-16-4-318

Task: Add a Reference to Product's ID

```
<!DOCTYPE letter [ ...
   <!ELEMENT enclosure (#PCDATA)>
   <!ATTLIST enclosure idref IDREF #IMPLIED>
]>
<letter>
   ...
   <body>
    ... <product name="credit card" id="ab2345"/> ...
   </body>
   <enclosure idref="ab2345">credit card</enclosure>
</letter>
```

e-Macao-16-4-319

Task: Add the Officer's Level

```
<!DOCTYPE letter [ ...
   <!ELEMENT officer (#PCDATA)>
   <!ATTLIST officer level
    (clerk | assistant | manager) #IMPLIED "clerk">
]>
<letter>
   ...
   <closure>
    Sincerely,
    <officer level="manager">Steven Rod</officer>
   </closure>
</letter>
```

e-Macao-16-4-320

Next Declaration Type

Reminder:

```
markupdecl ::=
  elementdecl |
  AttlistDecl |
  EntityDecl |
  NotationDecl |
  PI | Comment
```

Entity declaration is next. . .

e-Macao-16-4-321

Entities

Entities are a convenient way to represent information that:

- either occurs repeatedly or
- is expected to change

The information can reside in both XML instances and DTDs.

e-Macao-16-4-322

Role of Entities

An entity may represent:

1. a block of repeated text
2. a special character
3. a constant (fixed) string
4. an entire XML document
5. part of an XML document
6. part of a larger DTD
7. a content model in a DTD
8. a set of attributes in a DTD
9. a binary image
10. etc.

e-Macao-16-4-323

Pre-defined Entities

We have already seen two kinds of entities:

- pre-defined entities: `amp`, `apos`, `quot`, `gt`, `lt` to represent markup characters `&`, `'`, `"`, `>` and `<`.
- character entities: `#x0144` (hexadecimal) and `#324` (decimal) to refer to the Unicode characters, in this case to ń.

They are a special kind of pre-defined entities.

e-Macao-16-4-324

Entity Declaration

Other entities have to be declared in a DTD.

Even though may be uninterested in validation, you need a DTD to have user-defined entities!

If the same entity is declared more than once, the first declaration encountered is binding. XML processor may then issue a warning.

e-Macao-16-4-325

Types of Entities

Two kinds of entities exist:

1. general entity – defined in DTD but used in XML
2. parameter entity – defined and used in DTD

```
EntityDecl ::= GEDecl | PEDecl
```

A parameter entity and a general entity with the same name are two distinct entities; they use different namespaces.

e-Macao-16-4-326

General Entities

They are declared in a DTD (internal or external subset) as follows:

```
GEDecl ::=
  '<!ENTITY' S Name S EntityDef S? '>'
```

General entities are further divided into internal and external:

```
EntityDef ::=
  EntityValue | (ExternalID NDataDecl?)
```

e-Macao-16-4-327

Internal Entities

Internal entities: replacement text defined within a document and referenced from one or more locations in this document

- declared like this:

```
EntityDef ::= EntityValue
EntityValue ::=
  '"' ([^%&"] | PEReference | Reference)* '"' |
  "'" ([^%&'] | PEReference | Reference)* "'"
```

- referred like this: `&name;`

`PEReference` is a reference to parameter entity.

e-Macao-16-4-328

Example: Declaration of Internal Entities

```
   text replacement -> <!ENTITY type "XML editor">
 markup replacement -> <!ENTITY product "
' used instead of " -> <product type='&type;'>
                          XML Typewriter
                       </product>
replacement text with: ">
 1.entity reference -> <!ENTITY disclaimer
                          "&company; accepts no responsibility
                          for damages due to corruption or loss
                          of your data caused by &product;."
                       >
2.predefined entity -> <!ENTITY company "First&amp;Best">
 3.character entity -> <!ENTITY copyright "Copyright &#xA9;
                          2003">
```

e-Macao-16-4-329

Example: Replacement of Internal Entities

```
&product; is the best
&type; available on
the market. It helps
writing XML without
worrying about those
silly syntax rules.

However, &disclaimer.

&copyright
```

```
<product type="XML editor"> XML
Typewriter </product> is the best
XML editor available on the market.
It helps writing XML without worrying
about those silly syntax rules.

However, First&Best accepts no
responsibility for damages due to
corruption or loss of your data
caused by <product type="XML editor">
XML Typewriter </product>.

Copyright © 2003
```

e-Macao-16-4-330

Example: Validation of Internal Entities

```
                       <!DOCTYPE advert [
entity declarations ->  <!ENTITY type "...">
                        <!ENTITY product "...">
                        <!ENTITY disclaimer "...">
                        <!ENTITY copyright "...">
element declarations -> <!ELEMENT advert (#PCDATA | product)*>
                        <!ELEMENT product (#PCDATA)>
                        <!ATTLIST product type CDATA #REQUIRED>
                       ]>
        root element -> <advert>
                          &product; is the best &type; available
                          on the market. It helps writing XML
                          without worrying about those silly
                          syntax rules.
                          However, &disclaimer;. &copyright;
                       </advert>
```

e-Macao-16-4-331

Demo: Internal Entities

```
> cd "demos/internal entities"
> ls
advert.xml
> emacs advert.xml
> java dom.Counter -v advert.xml
```

e-Macao-16-4-332

Well-Formed Internal Entities

An internal general parsed entity is well-formed if its replacement text matches content:

```
content ::= CharData?
  ( (element | Reference | CDSect | PI | Comment)
  CharData?
  )*
```

The replacement text is the content of the entity, after replacement of character references and parameter-entity references.

e-Macao-16-4-333

Example: Well-Formed Internal Entities

```
   parameter entity ->  <!ENTITY % pub "O'Reilly &amp; Associates" >
     general entity ->  <!ENTITY rights "All rights reserved" >
     general entity ->  <!ENTITY book
                           "Learning XML: Erik T. Ray,
character reference ->     &#xA9; 1947
parameter reference ->     %pub;.
  general reference ->     &rights;" >

  replacement text for
the entity ''book'' ->  Learning XML: Erik T.Ray, © 2001
                        O'Reilly & Associates. &rights;
```

e-Macao-16-4-334

Pre-Defined Entities Revisited

Character entities are instances of internal general entities, so are the predefined entities `amp`, `quot`, `apos`, `lt` and `gt`.

All XML processors must recognize them whether they are declared or not. If declared, the following should be used:

```
<!ENTITY lt "&#38;#60;">
<!ENTITY gt "&#62;">
<!ENTITY amp "&#38;#38;">
<!ENTITY apos "&#39;">
<!ENTITY quot "&#34;">
```

Notice the double-escape for `lt` and `amp`:

```
lt -> &#38;#60; -> &#60; -> <
```

e-Macao-16-4-335

Task: Customize Decision – Approved

```
<!DOCTYPE letter [ ...
   <!ENTITY polite "are pleased">
   <!ENTITY decision "approved">
]>
<letter decision="reject">
   ...
   <body>
    We &polite; to inform you that your
    <product>credit card</product>
    application has been &decision;.
   </body>
</letter>
```

e-Macao-16-4-336

Task: Customize Decision – Rejected

```
<!DOCTYPE letter [ ...
   <!ENTITY polite "regret">
   <!ENTITY decision "rejected">
]>
<letter decision="reject">
   ...
   <body>
    We &polite; to inform you that your
    <product>credit card</product>
    application has been &decision;.
   </body>
</letter>
```

e-Macao-16-4-337

External Entities

Reminder:

```
EntityDef ::= EntityValue | (ExternalID
  NDataDecl?)
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
```

External entities match `ExternalID NDataDecl?`.

e-Macao-16-4-338

Types of External Entities

They refer to the data (XML or non-XML) stored in an external file:

- external parsed entity – without `NDataDecl`

  ```
  EntityDef ::= ExternalID
  ```

- external unparsed entity – with `NDataDecl`

  ```
  EntityDef ::= ExternalID NDataDecl?
  ```

e-Macao-16-4-339

External Parsed Entities

External parsed entity is used to include fragments of XML from an external file into the current XML document.

Those fragments:

- need not be well-formed
- cannot contain document type declarations

A useful technique for constructing large XML documents from a set of smaller files and sharing content in many XML documents.

e-Macao-16-4-340

Well-Formed External Parsed Entities

An external parsed entity is well-formed if the content of its replacement file matches the production `extParsedEnt`:

```
extParsedEnt ::= TextDecl? Content
```

Such files may begin with a text declaration:

```
TextDecl ::=
   '<?xml' VersionInfo? EncodingDecl S? '?>'
```

Similar to the XML declaration but the version declaration is optional and the standalone declaration is forbidden.

e-Macao-16-4-341

Example: External Parsed Entities

```
                <?xml version="1.0"?>
                <!DOCTYPE card SYSTEM "card.dtd" [
external ->       <!ENTITY visitor SYSTEM "visitor.xml">
  parsed ->       <!ENTITY document SYSTEM "document.xml">
  entity ->       <!ENTITY addresses SYSTEM "addresses.xml">
declarations ->   <!ENTITY travel SYSTEM "travel.xml">
                ]>
                <card type="arrival">
external ->       &visitor;
  parsed ->       &document;
  entity ->       &addresses;
references ->     &travel;
                  <signature sigfile="mysig"/>
                </card>
```

e-Macao-16-4-342

Example: External Parsed Entities Fragments

visitor.xml ->
well-formed
text declaration:
UTF-8 encoding

```
<?xml encoding="UTF-8"?>
<visitor>
  <name type="surname">Kowalski</name>
  <name type="given">Jan</name>
  ...
</visitor>
```

addresses.xml ->
ill-formed
text declaration:
UTF-16 encoding

```
<?xml encoding="UTF-16"?>
<address where="home">
  Morska 24B, 24-650 Gda&#x0144;sk,
  Poland
</address>
<address where="Macao">
  Flower Garden 12B, Taipa
</address>
```

e-Macao-16-4-343

Demo: External Parsed Entities

```
> cd "demos/external parsed entities"
> ls
card.xml card.dtd visitor.xml
document.xml addresses.xml travel.xml
> emacs *.xml
> java dom.Counter -v card.xml
> java dom.Counter visitor.xml
> java dom.Counter document.xml
> java dom.Counter addresses.xml
> java dom.Counter travel.xml
```

e-Macao-16-4-344

Task: Letter Body as External Parsed Entity

```
<!DOCTYPE letter [
   ...
   <!ELEMENT body (#PCDATA | product)*>
   <!ENTITY body SYSTEM "body.xml">
]>
<letter>
   <opening>...</opening>
   &body;
   <closure>...</closure>
</letter>
```

e-Macao-16-4-345

Next Declaration Type

Before we introduce unparsed entities a detour to XML Notations.

Reminder:

```
markupdecl ::=
   elementdecl |
   AttlistDecl |
   EntityDecl |
   NotationDecl |
   PI | Comment
```

e-Macao-16-4-346

Notations

Notations identify by name:

- the format of unparsed entities,
- the format of elements which bear a notation attribute, or
- the application to which a processing instruction is addressed

e-Macao-16-4-347

Notation Declarations

Notations are declared as follows:

```
NotationDecl ::=
  '<!NOTATION' S
  Name S (ExternalID | PublicID) S? '>'
```

e-Macao-16-4-348

Notation Declarations

System literal (typically URL or MIME type), public literal (`-//org//desc//lang`), or both:

```
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
PublicID ::= 'PUBLIC' S PubidLiteral
```

Only one notation can be defined for a given name.

e-Macao-16-4-349

Notations and XML Processors

XML processors:

- must provide applications with the name and external identifiers of any notation used
- may resolve the external identifier to find a processor for data in the notation described

It is not an error if an XML document declares a notation for which no notation-specific processor is available on the system.

e-Macao-16-4-350

Example: Notation Declarations

1. MIME type for jpg	`<!NOTATION jpeg SYSTEM "image/jpg">`
2. MIME type for troff-encoded text	`<!NOTATION troff SYSTEM "application/x-troff">`
3. application to process data	`<!NOTATION gif SYSTEM "/usr/local/bin/xv">`
4. URL to a technical document about formats	`<!NOTATION date SYSTEM "http://www.w3.org/TR/date.pdf">`
5. A formal public identifier to an online resource	`<!NOTATION date PUBLIC "-//W3C//Date Format//EN" SYSTEM "http://www.w3.org/TR/date">`

e-Macao-16-4-351

External Unparsed Entities

Reminder:

```
EntityDef ::= EntityValue | (ExternalID
  NDataDecl?)
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
```

External unparsed entity declaration contains `ExternalID` as well as `NDataDecl`, where `Name` is the declared name of a notation:

```
NDataDecl ::= S 'NDATA' S Name
```

e-Macao-16-4-352

External Unparsed Entities: Declaration

External unparsed entity is used to incorporate non-XML, typically binary data into the XML document:

```
<!ENTITY view SYSTEM "view.jpg" NDATA jpeg>
```

It is declared in conjunction with a declared notation:

```
<!NOTATION jpeg SYSTEM "image/jpg">
```

e-Macao-16-4-353

External Unparsed Entities: Use

It is used as the value of the attribute of type `ENTITY` or `ENTITIES`, not via the `&view;` reference:

```
<!ELEMENT product (#PCDATA)>
<!ATTLIST product
   img ENTITY #IMPLIED>
```

```
<product img='view' ...>
   XML Typewriter
</product>
```

e-Macao-16-4-354

Example: External Unparsed Entities

```
                        <!DOCTYPE advert [
        root element ->   <!ELEMENT advert (#PCDATA | product)*>
     'product' child ->   <!ELEMENT product (#PCDATA)>
 attributes of product:   <!ATTLIST product
      text attribute ->        type CDATA #REQUIRED
    entity attribute ->        img ENTITY #IMPLIED>
notation declaration ->   <!NOTATION jpeg SYSTEM "image/jpg">
     unparsed entity ->   <!ENTITY view SYSTEM
                               "view.jpg" NDATA jpeg>
                          ...
        unparsed entity ]>
     reference as value <advert>
  of 'img' attribute ->  <product img='view' ...>
                          XML Typewriter
                         </product> is the best &type; ...
                        </advert>
```

e-Macao-16-4-355

Demo: External Unparsed Entities

```
> cd "demos/external unparsed entities"
> ls
advert.xml
> emacs advert.xml
> java dom.Counter -v advert.xml
```

e-Macao-16-4-356

Task: Signature as External Unparsed Entity

```
<!DOCTYPE letter [ ...
   <!ATTLIST officer
        level (clerk | assistant | manager) "clerk"
        signature ENTITY #IMPLIED>
   <!ENTITY rod SYSTEM "rod.jpeg" NDATA jpeg>
   <!NOTATION jpeg SYSTEM "image/jpg">
]>
<letter> ...
   Sincerely,
   <officer level="manager" signature="rod">
   Steven Rod
   </officer> ...
</letter>
```

e-Macao-16-4-357

Parameter Entities

Reminder:

```
EntityDecl ::= GEDecl | PEDecl
```

Parameter entities provide a reuse mechanism for building a DTD.

e-Macao-16-4-358

Parameter Entities

They are declared and used in DTD only:

- declaration:
  ```
  PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
  PEDef ::= EntityValue | ExternalID
  ```
- reference: `%name;`

Parameter-entity replacement text must be properly nested with markup declarations (and parenthesized groups).

e-Macao-16-4-359

Parameter versus General Entities

Compare with general entities:

1. declaration of parameter entity uses `%` after `ENTITY`
2. declaration of parameter entity has no `NDataDecl`: there are no unparsed external parameter entities
3. parameter entities are references `%name;` not `&name;`
4. parameter entity reference is recognized in DTD only

e-Macao-16-4-360

Internal and External Parameter Entities

- internal entities – to define internally a replacement text for building a DTD from reusable pieces

```
PEDecl ::=
  '<!ENTITY' S '%' S Name S EntityValue S? '>'
```

- external entities – bring DTD content from an outside file

```
PEDecl ::=
  '<!ENTITY' S '%' S Name S ExternalID S? '>'
```

e-Macao-16-4-361

Internal Parameter Entity Usage

Common usage scenarios:

1. to serve as documentation of an intended specific datatype for an attribute that is actually declared as `CDATA` or `NMToken`:

```
<!ENTITY % Margin "CDATA">
<!ENTITY % Color "CDATA">
<!ENTITY % URI "CDATA">
```

e-Macao-16-4-362

Internal Parameter Entity Usage

2. to specify an element content model that is common in a DTD:

```
<!ENTITY % fontstyle "tt | i | b | big | small">
<!ENTITY % phrase "em | strong | code | cite |
   code">
<!ENTITY % inline "a | %fontstyle; | %phrase;">
```

e-Macao-16-4-363

Internal Parameter Entity Usage

3. to collect a group of related attribute declarations that are used repeatedly in various content models in a DTD:

```
<!ATTLIST frame %standard; %color; %margins;>
<!ENTITY % standard "
   id ID #IMPLIED
   xml:base %URI; #IMPLIED">
<!ENTITY % color "
   background %Color; #IMPLIED
   foreground %Color; #IMPLIED">
<!ENTITY % margins "
   left %Margin; #IMPLIED
   right %Margin; #IMPLIED">
```

e-Macao-16-4-364

Parameter Entity References

In entity declarations, as entity value:

```
EntityValue ::=
  '"' ([^%&"] | PEReference | Reference)* '"' |
  "'" ([^%&'] | PEReference | Reference)* "'"
```

This case is only allowed in the external DTD subset (`ExternalID`), not the internal subset (`markupdecl`):

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

e-Macao-16-4-365

Parameter Entity References

In document type declarations, as markup separators:

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  '[' (markupdecl | DeclSep)* ']' S?)? '>'
DeclSep ::= PEReference | S
```

This case is allowed in both internal and external DTD subsets.

e-Macao-16-4-366

Example: External Parameter Entities

```
                    <?xml version="1.0"?>
                    <!DOCTYPE card SYSTEM "card.dtd">
                    <card type="arrival">
external general ->    &visitor;
entity references ->   &document;
                ->     &addresses;
                ->     &travel;
                       <signature sigfile="mysig"/>
                    </card>
```

e-Macao-16-4-367

Example: External Parameter Entities

```
internal parameter entity
           declaration ->  <!ENTITY % details "document, address, ...">
             reference ->  <!ELEMENT card (visitor, %details;, ...)>
external parameter entity
           declaration ->  <!ENTITY % visitor SYSTEM "visitor.dtd">
                       ->  <!ENTITY % document SYSTEM "document.dtd">
                       ->  <!ENTITY % address SYSTEM "address.dtd">
                       ->  <!ENTITY % travel SYSTEM "travel.dtd">
             reference ->  %visitor;
                       ->  %document;
                       ->  %address;
                       ->  %travel;
  external general entity
           declaration ->  <!ENTITY visitor SYSTEM "visitor.xml">
                       ->  <!ENTITY document SYSTEM "document.xml">
                       ->  <!ENTITY addresses SYSTEM "addresses.xml">
                           <!ENTITY travel SYSTEM "travel.xml">
```

e-Macao-16-4-368

Demo: Parameter Entities

```
> cd "demos/parameter entities"
> ls
card.xml visitor.xml document.xml
addresses.xml travel.xml
card.dtd date.dtd visitor.dtd document.dtd
address.dtd travel.dtd signature.dtd
> java dom.Counter -v card.xml
```

e-Macao-16-4-369

Task: Officer Level as Parameter Entity

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ENTITY % levels "clerk | assistant | manager">
  <!ATTLIST officer level (%levels;) "clerk">
]>
<letter>...</letter>
```

Doesn't work?

e-Macao-16-4-370

Task: Officer Level as Parameter Entity

XML document:

```
<?xml version="1.0"?>
<!DOCTYPE letter SYSTEM "letter.dtd">
<letter>...</letter>
```

DTD document:

```
...
<!ENTITY % levels "clerk | assistant | manager">
<!ATTLIST officer level (%levels;) "clerk">
```

e-Macao-16-4-371

Task: Entity Declarations in External File

DTD document:

```
<!ENTITY % entities SYSTEM "entities.dtd">
%entities;
...
```

Another DTD document:

```
<!ENTITY % levels "clerk | assistant | manager">
<!ENTITY polite "are sorry">
<!ENTITY decision "rejected">
```

e-Macao-16-4-372

Well-Formed Entities

- internal general entity – replacement text (text after expansion of character and parameter references) matches content

```
content ::=
  CharData?
  (
   ( element |
      Reference |
      CDSect |
      PI |
      Comment)
    CharData?
  )*
```

e-Macao-16-4-373

More Well-Formed Entities

- external general entity – it matches `extParsedEnt`

```
extParsedEnt ::= TextDecl? Content
```

- parameter entities are well-formed by definition

e-Macao-16-4-374

Nesting of Logical and Physical Structures

The consequence of well-formedness is that the logical and physical structures in an XML document are properly nested.

None of:

start-tag	comment
end-tag	processing instruction
empty-element tag	character reference
element	entity reference

can begin in one entity and end in another.

e-Macao-16-4-375

External DTD Fragments

External DTD fragments are linked from XML document in two ways:

- As replacement text of an external parameter entity reference, occurring as the declaration separator (`DeclSep`), matching:

```
extSubsetDecl ::=
   (markupdecl | conditionalSect | DeclSep)*
```

A list of markup declarations, conditional sections and declaration separators.

e-Macao-16-4-376

External DTD Fragments

- as `ExternalID` in the document type declaration:

```
doctypedecl ::=
 '<!DOCTYPE' S Name (S ExternalID)? S?
 ('[' (markupdecl | DeclSep)* ']' S?)? '>'
 DeclSep ::= PEReference | S
```

This is called the external subset. It has to match:

```
extSubset ::= TextDecl? extSubsetDecl
```

A replacement text for external parameter references, with an optional text declaration.

e-Macao-16-4-377

Conditional Sections

Portions of the document type declaration external subset which are included in, or excluded from, the DTD:

```
conditionalSect ::=
  '<![' S? conditionTest S? '[' extSubsetDecl
  ']]>'
```

e-Macao-16-4-378

Conditional Sections Tests

The test is one of:

- `INCLUDE`
- `IGNORE` or
- a parameter entity reference who may be either of them

```
conditionTest ::=
    'INCLUDE' |
    'IGNORE' |
    PEReference
```

e-Macao-16-4-379

Example: Conditional Sections

always include ->
```
<![INCLUDE[
   <!ELEMENT travel (place, flight?)>
   <!ATTLIST travel type (from) #IMPLIED>
]]>
```

always ignore ->
```
<![IGNORE[
   <!ELEMENT travel (place, flight?)>
    <!ATTLIST travel type (to) #IMPLIED>
]]>
```

parameter entity ->
```
<!ENTITY % direction "IGNORE">
```

include or ignore ->
```
<![%direction;[
   <!ELEMENT travel (place, flight?)>
   <!ATTLIST travel type (to) #IMPLIED>
]]>
```

e-Macao-16-4-380

Task: Conditional Acceptance/Rejection

```
<!ENTITY % accept "INCLUDE">
<!ENTITY % reject "IGNORE">

<![%accept;[
  <!ENTITY polite "are pleased">
  <!ENTITY decision "approved">
]]>

<![%reject;[
  <!ENTITY polite "are sorry">
  <!ENTITY decision "rejected">
]]>
```

e-Macao-16-4-381

XML Processors

XML processors: validating and non-validating.

Both must report violations of well-formedness encountered in the main document and any other parsed entity that they read.

- validating processors must report violations of the constraints expressed by the declarations in the DTD. They must read and process the entire DTD and all external parsed entities referenced in the document.
- non-validating processors have to check the main document, including the entire internal DTD subset, for well-formedness.

A.2.4. Namespaces

Namespaces

e-Macao-16-4-383

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-384

Mixing Vocabularies

Suppose we want to embed SVG image in a DocBook document.

Both SVG and DocBook have the `title` elements.

How to distinguish them?

```
<book>
 <title>SVG Manual</title>
 <chapter>
  ...
  <mediaobject>
   <imageobject>
    <svg>
     <title>SVG Example</title>
     ...
    </svg>
   </imageobject>
  </mediaobject>
 </chapter>
</book>
```

In general, how to mix different vocabularies in an XML document to avoid name conflicts?

e-Macao-16-4-385

Name Disambiguation with URI references

Solution: leverage the uniqueness of Universal Resource Identifiers (URI) ensured by the DNS (Domain Name System):

- SVG – http://www.w3.org/TR/SVG
- DocBook – http://www.oasis-open.org/docbook
- XML Schema - http://www.w3.org/2001/XMLSchema
- etc.

Apply URI as a prefix to element/attribute names.

e-Macao-16-4-386

URI as Name Prefixes

Consider this:

```
<book>
 <http://www.oasis-open.org/docbook:title>
  SVG Manual
 </http://www.oasis-open.org/docbook:title>
 ...
 <http://www.w3.org/TR/SVG:title>
  SVG Example
 </http://www.w3.org/TR/SVG:title>
</book>
```

Two problems with this solution:

- element names become very long
- URIs include characters not allowed in XML names

e-Macao-16-4-387

XML Namespaces

Instead, associate URI with a short name – namespace – then use this namespace as a prefix to qualify element/attribute names.

```
<book
  xmlns:docbook="http://www.oasis-open.org/docbook"
  xmlns:svg="http://www.w3.org/TR/SVG">
  ...
  <docbook:title>SVG Manual</docbook:title>
  ...
  <svg:title>SVG Example</svg:title>
  ...
</book>
```

e-Macao-16-4-388

Namespace Declaration

A namespace is declared using a family of reserved attributes. Such attribute names have `xmlns` or `xmlns:` as a prefix.

```
NSAttName ::= PrefixedAttName | DefaultAttName
```

Two kinds of namespace declarations:

– regular – include a prefix

```
PrefixedAttName ::= 'xmlns:' NCName
```

– default – without a prefix

```
DefaultAttName ::= 'xmlns'
```

e-Macao-16-4-389

Namespace Prefix

`NCName` is the namespace prefix - any legal XML name without a colon:

```
NCName ::= (Letter | '_') (NCNameChar)*
NCNameChar ::=
  Letter | Digit | '.' | '-' | '_' |
  CombiningChar | Extender
```

Prefixes beginning with the sequence `xml` in any case combination, are reserved by XML and XML-related specification.

e-Macao-16-4-390

Namespace Value

The value of the namespace attribute is the namespace name:

- this should be a URI reference
- empty string is also allowed, but only for a default namespace
- the URI may, but need not, point to an existing address
- typically, the namespace points to formal description of a vocabulary: W3C Recommendation, DTD or XML Schema

Example: Namespace declarations

```
xmlns:docbook="http://www.oasis-open.org/docbook"
xmlns:svg="http://www.w3.org/TR/SVG"
xmlns="http://www.w3.org/TR/REC-xml-names"
xmlns=""
```

e-Macao-16-4-391

Example: Namespace Declarations

```
xmlns:docbook="http://www.oasis-open.org/docbook"
xmlns:svg="http://www.w3.org/TR/SVG"
xmlns="http://www.w3.org/TR/REC-xml-names"
xmlns=""
```

e-Macao-16-4-392

Qualified Names

A qualified name is an XML name without ":", or with one ":" separating the prefix and the local part:

```
QName ::= (Prefix ':')? LocalPart
Prefix ::= NCName
LocalPart ::= NCName
```

The prefix must be declared in a namespace declaration.

e-Macao-16-4-393

Example: Qualified Names

```
<book xmlns:docbook="http://www.oasis-open.org/docbook">
       <docbook:title>
                …
       </docbook:title>
       …
</book>
```

e-Macao-16-4-394

Element and Attributes Revisited

Start tags, end tags and empty element tags with qualified names:

```
STag ::= '<' QName (S Attribute)* S? '>'
ETag ::= '</' QName S? '>'
EmptyElemTag ::= '<' QName (S Attribute)* S? '/>'
```

Attributes are either namespace declarations or their names are given as qualified names:

```
Attribute ::=
  NSAttName Eq AttValue | QName Eq AttValue
```

e-Macao-16-4-395

Namespace Constraint

The namespace prefix used in element- and attribute-names:

- `xmlns` is used only for namespace binding and is not itself bound to any namespace name
- `xml` is by definition bound to `http://www.w3.org/XML/1998/namespace`
- any other prefix must have been declared:
  - in the start-tag of the element where the prefix is used, or
  - in an ancestor element

e-Macao-16-4-396

DTDs Revisited 1

Qualified names appear in:

1) document type declarations, to refer to the root element:

```
doctypedecl ::= '<!DOCTYPE' S
 QName (S ExternalID)? S?
 ('[' (markupdecl | PEReference | S)* ']' S?)? '>'
```

e-Macao-16-4-397

DTDs Revisited 2

2) in element declarations, content particles and mixed models:

```
elementdecl ::=
   '<!ELEMENT' S QName S contentspec S? '>'

cp ::= (QName | choice | seq) ('?' | '*' | '+')?

Mixed ::=
   '(' S? '#PCDATA' S? ')' |
   '(' S? '#PCDATA' (S? '|' S? QName)* S? ')*'
```

e-Macao-16-4-398

DTDs Revisited 3

3) in attribute list declarations and attribute definitions:

```
AttlistDecl ::=
   '<!ATTLIST' S QName AttDef* S? '>'

AttDef ::=
   S (QName | NSAttName) S AttType S DefaultDecl
```

e-Macao-16-4-399

Location of Namespace Declarations

Due to namespace declaration constraints on XML documents, namespace declarations should be provided:

- directly in the XML document
- via a default attribute declared in the internal DTD subset

e-Macao-16-4-400

Namespace Scoping

The namespace declaration applies to:

- the element where it is specified and
- to descendants of this element, unless overridded by another namespace declaration with the same `NSAttName` part.

Multiple namespace can be declared within a single element.

e-Macao-16-4-401

Example: Namespace Scoping

```
http://www.nul1.org -> <nul:book xmlns:nul="http://www.nul1.org">
                            …
                 none ->    <title>
                               …
                            </title>
                 none ->    <chap xmlns:nul="http://www.nul2.org">
                               …
http://www.nul2.org ->         <nul:title>
                                  …
                               </nul:title>
                            </chap>
http://www.nul1.org ->      <nul:chap>
                               …
                            </nul:chap>
                         </nul:book>
```

e-Macao-16-4-402

Example: Multiple Namespaces

```
                 none -> <book
                          xmlns:ns1="http://www.nul1.org"
                          xmlns:ns2="http://www.nul2.org">
http://www.nul1.org ->    <ns1:title>...</ns1:title>
http://www.nul2.org ->    <ns2:title>...</ns2:title>
http://www.nul2.org ->    <ns3:chap
                                xmlns:ns3="http://www.nul2.org">
                           <title
                 ERROR ->   ns2:status=...
                       ->   ns3:status=... >
                           ...
                           </title>
                          </ns3:chap>
                         </book>
```

e-Macao-16-4-403

Default Namespace

A default namespace applies to:

- the element where it is declared, provided it has no prefix
- to all its descendants with no prefix

If the URI reference is empty, then un-prefixed elements within its scope are considered not to be in any namespace.

Default namespaces do not apply to attributes.

e-Macao-16-4-404

Example: Default Namespace

```
 http://www.nul.org -> <book
                         xmlns="http://www.nul.org"
                         xmlns:ns1="http://www.nul1.org"
                         xmlns:ns2="http://www.nul2.org">
 http://www.nul.org ->   <title>...</title>
http://www.nul1.org ->   <ns1:title>...</ns1:title>
http://www.nul2.org ->   <ns2:chap xmlns="">
               none ->    <title>...</title>
                          ...
                        </ns2:chap>
                      </book>
```

e-Macao-16-4-405

Task: Arrival Card with Namespaces

Retype a simple arrival card:

```
<?xml version="1.0"?>
<card>
   <visitor>
    <name>Jan Kowalski</name>
   </visitor>
   <document>passport</document>
</card>
```

1. Declare the default namespace `http://www.macau.gov.mo` for all elements of the document.
2. Declare the namespace `http://www.kowalski.org` for the `visitor` element.
3. Make sure that `name` does not belong to any namespace.

e-Macao-16-4-406

XML Namespace Recommendation

Namespaces in XML:

- W3C Recommendation published in January 1999
- Editors: Tim Bray (Textuality), Dave Hollander (Hewlett-Packard), Andrew Layman (Microsoft)
- abstract:

  XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

e-Macao-16-4-407

XML Namespace Conformance

Conformance to the XML namespace specification:

- element and attribute names contain either zero or one colon
- no entity name, PI target or notation name contains any colon
- values of the attributes of types `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES` and `NOTATION` should contain no colon

A.3. XML Technologies

A.3.1. XML Schema

XML Schema

e-Macao-16-4-409

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-410

DTD Limitations 1-2

1. document-centric – DTDs are a simplification of SGML DTDs, which themselves are document-focused.

   As a result, DTDs are more suited to describe the content of documents, and less to describe structured data.

2. no meta-data access - applications cannot access the content of the DTD, once the document is processed by an XML parser, e.g. via DOM.

   Grouping, sharing and reuse of markup declarations and all metadata information (data about data) in a DTD are lost.

e-Macao-16-4-411

DTD Limitations 3

3. limited datatyping – DTDs provide very limited datatyping

elements	attributes	
EMPTY	CDATA	
PCDATA	ID	NOTATION
element	IDREF	IDREFS
mixed	NMTOKEN	NMTOKENS
ANY	ENTITY	ENTITIES

e-Macao-16-4-412

DTD Limitations 4-5

4. ranges or sets are hard to define - DTD enable enumeration of legal values for attributes and this is helpful for only very small sets – but not element content.

```
<!ATTLIST date dayofweek
 (monday | tuesday | wednesday | thursday |
 friday | saturday | sunday)
 #IMPLIED
>
```

5. no subclassing - DTD does not permit describing common data structures in a class definition, and capturing all variations in subclasses.

e-Macao-16-4-413

DTD Limitations 6-7

6. order of children is too rigid - DTD require us either:
   - list all children elements (optional or not) in the order in which they must occur

     ```
     <!ELEMENT P (A, B+, C?)>
     ```

   - or use a mixed model where no order constraints are imposed

     ```
     <!ELEMENT P (#PCDATA| A, B, C)*>
     ```

7. no namespace support – to check validity, we must keep prefixes in XML in synch with the DTD: if you change one, you have to change the other.

e-Macao-16-4-414

DTD Limitations 8-9

8. limited ways to express repetitions

   element B must occur exactly 15 times:

```
<!ELEMENT P (A,B,B,B,B,B,B,B,B,B,B,B,B,B,B,C)>
```

   element B may occur between 13 and 15 times:

```
<!ELEMENT P
 (A,B,B,B,B,B,B,B,B,B,B,B,B,B,C)
 (A,B,B,B,B,B,B,B,B,B,B,B,B,B,C)
 (A,B,B,B,B,B,B,B,B,B,B,B,B,B,C)>
```

9. DTDs are written in native, non-XML syntax - XML tools cannot be used to process DTD documents.

e-Macao-16-4-415

XML Schema – W3C Recommendation 1

W3C Recommendation May 2001.

Three parts:

1. Primer - A non-normative document intended to provide an easily readable description of the XML Schema language. XML Schema Part 1: Structures and XML Schema Part 2: Datatypes provide the complete normative description of the XML Schema language.

e-Macao-16-4-416

XML Schema – W3C Recommendation 2

2. Structures - XML Schema language offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespaces. The schema language, which is itself represented in XML 1.0 and uses namespaces, considerably extends the capabilities found in DTDs.

3. Datatypes - Defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 DTDs for specifying datatypes on elements and attributes.

e-Macao-16-4-417

Example: XML versus DTD

```
XML instance ->  <?xml version="1.0"?>
    document ->  <!DOCTYPE date SYSTEM "date.dtd">
             ->  <date>
             ->   <day>14</day>
             ->   <month>September</month>
             ->   <year>2003</year>
             ->   <weekday>Sunday</weekday>
             ->  </date>

Document Type ->  <!ELEMENT date (day, month, year, weekday?)>
   Definition ->  <!ELEMENT day (#PCDATA)>
             ->  <!ELEMENT month (#PCDATA)>
             ->  <!ELEMENT year (#PCDATA)>
             ->  <!ELEMENT weekday (#PCDATA)>
```

e-Macao-16-4-418

Example: XML Schema

```
                  <?xml version="1.0"?>
  schema element -> <xsd:schema
schema namespace ->  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    date element ->  <xsd:element name="date">
    complex type ->   <xsd:complexType>
     sequence of ->    <xsd:sequence>
             day ->     <xsd:element name="day" type="xsd:string"/>
           month ->     <xsd:element name="month" type="xsd:string"/>
            year ->     <xsd:element name="year" type="xsd:string"/>
         weekday ->     <xsd:element name="weekday"
      is optional ->       minOccur="0" type="xsd:string"/>
                    </xsd:sequence>
                   </xsd:complexType>
                  </xsd:element>
                 </xsd:schema>
```

e-Macao-16-4-419

Example: XML Referring Schema

```
Location of
the schema
document   ->
Namespace ->
declaration
for XML
instance
documents
```

```
<?xml version="1.0"?>
<date
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="date.xsd"
>
  <day>14</day>
  <month>September</month>
  <year>2003</year>
  <weekday>Sunday</weekday>
</date>
```

e-Macao-16-4-420

Example: XML Referring Schema and DTD

```
                  <?xml version="1.0"?>
  external DTD -> <!DOCTYPE date SYSTEM "date.dtd"
  internal DTD -> [
                   <!ATTLIST date
     attribute ->  xsi:noNamespaceSchemaLocation CDATA #IMPLIED
  declarations ->  xmlns:xsi CDATA #FIXED "...XMLSchema-instance">
                  ]>
                  <date
  schema location  xsi:noNamespaceSchemaLocation="date.xsd"
                ->  xmlns:xsi="...XMLSchema-instance">
     namespace ->   <day>14</day>
      declaration   <month>September</month>
                    <year>2003</year>
                    <weekday>Monday</weekday>
                  </date>
```

e-Macao-16-4-421

Demo: DTD versus Schema Validation

```
> cd "demos/dtd versus schema validation"
> ls
date.xml date.dtd date.xsd
dateDTD.xml dateSchema.xml dateDTDSchema.xml
> java dom.Counter date.xml
> java dom.Counter -v dateDTD.xml
> java dom.Counter -s dateSchema.xml
> java dom.Counter -v dateDTDSchema.xml
> java dom.Counter -v -s dateDTDSchema.xml
```

e-Macao-16-4-422

What Have Been Gained?

So we replace a 5-line DTD with a 14-line XML Schema.

What have been gained?

A solid foundation to build a better schema:

- more stringent datatyping
- reuse of data structures
- more expressive content model
- XML syntax reused
- XML tools reapplied
- self-description and -validation
- etc.

e-Macao-16-4-423

Schema Document Structure

```
root element ->   <schema …>
any number of ->    <include …/>
                    <import>…</import>
                    <redefine>…</redefine>
                    <annotation>…</annotation>

any number of ->    <simpleType>…</simpleType>
                    <complexType>…</complexType>
                    <element>…</element>
                    <attribute …/>
                    <attributeGroup>…</attributeGroup>
                    <group>…</group>
                    <annotation>…</annotation>
                  </schema>
```

e-Macao-16-4-424

Schema Namespace

XML schema standards is namespace-sensitive:

1. it can describe documents with elements and attributes that belong to different namespaces
2. namespaces distinguish between references to built-in data types and other types defined by the schema author

Schema belongs to `http://www.w3.org/2001/XMLSchema` namespace:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  …
</schema>
```

e-Macao-16-4-425

Referring to Schema: No Namespace 1

XML instance document with no namespace:

```
<elem
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schemaNoNamespace.xsd">
  this is text
</elem>
```

The attribute `noNamespaceSchemaLocation` determines the schema location for those elements that do not belong to any namespace.

The attribute belongs itself to the namespace `http://www.w3.org/2001/XMLSchema-instance.`

e-Macao-16-4-426

Referring to Schema: No Namespace 2

XML Schema file `schemaNoNamespace.xsd`:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="elem" type="xsd:string"/>
</xsd:schema>
```

e-Macao-16-4-427

Demo: No Namespace Validation

```
> cd "demos/schema no namespace"
> dir
schemaNoNamespace.xsd noNamespace.xml
> java dom.Counter -v -s noNamespace.xml
```

e-Macao-16-4-428

Comments

In addition to the normal XML comments, annotation elements distinguish between human- and software-aimed comments:

```
<annotation>
   <documentation source="…">
      this is documentation
   </documentation>
   <appinfo source="…">processing instruction</appinfo>
</annotation>
```

The source element includes a URL to the document with further information about the issue.

e-Macao-16-4-429

Element Declarations

Element declaration requires that there exists an element in the instance document:

- whose name is given by the `name` attribute and
- whose content is of the type specified

```
<xsd:element name="date">
  <xsd:complexType>...</xsd:complexType>
</xsd:element>
```

e-Macao-16-4-430

Named and Anonymous Types

The type of the component may be given:

- as anonymous type embedded directly inside `element`:

```
<xsd:element name="date">
  <xsd:complexType>...</xsd:complexType>
</xsd:element>
```

- as the named type referred by `element` via its `type` attribute

```
<xsd:element name="date" type="Date"/>
<xsd:complexType name="Data">
  ...
</xsd:complexType>
```

e-Macao-16-4-431

Named Type Reuse

Named types can be reused by several element declarations:

```
<xsd:element name="date" type="Date"/>
<xsd:element name="mydate" type="Date"/>

<xsd:complexType name="Data">
  ...
</xsd:complexType>
```

Anonymous types are only used within one element declaration.

e-Macao-16-4-432

Top-Level Elements

Element declaration can occur at:

- top level – element must exist in the instance document

```
<xsd:element name="date" type="Date"/>
```

When several top-level element declaration are given, one of the elements must exist in the document.

e-Macao-16-4-433

Local-Level Elements

- local level – part of the type's definition

```
<xsd:complexType name="Data">
  ...
  <xsd:element name="day" type="xsd:string">
  ...
</xsd:complexType>
```

e-Macao-16-4-434

Task: Schema for Arrival Card

Return the arrival card to the no-namespace version:

```
<?xml version="1.0"?>
<card>
   <visitor>
    <name>Jan Kowalski</name>
   </visitor>
   <document>passport</document>
</card>
```

Design the schema for this card.

Refer to the schema from the XML document.

Schema-validate the document.

e-Macao-16-4-435

Element Repetition

Local-level elements can contain repetition attributes:

- `minOccurs` – minimal number of occurrences, default 1

```
<xsd:element name="elem" minOccurs="0"/>
```

- `maxOccurs` – maximum number of occurrences, default 1

```
<xsd:element name="elem"
  minOccurs="2" maxOccurs="unbounded"/>
```

e-Macao-16-4-436

Task: Repeated Names

In the schema allow the visitor to have from one to three names.

Modify the XML document.

Validate.

e-Macao-16-4-437

Element Reference

Top-level elements can be referred when declaring local-level elements; no `name` and `type` attributes are needed.

```
<xsd:element name="day" type="xsd:string"/>

<xsd:complexType name="Date">
  ...
  </xsd:element ref="day"/>
  ...
</xsd:complexType>
```

e-Macao-16-4-438

Task: Top-Level Visitor Element

In the schema define the visitor element on the top-level.

Refer to the visitor from the card type.

Validate.

e-Macao-16-4-439

Simple and Complex Types

Types can be simple or complex:

- simple types describe values of attributes, as well as elements that contain text and attributes (but no children)

```
<xsd:simpleType name="productNumber">
  ...
</xsd:simpleType>
```

- complex types describe elements with text, attributes and children

```
<xsd:complexType name="productSpecification">
  ...
</xsd:complexType>
```

e-Macao-16-4-440

Pre-Defined and User-Defined Types

XML Schema provides 44 pre-defined simple types, usually referred to with a prefix. Other types have to be declared:

- pre-defined type `xsd:string`

```
<xsd:element name="day" type="xsd:string"/>
```

- user-defined type `Date`

```
<xsd:element name="date" type="Date"/>
<xsd:complexType name="Data">
  ...
</xsd:complexType>
```

e-Macao-16-4-441

Pre-Defined Types: DTD and String

DTD type	Schema type	example
ID	string	a practical guide
IDREF	normalizedString	a practical guide
IDREFS	token	a practical guide
ENTITY	Name	my:book, book
ENTITIES	NCName	book
NMTOKEN	QName	my:book
NMTOKENS	language	de, en
NOTATION	anyURI	http://www.iist.unu.edu

e-Macao-16-4-442

Pre-Defined Types: Numeric

Schema type	Schema type
boolean	byte
float	short
double	int
decimal	long
integer	unsignedByte
nonNegativeInteger	unsignedShort
positiveInteger	unsignedInt
negativeInteger	unsignedLong
nonPositiveInteger	base64Binary
	hexBinary

e-Macao-16-4-443

Pre-Defined Types: Date and Time

Schema type	example
duration	P2Y4M7DT10H30M17.5S
date	2003-09-15
time	15:07:01
dateTime	2003-09-15T15:07:00
gYear	2003
gMonth	--02
gYearMonth	2003-09
gDay	---15
gMonthDay	--09-15

e-Macao-16-4-444

Task: Predefined Versus User-Defined Date

In the schema for the arrival card, define date of birth as:

1. pre-defined type
2. use-defined type with separate elements for:

   a) day
   b) month
   c) year

e-Macao-16-4-445

User-Defined Simple Types: Restriction

Given pre-defined simple types, we can derive new simple types.

This can be done by:

- restriction

```
<xsd:simpleType name="myType">
  <xsd:restriction base="simpleType">
    <facet1 value="..."/>
    <facet2 value="..."/>
    <facet3 value="..."/>
  </xsd:restriction>
</xsd:simpleType>
```

e-Macao-16-4-446

User-Defined Simple Types: List and Union

- list

```
<xsd:simpleType name="myType">
  <xsd:list itemType="simpleType">
</xsd:simpleType>
```

- union

```
<xsd:simpleType name="myType">
  <xsd:union memberTypes="simpleType1   simpleType2">
</xsd:simpleType>
```

e-Macao-16-4-447

Facets for Simple Type Restriction

XML Schema defines 12 constraining facets:

string types	numeric types
length	minInclusive
minLength	maxInclusive
maxLength	minExclusive
pattern	maxExclusive
enumeration	totalDigits
whiteSpace	fractionDigits

e-Macao-16-4-448

Facets: MinLength and MaxLength

Strings of 5 to 10 characters:

```
<xsd:simpleType name="string5to10">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="5"/>
    <xsd:maxLength value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

e-Macao-16-4-449

Task: Restricted String Types

In the schema for the arrival card, define the travel document element with the number element.

Define the number element as the string of exactly 10 characters.

e-Macao-16-4-450

Facets: MinInclusive and MaxInclusive

Numbers in the range from 1 to 31:

```
<xsd:simpleType name="days">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="31"/>
  </xsd:restriction>
</xsd:simpleType>
```

e-Macao-16-4-451

Task: Restricted Numerical Types

In the schema for the arrival card, define

a) day
b) month
c) year

as restricted numerical types.

e-Macao-16-4-452

Facets: Enumeration

```
<xsd:simpleType name="weekDays">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Monday"/>
      <xsd:enumeration value="Tuesday"/>
      <xsd:enumeration value="Wednesday"/>
      <xsd:enumeration value="Thursday"/>
      <xsd:enumeration value="Friday"/>
      <xsd:enumeration value="Saturday"/>
      <xsd:enumeration value="Sunday"/>
   </xsd:restriction>
</xsd:simpleType>
```

Only `enumeration` and `pattern` can appear many times.

e-Macao-16-4-453

Task: Enumerated String Types

Inside schema for the arrival card, define the sex element inside the visitor element using string enumeration.

e-Macao-16-4-454

Facets: TotalDigits and FractionDigits

Price type with 2 fraction digits and 8 in total:

```
<xsd:simpleType name="Price">
   <xsd:restriction base="xsd:float">
      <xsd:totalDigits value="8"/>
      <xsd:fractionDigits value="2"/>
   </xsd:restriction>
</xsd:simpleType>
```

e-Macao-16-4-455

Facets: Pattern

Strings of upper-case or lower-case letters (not mixed):

```
<xsd:simpleType name="myName">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]+"/>
      <xsd:pattern value="[a-z]+"/>
   </xsd:restriction>
</xsd:simpleType>
```

The value of a pattern is a regular expression.

e-Macao-16-4-456

Facets: Regular Expressions for Patterns

`*`	zero or more
`+`	one or more
`?`	zero or one
`.`	any character
`(a|b)`	a or b
`(abc)`	sequence of a, b and c
`[abc]`	any or a, b or c
`[^a-z]`	not a letter in the range
`(expr){n}`	expr repeated exactly n times
`(expr){m,n}`	expr repeated from m to n times
`\p{X}`	one character from the Unicode character class X

e-Macao-16-4-457

Task: Patterns for String Types

Inside schema for the arrival card, define the travel document identifier following three constraints:

1. the identifier consists of 10 characters
2. the first two characters are letters
3. the last eight characters are digits

e-Macao-16-4-458

Declarations with Simple Types

- elements can have simple or complex types:

```
<xsd:element name="price" type="Price"/>
<xsd:simpleType name="Price">
   ...
</xsd:simpleType>
```

- attributes always have simple types:

```
<xsd:attribute name="price">
   <xsd:simpleType>
   ...
   </xsd:simpleType>
</xsd:attribute>
```

e-Macao-16-4-459

Attribute Declarations

Similar to element declarations, with `name`, `type` and `ref` attributes.

Attribute declaration is related to the parent element, it must follow the declarations of the element's children:

```
<xsd:complexType name="Collection">
   ...
   <xsd:element ref="book"/>
   <xsd:element ref="CD"/>
   <xsd:attribute name="version" type="xsd:string"/>
</xsd:complexType>
```

Only elements with complex types may contain attributes!

e-Macao-16-4-460

Top- and Local-Level Attributes

Like elements, attributes can be defined at top and local levels:

- top level attribute – referred by local-level attributes

```
<xsd:attribute name="version" type="xsd:string"/>
```

- local level attribute – content of its parent element:

```
<xsd:complexType name="Collection">
   ...
   <xsd:element ref="CD"/>
   <xsd:attribute ref="version"/>
</xsd:complexType>
```

e-Macao-16-4-461

Attributes and Types

- attribute with named type

```
<xsd:attribute name="version" type="xsd:string"/>
```

- attribute with anonymous type

```
<xsd:attribute name="versions">
   <xsd:simpleType>
      <xsd:list itemType="Version"/>
   </xsd:simpleType>
</xsd:attribute>
```

- local-level attribute referring to top-level attribute

```
<xsd:attribute ref="version"/>
```

e-Macao-16-4-462

Attribute Occurrence

The `use` attribute determines the attribute's occurrence:

- attribute is required:

```
<xsd:attribute name="version" use="required" .../>
```

- attribute is optional (default):

```
<xsd:attribute name="version" use="optional" .../>
```

- attribute is prohibited:

```
<xsd:attribute name="version" use="prohibited" .../>
```

e-Macao-16-4-463

Attribute Default and Fixed Values

Two more attributes:

- `fixed` - fixed value for the attribute:

```
<xsd:attribute name="version" fixed="1.0" .../>
```

- `default` - default value for the attribute:

```
<xsd:attribute name="version" default="1.0" .../>
```

If an element is missing, the defaults for its attributes are not supplied.

e-Macao-16-4-464

Complex Type Definition

Consists of element declarations/references embedded inside a content model, and attribute declarations:

```
<xsd:complexType name="...">
   <xsd:sequence>
      <xsd:element .../>
      <xsd:element .../>
   </xsd:sequence>
   <xsd:attribute .../>
   <xsd:attribute .../>
</xsd:complexType>
```

e-Macao-16-4-465

Content Models

Three content models:

- `sequence` – similar to `(a,b)` in DTD
- `choice` – similar to `a|b` in DTD
- `all` – does not occur in DTD

They can be nested within each other to any level, and may have `minOccurs` and `maxOccurs` attributes.

e-Macao-16-4-466

Sequence Content Model

Elements must occur exactly in the order indicated, and they all must occur (unless `minOccurs="0"` for individual elements):

```
<xsd:complexType>
   <xsd:sequence>
      <xsd:element name="day" type="day"/>
      <xsd:element name="month" type="month"/>
      <xsd:element name="year" type="xsd:integer"/>
      <xsd:element name="weekeday"
        minOccurs="0" type="weekDay"/>
   </xsd:sequence>
</xsd:complexType>
```

e-Macao-16-4-467

Choice Content Model

Exactly one of elements may occur; elements are mutually exclusive:

```
<xsd:complexType name="address">
   <xsd:sequence>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:choice minOccurs="0">
         <xsd:element name="state" type="xsd:string"/>
         <xsd:element name="province" type="xsd:string"/>
      </xsd:choice>
      <xsd:element ref="zip"/>
   </xsd:sequence>
</xsd:complexType>
```

e-Macao-16-4-468

All Content Model

This model does not occur in DTD, but it does occur in SGML.

All of the element may appear, in any order.

Limitations:

- each element may occur no more than once: `minOccurs` is zero or one, `maxOccurs` is one
- cannot contain `sequence` and `choice` models
- must occur as the only immediate child at the beginning of the content model, and occur no more than once

e-Macao-16-4-469

All Content Model

Address must contain: street, city, country and zip, and may contain state. Any order of elements is allowed:

```
<xsd:complexType name="address">
   <xsd:all>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state"
         minOccurs="0" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element ref="zip"/>
   </xsd:all>
</xsd:complexType>
```

e-Macao-16-4-470

Complex Types Again

Formally, there are two ways to define complex types:

- `simpleContent` – permits character data and attributes

```
<xsd:complexType>
   <xsd:simpleContent>...</xsd:simpleContent>
</xsd:complexType>
```

- `complexContent` – permits children elements and attributes; this is the default, so may be omitted

```
<xsd:complexType>
   <xsd:complexContent>...</xsd:complexContent>
</xsd:complexType>
```

e-Macao-16-4-471

Derivation of Complex Types

Four ways to derive a complex type from another type:

1. extension of any simple type
2. extension of another complex type
3. restriction of another complex type
4. restriction of the generic `anyType`

e-Macao-16-4-472

Adding Attributes

Adding `version` and `status` attributes (both `string`) to an element with simple content (`integer`):

```
<xsd:complexType>
   <xsd:simpleContent>
      <xsd:extension base="xsd:integer">
        <xsd:attribute name="version" type="xsd:string"/>
        <xsd:attribute name="status" type="xsd:string"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

e-Macao-16-4-473

Adding Elements

Adding `new` and `more` elements to an existing `myType`, at the end of its children list:

```
<xsd:complexType>
   <xsd:complexContent>
      <xsd:extension base="myType">
         <xsd:sequence>
            <xsd:element name="new" type="xsd:string"/>
            <xsd:element name="more" type="xsd:string"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

e-Macao-16-4-474

Removing Children Elements

When removing elements of an existing `myType`, we have to repeat all its children (modified or not), except those that are being removed:

```
<xsd:complexType>
   <xsd:complexContent>
      <xsd:restriction base="myType">
       <xsd:sequence>
        <xsd:element name="old" type="xsd:string"/>
        <xsd:element name="modified" type="xsd:integer"/>
       </xsd:sequence>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
```

e-Macao-16-4-475

Empty Elements

- element without content/attributes:

```
<xsd:complexType name="empty"/>
```

- element without content but with attributes:

```
<xsd:complexType name="empty">
    <xsd:attribute name="price" type="xsd:integer"/>
    <xsd:attribute name="version" type="xsd:string"/>
</xsd:complexType>
```

e-Macao-16-4-476

Limiting Derivation

We can limit the derivation of types with the `final` attribute:

- extended but not restricted

```
<xsd:complexType final="extension">...<xsd:complexType>
```

- restricted but not extended

```
<xsd:complexType final="restriction">...<xsd:complexType>
```

- cannot be derived at all

```
<xsd:complexType final="#all">...<xsd:complexType>
```

e-Macao-16-4-477

Mixed Content Model

- element content consists of either sub-elements or character data, but not both (default):

```
<xsd:complexType mixed="false">...<xsd:complexType>
```

- element content is a mixture of sub-elements and character data:

```
<xsd:complexType mixed="true">...<xsd:complexType>
```

The order and number of elements in the mixed model is constrained by the schema, like in the non-mixed case; not possible for DTD.

e-Macao-16-4-478

Any Content Model

All simple and complex types are derived from `xsd:anyType`.

Restriction of `anyType` with complex content is the default:

```
<xsd:complexType>
   <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
         <xsd:sequence>
            <xsd:element name="..." .../>
            <xsd:element name="..." .../>
         </xsd:sequence>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
```

e-Macao-16-4-479

Element Groups: Declaration

Element group – a set of elements defined with a name:

```
<xsd:group name="myGroup">
   <xsd:sequence>
      <xsd:element ref="name"/>
      <xsd:element ref="scope"/>
   </xsd:sequence>
</xsd:group>
```

Must be the immediate child of `schema`, and may contain `sequence`, `choice` or `all` only.

e-Macao-16-4-480

Element Groups: Reference

Element group referenced in the complex type definition:

```
<xsd:complexType name="myType">
   <xsd:sequence>
      <xsd:group ref="myGroup"/>
      <xsd:element ref="price"/>
   </xsd:sequence>
</xsd:complexType>
```

Element groups play a similar role as parameter entities in DTD.

e-Macao-16-4-481

Attribute Groups: Declaration

Grouping attributes with `attributeGroup`:

```
<xsd:attributeGroup name="margins">
   <xsd:attribute name="top" type="xsd:float"/>
   <xsd:attribute name="bottom" type="xsd:float"/>
   <xsd:attribute name="left" type="xsd:float"/>
   <xsd:attribute name="right" type="xsd:float"/>
</xsd:attributeGroup>
```

Like `group`, `attributeGroup` is an immediate child of `schema`.

e-Macao-16-4-482

Attribute Groups: Reference

Referencing an attribute group by its name:

```
<xsd:complexType name="myType">
   <xsd:sequence>
      <xsd:group ref="myGroup"/>
      <xsd:element ref="price"/>
   </xsd:sequence>
   <xsd:attributeGroup ref="margins"/>
</xsd:complexType>
```

They also play the role similar to DTD's parameter entities.

e-Macao-16-4-483

Referring to Schema: One Namespace 1

XML instance document with the default namespace `http://www.w3c.org` for all its elements:

```
<elem
  xmlns="http://www.w3c.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3c.org schemaNamespace.xsd">
  text
</elem>
```

The attribute `SchemaLocation` determines the schema location for the elements that belong to the specified namespace:

```
xsi:schemaLocation="namespace schema"
```

e-Macao-16-4-484

Referring to Schema: One Namespace 2

XML Schema file `schemaNamespace.xsd`:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3c.org">
  <xsd:element name="elem" type="xsd:string"/>
</xsd:schema>
```

The `TargetNamespace` attribute of `schema` determines the namespace this schema is used to validate.

Every schema is used to validate a single namespace!

e-Macao-16-4-485

Demo: One Namespace Validation

```
> cd "demos/schema one namespace"
> dir
schemaNamespace.xsd namespace.xml
> java dom.Counter -v -s namespace.xml
```

e-Macao-16-4-486

Referring to Schema: Two Namespaces 1

XML instance document with two namespaces
`http://www.w3c.org/1` and `http://www.w3c.org/2`:

```
<ns1:outside
  xmlns:ns1="http://www.w3c.org/1"
  xmlns:ns2="http://www.w3c.org/2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
   http://www.w3c.org/1 schemaNamespace1.xsd
   http://www.w3c.org/2 schemaNamespace2.xsd">

  <ns2:inside>text</ns2:inside>

</ns1:outside>
```

e-Macao-16-4-487

Referring to Schema: Two Namespaces 2

The attribute `SchemaLocation` determines the schema locations for the elements that belong to specified namespaces:

```
xsi:schemaLocation="
   namespace1 schema1
   namespace2 schema2
   namespace3 schema3…"
```

e-Macao-16-4-488

Referring to Schema: Two Namespaces 1

XML instance document with two namespaces
`http://www.w3c.org/1` and `http://www.w3c.org/2`:

```
<ns1:outside
  xmlns:ns1="http://www.w3c.org/1"
  xmlns:ns2="http://www.w3c.org/2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
   http://www.w3c.org/1 schemaNamespace1.xsd
   http://www.w3c.org/2 schemaNamespace2.xsd">

  <ns2:inside>text</ns2:inside>

</ns1:outside>
```

e-Macao-16-4-489

Referring to Schema: Two Namespaces 3

The file `schemaNamespace1.xsd` used to validate the elements from the namespace `http://www.w3c.org/1`:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.iist.unu.edu/1">

  <xsd:element name="outside">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="http://www.iist.unu.edu/2"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

e-Macao-16-4-490

Referring to Schema: Two Namespaces 4

The file `schemaNamespace2.xsd` used to validate the elements from the namespace `http://www.w3c.org/2`:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.iist.unu.edu/2">

  <xsd:element name="inside" type="xsd:string"/>

</xsd:schema>
```

e-Macao-16-4-491

Demo: Two Namespace Validation

```
> cd "demos/schema two namespaces"
> dir
schemaNamespace1.xsd schemaNamespace1.xsd
namespaces.xml
> java dom.Counter -v -s namespaces.xml
```

e-Macao-16-4-492

Instance Documents and Qualification

Should elements/attributes in an instance document be qualified?

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.iist.unu.edu/xml"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <xsd:element name="top">...</xsd:element>
    ...
</xsd:schema>
```

- `qualified` – they must be in the target namespace
- `unqualified` – they should not be in any namespace; default

e-Macao-16-4-493

Schema Modularisation

Two top-level elements permit the inclusion of an external schema:

- `import` – allows access to elements and type definitions from different namespaces
- `include` – the target namespace of the included schema must be the same as the target namespace of the including schema

They must occur prior to any other definitions in a schema.

e-Macao-16-4-494

DTD for XML Schema

As XML Schema documents apply XML syntax, they can be checked for validity. Fragment of the official XML Schema DTD:

```
          namespace prefix -> | <!ENTITY % p 'xs:'>
           namespace sufix -> | <!ENTITY % s ':xs'>
      namespace declaration -> | <!ENTITY % nds 'xmlns%s;'>
      schema qualified name -> | <!ENTITY % schema "%p;schema">
schema element declaration -> | <!ELEMENT %schema;
                           -> |  (%simpleType; | %complexType; |
                           -> |   %element; | %attribute; | ...)*>
   schema attributes         | <!ATTLIST %schema;
                  version -> |   version CDATA #IMPLIED
           schema namespace -> |   %nds; %URIref; #FIXED '...XMLSchema'
         default namespace -> |   xmlns CDATA #IMPLIED
         unique identifier -> |   id ID #IMPLIED
          other attributes -> |   %schemaAttrs;>
```

e-Macao-16-4-495

DTD Validation of XML Schema Documents

Validation statement:

> Any XML document which is not valid per this DTD given redefinitions in its internal subset of the 'p' and 's' parameter entities appropriate to its namespace declaration of the XML Schema namespace is almost certainly not a valid schema.

A.3.2. XPath

XPath

e-Macao-16-4-497

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-498

XPath

A language to address parts of the XML document.

Apart from this main goal, it serves additional goals:

1. manipulation of text, numbers and logical values
2. formulating patterns for document tree nodes
3. checking if a node satisfies a pattern

Intended for the use by XSLT and XPointer.

W3C Recommendation November 1999.

e-Macao-16-4-499

XPath Features

XPath provides a compact non-XML notation.

XPath operates on the logical tree structures of an XML document.

Its name originates from the path notation to navigate the hierarchical tree structures of XML documents.

e-Macao-16-4-500

Evaluating XPath Expressions

Evaluating an XPath expression returns one of four kinds of objects:

1. set of nodes
2. boolean (true or false)
3. number (floating point)
4. string (list of UCS characters)

e-Macao-16-4-501

Evaluation Context

Expression is evaluated with respect to the context:

1. context node
2. a pair of positive integers: position and size of the context
3. values of the variables; one of the four types
4. function library with arguments are results of the four types
5. namespace declarations

During evaluation:

- variables, functions and namespaces remain unchanged
- context node, position and size may change

e-Macao-16-4-502

Task: Create XML Document

Create `collection.xml`:

```
<?xml version="1.0"?>
<collection>
  <book>
    <title>Learning XML</title>
    <review>4</review>
  </book>
  <book id="abc">
    <title>Java and XML</title>
  </book>
  <article>
    <title>XML and Semantic Web</title>
    <journal><title>CACM</title></journal>
  </article>
</collection>
```

e-Macao-16-4-503

Task: Create XSLT Document

Create `collection.xsl`:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:for-each select="CONTEXT">
    <xsl:value-of select="XPATH"/>
    <xsl:text> </xsl:text>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

e-Macao-16-4-504

Task: XPath Experiments

Use the values of `CONTEXT` and `XPATH` to test XPath expressions.

For instance:

```
CONTEXT = /
XPATH   = .
```

Run:

```
> java org.apache.xalan.xslt.Process
   -in collection.xml
   -xsl collection.xsl
```

e-Macao-16-4-505

Relative Paths

The simplest path is the reference to an element name.

Let:

```
CONTEXT = collection
XPATH   = .
```

e-Macao-16-4-506

Multiple Steps

A path with two steps:

```
collection/article
```

Let:

```
CONTEXT = collection/article
XPATH   = .
```

e-Macao-16-4-507

Wildcard Steps

Any element may occur between `collection` and `title`:

```
collection/*/title
```

Let:

```
CONTEXT = collection/*/title
XPATH   = .
```

e-Macao-16-4-508

Descendants Selection

Any number of elements may occur between `collection` and `title`:

```
collection//title
```

Let:

```
CONTEXT = collection//title
XPATH   = .
```

e-Macao-16-4-509

Self, Parents and Grandparents

Self node: `.`
Parent node: `..`
Grandparents: `../..`

Let:

```
CONTEXT = collection/article
XPATH   = .
XPATH   = ../book
XPATH   = ../..
```

e-Macao-16-4-510

Absolute Path

`/collection/article` selects `article` from every context node.

Let:

```
CONTEXT = collection/book
XPATH   = /collection/article
```

`//title` selects all `title` elements in the document:

Let:

```
CONTEXT = //title
XPATH   = .
```

e-Macao-16-4-511

Predicates

Location paths are indiscriminate in node selection.

Predicate:

```
book[…]/title
```

can be used to quality any step in the path.

e-Macao-16-4-512

Position Tests

Selects the second book:

```
para[position()=2]
```

Let:

```
CONTEXT = collection/book[position()=2]
XPATH   = .
```

e-Macao-16-4-513

Presence Tests

Selects a books with a review element:

```
collection/book[review]
```

Let:

```
CONTEXT = collection/book[review]
XPATH   = .
```

e-Macao-16-4-514

Value Tests

Selects a books with a given title:

```
collection/book[title='Java and XML']
```

Let:

```
CONTEXT = collection/book[title='Java and XML']
XPATH   = .
```

e-Macao-16-4-515

Attribute Presence Test

Selects `book` if it contains an attribute `id`:

```
book[@id]
```

Let:

```
CONTEXT = //book[@id]
XPATH   = .
```

e-Macao-16-4-516

Attribute Value Test

Selects `book` if it contains an attribute `id` which has the value `"abc"`:

```
book[@id="abc"]
```

Let:

```
CONTEXT = //book[@id="abc"]
XPATH   = .
```

e-Macao-16-4-517

Boolean Tests: Negation

Selects `book` if it does not contain an attribute `id`:

```
book[not(@id)]
```

Let:

```
CONTEXT = //book[not(@id)]
XPATH   = .
```

e-Macao-16-4-518

Boolean Tests: Disjunction

Selects the element `book` if it does not contain an attribute `id` or occurs on the last position:

```
book[not(@id) or position()=2]
```

Let:

```
CONTEXT = //book[not(@id) or position()=last()]
XPATH   = .
```

e-Macao-16-4-519

String Calculation: Contains

Selects the element `book` which `title` contains `'Java'`:

```
//book/title[contains(text(),'Java')]
```

Let:

```
CONTEXT = //book/title[contains(text(),'Java')]
XPATH   = .
```

e-Macao-16-4-520

String Calculation: Substring

Selects the element `book` which `title` contains `'Java'`:

```
//book/title[contains(text(),'Java')]
```

Displays the first four characters of the title:

```
substring(.,1,4)
```

Let:

```
CONTEXT = //book/title[contains(text(),'Java')]
XPATH   = substring(.,1,4)
```

e-Macao-16-4-521

Multiple Predicates

Predicates can be combined.

First select all books with the `id` attribute, then among them all books with the `"Java and XML"` title:

```
//book[@id][title='Java and XML']
```

Let:

```
CONTEXT = //book[@id][title='Java and XML']
XPATH   = .
```

A.3.3. XSLT

XSLT

e-Macao-16-4-523

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

e-Macao-16-4-524

XSL

XSL = eXtensible Stylesheet Langugage

Consists of three parts:

- XSLT – an XML language for transforming XML documents
- XSL-FO – an XML language for formatting semantics
- XPath – a non-XML syntax for addressing parts of an XML document. Also used in XLink, XPointer and XQuery.

e-Macao-16-4-525

XSLT

- XSLT = eXtensible Stylesheet Language Transformations
- XSLT is a fully-fledged declarative programming language specializing in XML transformations
- XSLT programs are themselves written in XML

e-Macao-16-4-526

XSLT

- XSLT = eXtensible Stylesheet Language Transformations
- XSLT is a fully-fledged declarative programming language specializing in XML transformations
- XSLT programs are themselves written in XML

e-Macao-16-4-527

Example: XML Input

```
<?xml version="1.0"?>
<greeting>
   Hello World!
</greeting>
```

e-Macao-16-4-528

Example: XSLT Program

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>

   <xsl:template match="/">
      <xsl:apply-templates select="greeting"/>
   </xsl:template>

   <xsl:template match="greeting">
      <html>
         <body>
            <h1><xsl:value-of select="."/></h1>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

e-Macao-16-4-529

Example: Invocation

```
java org.apache.xalan.xslt.Process
   -in greeting.xml
   -xsl greeting.xsl
   -out greeting.html
```

e-Macao-16-4-530

Example: HTML Output

```
<html>
   <body>
      <h1>
         Hello World!
      </h1>
   </body>
</html>
```

e-Macao-16-4-531

XSLT Document Structure

Root element and namespace declaration:

```
<xsl:stylesheet
   version="1.0"
   xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
   …
</xsl:stylesheet>
```

e-Macao-16-4-532

Output Format 1

What is the format of the output document?

```
<xsl:stylesheet version="1.0"
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>

    <xsl:output method="html"/>

    …
</xsl:stylesheet>
```

Possibilities: text, HTML, XML.

e-Macao-16-4-533

Output Format 2

Each output method provides its own formatting attributes.

For instance:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>

    <xsl:output method="html" ident="yes"/>

    …
</xsl:stylesheet>
```

e-Macao-16-4-534

XSLT Processing

XSLT program describe a set of transformation rules from the input document to the output document.

Transformation is defined with:

1. templates matched against the input elements
2. patterns describing fragments of the output document

e-Macao-16-4-535

XSLT Templates

General format of templates:

```
<xsl:template match="xpath-expr">
    …
</xsl:template>
```

The value of the match attribute is the XPath expression.

The template matching the root element only:

```
<xsl:template match="/">
    …
</xsl:template>
```

e-Macao-16-4-536

Applying XSLT Templates

The template calling recursively other templates (perhaps itself) given `greeting` as the current element:

```
<xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
</xsl:template>
```

e-Macao-16-4-537

Templates with Text

The only matching template is this:

```
<xsl:template match="greeting">
    <html>
        <body>
            <h1>
                <xsl:value-of select="."/>
            </h1>
        </body>
    </html>
</xsl:template>
```

Produces text on output (HTML markup).

e-Macao-16-4-538

Referring to the Current Element

The content of the current element (greeting) is sent to output:

```
<xsl:template match="greeting">
    <html>
        <body>
            <h1>
                <xsl:value-of select="."/>
            </h1>
        </body>
    </html>
</xsl:template>
```

e-Macao-16-4-539

Template Processing

Let the current context contain the root element.

1) are there any nodes to process in the current context?

2) for every node in the context:

   a) are there any templates matching the node?
      i. if there are several templates choose the most specific
      ii. if there is no template, choose the default template

   b) invoke the template recursively for the next context

e-Macao-16-4-540

Default Templates 1

Applied when lacking specific templates:

1. ensures continued processing even if there is no template for a given element:

```
<xsl:template  match="*|/">
   <xsl:apply-templates/>
</xsl:template>
```

e-Macao-16-4-541

Default Templates 2

2. the elements are processed regardless of the mode:

```
<xsl:template match="*|/" mode="x">
   <xsl:apply-templates mode="x"/>
</xsl:template>
```

3. text and attributes of selected elements are send to output:

```
<xsl:template match="text()|@*">
   <xsl:value-of select="."/>
</xsl:template>
```

e-Macao-16-4-542

Default Templates 3

4. comment and processing instruction nodes are not processed:

```
<xsl:template
   match="comment() |
   processing-instruction()"/>
```

5. namespace nodes are not processed:

```
<xsl:template match="namespace()"/>
```

e-Macao-16-4-543

Template Attributes

The template must contain one of the following attributes:

1. `match` – the template invoked when matching elements are found:

```
<xsl:template match="...">...</xsl:template>
```

2. `name` – the template is invoked by name:

```
<xsl:template name="...">...</xsl:template>
```

e-Macao-16-4-544

Calling Template by Name

Definition of the named template:

```
<xsl:template name="myName">...</xsl:template>
```

Calling the template:

```
<xsl:template ...>
   <xsl:call-template name="myName"/>
</xsl:template>
```

e-Macao-16-4-545

Example: Named Template

```
<xsl:template match="root">
  <xsl:call-template name="myName"/>
  <xsl:call-template name="myName"/>
</xsl:template>

<xsl:template name="myName">
  <xsl:text>text</xsl:text>
</xsl:template>
```

e-Macao-16-4-546

Matched Template

Definition of the matched template:

```
<xsl:template match="elem">...</xsl:template>
```

Invoking the template:

1. applies to all children of the current node:

```
<xsl:apply-templates/>
```

2. selects the nodes to which the template applies:

```
<xsl:apply-templates select="..."/>
```

e-Macao-16-4-547

Example: Matched Template

```
<xsl:template match="root">
  <xsl:apply-templates/>
  <xsl:apply-templates select="@att"/>
</xsl:template>

<xsl:template match="elem">
  <xsl:text>elementy</xsl:text>
</xsl:template>

<xsl:template name="@att">
  <xsl:text>atrybuty</xsl:text>
</xsl:template>
```

e-Macao-16-4-548

Templates with Mode

The template may contain the mode attribute that allows to process the same set of nodes several times.

Definition of the template with mode:

```
<xsl:template match="..." mode="...">
   ...
</xsl:template>
```

Invoking the mode template:

```
<xsl:apply-templates select="..." mode="...">
```

e-Macao-16-4-549

Example: Templates with Mode

```
<xsl:template match="root">
  <xsl:apply-templates mode="mode1"/>
  <xsl:apply-templates mode="mode2"/>
</xsl:template>

<xsl:template match="elem" mode="mode1">
  <xsl:text>mode1:</xsl:text>
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="elem" mode="mode2">
  <xsl:text>mode2:</xsl:text>
  <xsl:value-of select="."/>
</xsl:template>
```

e-Macao-16-4-550

Task: Simplify Credit Card Letter

From document-oriented to data-oriented XML:

```
<?xml version="1.0"?>
<letter>
   <customer>Simon White</customer>
   <product>credit card</product>
   <officer level="manager">Steven Rod</officer>
   <enclosure>credit card</enclosure>
   <enclosure>initial PIN</enclosure>
</letter>
```

e-Macao-16-4-551

Task: Create the Transformation Stylesheet

Transformation to generate the acceptance letter:

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <xsl:template match="letter">
      Dear <xsl:value-of select="customer"/>,
      We are pleased to inform you that your
      <xsl:value-of select="product"/> application
      has been accepted.
      Sincerely, <xsl:value-of select="officer"/>
   </xsl:template>
</xsl:stylesheet>
```

e-Macao-16-4-552

Task: Run the Transformation

Execute `xalan`:

```
> xalan letter.xml letter.xsl output
```

What is the result?

e-Macao-16-4-553

Task: Change Default Output Format

```
<?xml version="1.0"?>
<xsl:stylesheet ...>

   <xsl:output method="text"/>

   <xsl:template match="letter">...</xsl:template>

</xsl:stylesheet>
```

Run `xalan`. Note the change.

e-Macao-16-4-554

Task: Modify the Stylesheet

Letter template:

```
<xsl:template match="letter">
   Dear <xsl:value-of select="customer"/>,
   <xsl:apply-templates select="product"/>
   Sincerely, <xsl:value-of select="officer"/>
</xsl:template>
```

Product template:

```
<xsl:template match="product">
   We are pleased to inform you that your
   <xsl:value-of select="."/> application
   has been accepted.
</xsl:template>
```

e-Macao-16-4-555

Task: Run the Transformation

Execute `xalan`:

```
> xalan letter.xml letter.xsl output
```

Any change?

e-Macao-16-4-556

Task: Add Officer's Level

Refer to the officer's level attribute:

```
<xsl:template match="letter">
   Dear <xsl:value-of select="customer"/>,
   <xsl:apply-templates select="product"/>
   Sincerely, <xsl:value-of select="officer"/>
   (<xsl:value-of select="officer/@level"/>)
</xsl:template>
```

Run `xalan`.

e-Macao-16-4-557

Which Template?

There is no template for a given context:

```
<xsl:template match="root">...</xsl:template>
<xsl:template match="sub">...</xsl:template>

<root att="value">
   <elem><sub>text1</sub></elem>
</root>
```

Apply the default template.

e-Macao-16-4-558

Which Template?

There is exactly one template for a given context:

```
<xsl:template match="root">...</xsl:template>
<xsl:template match="elem">...</xsl:template>

<root att="value">
   <elem><sub>text1</sub></elem>
</root>
```

Apply this template.

e-Macao-16-4-559

Which Template?

There are two templates for a given context.

```
<xsl:template match="root">...</xsl:template>
<xsl:template match="sub">...</xsl:template>
<xsl:template match="elem/sub">...</xsl:template>

<root att="value">
   <elem><sub>text1</sub></elem>
</root>
```

Apply the more specific template.

e-Macao-16-4-560

Which Template?

There are two equally-specific templates for a given context:

```
<xsl:template match="elem" priority="1">
  <xsl:text>first</xsl:text>
</xsl:template>

<xsl:template match="elem" priority="0">
  <xsl:text>second</xsl:text>
</xsl:template>
```

Apply the template with the higher priority.

e-Macao-16-4-561

Which Template?

There are two equally-specific template for a given context and with equal priorities:

```
<xsl:template match="elem" priority="0">
  <xsl:text>pierwszy</xsl:text>
</xsl:template>

<xsl:template match="elem" priority="0">
  <xsl:text>drugi</xsl:text>
</xsl:template>
```

Apply the template that occurs later in the stylesheet.

e-Macao-16-4-562

Template with Parameters

Template definition with two parameters:

```
<xsl:template name="area">
  <xsl:param name="height"/>
  <xsl:param name="width"/>
  <xsl:value-of select="$height * $width"/>
</xsl:template>
```

Calling the template with concrete arguments:

```
<xsl:call-template name="area">
  <xsl:with-param name="height" select="10"/>
  <xsl:with-param name="width" select="20"/>
</xsl:call-template>
```

e-Macao-16-4-563

Default Parameters

Template with default values of the parameters:

```
<xsl:template name="area">
  <xsl:param name="height">10</xsl:param>
  <xsl:param name="width">20</xsl:param>
  <xsl:value-of select="$height * $width"/>
</xsl:template>
```

Calling the template with one parameter:

```
<xsl:call-template name="area">
  <xsl:with-param name="height" select="30"/>
</xsl:call-template>
```

e-Macao-16-4-564

Default Parameters with Choice

Template with parameters which default values are associated with the input document:

```
<xsl:template name="area">
  <xsl:param name="height" select="hi"/>
  <xsl:param name="width" select="wi"/>
  <xsl:value-of select="$height * $width"/>
</xsl:template>
```

Calling the template without parameters:

```
<xsl:call-template name="area"/>
```

e-Macao-16-4-565

Global Parameters

Parameters which scope expands the whole stylesheet.

They occur as the children of the `stylesheet` element:

```
<xsl:stylesheet ...>
   <xsl:param name="depth">30</xsl:param>
   ...
</xsl:stylesheet>
```

Default values like for local parameters.

e-Macao-16-4-566

Use of Global Parameters

Using the global parameter in the template:

```
<xsl:template name="volume">
  <xsl:param name="height" select="height"/>
  <xsl:param name="width" select="width"/>
  <xsl:value-of select="$height * $width* $depth"/>
</xsl:template>
```

e-Macao-16-4-567

Global Parameter Assignment

Global parameters can be assigned value in the command line.

For instance:

```
java org.apache.xalan.xslt.Process
   -in file.xml
   -xsl file.xsl
   -out file.out
   -param depth 50
```

For the depth parameter:

```
<xsl:param name="gleb"/>
```

e-Macao-16-4-568

Variables

Variable declarations:

1. name `x` and empty value:

```
<xsl:variable name="x"/>
```

2. name `x` and value `test`:

```
<xsl:variable name="x" select="'test'"/>
```

3. name `x` and value of the `test` element:

```
<xsl:variable name="x" select="test"/>
```

e-Macao-16-4-569

Example: Variables

The value of the `depth` variable depends on the choice expression that refers to the `depth` element:

```
<xsl:template name="depth">
  <xsl:param name="heigth" select="height"/>
  <xsl:param name="width" select="width"/>
  <xsl:variable name="depth">
    <xsl:choose>
      <xsl:when test="depth = 'one'">1</xsl:when>
      <xsl:when test="depth = 'two'">2</xsl:when>
      <xsl:otherwise>3</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:value-of select="$height * $width * $depth"/>
</xsl:template>
```

e-Macao-16-4-570

Variable Scope

Visible in the element in which it is declared:

1. local variable

```
<xsl:template name="...">
  <xsl:variable name="...">...</xsl:variable>
</xsl:template>
```

2. global variable

```
<xsl:stylesheet ...>
   <xsl:variable name="...">...</xsl:variable>
   ...
</xsl:template>
```

e-Macao-16-4-571

Variable or Constant?

Constant.

The values of variables cannot be modified.

e-Macao-16-4-572

Conditional Execution

Element `if`:

```
<xsl:if test="...">
   ...
</xsl:if>
```

If the calculated value of the test attribute is:

1. `true` – the content of the element is processed
2. `false` – the content of the element is ignored

e-Macao-16-4-573

Value of the test Attribute

The value is converted to `boolean`:

1. number – if zero or NaN (non-number) then false, otherwise true
2. node-set – if empty then false, otherwise true
3. string – if empty then false, otherwise true

e-Macao-16-4-574

Example: test Attributes

```
<xsl:if test="count(elem) &gt;=2">
<xsl:if test="$x">
<xsl:if test="true()">
<xsl:if test="true">
<xsl:if test="'true'">
<xsl:if test="'false'">
<xsl:if test="not(3)">
<xsl:if test="section/section">
```

e-Macao-16-4-575

Example: Conditional Execution

```
<xsl:if test="count(elem) = 2  and @att">
   <xsl:text>condition fulfilled</xsl:text>
</xsl:if>

<root att="wartosc">
   <elem>text1</elem>
   <elem>text2</elem>
</root>
```

e-Macao-16-4-576

Choice Execution

At least one `when`, optional `otherwise`:

```
<xsl:choose>
   <xsl:when test="...">...</xsl:when>
   ...
   <xsl:when test="...">...</xsl:when>
   <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

The first element with the `test` attribute returning true is processed.

If one does not exist, the content of `otherwise` is processed.

e-Macao-16-4-577

Example: Choice Execution

```
<xsl:choose>
   <xsl:when test="not(@att)">wybor 1</xsl:when>
   <xsl:when test="contains(elem[1],elem[2])">
      wybor 2
   </xsl:when>
   <xsl:otherwise>wybor 3</xsl:otherwise>
</xsl:choose>
```

```
<root att="wartosc">
   <elem>maly przyklad</elem>
   <elem>przyklad</elem>
</root>
```

e-Macao-16-4-578

Task: Make Decision Explicit

Modify XML:

```
<?xml version="1.0"?>
<letter decision="accepted">...</letter>
```

e-Macao-16-4-579

Task: One Template for Each Decision

Accept:

```
<xsl:template name="accepted">
   We are pleased to inform you that your
   <xsl:value-of select="product"/> application
   has been accepted.
</xsl:template>
```

Reject:

```
<xsl:template name="rejected">
   We are sorry to inform you that your
   <xsl:value-of select="product"/> application
   has been rejected.
</xsl:template>
```

e-Macao-16-4-580

Task: Conditionally Call Templates

Choose the template depending on the `decision` attribute:

```
<xsl:choose>
   <xsl:when test="@decision='accepted'">
      <xsl:call-template name="accepted"/>
   </xsl:when>
   <xsl:when test="@decision='rejected'">
      <xsl:call-template name="rejected"/>
   </xsl:when>
</xsl:choose>
```

e-Macao-16-4-581

Task: Run Xalan for Each Case

- for accept

```
<?xml version="1.0"?>
<letter decision="accepted">...</letter>
```

- for reject

```
<?xml version="1.0"?>
<letter decision="rejected">...</letter>
```

e-Macao-16-4-582

Task: Modify Decision Templates

Make the decision explicit:

- accepted template

```
<xsl:template name="accepted">
   We are pleased ... application has been
   <xsl:value-of select="@decision"/>.
</xsl:template>
```

- reject template

```
<xsl:template name="rejected">
   We are sorry ... application has been
   <xsl:value-of select="@decision"/>.
</xsl:template>
```

e-Macao-16-4-583

Task: Unify Decision Templates

Make a `body` template with `polite` parameter:

```
<xsl:template name="body">
   <xsl:param name="polite"/>
   We are <xsl:value-of select="polite"/>
   to inform you that your
   <xsl:value-of select="product"/> application
   has been <xsl:value-of select="@decision"/>.
</xsl:template>
```

e-Macao-16-4-584

Task: Call Template with Parameter

```
<xsl:choose>
    <xsl:when test="@decision='accepted'">
        <xsl:call-template name="body">
            <xsl:with-param name="polite">pleased</xsl:with-param>
        </xsl:call-template>
    </xsl:when>
    <xsl:when test="@decision='rejected'">
        <xsl:call-template name="body">
            <xsl:with-param name="polite">sorry</xsl:with-param>
        </xsl:call-template>
    </xsl:when>
</xsl:choose>
```

Run `xalan` in both cases.

e-Macao-16-4-585

Task: Remove Second Test

```
<xsl:choose>
    <xsl:when test="@decision='accepted'">
        ...
    </xsl:when>
    <xsl:otherwise>
        ...
    </xsl:otherwise>
</xsl:choose>
```

Run `xalan` in both cases.

e-Macao-16-4-586

Iterative Execution

Selects the set of node, then processes each of them in order selecting every time the new current node:

```
<xsl:for-each select="...">
    ...
</xsl:for-each>
```

It may initially contain several sort elements which are ordering the set before processing.

e-Macao-16-4-587

Example: Iterative Execution

```
<xsl:for-each select="elem">
   <xsl:value-of select="substring(.,1,5)"/>
</xsl:for-each>
```

```
<root att="wartosc">
    <elem>small example</elem>
    <elem>big example</elem>
</root>
```

e-Macao-16-4-588

Sorting

Sorting the set of node after selection by `apply-templates`:

```
<xsl:template match="root">
  <xsl:apply-templates select="elem">
    <xsl:sort/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="elem">
  <xsl:value-of select="."/>
</xsl:template>
```

Also a child of `for-each`.

e-Macao-16-4-589

Sorting Order

Inverse sorting order:

```
<xsl:template match="root">
  <xsl:apply-templates select="elem">
    <xsl:sort order="descending"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="elem">
  <xsl:value-of select="."/>
</xsl:template>
```

e-Macao-16-4-590

Task: Generate a List of Enclosures

Use a `for-each` loop to process each `enclosure` element:

```
<xsl:template match="letter">
    ...
    Enclosures:
    <xsl:for-each select="enclosure">
        <xsl:sort/>
        <xsl:value-of select="position()"/>.
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
    </xsl:for-each>
</xsl:template>
```

Run `xalan`.

e-Macao-16-4-591

Task: Sort the Enclosures

Use a `sort` element to sort the enclosures in ascending lexicographic order:

```
<xsl:template match="letter">
    ...
    Enclosures:
    <xsl:for-each select="enclosure">
        <xsl:sort/>
        ...
    </xsl:for-each>
</xsl:template>
```

Run `xalan`.

e-Macao-16-4-592

Task: Calculate the Number of Enclosures

Use the XPath `count` function:

```
<xsl:template match="letter">
    ...
    Attached, please see
    <xsl:value-of select="count(enclosure)"/>
    enclosures.
    ...
</xsl:template>
```

Run `xalan`.

e-Macao-16-4-593

Creating New Nodes

1. elements
2. attributes
3. text
4. processing instructions

e-Macao-16-4-594

Creating Elements 1

A template creating an element with name `elem` and the value equal that of the attribute `att`:

```
<xsl:template match="root">
  <xsl:element name="elem">
    <xsl:value-of select="@att"/>
  </xsl:element>
</xsl:template>
```

e-Macao-16-4-595

Creating Elements 2

A template creating an element with the name equal to the value of the attribute `att` and the value equal to the name of that attribute:

```
<xsl:template match="root">
  <xsl:element name="{@att}">
    <xsl:value-of select="name(@att)"/>
  </xsl:element>
</xsl:template>
```

e-Macao-16-4-596

Creating Attributes

A template creating an element `root` with attribute `att` which value is the content of the element `elem`:

```
<xsl:template match="root">
  <root>
    <xsl:attribute name="att">
      <xsl:value-of select="elem"/>
    </xsl:attribute>
  </root>
</xsl:template>
```

A.4. XML Java Processing

A.4.1. SAX

SAX

e-Macao-16-4-598

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) event-based programming (SAX)
   b) tree-based programming (DOM)
   c) rule-based programming (XSLT)

e-Macao-16-4-599

XML Programming Models

Three programming models:

- based on events – SAX
- based on trees – DOM
- based on templates – XSLT

How to program applications using XML syntax?

e-Macao-16-4-600

XML Processing Applications

The application has a built-in XML parser:

- the parser is rarely implemented, more often used off-the-shelf
- the parser processes XML syntax in various phases of the application's execution

The application and the parser are both using the same model (API) and representation of the input XML file.

Here we concentrate on the XML API for Java.

e-Macao-16-4-601

Java APIs for XML 1

Five different interfaces:

- JAXP – Java API for XML Processing
  Programming XML applications in Java using SAX, DOM and XSLT programming models.

2) JAXB – Java Architecture for XML Binding
   Writing Java objects in XML (marshalling), converting XML back to Java (unmarshalling)

3) JAXR – Java API for XML Registries
   Recording available services in an external registry, looking up the services in the registry.

e-Macao-16-4-602

Java APIs for XML 2

4) JAXM – Java API for XML Messaging
   Asynchronous exchange mechanism (send and forget) for XML messages exchanged between applications.

5) JAX-RPC – Java API for XML RPC
   Synchronous exchange mechanism (send and wait for reply) for XML messages exchanged between applications.

Here we describe JAXP – Java API for XML Processing.

e-Macao-16-4-603

Java API for XML Processing

JAXP is available in the package `javax.xml.parsers`.

Two abstract classes are contained in the package:

1) `SAXParserFactory` – enables the applications to create and configure a SAX parser
2) `DocumentBuilderFactory` – enables the applications to create and configure the DOM parser

e-Macao-16-4-604

Replacing API Implementations

Factory classes allow to replace parser implementations without the need to change the application's source code.

The implementation used depends on the setting of the properties:

1) `javax.xml.parsers.SAXParserFactory`
2) `javax.xml.parsers.DocumentBuilderFactory`

e-Macao-16-4-605

SAX API

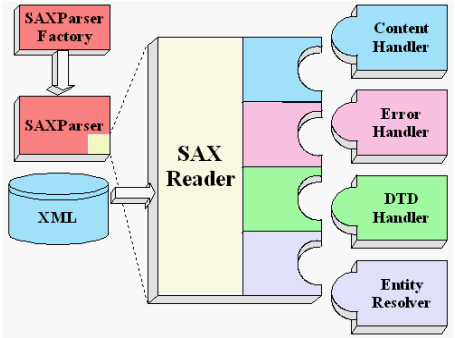
Simple API for XML.

A mechanism for processing XML documents element after element - serial and event-driven.

Most often used in server applications which are subject to the high performance requirements.

e-Macao-16-4-606

SAX API Architecture



e-Macao-16-4-607

SAX API – Operation

1) the object of the factory class `SAXParserFactory` class creates the parser – object of the `SAXParser` class

2) the parser encapsulates the SAXReader object which is used to read the input XML document

3) during parsing, SAXReader invokes the methods that belong to the following four interfaces:
   a) `ContentHandler`
   b) `ErrorHandler`
   c) `DTDHandler`
   d) `EntityResolver`

4) Those methods are realized by the application.

e-Macao-16-4-608

Create a SAX Parser Factory

Create an object of the SAX parser factory class:

```
SAXParserFactory factory =
   SAXParserFactory.newInstance();
```

The `newInstance()` method is:

```
public static SAXParserFactory newInstance()
   throws FactoryConfigurationError
```

It raises an error when there is no implementation.

e-Macao-16-4-609

Create a SAX Parser

Create the SAX parser through the new factory object:

```
SAXParser saxParser = factory.newSAXParser();
```

according to the current parameters set for the factory object.

The `newSAXParser()` method is:

```
public abstract SAXParser newSAXParser()
   throws ParserConfigurationException, SAXException
```

The parser generator raises an exception if the factory does not support the required combination of the parser features.

e-Macao-16-4-610

Configuring the SAX Parser Factory

1. the parser handles namespaces

```
void setNamespaceAware(boolean awareness)
boolean isNamespaceAware()
```

2. the parser validates documents

```
void setValidating(boolean validating)
boolean isValidating()
```

e-Macao-16-4-611

Parsing XML Documents

The `SAXParser` class enables parsing of XML documents that originate from different sources:

```
void parse(java.io.File f, ...)
void parse(java.io.InputStream is, ...)
void parse(java.lang.String uri, ...)
void parse(InputSource is, ...)
```

`InputSource` helps decide how the XML document should be read by the parser: as character stream, byte stream or the URL-addressed file.

e-Macao-16-4-612

Handling SAX Events

The second argument of the parse method is the object for handling events generated during parsing:

```
void parse(..., DefaultHandler dh)
```

The `DefaultHandler` class includes default implementations for the event-handling methods, declared by the interfaces:

1. `EntityResolver`
2. `DTDHandler`
3. `ContentHandler`
4. `ErrorHandler`

e-Macao-16-4-613

Entity-Resolving Events

Interface `EntityResolver`.

The parser will invoke this method before opening any external entity:

```
public InputSource
   resolveEntity(String publicId, String systemId)
    throws SAXException, java.io.IOException
```

where:

1. `publicId` is the Formal Public Identifier of the external entity, if one exists, otherwise `null`
2. `systemId` – is the system identifier of the external entity

e-Macao-16-4-614

Error-Handling Events

Interface `ErrorHandler`. Reporting errors:

1. reporting a warning
   `void warning(SAXParseException exc)`
2. reporting a non-critical error
   `void error(SAXParseException exc)`
3. reporting a fatal error
   `void fatalError(SAXParseException exc)`

The SAX parser is required to use those interfaces for reporting errors or warnings related to XML document processing, not exceptions.

This limitation does not apply to applications.

e-Macao-16-4-615

DTD-Handling Events

Interface `DTDHandler`. Events related to DTD processing:

Encountering notation declaration:

```
void notationDecl(
   String name, String publicId, String systemId)
```

Encountering unparsed entity declaration:

```
void unparsedEntityDecl(
   String name, String publicId, String systemId,
   java.lang.String notationName)
```

e-Macao-16-4-616

Content-Handling Events

Interface `ContentHandler`.

Handling events informing about the logical content of the document.

The main interface implemented by SAX applications.

If an application wants to be informed about events generated during document parsing, then it:

1. implements this interface
2. registers the implementation with the SAX parser using the `setContentHandler` method

e-Macao-16-4-617

Types of Content-Handling Events

Events informing about various kinds of content:

- informing about the document beginning
- informing about the document end
- informing about character data
- informing about ignorable white characters
- informing about the start of an element
- informing about the end of an element
- informing about the entry to and exit from a new namespace
- informing about processing instructions
- informing about ignorable entity

e-Macao-16-4-618

Content-Handling: Document Start/End

Informing about the start of the document:

```
void startDocument()
```

Informing about the end of the document:

```
void endDocument()
```

e-Macao-16-4-619

Content-Handling: Character Data

Informing about encountered character data:

```
void characters(char[] ch, int start, int length)
```

where

1. `ch` – character data of the document
2. `start` – initial table index
3. `length` – table length

e-Macao-16-4-620

Content-Handling: Ignorable Whitespace

Informing about encountered ignorable whitespaces:

```
void ignorableWhitespace(
    char[] ch, int start, int length)
```

where

1. `ch` – character data of the document
2. `start` – initial table index
3. `length` – table length

e-Macao-16-4-621

Content-Handling: Element Start 1

Informing about the start of an element:

```
public void startElement(
   String namespaceURI, String localName,
   String qName, Attributes atts)
```

where

1. `String namespaceURI` – URI of the element's namespace
2. `String localName` – local name (without prefix)

   Required when `http://xml.org/sax/features/namespaces` (system property) is true; default case.

e-Macao-16-4-622

Content-Handling: Element Start 2

More parameters:

3. `String qName` – the element's qualified name (with prefix).

   Optional when `http://xml.org/sax/features/namespace-prefixes` (system property) is false; default case.

4. `atts` – the attributes of the element; only the attributes with values given directly (not `#IMPLIED`) are included.

   This includes namespace declarations (`xmlns:*`) given the system property `http://xml.org/sax/features/namespace-prefixes`.

e-Macao-16-4-623

Content-Handling: Element End

Informing about the end of an element:

```
void endElement(
   String namespaceURI, String localName, String qName)
```

The end-tag event is also called for the empty element.

e-Macao-16-4-624

Content-Handling: Namespace Begin/End

Informing about the beginning of the namespace, occurs just before the corresponding `startElement`:

```
public void startPrefixMapping(
   String prefix, String uri)
```

Informing about the end of the namespace, occurs just after the corresponding `endElement`:

```
public void startPrefixMapping(
   String prefix, String uri)
```

e-Macao-16-4-625

Content-Handling: Processing Instruction

Informing about processing instruction:

```
public void processingInstruction(
   String target, String data)
```

e-Macao-16-4-626

Content-Handling: Ignorable Entity

Informing about encountering an ignorable entity:

```
public void skippedEntity(String name)
```

Non-validating parsers are allowed to ignore entities when they did not see their declarations (e.g. in external DTD).

Both validating and non-validating parsers may ignore external entities depending on the system properties:

```
http://xml.org/sax/features/external-general-entities
http://xml.org/sax/features/external-parameter-entities
```

e-Macao-16-4-627

Registering Event Handlers 1

Methods for registering/retrieving event handlers:

- content handler:

```
ContentHandler getContentHandler()
void setContentHandler(ContentHandler handler)
```

2. DTD handler

```
DTDHandler getDTDHandler()
void setDTDHandler(DTDHandler handler)
```

e-Macao-16-4-628

Registering Event Handlers 2

3. entity resolver

```
EntityResolver getEntityResolver()
void setEntityResolver(EntityResolver resolver)
```

4. error handler

```
ErrorHandler  getErrorHandler()
void setErrorHandler(ErrorHandler handler)
```

An application may register a new event handler in the middle of parsing a document. The parser would switch immediately.

e-Macao-16-4-629

Packages for SAX

Where is this all located?

- input and output:

```
import java.io.*;
```

2. all interfaces of SAX parsers:

```
import org.xml.sax.*;
```

3. handling of parser-generated events:

```
import org.xml.sax.helpers.DefaultHandler;
```

e-Macao-16-4-630

Packages 2

4. creating the SAX parser:

```
import javax.xml.parsers.SAXParserFactory;
```

5. parser-generation exception:

```
import javax.xml.parsers.ParserConfigurationException;
```

6. the SAX parser:

```
import javax.xml.parsers.SAXParser
```

e-Macao-16-4-631

Application Skeleton

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser

public class App {
   public static void main(String argv[]) {
       …
   }
}
```

e-Macao-16-4-632

Event Handling

The application must be able to catch and handle events issued by the XML parser about the encountered XML document content.

Implement all four interfaces `EntityResolver`, `DTDHandler`, `ErrorHandler`, `ContentHandler`? No.

Instead:

1. Inherit the `DefaultHandler` class that provides empty methods implementing all four interfaces.
2. Implement within `App` the handlers to those events that require a specific response.

e-Macao-16-4-633

Application Skeleton Revisited

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser

public class App extends DefaultHandler {
   public static void main(String argv[]) {…}
    …
    public void startElement(…){…}
    public void endElement(…){…}
    public void characters(…){…}
    …
   }
}
```

e-Macao-16-4-634

Main Method 1

```
public static void main(String argv[]) {
```

Check if there is a command-line argument:

```
   if (argv.length != 1) {
    System.err.println("Usage: cmd filename");
    System.exit(1);
   }
```

e-Macao-16-4-635

Main Method 2

The current class provides handling of the SAX events:

```
DefaultHandler handler = new App();
```

Create the SAX parser factory object:

```
SAXParserFactory factory =
   SAXParserFactory.newInstance();
```

e-Macao-16-4-636

Main Method 3

```
try {
```

Obtain the parser from the factory object:

```
    SAXParser saxParser = factory.newSAXParser();
```

Invoke the parser passing on the input document and the object of the current class to handle the events:

```
    saxParser.parse(new File(argv[0]), handler);

  } catch (Throwable t) { t.printStackTrace(); }

  System.exit(0);
}
```

e-Macao-16-4-637

Demo: Empty SAX Application

```
> cd "sax empty"
> dir
date.xml App.java
> javac App.java
> java App date.xml
```

e-Macao-16-4-638

Example: Element Counter 1

```
public class App extends DefaultHandler {
```

Declare a counter variable:

```
   int counter;

   public static void main(String argv[]) {
      …
   }
```

e-Macao-16-4-639

Example: Element Counter 2

Increment the counter for every new element:

```
public void startElement(
    String namespaceURI, String sName,
    String qName, Attributes attrs) throws SAXException {
    counter++;
}
```

Print the counter when encountering the end of the document:

```
public void endDocument() throws SAXException {
    System.out.println(counter);
}
}
```

e-Macao-16-4-640

Demo: Element Counter

```
> cd "sax counter"
> dir
date.xml App.java
> javac App.java
> java App date.xml
```

e-Macao-16-4-641

Task: Document Depth

Design a SAX application to calculate the depth of an XML document, that is the longest nesting of element within each other.

Implement event handlers `startElement`, `endElement` and `endDocument`.

e-Macao-16-4-642

Attribute Interface

Reconsider:

```
public void startElement(
 String namespaceURI, String sName,
    String qName, Attributes attrs) throws SAXException {
…
}
```

What is `Attributes`? An interface for a list of XML attributes.

1. `int getLength()`
2. `String getLocalName(int index)`
3. `String getValue(int index)`
4. etc.

e-Macao-16-4-643

Task: Document Statistics

Write a SAX application to print the names of all elements encountered and the number of attributes for each of them.

A.4.2. DOM

DOM

e-Macao-16-4-645

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) event-based programming (SAX)
   b) tree-based programming (DOM)
   c) rule-based programming (XSLT)

e-Macao-16-4-646

DOM

Document Object Model:

- Document – written with HTML, XML and others
- Object – representing parts of a document
- Model – document modeled as a tree

e-Macao-16-4-647

DOM Standard

A standard application programming interface (API) to access and update the structure of a document:

- standard method to access and update XML
- widely used in all major programming languages
- very useful in web browsers

e-Macao-16-4-648

DOM Components

What is DOM:

1) API
2) W3C Recommendation
3) document represented as a tree

e-Macao-16-4-649

DOM Components: API

DOM is a tree-based API - a set of interface definitions to access and update the tree representation of a document.

Overhead versus convenience:

- overhead - the document is loaded into memory
- convenience – documents can be randomly accessed

e-Macao-16-4-650

DOM Components: W3C Recommendation 1

DOM is the W3C Recommendation:

> *…a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents…*

DOM is an open standard.

Defined in IDL.

http://www.w3.org/DOM

e-Macao-16-4-651

DOM Components: W3C Recommendation 2

Evolution of DOM W3C Recommendation:

- level 1 – October 1998
  programmatic interface to manipulate XML and HTML

2) level 2 – November 2000
   multiple interfaces: core, views, events, CSS, traversal, range

- level 3 – April 2004
  support for information sets, XBase, attaching user information

e-Macao-16-4-652

DOM Components: Document Tree 1

Document components represented as node objects.

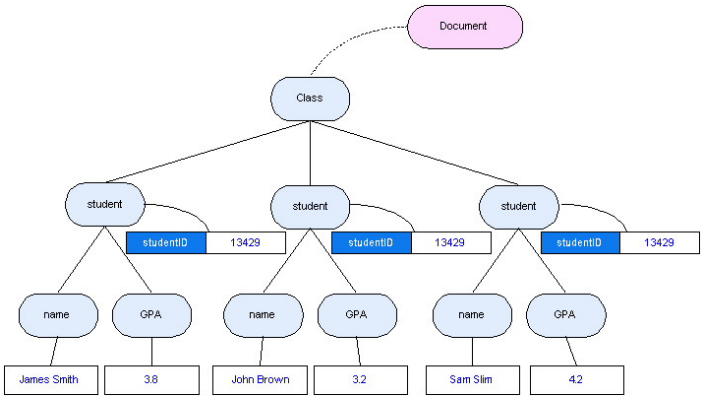
Nodes related to each other through properties.

Nodes are of various types:

1) elements
2) attributes
3) comments
4) text
5) etc.

e-Macao-16-4-653

DOM Components: Document Tree 2



e-Macao-16-4-654

DOM Components: Document Tree 3

Why represent a document as a tree?

1. suitable for recursive processing
2. suitable for accessing the content randomly
3. natural fit
   - documents have parts, trees have parts
   - documents have hierarchies, trees have hierarchies

e-Macao-16-4-655

Usage of DOM

When is DOM typically used?

1. applications that require random access to parts of a document
2. applications that require a document to be modified

For example:

1. scripting HTML pages
2. XML authoring tools
3. SVG viewers

e-Macao-16-4-656

DOM Strengths and Weaknesses 1

DOM is build to address some classes of problems better than others.

DOM stores a document in memory:

- suitable for random access
- heavy memory usage

DOM is unsuitable for:

- large documents
- devices with limited memory

e-Macao-16-4-657

DOM Strengths and Weaknesses 2

DOM is language-independent.

This:

- encourages open standards and implementations
- no need to switch models when switching languages

but:

- choosing the lowest common denominator – one cannot take advantage of specific language support

e-Macao-16-4-658

DOM versus SAX

SAX (Simple API for XML):

1. does not store the document in memory
2. parses XML by triggering callbacks to an application

Both build to address different classes of problems:

1. DOM
   - randomly access a document
   - update a document
2. SAX
   - large documents
   - document processing

e-Macao-16-4-659

DOM Implementations

W3C defines the interface.

Many language bindings: Java, JavaScript, C++, Perl, C#, etc.

A wide variety of implementations:

1. open-source and commercial
2. many different languages
3. complete and incomplete

e-Macao-16-4-660

Java Example

Create a DOM from an SVG image file:

```
DOMParser parser = new DOMParser();
Document doc = parser.parse("hello_world.svg");
Element docEl = doc.getDocumentElement();
System.out.println(docEl.hasChildNodes());
```

1. instantiate a DOM Parser
2. load the image into a DOM
3. get the root element of the document
4. print out if the root element has children

e-Macao-16-4-661

DOM Read Application 1

Defines the API to obtain DOM Document instances from XML:

```
import javax.xml.parsers.DocumentBuilder;
```

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents:

```
import javax.xml.parsers.DocumentBuilderFactory;
```

Exception classes for parser configuration errors:

```
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
```

SAX parser exceptions:

```
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```

e-Macao-16-4-662

DOM Read Application 2

Required I/O classes:

```
import java.io.File;
import java.io.IOException;
```

The Document interface represents the entire HTML or XML document:

```
import org.w3c.dom.Document;
```

DOM exceptions are raised when an operation is impossible for logical reasons, lost data, or because the implementation became unstable:

```
import org.w3c.dom.DOMException;
```

e-Macao-16-4-663

DOM Read Application 3

DOM application class:

```
public class Dom {
```

Static document object:

```
   static Document document;
```

The main method:

```
   public static void main(String[] argv) {
    …
   }

}
```

e-Macao-16-4-664

DOM Read Application 4

The main method:

Checking the presence of a command-line argument:

```
   if (argv.length != 1) {
   System.err.println("Usage: java Dom filename");
   System.exit(1);
    }
```

Creating a new DOM parser factory object:

```
   DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
```

e-Macao-16-4-665

DOM Read Application 5

Creating the DOM builder from the factory object:

```
   try {
      DocumentBuilder builder =
         factory.newDocumentBuilder();
```

Creating a DOM tree for the input document:

```
      document = builder.parse(new File(argv[0]));
```

e-Macao-16-4-666

DOM Read Application 6

Catching four different kinds of exceptions:

```
    } catch (SAXParseException spe) {
       System.out.println(spe.getMessage());
    } catch (SAXException sxe) {
       System.out.println(sxe.getMessage());
    } catch (ParserConfigurationException pce) {
       System.out.println(pce.getMessage());
    } catch (IOException ioe) {
       System.out.println(ioe.getMessage());
    }
   }
}
```

e-Macao-16-4-667

Task: DOM Read Application

Type the DOM read application skeleton.

Compile.

Run.

e-Macao-16-4-668

Demo: DOM Read Application

```
> cd "dom read"
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

e-Macao-16-4-669

DOM Read-Write Application 1

Previous DOM application could only read XML.

Lets write one that both reads and writes.

As in the read application:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.File;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
```

e-Macao-16-4-670

DOM Read-Write Application 2

Setting up the transformer and stream packages:

```
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
```

e-Macao-16-4-671

DOM Read-Write Application 3

As before:

```
public class App {

static Document document;

   public static void main(String[] argv) {
   if (argv.length != 1) {
      System.err.println("Usage: java App filename");
      System.exit(1);
   }

   DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
   try {
      DocumentBuilder builder =
         factory.newDocumentBuilder();
            document = builder.parse(new File(argv[0]));
```

e-Macao-16-4-672

DOM Read-Write Application 4

Creating the source, transformer and result:

```
   TransformerFactory tFactory =
   TransformerFactory.newInstance();
   Transformer transformer = tFactory.newTransformer();
   DOMSource source = new DOMSource(document);
   StreamResult result = new StreamResult(System.out);
   transformer.transform(source, result);
```

Catching transformer exceptions:

```
} catch(TransformerConfigurationException tce){
   System.out.println(tce.getMessage());
} catch(TransformerException te){
   System.out.println(te.getMessage());
}
```

e-Macao-16-4-673

DOM Read-Write Application 5

As before:

```
  catch (SAXParseException spe) {
   System.out.println(spe.getMessage());
} catch (SAXException sxe) {
   System.out.println(sxe.getMessage());
} catch (ParserConfigurationException pce) {
   System.out.println(pce.getMessage());
} catch (IOException ioe) {
   System.out.println(ioe.getMessage());
  }
 }
}
```

e-Macao-16-4-674

Task: DOM Read-Write Application

Type the DOM read-write application by extending the read application.

Compile.

Run.

e-Macao-16-4-675

Demo: DOM Read-Write Application

```
> cd "dom read write"
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

e-Macao-16-4-676

DOM Data Types

1. `Node` – the main DOM data type
2. `NodeList` – ordered collection of nodes
3. `NamedNodeMap` – name-indexed collection of nodes
4. `DOMString` – UTF16-encoded string
5. `DOMImplementation` – current implementation
6. `DOMException` – error codes
7. `DOMTimeStamp` – time value

e-Macao-16-4-677

DOM Interface

```
NodeList
NamedNodeMap
Node        |  Node
            |  Document
            |  DocumentFragment
            |  Element
            |  Attr
            |  CharacterData  |  Text  |  CDATASection
            |                 |  Comment
            |  Entity
            |  EntityReference
            |  ProcessingInstruction
DOMString
DOMTimeStamp
DOMImplementation
DOMException
```

e-Macao-16-4-678

DOM Interface: Node

The base interface for DOM.

Many different types of Nodes.

Defines common properties and methods.

Each of the specific node types inherit these.

Possible to completely access the content and structure.

e-Macao-16-4-679

DOM Interface: Node Types

Tree-building node types:

```
ELEMENT_NODE                1
ATTRIBUTE_NODE              2
ENTITY_REFERENCE_NODE       5
ENTITY_NODE                 6
DOCUMENT_NODE               9
DOCUMENT_FRAGMENT_NODE     11
```

Leaf node types:

```
TEXT_NODE                   3
CDATA_SECTION_NODE          4
PROCESSING_INSTRUCTION_NODE 7
COMMENT_NODE                8
DOCUMENT_TYPE_NODE         10
NOTATION_NODE              12
```

e-Macao-16-4-680

DOM Interface: Node Properties

1. `node.nodeName` - depends on `nodeType`
2. `node.nodeValue` - depends on `nodeType`
3. `node.nodeType` - one of defined types
4. `node.childNodes` - if any
5. `node.attributes` - if `nodeType` is `Element`

e-Macao-16-4-681

DOM Interface: Node Navigation

1. `node.ownerDocument` - if `nodeType` isn't `Document`
2. `node.parentNode` - if one exists
3. `node.firstChild` - if one exists
4. `node.lastChild` - if one exists
5. `node.nextSibling` - if one exists
6. `node.previousSibling` - if one exists

e-Macao-16-4-682

DOM Interface: Node Methods

1. `node.insertBefore(newChild,refChild)`
2. `node.replaceChild(newChild,oldChild)`
3. `node.removeChild(oldChild)`
4. `node.appendChild(newChild)`
5. `node.hasChildNodes()`
6. `node.cloneNode(deep)`
7. `node.hasAttributes()`
8. `node.isSupported()`
9. `node.normalize()`

e-Macao-16-4-683

DOM Interface: Element 1

`Element` extends `Node`

Most common node type in a document.

The only node type relevant for the `Attrs` interface.

Property: `Element.tagName` – the name of the element.

e-Macao-16-4-684

DOM Interface: Element 2

General methods:

```
element.getElementsByTagName(name)
element.hasAttribute(name)
```

Attribute node methods:

```
element.getAttributeNode(name)
element.setAttributeNode(newAttr)
element.removeAttributeNode(oldAttr)
```

e-Macao-16-4-685

Example: Inserting the Text Node 1

Consider the credit card application document:

```
<?xml version="1.0"?>
<letter decision="rejected">
   <customer>Simon White</customer>
   <product>credit card</product>
   <officer level="manager">Steven Rod</officer>
   <enclosure>credit card</enclosure>
   <enclosure>initial PIN</enclosure>
   <cc></cc>
</letter>
```

Suppose we would like to transform this document by adding to the element `cc` the text “to archives”.

e-Macao-16-4-686

Example: Inserting the Text Node 2

Add to the read-write application after:

```
document = builder.parse(new File(argv[0]));
```

Retrieve the root element from the document:

```
Element elem = document.getDocumentElement();
```

Create the next text node to hold the text “to archives”:

```
Text text = document.createTextNode("to archives");
```

e-Macao-16-4-687

Example: Inserting the Text Node 3

Get the last child of the root element:

```
Node cc = elem.getLastChild();
```

Append the new text node as the last child of the cc element:

```
cc.appendChild(text);
```

e-Macao-16-4-688

Demo: DOM Inserting Application

```
> cd “dom new child”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

e-Macao-16-4-689

Task: Element-Removing Application

Remove the second enclosure element from the credit card document.

Use:

1. `getElementByTagName`
2. `removeChild` of the `Element` interface

e-Macao-16-4-690

Example: Element-Removing Application

Add to the read-write application after:

```
document = builder.parse(new File(argv[0]));
```

Retrieve the root element:

```
Element elem = document.getDocumentElement();
```

Obtain the list of all `enclosure` children of the root:

```
NodeList nodes = elem.getElementsByTagName("enclosure");
```

Remove the second child from this list:

```
elem.removeChild(nodes.item(1));
```

e-Macao-16-4-691

Demo: DOM Element-Removing Application

```
> cd "dom remove child"
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

e-Macao-16-4-692

Task: Attribute-Changing Application

Change the letter's `decision` attribute from `rejected` to `accepted`.

Use:

1. `getElementByTagName`
2. `removeChild` of the `Element` interface

e-Macao-16-4-693

Example: Attribute-Changing Application

Add to the read-write application after:

```
document = builder.parse(new File(argv[0]));
```

Retrieve the root element, change the attribute value:

```
Element elem = document.getDocumentElement();
elem.setAttribute("decision", "accepted");
```

e-Macao-16-4-694

Demo: DOM Attribute-Changing Application

```
> cd "dom change attribute"
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

e-Macao-16-4-695

DOM Interface: Attributes

`Attr` interface extends `Node`.

Properties:
1. `attr.ownerElement` – the element this attribute is attached to
2. `attr.name` – the name of the attribute
3. `attr.specified` – true if value was specified in the document
4. `attr.value` – value of the attribute
   - if read – returned as string
   - if write – text node with a given string

However, attributes are not part of the tree.

Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree.

e-Macao-16-4-696

DOM Interface: Character Data

`CharacterData` interface extends `Node`.

Used to access character data.

Properties:

1. `node.data` – character data of the node
2. `node.length` – number of characters

Methods:

1. `cData.substringData(offset, count)`
2. `cData.appendData(arg)`
3. `cData.insertData(offset,arg)`
4. `cData.deleteData(offset,count)`
5. `cData.replaceData(offset,count,arg)`

e-Macao-16-4-697

DOM Interface: Document 1

`Document` interface extends `Node`.

Represents the entire document.

Contains information about the document type, document element and implementation.

Provides methods for:

1. abstract factory for creating the document's components
2. finding specific components in the document

e-Macao-16-4-698

DOM Interface: Document 2

Properties:

1. `node.doctype` – the document type of the document
2. `node.implementation` – the document's DOM implementation
3. `node.documentElement` – convenient access to the root element

Search methods:

1. `document.getElementsByTagName(tagname)`
2. `document.getElementByID(tagname)`

e-Macao-16-4-699

DOM Interface: Document 3

Abstract factory methods:

1. `document.createElement(tagName)`
2. `document.createDocumentFragment()`
3. `document.createTextNode(data)`
4. `document.createComment(data)`
5. `document.createCDATASection(data)`
6. `document.createProcessingInstruction(target,data)`
7. `document.createAttribute(name)`
8. `document.createEntityReference(name)`

e-Macao-16-4-700

DOM Interface: DOM Implementation

Represents the current implementation of the DOM.

Methods:

1. `DOMImplementation.hasFeature(feature, version)`
2. `createDocumentType(qualifiedName,publicID,systemID)`
3. `createDocument(namespaceURI,qualifiedName,docType)`

e-Macao-16-4-701

DOM Interface: DOM Implementation Features

XML Module:	`XML`
HTML Module:	`HTML`
Views Module:	`Views`
StyleSheet Module:	`StyleSheets`
CSS Module:	`CSS`
CSS (extended) Module:	`CSS2`
Event Module:	`Events`
User Interface Events:	`UIEvents`
Mouse Events Moudule:	`MouseEvents`
Mutation Events Module:	`MutationEvents`
Traversal Module:	`Traversal`
Range Module	`Range`

e-Macao-16-4-702

DOM Interface: DOM Exception

Defines error codes for specific processing situations:

```
INDEX_SIZE_ERR                 1
DOMSTRING_SIZE_ERR             2
HIERARCHY_REQUEST_ERR          3
WRONG_DOCUMENT_ERR             4
INVALID_CHARACTER_ERR          5
NO_DATA_ALLOWED_ERR            6
NO_MODIFICATION_ALLOWED_ERR    7
NOT_FOUND_ERR                  8
NOT_SUPPORTED_ERR              9
INUSE_ATTRIBUTE_ERR           10
INVALID_STATE_ER       11
SYNTAX_ERR          12
INVALID_MODIFICATION_ERR      13
NAMESPACE_ERR          14
INVALID_ACCESS_ERR         15
```

e-Macao-16-4-703

DOM Validation with DTD 1

How to validate with the DOM parser?

Extend the credit card application with DTD declaration:

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter
   (customer,product,officer,enclosure*)>
  <!ATTLIST letter decision CDATA #REQUIRED>
  <!ELEMENT customer (#PCDATA)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT officer (#PCDATA)>
  <!ATTLIST officer level CDATA #IMPLIED>
  <!ELEMENT enclosure (#PCDATA)>
  <!ELEMENT cc (#PCDATA)>
]>
<letter decision="rejected">
  …
</letter>
```

e-Macao-16-4-704

DOM Validation with DTD 2

How to validate with the DOM parser?

```
…
DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
```

Set the validating feature for the DOM factory objects:

```
factory.setValidating(true);
try {
   DocumentBuilder builder = factory.newDocumentBuilder();
```

Register an instance of the current class as the error handler:

```
   builder.setErrorHandler(new App());
   document = builder.parse(new File(argv[0]));
}
catch {…}
```

e-Macao-16-4-705

DOM Validation with DTD 3

Implement error handlers:

```
public void error(SAXParseException exception) {
   System.out.println("Error: " + exception.getMessage());
}

public void fatalError(SAXParseException exception) {
   System.out.println("Fatal Error: " +
    exception.getMessage());
}

public void warning(SAXParseException exception) {
   System.out.println("Warning :" +
    exception.getMessage());
}
```

e-Macao-16-4-706

Demo: DOM Validation with DTD

```
> cd "dom validate DTD"
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

e-Macao-16-4-707

DOM Validation with Schema 1

How to validate with the Schema parser?

Packages as before plus `XMLConstants`:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import org.xml.sax.helpers.DefaultHandler;
import java.io.File;
import org.w3c.dom.*;
import org.w3c.dom.DOMException;
import javax.xml.validation.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;

import javax.xml.XMLConstants;
```

e-Macao-16-4-708

DOM Validation with Schema 2

```
public class App {

   static Document document;
```

Set the schema validation feature:

```
    static final String SCHEMA_VALIDATION_FEATURE_ID =
    "http://apache.org/xml/features/validation/schema";
```

Schema is referred to from the command line:

```
   public static void main(String[] argv) {
    if (argv.length != 2) {
                System.err.println("Usage: java App
          xmlfile xmlschemafile");
                System.exit(1);
          }
```

e-Macao-16-4-709

DOM Validation with Schema 3

```
try {
```

Parse an XML document into a DOM tree:

```
DocumentBuilder parser =
   DocumentBuilderFactory.newInstance().newDocumentBuilder(
   );
Document document = parser.parse(new File(argv[0]));
```

Create a `SchemaFactory` capable of understanding schemas:

```
SchemaFactory factory =
   SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS
   _URI);
```

Load a WXS schema, represented by a Schema instance:

```
Source schemaFile = new StreamSource(new File(argv[1]));
Schema schema = factory.newSchema(schemaFile);
```

e-Macao-16-4-710

DOM Validation with Schema 4

Load a schema:

```
Source schemaFile = new StreamSource(new File(argv[1]));
Schema schema = factory.newSchema(schemaFile);
```

Create a Validator instance to validate an instance document:

```
Validator validator = schema.newValidator();
```

Validate the DOM tree:

```
   validator.validate(new DOMSource(document));
} catch (Exception e) {
   System.out.println(e.getMessage());
}
}
}
```

e-Macao-16-4-711

Demo: DOM Validation with Schema

```
> cd "dom validate schema"
> dir
App.java noNamespace.xml schemaNoNamespace.xsd
> javac App.java
> java App noNamespace.xml schemaNoNamespace.xsd
```

A.4.3. XSLT

Java and XSLT

e-Macao-16-4-713

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C
2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces
3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)
4) XML Java Processing
   a) event-based programming (SAX)
   b) tree-based programming (DOM)
   c) rule-based programming (XSLT)

e-Macao-16-4-714

XSLT and Java

XSLT – Extensible Stylesheet Language Transformation has been introduced before.

How to use XSLT from Java applications?

- formulate the transformation template in external XSLT file
- use the transformer class as in the read-write DOM application

e-Macao-16-4-715

Java XSLT Application 1

Import the packages as before:

```
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import java.io.FileOutputStream;
import javax.xml.XMLConstants;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
```

e-Macao-16-4-716

Java XSLT Application 2

The beginning as the read-write DOM application:

```
public class App {

    static Document document;

    public static void main(String[] argv) {
        if (argv.length != 3) {
            System.err.println("Usage:
               java App xmlfile xslfile outfile");
            System.exit(1);
        }
```

e-Macao-16-4-717

Java XSLT Application 3

Parser an XML document into a DOM tree:

```
try {
    DocumentBuilder parser =
    DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document document = parser.parse(new File(argv[0]));
```

Setup the transformer according to the external XSLT file:

```
    TransformerFactory tFactory =
       TransformerFactory.newInstance();
    Transformer transformer =
       tFactory.newTransformer(new StreamSource(argv[1]));
```

e-Macao-16-4-718

Java XSLT Application 4

Perform the transformation on the parser-generate document model, sending the output to the specified file:

```
   transformer.transform(
       new DOMSource(document),
       new StreamResult(new FileOutputStream(argv[2])));

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

e-Macao-16-4-719

DEMO: Java XSLT Application

```
> cd "dom xslt"
> dir
App.java farewell.xml farewell.xsl
> javac App.java
> java App farewell.xml farewell.xsl output
> type output
```

e-Macao-16-4-720

Acknowledgements

The author would like to thank Milton Chau, Elsa Estevez, Brian Iu, Adegboyega Ojo, Gabriel Oteniya and Frank Wong for their comments, code and support during the preparation and delivery of this course.

B. Assessment

B.1. Set 1

1	What is the latest version of XML?		tick
	a	1.0	
	b	1.1	x
	c	1.4	
	d	2.0	

2	Which of the following languages apply XML syntax?		tick
	a	DTD	
	b	XSLT	x
	c	PDF	
	d	SVG	x

3	How many bytes are used in UTF-8 to represent a single character?		tick
	a	1, if the character belongs to the Latin alphabet	x
	b	2, for all characters	
	c	between 1 and 2	
	d	between 1 and 4	x

4	Which character encodings must be supported by an XML processors?		tick
	a	ASCII	
	b	UTF-32	
	c	UTF-8	x
	d	big5	

5	Which attributes must appear in an XML declaration?		tick
	a	xml	
	b	xml-stylesheet	
	c	version	x
	d	encoding	

6	Which of the following strings are legal XML names?		tick
	a	bed-breakfast	x
	b	bed&breakfast	
	c	-done	
	d	xml.class	x

7	Consider the document on the right. Which of the statements below are true?	<!DOCBOOK publisher [<!ELEMENT publisher (#PCDATA)>]> <publisher>Chapman&Hall</publisher>	tick
	a	The document is not well-formed and not valid	x
	b	The document is not well-formed but valid.	
	c	The document is well-formed but not valid.	
	d	The document is well-formed and valid.	

8	Consider the XML fragment on the	<e e=””/>	tick

	right. What is wrong with it?			
	a	no XML declaration		
	b	missing end-tag		
	c	element and attribute have the same names		
	d	nothing		x

9	Consider the XML fragment on the right. How to fill "…" to make it well-formed?		<e1><e2 a=…	tick
	a	<e2/><e1/>		
	b	<e1/><e2/>		
	c	` `</e2></e1>		x
	d	" "</e2></e1><!- the end ->		x

10	Consider the XML fragment on the right. How to fill "…" to make it valid?		<!DOCTYPE e [<!ELEMENT e (d?)+> <!ELEMENT d EMPTY>]> <e>…</e>	tick
	a	<d>text</d>		
	b	<d/>		x
	c			x
	d	<d/><d/>		x

11	Consider the XML fragment on the right. How to fill "…" to make it valid?		<!DOCTYPE e [<!ELEMENT e (#PCDATA)> …]> <e>text</e>	tick
	a			x
	b	<!ATTLIST e a CDATA #REQUIRED>		
	c	<!ATTLIST e a CDATA #IMPLIED>		x
	d	<!ATTLIST e a CDATA "text">		x

12	What kind of markup may occur between a start-tag and an end-tag?		tick
	a	element	x
	b	attribute	
	c	entity reference	x
	d	entity declaration	

13	Which of the following are XML well-formedness constraints:		tick
	a	an entity must not contain a recursive call to itself	x
	b	an attribute name may not repeat, unless in empty-element tag	
	c	all entities referenced in a document must be declared	
	d	the name in the start-tag must match the name in the end-tag	x

14	What kind of entity is declared here? <!ENTITY e SYSTEM "f">		tick
	a	parameter	
	b	general internal	
	c	external parsed	x
	d	external unparsed	

15	Which namespace does the element f belong to?	<e xmlns:n=”1”> <n:f xmlns:n=”2”/> </e>	tick
	a	1	
	b	2	x
	c	default	
	d	none	

16	Which namespace does the element f belong to?	<e xmlns=”1”> <f xmlns=””/> </e>	tick
	a	1	
	b	default	
	c	none	x
	d	the document is not well-formed	

17	How many times can the element e occur?	<element name=”e” minOccurs=”2”/>	tick
	a	2	
	b	1	
	c	0	x
	d	not well-formed schema	

18	Which statements below about XML are true?		tick
	a	Elements can have simple types.	x
	b	Attributes can have simple types.	x
	c	Elements can have complex types.	x
	d	Attributes can have complex types.	

19	What is the DTD corresponding to the following schema content model?	<all> <element ref=”a”/> <element ref=”b”/> </all>	tick
	a	(a,b)	
	b	(a|b)	
	c	(a?|b?)	
	d	none	x

20	How to write an Xpath expression to access all grandchildren “note” of a “book” child element, which contain the attribute “type”?		tick
	a	book/note[type]	
	b	book//note[@type]	
	c	book/*/note[@type]	x
	d	book/*/note[@type=’check it’]	

B.2. Set 2

1	Which of the following languages apply XML syntax?		tick
	a	DTD	
	b	XSLT	x
	c	PDF	
	d	SVG	x

2	How many bytes are used in UTF-8 to represent a single character?		tick
	a	1, if the character belongs to the Latin alphabet	x
	b	2, for all characters	
	c	between 1 and 2	
	d	between 1 and 4	x

3	Which attributes must appear in an XML declaration?		tick
	a	xml	
	b	xml-stylesheet	
	c	version	x
	d	encoding	

4	Which of the following strings are legal XML names?		tick
	a	bed-breakfast	x
	b	bed&breakfast	
	c	-done	
	d	xml.class	x

5	Consider the document on the right. Which of the statements below are true?	<!DOCBOOK publisher [<!ELEMENT publisher (#PCDATA)>]> <publisher>Chapman&Hall</publisher>	tick
	a	The document is not well-formed and not valid	x
	b	The document is not well-formed but valid.	
	c	The document is well-formed but not valid.	
	d	The document is well-formed and valid.	

6	Consider the XML fragment on the right. What is wrong with it?	<e e=””/>	tick
	a	no XML declaration	
	b	missing end-tag	
	c	element and attribute have the same names	
	d	nothing	x

7	Consider the XML fragment on the right. How to fill “…” to make it valid?	<!DOCTYPE e [<!ELEMENT e (d?)+> <!ELEMENT d EMPTY>]> <e>…</e>	tick
	a	<d>text</d>	
	b	<d/>	x
	c		x

	d	<d/><d/>	x

8	What kind of markup may occur between a start-tag and an end-tag?		tick
	a	element	x
	b	attribute	
	c	entity reference	x
	d	entity declaration	

9	Which of the following are XML well-formedness constraints:		tick
	a	an entity must not contain a recursive call to itself	x
	b	an attribute name may not repeat, unless in empty-element tag	
	c	all entities referenced in a document must be declared	
	d	the name in the start-tag must match the name in the end-tag	x

10	What kind of entity is declared here? <!ENTITY e SYSTEM "f">		tick
	a	parameter	
	b	general internal	
	c	external parsed	x
	d	external unparsed	

11	Which namespace does the element f belong to?	<e xmlns:n="1"> <n:f xmlns:n="2"/> </e>	tick
	a	1	
	b	2	x
	c	default	
	d	none	

12	How many times can the element e occur?	<element name="e" minOccurs="2"/>	tick
	a	2	
	b	1	
	c	0	x
	d	not well-formed	

13	Which statements below about XML Schema are true?		tick
	a	Elements can have simple types.	x
	b	Attributes can have simple types.	x
	c	Elements can have complex types.	x
	d	Attributes can have complex types.	

14	What is the DTD corresponding to the following schema content model?	<all> <element ref="a"/> <element ref="b"/> </all>	tick
	a	(a,b)	
	b	(a|b)	
	c	(a?|b?)	
	d	none	x

15	How to write an Xpath expression to access all grandchildren "note" of a	tick

	"book" child element, which contain the attribute "type"?		
	a	book/note[type]	
	b	book//note[@type]	
	c	book/*/note[@type]	x
	d	book/*/note[type]	

16		How to invoke <template match="…">…</template> in XSLT?	tick
	a	<match-template match="…"/>	
	b	<apply-template select="…"/>	x
	c	<call-template name="…"/>	
	d	wrong syntax	

17		What elements may occur within <choose>…</choose> in XSLT?	tick
	a	case	
	b	otherwise	x
	c	when	x
	d	between	

18		What elements may contain the element sort in XSLT?	tick
	a	none	
	b	template	
	c	apply-templates	x
	d	for-each	x

19		What content-handling events are provided by SAX?	tick
	a	character	x
	b	startElement	x
	c	warning	
	d	StartPrefixMapping	x

20		Which methods below belong to the DOM API?	tick
	a	getElementbyTagName	x
	b	getElement	
	c	createTextNode	x
	d	removeNode	

XML Technology and Java

Tomasz Janowski

The United Nations University IIST, Macau
University of Gdańsk, Poland

Objective

The course has two main objectives:

1) To provide students with the solid foundation and understanding of XML and XML-related technologies.
2) To develop skills in writing XML-processing Java applications using:
   a) SAX (Simple API for XML),
   b) DOM (Document Object Model) and
   c) XSLT (Extensible Stylesheet Language Transformations).

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

Timetable

7 days timetable:

- Introduction, Unicode
- XML
- DTD
- Namespaces, XML Schema
- XPath, XSLT
- Java XML with DOM
- Java XML with SAX and XSLT

Organization

The course consists of:

- lectures
- demonstrations
- assignments (tasks to perform)
- project work

Literature

The basis for the course are official technical documents of W3C:

- World Wide Web Consortium, Technical Reports, http://www.w3c.org/TR/

Recommended general books:

- Erik T. Ray, Learning XML, O'Reilly, 2001
- Kenneth B. Stall, XML Family of Specifications, Addison Wesley, 2003

More Literature

Recommended specialised books:

- Processing XML with Java, E. R. Harold, Addison Wesley
- XML Internationalisation and Localization, Yves Savourel, SAMS
- XML Topic Maps, Jack Park (Ed.), Addison Wesley
- Secure XML, D. E. Eastlake III and Kitty Niles, Addison Wesley
- XML Data Management, A. Chaudhri et. al., Addison Wesley
- ebXML, A. Walsh, Prentice Hall
- XML Distributed Systems Design, A.M. Rambhia, SAMS
- Modelling XML Applications with UML, D. Carlson, Addison Wesley
- etc.

History

This course was delivered before:

1) September 2003, UNU-IIST, Macau, 40 hours course for Macau IT staff from government, academia and industry.
2) October 2003 – January 2004, University of Gdańsk, Poland, 120 hours (60 hours of lectures and 60 of exercises) monograph elective course for Master degree students.

Motivation

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

World Wide Web

If it ain't broke, don't fix it.
Millions of people surf the Web every day:

- home-makers searching for do-it-yourself recipes
- students looking for help in their mid-term projects
- investors seeking the latest stock quotes
- tourists exploring the best holiday packages
- readers purchasing books from on-line bookshops
- researchers learning the latest results by their peers

World Wide Web works well for them. Or does it?

Problems with the Web

1) Browser-specific extensions
2) Explicit browser support
3) Browser orientation
4) Structure and style intermixed
5) Data exchange is problematic
6) Monitor orientation
7) Unfocused searches
8) Static content
9) One-page limitation
10) One-way linking only
11) etc.

Problems: Browser-Specific Extensions

HTML standards take a long time to agree. Not willing to wait, vendors decide to introduce browser-specific extensions.

This HTML displays differently in every browser we tested:

```
<html>
  <head>
    <title>Welcome Message</title>
  </head>
  <body>
    <marquee>Welcome</marquee>
    to the
    <blink>XML Technology</blink>
    course!
    <img src="smiley.gif"/>
  </body>
</html>
```

Demo: Browser-Specific Extensions

```
> cd "demos/browser-specific extensions"
> dir
smiley.gif welcome.html
> opera welcome.html
> netscape welcome.html
> iexplorer welcome.html
> amaya welcome.html
```

Problems: Explicit Browser Support

Browser-specific features cause a dilemma for content providers:

- use old features only
- support one browser

- support multiple browsers:

```
<script language="javascript">
  if (version < 4.0)
    location.href='index1.html';
  if (vendor == 'Netscape')
    location.href='index2.html';
  if (vendor == 'Microsoft')
    location.href='index3.html';
</script>
<noscript>
  <a href="index4.html">
    No scripting.
  </a>
</noscript>
```

Problems: Browser Orientation

A Web browser became a launcher for applications, not just a tool for surfing the Web.

This paradigm is too restrictive. Our everyday applications

1. editors
2. spreadsheets
3. media players, . . .

should access the Web directly.

HTML calling a Java applet:

```
<html>
<body>
<h1>XML Technology Course</h1>
<applet code="menuscroll2.class">
<param name="text1" value="XML"/>
<param name="text2" value="DTD"/>
<param name="text3" value="DOM"/>
...
</applet>
</body>
</html>
```

Demo: Browser Orientation

```
> cd "demos/browser orientation"
> dir
menuScroll.html menuscroll2.class
> opera menuScroll.html
```

Problems: Structure and Style Intermixed

- Structural elements (title) and style elements (i) are freely intermixed in HTML.
- Such documents are tied to a single style vocabulary, and difficult to convert.

```
<html> ...
XML is <i>fun</i>.
In Polish we say
<i>fajny</i>
</html>
```

```
<html> ...
XML is
<emphasis>fun</emphasis>.
In Polish we say
<foreign>fajny</foreign>.
</html>
```

Problems: Monitor Orientation

With the growing number of Internet-connected devices:

1) computers
2) phones
3) hand-held devices
4) TV, etc.

data description must be independent of the device on which it will be displayed.

```
<html>
<head>
<title>Framed Page</title>
</head>
<frameset cols="100,*">
<frame name="navigation" .../>
<frame name="main" .../>
<noframes>
<p> your browser does
not support frames</p>
</noframes>
</frameset>
</html>
```

Problems: Unfocused Search Engines

Most search engines can only index the frequency of words in a document, producing thousands of hits, most missed. We need to convey semantic information about the document content.

```
<html>
  <head>
    <meta name="keywords" content="XML course macao"/>
    <meta name="description" content="This site ..."/>
    <title>XML Technology Course</title>
  </head>
  <body>
    <p> We invite applications to attend ... </p>
  </body>
</html>
```

More Problems

- Data exchange is problematic

  HTML is not suited for data exchange: filtering, integrating from different sources, verifying that data has required types.

- Static content

  On data-intensive web sites, content changes frequently. Presentation should be generated for a given content, and re-generated every time the content changes.

More Problems

- One-page limitation

  The Web does not support handling collections of inter-related documents, except one at a time. We needs ways to express relations within a document and between different documents.

- One-way linking

  The current one-way linking mechanism is too restrictive. We need more powerful mechanisms: links with multiple targets, multi-directional links and external link-bases.

Overview

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

XML

XML is called upon to solve such problems.

- XML is not to replace HTML.
- HTML will be (is currently) absorbed into XML as a pure version of itself: XHTML.
- XML is to create a more solid and flexible foundation for the next generation Internet technology.

What is XML?

1) a protocol for containing and managing information

2) a family of technologies that can do anything from writing, validating, presenting, to processing documents

3) a philosophy promoting the maximum usefulness for data by refining it to its purest and most structured form

What is not XML?

- It is not a programming language:

  XML does not prescribe an execution to be carried out by a machine, unlike e.g. Java

- It is not a presentation language:

  XML does not contain instructions to render a document, unlike e.g. Postscript

XML as a Language

"L" in XML stands for *Language*.

XML provides a notation to write self-describing data. It does so by capturing the structure of data separately from its style.

- **syntax:** XML documents have well-defined syntax.
- **semantics:** XML does not assign semantics to its documents, but permits external applications to do so.

XML as a Meta-Language

“X” in XML stands for *eXtensible*.

XML is a *meta-language* - a syntax to describe other languages.

Those languages can span diverse industrial domains.
Notably, XML can be applied to describe its own extensions.

XML Domains

vertical domains		horizontal domains	
law	LegalXML	validation	XML Schema
news	NewsML	transformation	XSLT
finances	Visa Invoice	presentation	XSL-FO
business	ebXML	navigation	XLink
telephony	VoiceXML	retrieval XML	Query
publishing	XHTML	distribution	SOAP
governance	GovML	security	XML Encryption
…		…	

XML as a Meta-Markup-Language

“M” in XML stands for *Markup*.

XML instance languages are *markup languages*: they annotate their expressions (documents) with *markup* to highlight their structure.

What is markup then?

Markup

- Markup is information added to a document to enhance its meaning in certain ways, by identifying parts of a document and how they relate to one another.

- Markup language is a set of symbols that can be placed in the text of a document to demarcate and label its parts.

Example: XML Markup

```
<?xml version="1.0"?>
<message date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

- boundaries – beginning and end tags
- roles – tag name to identify the element's role
- meta-data – attributes to inform about content
- position – one element comes before another
- containment – one element is inside another
- relationships – linking to external resources

Demo: XML in Browsers

```
> cd "demos/xml in browsers"
> ls
welcome.xml
> opera welcome.xml
> iexplore welcome.xml
```

Overview of XML

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up

Overview of XML: Self-Describing Data

The name of an element describes the meaning or function of the data it contains:

- from – sender
- to – receiver
- emphasis – important text

An application can then process the document taking into account this semantic information

```
<?xml version="1.0"?>
<message
   date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

Overview of XML: Flexible Structuring

Two approaches to data modeling:

1. Data-centric XML stores all data inside elements.

2. Document-centric XML interleaves elements with text.

XML permits both approaches.

Example: Data- versus Document-Centric

```
<?xml version="1.0"?>
<message date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

```
<?xml version="1.0"?>
<message>
<from>Tomasz</from> sends
a welcome message to
<to>Participants</to>:
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</message>
```

Overview of XML: Structure-Style Separation

XML versus HTML:

1. XML describes the structure of data, regardless of how it will be formatted for presentation.

2. HTML contains a mixture of the structural (title) and presentation (table) markup.

Example: Structure versus Style

```
<?xml version="1.0"?>
<message date="15.09.2003">
<from>Tomasz</from>
<to>Participants</to>
<subject>Welcome</subject>
<body>
Welcome to the
<emphasis>XML Technology
</emphasis> course!
<cheers img="smiley.jpg"/>
</body>
</message>
```

```
<html>
<table>
<tr>
<td><b>from</b></td>
<td>Tomasz</td>
...
</tr>
</table>
Welcome to the
<i>XML Technology</i> course!
<img src="smiley.gif"/>
</html>
```

Overview of XML: Style Kept Externally

XML versus HTML:

- HTML is restricted to particular presentation.
- One XML document can be formatted in different ways.
- The formatting information is kept in a stylesheet document.
- The stylesheet document is external to the instance document.
- The stylesheet may be referred from the instance document.

Example: External Stylesheet

Given this CSS (Cascading Stylesheets) document (welcome.css)

```
to:before {content: "to:"}
from:before {content: "from:"}
subject:before {content: "subject:"}
body {display:block}
```

and this XML document:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="welcome.css"?>
<message>...</message>
```

a CSS-aware browser produces this result:

```
from: Tomasz to: Participants subject: Welcome
Welcome to the XML Technology course!
```

Example: Another External Stylesheet

while applying this stylesheet:

```
to:before {content: "to:"}
from:before {content: "from:"}
to, from {display:block; font-weight: bold}
subject {display:none}
emphasis {font-style: italic}
```

the result is as follows:

from: Tomasz
to: Participants
Welcome to the *XML Technology* course!

Demo: Different Presentations

```
> cd "demo/different presentations"
> dir
welcome1.css welcome2.css welcome1.xml welcome2.xml
> opera welcome1.xml
> opera welcome2.xml
```

Overview of XML: Document Transformations

More radical results can be obtained through XSLT:

- eXtensible Stylesheet Language Transformations
- XSLT is a fully-fledged declarative programming language specializing in XML transformations
- XSLT programs are themselves written in XML

Example: XML Transformations with XSLT 1

This program generates HTML to display the welcome message.

Elements prefixed by xsl are formatting instructions of XSLT.

```
<xsl:stylesheet
   xmlns:xsl="www.w3.org...">
  <xsl:template match="message">
    <html>
      <table> ...
        <td><b>from</b></td>
        <td>
          <xsl:value-of select="from"/>
        </td>
      </table>
      <xsl:value-of select="body"/>
      <img src="smiley.gif"/>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Example: XML Transformations with XSLT 2

Here is the generated HTML:

```
<html>
   <table>
    <tr><td><b>from</b></td><td>Tomasz</td></tr>
    <tr><td><b>to</b></td><td>Participants</td></tr>
   </table>
   Welcome to the <i>XML Technology</i> course!
   <img src="smiley.gif"/>
</html>
```

and the output rendered by a browser:

from Tomasz
to Participants
subject Hello
Welcome to the *XML Technology* course!

Demo: HTML Generated by XSLT

```
> cd "demos/html generated by xslt"
> dir
welcome.xml welcome.xsl smiley.gif
> xalan welcome.xml welcome.xsl welcome.html
> ls
welcome.xml welcome.xsl welcome.html smiley.gif
> opera welcome.html
```

Demo: Built-in XSLT Processing

```
> cd "demos/built-in xslt processing"
> dir
welcome.xml welcome.xsl smiley.gif
> iexplore welcome.xml
```

Overview of XML: Where are We?

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations HERE!
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up

Overview of XML: Document Processing

Transformations from XML to HTML is just one kind of processing.

Many other transformations are possible:

- XML to XML
- XML to text
- XML to PDF
- XML to Latex
- XML to troff
- etc.

They may have nothing to do with the Web.

Example: Generating LaTeX from XML

- XSLT to build the welcome message
- the input in XML
- the output in LaTeX
- notice the interleaving of different kinds of markup

```
<xsl:stylesheet
   xmlns:xsl="www.w3.org...">

<xsl:template match="message">
   \documentclass{article}
   \title{<xsl:value-of
    select="subject"/>}
   \begin{document}
   ...
   <xsl:apply-templates
    select="body"/>
   \end{document}
</xsl:template>

<xsl:template match="emphasis">
   {\it <xsl:value-of select="."/>}
</xsl:template>

</xsl:stylesheet>
```

Demo: LaTeX Generated by XSLT

```
> cd "demos/latex generated by xslt"
> dir
welcome.xml welcome.xsl smiley.eps
> xalan welcome.xml welcome.xsl welcome.tex
> dir
welcome.xml welcome.xsl welcome.tex smiley.eps
> latex welcome.tex
> dir
welcome.xml welcome.xsl welcome.tex welcome.aux
welcome.dvi welcome.log smiley.eps
> yap welcome.dvi
```

Overview of XML: Programming Support

Major programming languages all provide support for XML.

Notably, these include C++, Java and Perl.

Programming support falls into:

- event-based: a program responds to the events generated by an XML parser, as it is processing an XML document (SAX)
- tree-based: a program traverses and modifies locally a parser-generated document tree (DOM)
- rule-based: a program executes recursive transformation rules, causing global modifications to the document tree (XSLT)

Example: Document Object Model API

Java code to process an XML document.

A parser is invoked on the input document.

DOM API is used to traverse recursively the parser-generated tree.

```
package dom;
public class Counter {
  public void count(Node node) {
    switch (node.getNodeType()){
      case Node.TEXT_NODE: { ... }
      case Node.ELEMENT_NODE: {
      Node child = node.getFirstChild();
      while (child != null) {
        count(child);
        child = child.getNextSibling();
  }}}}
  public static void main(String argv[]) {
    Counter counter = new Counter();
    document = parser.parse(argv[2]);
    counter.count(document);
  }
}
```

Demo: XML Processing with Java DOM API

```
> cd "demos/xml processing with java dom api"
> dir
welcome.xml
> java dom.Writer welcome.xml
> java dom.Counter welcome.xml
> java dom.GetElementsByTagName -e message welcome.xml
> java dom.GetElementsByTagName -a img welcome.xml
```

Overview of XML: International Support

XML supports Unicode:

- UTF8 – 8-bit Unicode – is the default encoding
- every XML processor must support both UTF8 and UTF16
- they may, and usually do, support alternative character sets
- text, element and attribute names can all be international
- several languages can co-exist in one document

Example: XML Document in Polish

Here is a welcome message in Polish.

iso-8859-2 is the Central-European encoding.

Text and tag/attribute names can all appear in Polish.

```
<?xml version="1.0"
   encoding="iso-8859-2"?>
<wiadomość data="15.09.2003">
   <od> Tomasz </od>
   <do> Uczestnicy </do>
   <tytuł> Powitanie</tytuł>
   <treść>
    Witam na kursie
    <ważne>Technologii
    XML</ważne>
    <czołem obraz="smiley.jpg"/>
    This is a message in Polish:
    ąćęłńóśźĄĆĘŁŃÓŚŹŻ
   </treść>
</wiadomość>
```

Demo: XML in Polish

```
> cd "demos/xml in polish"
> ls
polish.xml
> opera polish.xml
> iexplore polish.xml
```

Overview of XML: Strict Rules of Syntax

An XML document must be well-formed to be process-able by XML-compliant applications:

- XML parsers are explicitly required not to process ill-formed XML, but to exit with a suitable error message
- Browsers accept ill-formed HTML, trying to guess the intentions of the document's author.
- A lot of browser code goes to processing ill-formed HTML, increasing complexity and decreasing predictability.

Example: Ill-Formed XML

1. date lacks quotes
2. emphasis is misspelled
3. cheers does not end
4. message and body overlap

```
<?xml version="1.0"?>
<message date=15.09.2003>
   <from>Tomasz</from>
   <to>Participants</to>
   <subject>Welcome</subject>
   <body>
    Welcome to the
    <emphasis>XML Technology
    </emphazis> course!
    <cheers img="smiley.jpg">
   </message>
</body>
```

Demo: Ill-Formed XML

```
> cd "demos/ill-formed xml"
> dir
welcome.xml welcomeIll.xml
> cp welcomeIll.xml welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
> emacs welcome.xml
> opera welcome.xml
```

Overview of XML: Enforcing Validation Rules

Well-formedness contains the most basic rules of syntax checking, common to all kinds of XML documents.

Validation permits the expression and checking of the properties common to particular XML instance languages:

- A document is valid if it complies with certain rules of correctness defined for a language.
- One way to express such rules is Document Type Definition.
- A ill-formed document is invalid, but an invalid document need not be ill-formed.

Example: Invalid Well-Formed Document

This message is well-formed, but who is the receiver?

```
<?xml version="1.0"?>
<message date="15.09.2003">
   <from>Tomasz</from>
   <subject>Welcome</subject>
   <body>
    Welcome to the
    <emphasis>XML Technology
    </emphasis> course!
    <cheers img="smiley.jpg"/>
   </body>
</message>
```

Example: Validating Welcome Documents

Validation rules:

1. message is the root element
2. it contains elements from, to, subject and body
3. also the optional date attribute that contains character data
4. etc.

```
<?xml version="1.0"?>
<!DOCTYPE message welcome.dtd>
<message date="15.09.2003">
...
</message>
```

welcome.dtd

```
<!ELEMENT message (from,to,subject,body)>
<!ATTLIST message date CDATA #IMPLIED>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA | emphasis |
cheers)*>
<!ELEMENT emphasis (#PCDATA)>
<!ELEMENT cheers EMPTY>
<!ATTLIST cheers img CDATA #REQUIRED>
```

Demo: Invalid XML

```
> cd "demos/invalid xml"
> dir
welcome.xml welcomeInvalid.xml
> cp welcomeInvalid.xml welcome.xml
> xerces welcome.xml
> emacs welcome.xml
> xerces welcome.xml
```

Overview of XML: Where are We?

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules HERE!
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up

Overview of XML: Creating Languages

DTD provides a mechanism for language definition:

- `welcome.dtd` file leaves:
  - some freedom as to structure of valid XML documents
  - a lot of freedom as the textual content of such documents
- The file can be attached to various XML documents to check their validity with respect to its definitions.
- All XML documents valid with respect to `welcome.dtd` constitute a language of “welcome messages”.

Example: Another ValidWelcome Document

Welcome for applications for the Software Project Management course.

```
<?xml version="1.0"?>
<!DOCTYPE message welcome.dtd>
<message>
   <from>UNU/IIST</from>
   <to>Students</to>
   <subject>Welcome</subject>
   <body>
    We invite applications to
    attend the Software Project
    Management course. The    deadline is
    <emphasis>15.09.2003 </emphasis>.
    Please apply to
    <emphasis>spm@iist.unu.edu
    </emphasis>.
   </body>
</message>
```

Demo: Valid XML

```
> cd "demos/valid xml"
> dir
projectManagement.xml welcome.dtd
> xerces projectManagment.xml
```

Overview of XML: Adopting Languages

Instead of defining our own language, chances are that a public XML language exists to suit our needs.

There are hundreds of XML languages defined:

1. MathML – mathematics in XML
2. SVG – vector graphics in XML
3. DocBook – authoring books in XML
4. CML – describing chemical molecules
5. Visa Invoice – writing invoices in XML
6. XHTML – authoring hypertext documents in XML
7. and many others

Example: MathML

Here is a fragment of MathML – an XML language to describe mathematical notation.

`mathml.dtd` is a public DTD for this language.

```
<?xml version="1.0"?>
<!DOCTYPE math
"http://www.w3.org/mathml.dtd">
<math>
<mi>x</mi><mo>=</mo>
<mfrac>
   <mrow>
    <mrow>
       <mo>-</mo><mi>b</mi>
    </mrow>
    <mo>&PlusMinus;</mo>
    <msqrt> ... </msqrt>
   </mrow>
   <mrow> ... </mrow>
</mfrac>
</math>
```

Demo: MathML

```
> cd "demos/mathml"
> dir
math.xml
> xerces math.xml
> opera math.xml
> amaya math.xml
```

Example: Scalable Vector Graphics

SVG is another example: a language to define scalable vector graphics.

Here the SVG markup to render a welcome text along a sinusoidal path.

`svg.dtd` is a public DTD for this language.

```
<?xml version="1.0"?>
<!DOCTYPE svg "http://www.w3.org/svg.dtd">
<svg width="12cm" height="3.6cm">
   <text fill="blue">
    <textPath xlink:href="#MyPath">
        Welcome to the
        <tspan dy="50" fill="red">
            XML Technology
        </tspan>
        course!
    </textPath>
   </text>
   <rect width="998" height="298"/>
</svg>
```

Demo: Scalable Vector Graphics

```
> cd "demos/scalable vector graphics"
> dir
welcome.svg
> xerces welcome.svg
> opera welcome.svg
> amaya welcome.svg
> iexplore welcome.svg
```

Overview of XML: Integrating Languages

Several vocabularies can be used in the same document:

- A prefix indicates which language an element comes from:

svg:title	title from SVG
xhtml:title	title from XHTML
math:title	title from MathML

- A prefix is declared referring to the language's official URL

- It tells XML processor how to process an element, or which external application should be invoked.

Example: XHTML and MathML and SVG

Here is a document using three languages:

1. XHTML,
2. MathML and
3. SVG.

```
<?xml version="1.0"?>
<html xmlns="www.w3.org/xhtml">
   <body>
    <p>Here is XHTML ... </p>
    <p>Here is MathML:
    <math xmlns="www.w3.org/MathML">
        ...
    </math>
    </p>
    <p>Here is SVG:
    <svg xmlns="www.w3.org/svg">
        ...
    </svg>
    </p>
   </body>
</html>
```

Demo: XHTML and MathML and SVG

```
> cd "demos/xhtml and mathml and svg"
> dir
mixed.xml
> amaya mixed.xml
```

Overview of XML: Self-Describing Technology

CSS/DTD predate XML and apply their own native syntax.

The general trend is to replace such legacy syntax with XML:

- XSLT – XML transformation language
- XML Schema – XML class description language

Example: XML Transformations in XML

XSLT is entirely expressed with XML.

An XML-transforming program is itself an XML document.

Here is an earlier transformation of the welcome message.

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="www.w3.org..."><xsl:template match="message">
<html>
   <table> ...
    <td><b>from</b></td>
    <td>
    <xsl:value-of select="from"/>
    </td>
   </table>
   <xsl:value-of select="body"/>
</html>
</xsl:template>
</xsl:stylesheet>
```

Example: XML Classes in XML

Also XML Schema – a language to define XML classes of XML documents – is entirely described in XML.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="www.w3.org/XMLSchema">
   <xsd:element name="message" type="Message"/>
   <xsd:complexType name="Message">
    <xsd:sequence>
        <xsd:element name="from" type="xsd:string"/>
        <xsd:element name="to" type="xsd:string"/>
        <xsd:element name="subject" type="xsd:string"/>
        <xsd:element name="body" type="Body"/>
    </xsd:sequence>
    <xsd:attribute name="date" type="xsd:date" use="optional"/>
   </xsd:complexType> ...
</xsd:schema>
```

Demo: XML Schema Validation

```
> cd "demos/xml schema validation"
> dir
welcome.xml welcome.xsd welcomeInvalid.xml
> cp welcomeInvalid.xml welcome.xml
> xercesSchema welcome.xml
> emacs welcome.xml
> xercesSchema welcome.xml
> emacs welcome.xml
> xercesSchema welcome.xml
> emacs welcome.xml
> xercesSchema welcome.xml
```

Overview of XML: Technology Clean-Up

Another trend is to reformulate pre-XML technologies in XML.

A case in point is XHTML, which comes in three flavors:

- strict – most XML-like and forward-moving
- transitional – retains enough HTML to use older browsers
- frameset – string XHTML plus frames

Legacy software is a concern when moving from non-XML to XML.

Example: XHTML

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <body>
    <table>
        <tr><td><b>from</b></td><td>Tomasz</td></tr>
        <tr><td><b>to</b></td><td>Participants</td></tr>
        <tr><td><b>subject</b></td><td>Welcome</td></tr>
    </table>
    <p>Welcome to the <i>XML Technology</i> course!
    <img src="smiley.gif" alt="Cheers"/></p>
   </body>
</html>
```

Demo: XHTML Validation

```
> cd "demos/xhtml validation"
> dir
welcome.html welcome.xhtml smiley.gif
> opera welcome.html
> opera welcome.xhtml
> emacs welcome.xhtml
> opera welcome.xhtml
```

Overview of XML: Where are We?

1. self-describing data
2. flexible structuring
3. structure-style separation
4. style kept externally
5. document transformations
6. document processing
7. programming support
8. international support
9. strict syntactic rules
10. enforcing validation rules
11. creating languages
12. adopting languages
13. integrating languages
14. self-describing technology
15. technology clean-up HERE!

Origins

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

XML Timeline

1967	GenCode
1969	Generalized Markup Language
1980's	GML adopted by government and industry
1986	Standard Generalized Markup Language
1989	World Wide Web = HTML + HTTP + URL
1991	World Wide Web is online
1994	XML is envisioned at the 2nd WWW Conference
1994	World Wide Web Consortium is founded
1996	W3C XML Activity Area starts
1998	XML becomes W3C Recommendation
1998	first applications of XML emerge
1999	Internet Explorer 5.0 – the first browser to support XML

Origins of XML: GenCode

1967	GenCode project of Graphic Communication Association (GCA) proposes the use of descriptive tags to aid the separation of content and presentation in electronic documents.

Origins of XML: GML

1969	Charles Goldfarb, Ed Mosher and Ray Lorie of IBM develop GML – the first document markup language using descriptive tags. GML added DTDs to GenCode.
1980's	GML is adopted by government and industry: 1. CALS Tables by the CALS group of the US DoD (format to represent tabular data). 2. Berglund of CERN – European Particle Physics Laboratory – develops a publishing system to test SGML.

Origins of XML: SGML

1978	American National Standards Institute (ANSI) forms a committee with Goldfarb and GCA Gen-Code group to work on the standard for GML.
1986	SGML – Standard Generalized Markup Language – becomes an ISO standard (8879:1986). SGML defines a syntax for creating application-specific markup languages with grammars expressed by DTDs.

Origins of XML: HTML

1989	Tim Berners-Lee makes the first proposal for WWW – a system to access hypertext documents on the Internet: 1. HTTP – HyperText Transfer Protocol 2. URL – Uniform Resource Locator 3. HTML – HyperText Markup Language HTML is an SGML document type for hypertext documents, simple and easy to program.
1991	Tim Berners-Lee announces his work on alt.hypertext and makes the first server and browser software freely available.

Origins of XML: XML is Envisioned

1994	C. M. Sperberg-McQueen and Robert F. Goldstein, „*HTML to the Max: A Manifesto for Adding SGML Intelligence to the WWW*”, 2nd WWW Conference: • HTML had to sacrifice the principles of GenCode to become truly useful. • Adapt SGML for the Web? SGML is too complex. • A new language is needed, as simple as HTML, but retaining the generality of SGML. • XML is envisioned.

Origins of XML: W3C is Formed

1994	WWW Consortium is founded by MIT, INRIA and Keio with headquarters in MIT, led by Tim Berners-Lee.
1995	HTML Working Group is formed by W3C.
1996	XML Activity Area, Phase 1, formed by W3C.
1997	First public XML specification draft presented, Tim Bray and C.M.Sperberg McQueen editors.
1998	HTML 4.0 becomes W3C Recommendation. HTML Working Group reforms to work on XHTML.

Origins of XML: XML is Formalized by W3C

1998	XML 1.0 becomes W3C Recommendation. From this moment on, 80% of W3C activities are related to XML.
1998	The first applications of XML emerge: Mathematical Markup Language (MathML) and Chemical Markup Language (1997).
1998	XML media types text/xml and application/xml proposed by Internet Engineering Task Force (IEFT).

Origins of XML: XML is Implemented

1998	DOM becomes W3C Recommendation.
1999	Namespaces in XML become W3C Recommendation.
1999	RDF becomes W3C Recommendation.
1999	Internet Explorer 5.0 is released – the first mainstream browser to support XML.
1999	“XML and the Second-Generation Web” by Jon Bosak and Tim Bray published in Scientific American.

Origins of XML: Open-Source XML Initiatives

1999	Apache XML project starts, soon producing a host of XML tools: xerces, xalan, cocoon, batik, etc.
1999	ebXML – a worldwide project to standardize XML business specifications – is initiated by: • UN/CEFACT (Trade Facilitation and Electronic Business organization) and • OASIS (Organization for the Advancement of Structured Information Standards).

Origins of XML: More Implementations

1999	XSL and XPath become W3C Recommendations.
2000	XHTML becomes W3C Recommendation.
2000	Opera 4.0 is released with XML support.
2000	Netscape 6.0 is released with XML support.
2000	Amaya 4.0 is released with support for HTML, XHTML, MathML, SVG and more.

Origins of XML: More W3C Recommendations

2001	XML Schema
2001	XLink and XBase
2001	Scalable Vector Graphics
2001	XSL Formatting Objects
2001	XML Information Sets

It is not history anymore. It is today.

W3C

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

W3C Founding

- Founded in 1994 by Tim Berners-Lee at the Massachusetts Institute of Technology, Laboratory for Computer Science.
- INRIA (Institut National de Recherche en Informatique et Automatique) joined as the first European W3C host in 1995.
- Keio University of Japan became the first Asian host in 1996.
- In 2003, European Research Consortium in Informatics and Mathematics took over the role of W3C Host from INRIA.

W3C Members

W3C consists of:

- a small full-time staff and
- hundreds of members: corporations, government agencies, universities, etc.

W3C issues Recommendations that explain how certain Web technologies should be used and integrated with existing ones.

Demo: Exploring W3C Site

```
> opera http://www.w3.org/
> opera http://www.w3.org/Consortium/
> opera http://www.w3.org/Consortium/Member/List
> opera http://www.w3.org/People/
```

W3C Recommendations

- W3C has no official jurisdiction to enforce its recommendations.
- The hope is that:

  once a consensus is reached on a particular specification, there will be sufficient vendor and developer support so that compliance results from “peer pressure”

- In the past, this hope now always materialized.

About W3C Recommendations

W3C Recommendations are written in a language that is precise and rigorous, albeit informal.

- They are not an easy reading: W3C Recommendations are aimed at XML developers, not XML users.
- There are 400+ XML books that explain and interpret W3C Recommendations at various levels, and courses like this one.

Demo: XML Resources

```
> opera http://www.w3.org/TR/
> opera www.amazon.com
> opera www.xml.org
```

W3C Recommendation Process

The emerge of a W3C Recommendation is the result of a formalized, often lengthy process:

1. An idea for a new technology arrives from:
   - a member,
   - industry or
   - the W3C team,

   perhaps published as a note on the W3C site.

2. The idea is added to existing Activity Area or, if none is appropriate, cause the creation a new Activity Area.

W3C Recommendation Process

3. The activity is assigned to:
   - one of Working Groups,
   - a special Interest Group or
   - a Coordination Group.

4. If sufficient interest is generated by the proposal, the Working Group eventually produced the first Working Draft.

5. W3C announces the Working Draft to solicit feedback from its members, the industry and the developer community.

W3C Recommendation Process

6. Subsequent revisions are published:
   - every three months,
   - until the Working Group decides to publish the Last Working Draft.

7. The Last Working Draft can be:
   - rejected at this point – sending back for more revisions, or
   - become a Candidate Recommendation.

W3C Recommendation Process

8. The Candidate Recommendation is a call to industry for:
   - implementations and
   - technical feedback

   This stage can be skipped if implementations already exist.

9. The outside experience may result in:
   - finalizing the Candidate Recommendation into Proposed Recommendation, or
   - sending it back into the Working Draft stage.

W3C Recommendation Process

10. Proposed Recommendation lasts from one to three months and is the last chance for members to suggest changes.

11. It is voted by members to become a W3C Recommendation:
    - as it is,
    - with changes,
    - return to the working draft, or
    - reject (drop from W3C activities).

Demo: W3C Recommendations

```
> opera http://www.w3.org/TR/
> opera http://www.w3.org/TR/#Recommendations
> opera http://www.w3.org/TR/#PR
> opera http://www.w3.org/TR/#PER
> opera http://www.w3.org/TR/#CR
> opera http://www.w3.org/TR/#WD
```

W3C Domains

The resources of W3C are spread across five domains:

1. Architecture Domain: to develop the underlying technologies of the Web (e.g. HTTP, XML, DOM, Jigsaw).

2. Document Formats Domain: to develop languages that Web content developers can use (e.g. SVG, XHTML, CSS).

W3C Domains

3. Interaction Domain: to improve user interaction with the Web(mobile access, multimedia, voice browsers)

4. Technology and Society Domain**:** to build infrastructure to address social, legal and public policy concerns (encryption).

5. Web Accessibility Initiative: to promote usability of the Web for people with disabilities (web content guidelines).

W3C XML Working Groups

There are five XML Working Groups:

1. XML CoreWorking Group: XML, namespaces
2. XML SchemaWorking Group: schema languages
3. XML Linking Working Group: XPointer, XLink, XML Base
4. XML Query Working Group: developing query data models
5. XML Coordination Group: coordination of all XML activities

W3C XML Working Groups

and three related ones:

1. XSL Working Group: XML transformation and display
2. DOM Working Group: models for manipulating documents
3. Web Services Working Group: XML-based protocols

Demo: W3C Areas and Working Groups

```
opera http://www.w3.org/Consortium/Activities
opera http://www.w3.org/XML/
```

Unicode

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

XML and Character Encoding

Coding matters:

- XML requires that after parsing, the XML document belongs to the Universal Character Set (UCS).
- XML processors must support UTF-8 and UTF-16 encodings of UCS.
- They may support other encodings of UCS and other character sets, but this is not required.

What does it all mean?

Coding Terminology

character: smallest component of a written language, such as: letters, digits, ideographs, punctuation and diacritical marks are all characters.

character set: a group of characters without associated numerical values, such as: Latin alphabet, Chinese phonetic symbols and Cyrillic alphabet.

coded character set: a character set where each character is associated with a code, such as ASCII or Unicode.

Universal Character Set

Character set developed by:

- ISO and
- the Unicode Consortium.

Supports most of the existing written languages.

Scripts in UCS

basic latin	0000-007F
latin supplement	0080-00FF
latin extended A	0100-017F
latin extended B	0180-024F
IPA extensions	0250-02AF
spacing modifier	02B0-02FF
diactritical	0300-036F
greek	0370-03FF
cyrillic	0400-04FF
armenian	0530-058F

hebrew	0590-05FF
arabic	0600-06FF
syriac	0700-074F
thanaa	0780-07BF
devanagari	0900-097F
bengali	0980-09FF
gurmukhi	0A00-0A7F
gujarati	0A80-0AFF
oriya	0B00-0B7F
tamil F	0B80-0BFF
...	

Scripts and Languages

One block is one script.

One script can support several languages.

For instance, Cyrillic is used to write:

1. Russian,
2. Bulgarian,
3. Ukrainian.

Some languages use several scripts.

Japanese uses:

1. Kanji
2. Hiragana
3. Katakana and
4. Romaji scripts.

Demo: Greek Letters in Unicode

```
> cd "demos/greek letters in unicode"
> acroread U0370.pdf
> opera polish-greek.xml
> iexplore polish-greek.xml
```

UCS Transformation Formats: UTF-8

UTF-8 – 8 bit encoding

- Each character is mapped to a sequence of 1-4 bytes.
- For documents in Latin script, UTF-8 is the same as ASCII.
- Default for XML.

UCS Transformation Formats: UTF-16

UTF-16 – 16 bit encoding

- Each character upto `FFFF` is encoded as a single 16-bit value.
- Characters above `FFFF` are represented as pairs of 16-bit values: high- and low-surrogates.
- Starts with a single character (Byte Order Mark):

  `FFFE` – most significant byte first

  `FEFF` – most significant byte second

Surrogates

`D800-DFFF` is a surrogate block:

- `D800-DBFF` low surrogate
- `DC00-DFFF` high surrogate

A pair of surrogate characters `(L,H)` represents a character:

```
(H - D800) * 400 + (L - DC00) + 10000
```

in the range `10000-10FFFF`.

Byte Order Mark

`BOM` – Byte Order Mark

The first character of a document:

- `FFFE` – the document is coded in UTF-16, big-endian
- `FEFF` – the document is coded in UTF-16, little-endian
- any other character – the document is coded in UTF-8

Example: UTF-16 and UTF-8

“Gulliver” in UTF-16 and UTF-8:

	G	u	l	l	i	v	e	r
FE FF	00 45	00 75	00 6C	00 6C	00 69	00 76	00 65	00 72
FF FE	45 00	75 00	6C 00	6C 00	69 00	76 00	65 00	72 00
	45	75	6C	6C	69	76	65	72

Demo: UTF-16 Encoding

```
> cd "demos/utf16 encoding"
> ls
doc.xml
> xvi32 doc.xml
> opera doc.xml
> iexplore doc.xml
```

Characters in XML 1.0

0001-0008	forbidden
0009	allowed – TAB
000A	allowed – NEW LINE
000B-000C	forbidden
000D	allowed – CARRIAGE RETURN
000E-001F	forbidden
0020	allowed – SPACE
0021-D7FF	allowed
D800-DBFF	forbidden – low surrogate
DC00-DFFF	forbidden – high surrogate
E000-FFFD	allowed
10000-10FFFF	allowed – encoded as pairs of surrogates

XML 1.1 permits representation of arbitrary Unicode characters.

XML and UCS

XML requires that:

- XML processors support the UTF-8 and UTF-16 encodings.

XML permits that:

- XML processors support alternative character sets/encodings.
- They are specified by the `encoding` attribute.

Alternative Encodings of UCS

iso-8859-1	Western Europe
iso-8859-2	Central Europe
iso-8859-3	Southern Europe
iso-8859-4	Northern Europe
iso-8859-5	Cyrillic
iso-8859-6	Arabic
iso-8859-7	Greek
iso-8859-8	Hebrew
iso-8859-9	Turkish
iso-8859-10	Nordic
...	...

big5	traditional Chinese
gb2312	simplified Chinese
euc-jp	Japanese (unix)
euc-kr	Korean (unix)
koi8-r	Russian
koi8-u	Ukrainian
tis-620	Thai
windows-*	Windows
cp-*	IBM
us-ascii	basic ASCII
...	...

XML

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

XML 1.0 – W3C Recommendation

- history:
  - first published in February 1998
  - revised October 2000
- editors:
  1. Tim Bray (Textuality and Netscape),
  2. Jean Paoli (Microsoft),
  3. C. M. Sperberg-McQueen (WorldWideWeb Consortium),
  4. Eve Maler (Sun Microsystems).

XML 1.0 – W3C Recommendation

- abstract:

  The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

- publication:

  `http://www.w3.org/TR/REC-xml`

XML 1.0 – W3C Recommendation

The core of the document is presentation of EBNF production rules to define the legal syntax of XML documents:

```
document ::= prolog element Misc*
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
XMLDecl ::= '<?xml' VerInfo EncodingDecl? SDDecl? S? '?>'
VerInfo ::= S 'version' Eq ("'" VerNum "'" | '"' VerNum '"')
VerNum ::= ([a-zA-Z0-9_.:] | '-')+
Eq ::= S? '=' S?
S ::= (#x20 | #x9 | #xD | #xA)+
```

There are also references to the behavior of XML processors:

> *Processors may signal an error if they receive documents labeled with versions they do not support.*

XML 1.1 – W3C Candidate Recommendation

Reasons for the new version:

- to keep up with the changing Unicode standard
- to add two more line-end characters
- to permit representation of arbitrary Unicode characters

XML 1.1 – W3C Candidate Recommendation

An excerpt from the 1.1 Candidate Recommendation:

> *Whereas XML 1.0 provided a rigid definition of names, wherein everything that was not permitted was forbidden, XML 1.1 names are designed so that everything that is not forbidden (for a specific reason) is permitted.*

Demo: XML W3C Recommendation

```
> opera http://www.w3.org/TR/REC-xml
> opera http://www.w3.org/TR/xml11/
```

XML Design Goals 1

Design goals for XML (XML 1.0 W3C Recommendation):

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs to process XML documents.

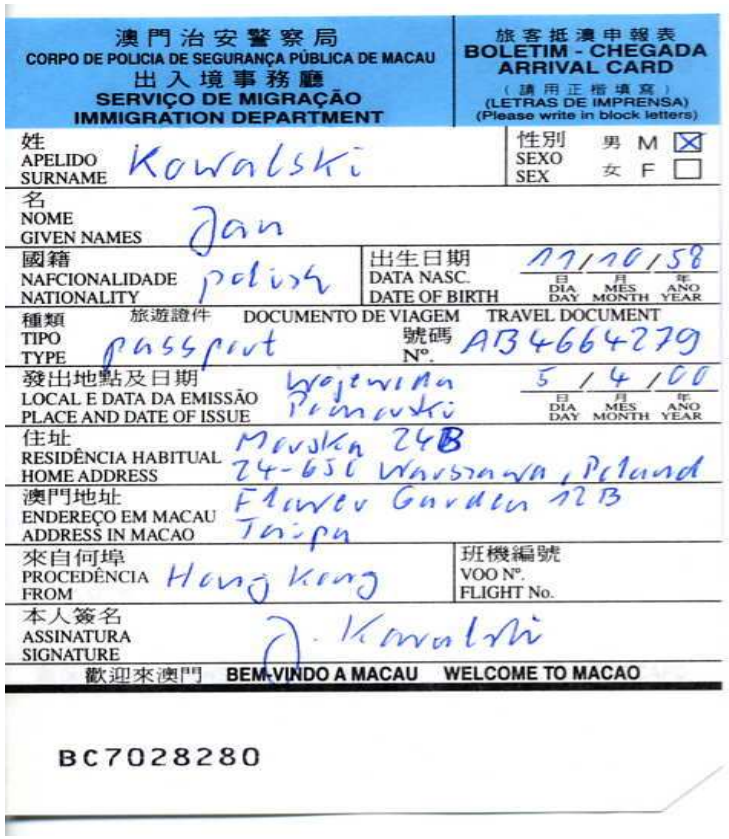
XML Design Goals 2

5. The number of optional features is to be kept to the minimum.
6. XML documents shall be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

Example: Macao Arrival Card

A case study for this section:

- Macao arrival card
- here filled by Jan Kowalski
- to be converted into XML



澳門治安警察局
CORPO DE POLICIA DE SEGURANÇA PÚBLICA DE MACAU
出入境事務廳
SERVIÇO DE MIGRAÇÃO
IMMIGRATION DEPARTMENT

旅客抵澳申報表
BOLETIM - CHEGADA
ARRIVAL CARD
(請用正楷填寫)
(LETRAS DE IMPRENSA)
(Please write in block letters)

姓 APELIDO SURNAME: Kowalski
性別 SEXO SEX: 男 M ☒ 女 F ☐
名 NOME GIVEN NAMES: Jan
國籍 NAFCIONALIDADE NATIONALITY: polish
出生日期 DATA NASC. DATE OF BIRTH: 11/10/58 (日 DIA DAY / 月 MÊS MONTH / 年 ANO YEAR)
種類 旅遊證件 TIPO TYPE: passport
DOCUMENTO DE VIAGEM TRAVEL DOCUMENT
號碼 Nº.: AB4664279
發出地點及日期 LOCAL E DATA DA EMISSÃO PLACE AND DATE OF ISSUE: Wojewoda Pomorski 5/4/00 (日 DIA DAY / 月 MÊS MONTH / 年 ANO YEAR)
住址 RESIDÊNCIA HABITUAL HOME ADDRESS: Morska 24B 24-651 Warszawa, Poland
澳門地址 ENDEREÇO EM MACAU ADDRESS IN MACAO: Flower Garden 12B Taipa
來自何埠 PROCEDÊNCIA FROM: Hong Kong
班機編號 VOO Nº. FLIGHT No.:
本人簽名 ASSINATURA SIGNATURE: J. Kowalski

歡迎來澳門 BEM-VINDO A MACAU WELCOME TO MACAO

BC7028280

Markup versus Character Data

XML document contains text that falls into two categories:

- markup – there are 12 different kinds of XML markup
- character data – parsed or unparsed

Example: Macao Arrival Card in XML

Character data:

- Kowalski,
- 24-630 Gdask, Poland

The rest is markup.

```
<?xml version="1.0"?>
<!DOCTYPE card SYSTEM "card.dtd">
<!-- arrival card for Jan Kowalski -->
<card type="arrival">
  <visitor>
    <name type="surname">
      Kowalski
    </name>
    ...
  </visitor>
  <address where="home">
    24-630 Gda&#x0144;sk, Poland
  </address>
  ...
  <signature sigfile="mysig"/>
</card>
```

XML Markup: 1-8

no	name	example
1	start tags	`<visitor>`
2	end tags	`</visitor>`
3	empty-element tags	`<signature/>`
4	entity references	`&copyright;`
5	character references	`ń`
6	comments	`<!-- whatever -->`
7	CDATA sections	`<![CDATA[ whatever ]]>`
8	document type declarations	`<!DOCTYPE ... >`

XML Markup: 9-12

no	name	example
9	processing instructions	`<?myApp ... ?>`
10	XML declarations	`<?xml version= ... ?>`
11	text declarations	`<?xml encoding= ... ?>`
12	white space at the top level	`<?xml version="1.0"?>` `<card>...</card>`

Unparsed Character Data

Character data falls into two kinds:

- unparsed (CDATA) – data that include entity/character references:
  - entity references – `&copyright;`
  - character references – `ń`

  For example:

```
Morska 24B, 24-630 Gda&#x0144;sk, Poland
```

Parsed Character Data

- parsed (PCDATA) – character data where entity references have been replaced by their definitions

  For example:

  `Morska 24B, 24-630 Gdańsk, Poland`

  `0144` (hexadecimal) is the Unicode value for the character ń.

Demo: Exploring Unicode Character Charts

```
> cd "../cdrom/unicode/Code Charts/"
> acroRd32 U0100.pdf
```

Document

A document is a sequence of:

- prolog (obligatory)
- element (obligatory)
- miscellenous markup (repetitive)

```
document ::= prolog element Misc*
```

Example: XML Document Structure

prolog ->

element ->

misc ->

```
<?xml version="1.0"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">
   <visitor>
    <name type="surname">
        Kowalski
    </name>
    ...
   </visitor>
   ...
</card>
<!-- end of document -->
```

Task: Prepare a Place on Your Hard Disk

```
> cd ``a convenient location''
> mkdir course course/tasks course/tasks/card
> cd course/tasks/card
```

Task: Write, Save and Open a Document

- Use your favorite text editor to create this document:

```
<?xml version="1.0"?>
<card type="arrival">
 <visitor>
     <name type="surname">your surname</name>
     <name type="given">your given names</name>
 </visitor>
</card>
```

- Save the file as card.xml
- Open the file with your favorite browser.

Task: Damage and Repair the Document

- Like this:

```
<xml version=2.0>
<card type=arrival>
 <visitor>
     <name type="surname">your surname
     <name type="given">your given names</name>
 </guest>
</card>
```

- Save and re-open.
- Follow error messages to repair the document.

Task: Modify Document Structure

- Add a comment:

```
<?xml version="1.0"?>
<card type="arrival">...</card>
<!-- the end -->
```

- Remove the declaration:

```
<card type="arrival">...</card>
<!-- the end -->
```

- Change the order:

```
<!-- the end -->
<card type="arrival">...</card>
```

Miscellaneous Markup

Miscellaneous markup is one of:

- a comment,
- a processing instruction
- or a whitespace

```
Misc ::= Comment | PI | S
```

Example: Miscellaneous Markup

comment ->

processing instruction ->

whitespace ->

```
<?xml version="1.0"?>
<!-- arrival card -->
<!DOCTYPE card SYSTEM "card.dtd">
<?myApplication date="12-03-2003"?>
  <card type="arrival">
  ...
</card>
```

Characters

A character is any Unicode character, excluding the surrogate blocks `D800-DBFF` and the characters `FFFE` and `FFFF`:

```
Char ::=     #x9 |
       #xA |
       #xD |
       [#x20-#xD7FF] |
       [#xE000-#xFFFD] |
       [#x10000-#x10FFFF]
```

Whitespace

A whitespace consists of one or more space characters, carriage returns, line feeds, or tabs:

```
S ::= (#x20 | #x9 | #xD | #xA)+
```

Characters are further divided into. . .

Characters: Letters and Digits

1. letters

```
Letter ::= BaseChar | Ideographic
BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] |...
Ideographic ::= [#x4E00-#x9FA5] | #x3007 | ...
```

2. digits

```
Digit ::= [#x0030-#x0039] | ...
```

Combining Characters

3. combining characters – characters that graphically combine with a preceding base character (e.g. diactrical marks)

```
CombiningChar ::= [#x0300-#x0345] | ...
```

Demo: Exploring Unicode Character Charts

```
> cd "../cdrom/unicode/Code Charts/"
> acroRd32 U0000.pdf
> acroRd32 U4E00.pdf
> acroRd32 U0300.pdf
```

XML Names

A Name begins with a letter, underscore or colon, and continues with letters, digits, hyphens, underscores, colons, or full stops:

```
NameChar ::=
    Letter | Digit |
    '.' | '-' | '_' | ':' |
    CombiningChar

Name ::= (Letter | '_' | ':') (NameChar)*

Nmtoken ::= (NameChar)+
```

Example: XML Names

wrong beginning ->	1class
	firstclass
illegal character ->	first&class
	first_class
reserved for namespaces ->	first:class
	first.class
reserved for XML ->	xml.class
->	Xml.class
->	XML.class
	class.xml
	class-Xml
	class_XML
strange but correct ->	_

Names beginning with `xml` in any combination of upper- and lower-case letters are reserved.

Task: Modify the Names

- Change all start tags into uppercase, re-open:

```
<?xml version="1.0"?>
<CARD type="arrival">
   ...
</card>
```

- Add the `my` prefix to card, re-open:

```
<?xml version="1.0"?>
<my:card type="arrival">
   ...
</my:card>
```

Comments

Reminder:

```
document ::= prolog element Misc*
Misc ::= Comment | PI | S
```

A comment is any sequence of characters surrounded by the literal strings `<!--` and `-->` that does not contain `--`:

```
Comment ::= '<!--'
((Char - '-') | ('-' (Char - '-')))* '-->'
```

Comments may appear anywhere outside markup. They are not part of the character data and need not be passed to applications.

Example: Comments

correct comment ->

incorrect comment ->

```
<?xml version="1.0"?>
<!-- arrival card for Jan Kowalski -->
<card type="arrival">
   ...
</card>
<!-- end of document --->
```

Task: Add Comments to Your Document

- Add comments to your document:

```
<?xml version="1.0"?>
HERE
<card type="arrival">
HERE
   <visitor>
     <name HERE type="surname">your surname</name>
     <name type="given">your given names</name>
   </visitor>
</card>
HERE
```

- Save, reopen and correct.

Processing Instruction

Reminder:

```
document ::= prolog element Misc*
Misc ::= Comment | PI | S
```

Processing instructions contain instructions for applications.

Processing Instruction

PI consists of a target – the application to which this instruction is directed – and the data to be passed, all inside `<?` and `?>`:

```
PI ::=
  '<?' PITarget
  (S (Char* - (Char* '?>' Char*)))? '?>'
PITarget ::=
  Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but must be passed through to the application.

XML Notation may be used for formal declaration of PI targets.

Example: Processing Instruction

processing instruction ->
processing instruction ->

```
<?xml version="1.0"?>
<?oneApp date="12-03-2003"?>
<?anotherApp whatever?>
<card type="arrival">
   ...
</card>
```

Task: Add Processing Instructions

Add two processing instructions to your document:

- to the mail application

```
<?mail message="coming back" date="30.09.2003"?>
```

- to the stylesheet processor

```
<?xml-stylesheet type="text/css"?>
```

Save and reopen in both cases. What happens?

Prolog

Reminder:

```
document ::= prolog element Misc*
```

A prolog contains optional XML declaration, `Misc*`, and optional document type declaration followed again by `Misc*`:

```
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

Example: Prolog

XML declaration ->	`<?xml version="1.0"?>`
document type declaration ->	`<!DOCTYPE card SYSTEM "card.dtd">`
processing instruction ->	`<?MyApp whatever ?>`
comment ->	`<!-- silly comment -->`
	`<element>`
	`...`
	`</element>`

XML Declaration

Indicates that the document is written in XML.

Its syntax resembles a processing instruction for the “xml” target:

```
XMLDecl ::=
  '<?xml' VersionInfo EncodingDecl?
  SDDecl? S? '?>'
```

If present, it must occur at the very beginning of a document.

Three attributes are allowed.

XML Version Declaration

1. The version of XML being used:

```
VersionInfo ::=
   S 'version' Eq
   ("'" VersionNum "'" | '"' VersionNum '"')
Eq ::= S? '=' S?
VersionNum ::= ([a-zA-Z0-9_.:] | '-')+
```

So far, the only allowed value is 1.0. In future, this feature should permit automatic version recognition.

XML Standalone Declaration

2. Is it possible to parse the document alone? Are there any external declarations that have to be consulted?

```
SDDecl ::=
   S 'standalone' Eq
   (("'" ('yes' | 'no') "'") |
   ('"' ('yes' | 'no') '"'))
```

Two possibilities:

- `yes` – there are no external markup declarations
- `no` – there are or may be such external declarations

The default is `no`.

XML Encoding Declaration

3. Character encoding used in the document:

```
EncodingDecl ::=
   S 'encoding' Eq
   ('"' EncName '"' | "'" EncName "'" )
EncName ::=
   [A-Za-z] ([A-Za-z0-9._] | '-')*
```

- XML processors have to support UTF-8 and UTF-16 only.
- Encoding name should be registered with IANA, or otherwise it should start with `x-`.
- If a document does not start with BOM, nor an encoding declaration, then it must use UTF-8.

Example: XML Declaration

minimum declaration ->	`<?xml version="1.0"?>`
predict new version ->	`<?xml version="1.1"?>`
incorrect declaration ->	`<?xml standalone="yes"?>`
standalone document ->	`<?xml version="1.0" standalone="yes"?>`
unnecessary ->	`<?xml version="1.0" encoding="UTF-8"?>`
east european encoding ->	`<?xml version="1.0" encoding="iso-8859-2"?>`
standalone, EE encoding ->	`<?xml version="1.0" encoding="iso-8859-2" standalone="yes"?>`

Task: Version and Encoding Declarations

1. change version into `1.1`:

```
<?xml version="1.1"?>
```

2. remove version attribute:

```
<?xml?>
```

3. add `UTF-8` encoding:

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

4. change to `UTF-16` encoding

Task: Standalone Declaration

1. add to `card.xml`:
   - declaration for entity `text`
   - entity reference `&text;`
   - declaration `standalone="yes"`

```
<?xml version="1.0" standalone="yes">
<!DOCTYPE card [<!ENTITY text "silly text">]>
<card type="arrival">
...
&text;
</card>
```

2. re-open the file

Task: Play with Standalone Declaration

3. create the external file `card.dtd`:

```
<!ENTITY text "silly text">
```

4. modify `card.xml` to refer to this file:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">
...
&text;
</card>
```

5. re-open and repair `card.xml`

Document Type Declaration

Reminder:

```
document ::= prolog element Misc*
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

Document type declaration:

- names the document's root element
- contains or points (or both) to markup declarations that provide a grammar for a class of documents

Document Type Declaration

Declaration rule:

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

- `Name` is the document's root element
- `ExternalId` points to the document's external DTD
- `[...]` contains the document's internal DTD

Document Type Declaration versus Definition

Document Type:

- declaration – markup `doctypedecl` that links/contains the document's type definition

- definition – document's class description rules (DTD): inside `markupdecl` or pointed by `ExternalID`

DTD = Document Type Definition

DTD is the native grammar description language used by XML.
It is described in detail in the next section.

External Document Type Declaration

External DTD is pointed by `ExternalId`, which starts with the keyword `SYSTEM` or `PUBLIC`:

```
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
```

SYSTEM Document Type Declaration

XML does not constrain the syntax much

```
SystemLiteral ::=
  ('"' [^"]* '"') | ("'" [^']* "'")
```

It is usually a file name, path, or URL.

Example: SYSTEM Declaration

```
<!DOCTYPE card SYSTEM "card.dtd">
<!DOCTYPE card SYSTEM "/usr/local/dtds/card.dtd">
<!DOCTYPE card SYSTEM "http://www.w3c.org/card.dtd">
```

PUBLIC Document Type Declaration

`PubidLiteral` is often called the Formal Public Identifier.

```
PubidLiteral ::=
  '"' PubidChar* '"' |
  "'" (PubidChar - "'")* "'"
PubidChar ::= #x20 | #xD | #xA |
  [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

FPI provides a formalized description of the content of the DTD, independent from its physical location.

PUBLIC Document Type Declaration

Here is the typical syntax:

```
<!DOCTYPE card PUBLIC
"-//UNUIIST//Immigration Card Language 1.0//EN"
"http://www.iist.unu.edu/dtds/card.dtd">
```

where:

- `UNUIIST`: the institution that maintains this DTD
- `-`: the institution is not a standards body (otherwise +)
- `Immigration Card Language 1.0`: short description
- `EN`: the language used

Example: XHTML DOCTYPE

DOCTYPE for an XHTML document:

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

Example: SVG DOCTYPE

DOCTYPE for an SVG document:

```
<!DOCTYPE svg
   PUBLIC
   "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-
   20010904/DTD/svg10.dtd">
```

Example: MathML DOCTYPE

DOCTYPE for a MathML document:

```
<!DOCTYPE math
   PUBLIC
   "-//W3C//DTD MathML 2.0//EN"
   "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
```

Task: External DTD

1. add another line to `card.dtd`:

```
<!ENTITY text "silly text">
<!ENTITY moreText "more silly text">
```

2. include both entity references:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">
... &text; &moreText;
</card>
```

3. re-open

Task: Internal DTD

4. include both declarations internally:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card [
   <!ENTITY text "silly text">
   <!ENTITY moreText "more silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

5. re-open, change standalone, re-open

Task: Internal and External DTD

6. make one declaration internal, another external:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM "card.dtd" [
   <!ENTITY text "silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

7. re-open

Task: Overlapping Internal and External DTD

8. modify `card.xml`:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM "card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

9. re-open

10. Which declaration takes precedence: internal or external?

Task: Local External DTD

11. copy `card.dtd` to your temp directory:

```
> copy card.dtd C:/temp
```

12. modify `card.xml`, re-open:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM "C:/temp/card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

Task: Remote External DTD, SYSTEM

13. copy `card.dtd` to remote server

```
> ftp ftp.iist.unu.edu
> put card.dtd
```

14. modify `card.xml`, re-open:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   SYSTEM
   "http:://www.iist.unu.edu/~tj/course/card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

Task: Remote External DTD, PUBLIC

15. modify `card.xml`, re-open:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE card
   PUBLIC
   "-//Small Steps Ltd.//Silly DTD Version 1.0//EN"
   "http:://www.iist.unu.edu/~tj/course/card.dtd" [
   <!ENTITY text "yet another silly text">
]>
<card type="arrival">
   ...
   &text;
   &moreText;
</card>
```

Elements

Reminder:

```
document ::= prolog element Misc*
```

Element is an empty element tag or a non-empty element:

```
element ::= EmptyElemTag | STag content ETag
```

A non-empty element consists of content surrounded by the opening and closing tags, with matching names.

Start and End Tags

There are three kinds of tags:

1. start tag – a name and a list of attribute declarations separated by whitespaces, all inside `<` and `>`:

```
STag ::= '<' Name (S Attribute)* S? '>'
```

2. end tag – a name inside `</` and `>`:

```
ETag ::= '</' Name S? '>'
```

Empty Element Tag

3. empty element tag – a name and a list of attribute declarations separated by whitespaces, all inside `<` and `/>`:

```
EmptyElemTag ::=
'<' Name (S Attribute)* S? '/>'
```

Example: Tags

start tag ->

end tag ->

empty element tag ->

```
<card type="arrival">
   <address where="home">
    24-630 Gda&#x0144;sk, Poland
   </address>
   ...
   <signature sigfile="mysig"/>
   ...
</card>
```

Attributes

Attribute declarations are used in start- and empty-element tags.

A declaration consists of:

- a name,
- the equal sign and
- a value

```
Attribute ::= Name Eq AttValue
```

Attribute Values

The value is any sequence of characters and references, except the characters used in markup (<, &, and " or '):

```
AttValue ::=
  '"' ([^<&"] | Reference)* '"' |
  "'" ([^<&'] | Reference)* "'"
```

It is surrounded by:

- double quotes (allowing single quotes inside) or
- single quotes (allowing double quotes inside)

Example: Attributes

attribute in start tag ->

attribute with references ->

attribute in empty element tag ->

```
<card type="arrival">
   <address
   where="home"
   city="Gda&#x0144;sk">
    ...
   </address>
   ...
   <signature sigfile="mysig"/>
</card>
```

Task: Arrival Card: Return to Simple Form

1. return `card.xml` to the simple form:

```
<?xml version="1.0"?>
<card type="arrival">
    <visitor>
        <name type="surname">your surname</name>
        <name type="given">your given names</name>
    </visitor>
</card>
```

Task: Add Visitor's Elements

2. add the elements to describe the visitor's: `sex` (empty element) and `nationality` (element with text)

```
<?xml version="1.0"?>
<card type="arrival">
    <visitor>
        <name type="surname">your surname</name>
        <name type="given">your given names</name>
        <sex type="..."/>
        <nationality>...</nationality>
    </visitor>
</card>
```

Task: Add More Visitor's Elements

3. add the `date` of birth (text)

```
<date>...</date>
```

4. expand `date` into `day`, `month` and `year` elements:

```
<date type="birth">
    <day>...</day>
    <month>...</month>
    <year>...</year>
</date>
```

Task: Add Travel Document Information

5. add travel document information:

   - document type
   - document number
   - issuing authority
   - date of issue

```
<document>
    <type>...</type>
    <number>...</number>
    <authority>...</authority>
    <date>...</date>
</document>
```

Task: Modify Travel Document Information

6. make document `type` an attribute:

```
<document type="...">
    <number>...</number>
    <authority>...</authority>
    <date>...</date>
</document>
```

7. make document `type` and `number` into attributes:

```
<document type="..." number="...">
    <authority>...</authority>
    <date>...</date>
</document>
```

Task: Modify Document Issue Date

8. differentiate dates of birth and document issue:

```
<card type="arrival">
    ...
    <visitor>
        <date type="birth">...</date>
    </visitor>
    <document type="..." number="...">
        <authority>...</authority>
        <date type="issue">...</date>
    </document>
    ...
</card>
```

Task: Add Visitor's Addresses

9. add the visitor's home and Macao address:

```
<card type="arrival">
    ...
    <address>...</address>
    <address>...</address>
</card>
```

10. differentiate between the addresses using the attribute `where`:

```
<card type="arrival">
    ...
    <address where="home">...</address>
    <address where="Macao">...</address>
</card>
```

Task: Expand Visitor's Address Information

11. expand address information into: `street`, `city`, `zip`, `state` and `country`.

```
<address where="...">
    <street>...</street>
    <city>...</city>
    <zip>...</zip>
    <state>...</state>
    <country>...</country>
</address>
```

Task: Expand Visitor's Street Address

12. expand `street` address into: `street`, `house`, `floor` (if any) and `flat` (if any).

```
<street>
    <street>...</street>
    <house>...</house>
    <floor>...</floor>
    <flat>...</flat>
</street>
```

Task: Add Travel Information

13. add `travel` information: origin (`place`), `mode` and `flight` (if any).

```
<card type="arrival">
    ...
    <travel>
        <place>...</place>
        <mode>...</mode>
        <flight>...</flight>
    </travel>
</card>
```

Task: Add Digital Signature

14. add the traveller's digital `signature`: empty element referring to the `my.sig` file.

```
<card type="arrival">
   ...
   <signature sigfile="my.sig"/>
</card>
```

References

A reference is one of:

- character reference or
- entity reference

```
Reference ::= CharRef | EntityRef
```

Character References

A character reference refers to a specific character in UCS.

```
CharRef ::=
  '&#' [0-9]+ ';' |
  '&#x' [0-9a-fA-F]+ ';'
```

Two forms:

- `&#` – decimal representation of the character's code
- `&#x` – hexadecimal representation of the character's code

Entity References

An entity reference refers to the content of a named entity:

```
EntityRef ::= '&' Name ';'
```

Such a name must be declared in a DTD.

Pre-Defined Entity References

There are five pre-defined entity names:

name	value
amp	&
apos	'
quot	"
gt	>
lt	<

They are used to include markup characters literally.

Example: References

pre-defined entity reference ->

character references:
decimal ->

hexadecimal ->

user-defined entity reference ->

```
<?xml version="1.0"?>
<hotel type="bed&amp;breakfast">
    ...
    <address>
        <city name="Gda&#324;sk"/>
        <street
            name="Okr&#x0119;&#x017D;na"/>
        ...
    </address>
    <note name="Read &legalstuff;"/>
</hotel>
```

Task: References for Common Places

1. define and refer to the entities for Hong Kong, Zhuhai and Macao:

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY hk "Hong Kong">
   <!ENTITY zh "Zhuhai">
   <!ENTITY mo "Macao">
]>
<card type="arrive">
   ...
   &hk; ...
   &mo; ...
   &zh; ...
</card>
```

Task: Define References for Two Sexes

2. define and refer to the entities representing two sexes:

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY male "<sex type="male"/>">
   <!ENTITY female "<sex type="female"/>">
]>
<card type="arrive">
   ... &female; ...
</card>
```

3. what is wrong? correct

Task: References for Common Origins

4. Define the entities to capture two common travel origins for visitors to Macao:

- from Zhuhai (on foot) or
- from Hong Kong (by ferry)

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY fromZhuhai "...">
   <!ENTITY fromHongKong "...">
]>
<card type="arrive">
   ... &fromHongKong; ...
</card>
```

Task: Reuse of Entities

5. Reuse the entities for common places when defining entities for common origins:

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY fromZhuhai "... &zh; ...">
   <!ENTITY fromHongKong "... &hk; ...">
]>
<card type="arrive">
   ...
</card>
```

6. Any other ideas for reuse in `card.xml`?

Content

Reminder:

```
document ::= prolog element Misc*
element ::= EmptyElemTag | STag content ETag
```

The text between start-tag and end-tag is called element content:

```
content ::=
   CharData?
   (  (element | Reference | CDSect | PI | Comment)
      CharData?
   )*
```

Types of Content

Element content may contain any number of:

- character data `CharData`
- elements `element`
- references `Reference`
- CDATA sections `CDSect`
- processing instructions `PI`
- comments `Comment`

Which are undefined yet? `CharData` and `CDSect`

CDATA Section

CDATA section is used to escape blocks of text containing characters which would otherwise be recognized as markup:

```
CDSect ::= CDStart CData CDEnd
CDStart ::= '<![CDATA['
CData ::= (Char* - (Char* ']]>' Char*))
CDEnd ::= ']]>'
```

Within CDATA, only `CDEnd` is recognized as markup.

CDATA cannot nest.

Example: CDATA Section

escaping markup without CDATA ->

```
<lecture about="XML">
   Here is a start-tag:
   &lt;message
   to=&quot;tj@iist.unu.edu&quot;&gt;
   and an end-tag: &lt;note/&gt;
</lecture>
```

escaping markup with CDATA ->

```
<lecture about="XML">
  <![CDATA[Here is a start-tag:
  <message to="tj@iist.unu.edu">
  and an end-tag: <note/>]]>
</lecture>
```

Character Data

All text that is not markup constitutes the character data of the document. More precisely:

```
CharData ::=
  [^<&]* - ([^<&]* ']]>' [^<&]*)
```

Character data is any string of characters which does not contain:

- the start-delimiter of any markup, and
- the CDATA-section-close delimiter `]]>`

Element Syntax Overview

```
document        ::= prolog element Misc*
element         ::= EmptyElemTag | STag content ETag
STag            ::= '<' Name (S Attribute)* S? '>'
Attribute       ::= Name Eq AttValue
AttValue        ::= '"' ([^<&"] | Ref)* '"' | ...
Ref             ::= EntityRef | CharRef
EntityRef       ::= '&' Name ';'
CharRef         ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'
ETag            ::= '</' Name S? '>'
EmptyElemTag    ::= '<' Name (S Attribute)* S? '/>'
Content         ::= CharData? ((element|Ref|CDSect|PI|Comment)...)*
CharData        ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
CDSect          ::= '<![CDATA[' CData ']]>'
CData           ::= (Char* - (Char* ']]>' Char*))
PI              ::= '<?' PITarget (S (Char*-(Char* '?>' Char*)))? '?>'
Comment         ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-- >'
```

Well-Formed XML Document

This concludes the explanation of the rules that define XML syntax.

What is a well-formed XML document?

1. Taken as a whole, it matches the rule:

   ```
   document ::= prolog element Misc*
   ```

2. It meets all well-formedness constraints (?)

3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed (?)

Entity Declarations

Entity declarations may:

- contain text as well as markup
- be parsed (XML content) or unparsed (non-XML)
- be defined within a document (internal) or outside (external)

Example: Parsed Entities

entity declaration ->

contains markup ->

entity reference ->

entity declaration ->

entity reference ->

```
<!DOCTYPE hotel SYSTEM [
  <!ENTITY legal "We reject any responsibility
   for damages due to errors in this
   <product type='software'/>.
   &copyright;">
  <!ENTITY copyright "&#x00A9;">
]>
<hotel type="bed&amp;breakfast">
  <note name="Read &legal;"/> ...
</hotel>
```

Well-Formed Parsed Entities

All parsed entities referred directly or indirectly from the body of the document, must be well-formed:

- internal parsed entity:

```
intParsedEnt ::= content
```

- external parsed entity:

```
extParsedEnt ::= TextDecl? content
```

Text Declaration

An external parsed entity may contain a text declaration:

```
TextDecl ::=
  '<?xml' VersionInfo? EncodingDecl S? '?>'
```

The text declaration allows for several different encoding methods to be used on the same document.

Note the differences with the XML declaration:

- `version` is optional,
- `encoding` is required,
- `standalone` is forbidden.

Task: Non-Well-Formed Entities

1. Modify `card.xml` so that some entities are not well-formed.

```
<?xml version="1.0"?>
<!DOCTYPE card [
   <!ENTITY male "type='male'/>">
   <!ENTITY female "type='female'/>">
]>
<card type="arrival">
   <visitor>...</visitor>
   <document>...</document>
   ... <sex &male; ...
</card>
```

2. Reopen, and repair.

Well-Formedness Constraints

The third component to well-formedness is satisfaction of the side-conditions on BNF production rules:

1. The name in the start-tag must match the name in the end-tag:

```
element ::= EmptyElemTag | STag content ETag
STag ::= '<' Name (S Attribute)* S? '>'
ETag ::= '</' Name S? '>'
```

Well-Formedness Constraints

2. No attribute name may appear more than once in the same start-tag or empty-element tag:

```
STag ::= '<' Name (S Attribute)* S? '>'
EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'
Attribute ::= Name Eq AttValue
```

3. Attribute values cannot contain direct or indirect entity references to external entities:

```
AttValue ::=
   '"' ([^<&"] | Reference)* '"' |
   "'" ([^<&'] | Reference)* "'"
```

Well-Formedness Constraints

4. The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a `<`.

```
AttValue ::=
   '"' ([^<&"] | Reference)* '"' |
   "'" ([^<&'] | Reference)* "'"
```

5. The name given in the entity reference is one of `amp`, `lt`, `gt`, `apos` or `quot`, or it must be declared in the document's DTD.

```
EntityRef ::= '&' Name ';'
```

Well-Formedness Constraints

6. An entity must not contain a recursive reference to itself, either directly or indirectly.

7. An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to in attribute values.

This concludes the definition of XML Syntax.

Certain aspects of well-formedness will be re-investigated after the introduction to Document Type Definitions in the next section.

Task: Violation of Well-Formedness

For each of the following well-formedness conditions modify `card.xml` to break this condition, watch the error message, and repair:

1. matching names
2. no repeated attributes
3. no `<` in entities referenced in attribute values
4. entity names must be declared or pre-defined
5. no recursive references in entities

Example: Macau Arrival Card

```
                                  <?xml version="1.0" standalone="yes"?>
                                  <card type="arrival">
                                  <!-- visitor information -->
                                  <visitor>
                                     <name type="surname">Kowalski</name>
less elements? ->                    <name type="given">Jan</name>
                <name>               <sex type="male"/>
          Jan Kowalski               <nationality>polish</nationality>
                </name>              <date type="birth">
less attributes? ->                   11 October 1958
                <date>               </date>
      11 October 1958             </visitor>
                </date>
```

Example: Macau Arrival Card

```
                                    <!-- travel document information -->
more attributes? ->                 <document type="passport">
                 <document             <number>AB4664279</number>
     type="passport"                    <authority>
number="AB4664279">                        Wojewoda Pomorski
                   ...                  </authority>
                 </document>            <date type="issue">
                                        5 April 2000
                                       </date>
                                    </document>
```

Example: Macau Arrival Card

```
                    more elements? ->
<address where="home">
  <street>Morska 24B</street>
  <city>Gda&#x0144;sk</city>
  <zip>24-650</zip>
  <country>Poland</country>
</address>

                    more entities? ->
&fromHongKong;
<!ENTITY fromHongKong
  "<travel type="from">
  <place>Hong Kong</place>
  </travel>„
>
```

```
<!-- address information -->
<address where="home">
  Morska 24B,
  24-650 Gda&#x0144;sk, Poland
</address>
<address where="Macao">
  Flower Garden 12B, Taipa
</address>

<travel type="from">
<place>Hong Kong</place>
</travel>

<signature sigfile="mysig"/>
</card>
```

Demo: Macao Arrival Card

```
> cd "demos/macao arrival card"
> dir
card.xml
> java dom.Counter card.xml
> cp card.xml card.xml.old
> emacs card.xml
> java dom.Counter card.xml
> cp card.xml.old card.xml
```

DTD

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

Well-Formed or Toast

All XML documents must be well-formed:

- an ill-formed document cannot be even called XML
- XML parsers are required:
  - not to process non-well-formed XML documents
  - to report the problem to the calling application, and exit
- As a result, XML parsers are lightweight, XML processing is predictable and consistent between different applications.

Well-Formedness versus Validity

A well-formed XML document may in addition be valid.

- An XML document is valid if:

  1. it has an associated Document Type Definition
  2. it complies with the constraints expressed in it

- Unlike for SGML, DTDs are optional for XML, so is validity.

In this section we explain the syntax of Document Type Definitions, and the notion of validity associated with them.

Demo: Validity of Macao Arrival Card

```
> cd "demos/validity of macao arrival card"
> ls
card.dtd cardNoDTD.xml cardYesDTD.xml
> java dom.Counter cardNoDTD.xml
> java dom.Counter -v cardYesDTD.xml
```

DTD Scope

With DTDs we can specify:

1. elements and their attributes
2. nesting and order of elements
3. whether elements can contain text
4. whether text and elements can be mixed
5. whether elements are optional/required
6. whether elements can be repeated
7. default or fixed values for attributes
8. limited datatypes for attributes
9. reusable sections of text called entities
10. non-XML content via notations
11. etc.

Document Type Declaration

Reminder:

```
document ::= prolog element Misc*
prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
```

Document type declaration:

- names the document's root element (`Name`)
- contains (inside [...]) or points (`ExternalID`) or both to the document's type definition.

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

External and Internal DTD

DTD consists of two subsets:

1. external – have been explained before

```
ExternalID ::=
   'SYSTEM' S SystemLiteral |
   'PUBLIC' S PubidLiteral S SystemLiteral
```

2. internal – a sequence of markup declarations (`markupdecl`) separated by declaration separators (`DeclSep`):

```
doctypedecl ::= ...
   ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

Example: DTD with External Subset

standalone=no ->
external subset ->
root ->

```
<?xml version="1.0"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">...</card>
```

Example: DTD with Internal Subset

standalone=yes ->
internal subset ->
root ->

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE card [
<!ELEMENT card (visitor, document, ... )>
<!ATTLIST card type (arrival | departure) ...>
<!ELEMENT visitor (name, name, sex, ... )>
<!ELEMENT name (#PCDATA)>
   ...
]>
<card type="arrival">...</card>
```

Example: DTD with External/Internal Subsets

standalone=no ->
external subset ->
internal subset ->

```
<?xml version="1.0"?>
<!DOCTYPE card SYSTEM “card.dtd” [
<!ELEMENT card (visitor, document, ... )>
<!ATTLIST card type (arrival | departure)...>
<!ELEMENT visitor (name, name, sex, ... )>
<!ELEMENT name (#PCDATA)>
   ...
]>
```

root ->

```
<card type="arrival">
   ...
</card>
```

Demo: Document Type Declarations

```
> cd "demos/document type declarations"
> ls
cardExternal.xml cardInternal.xml cardMixed.xml
cardBig.dtd cardSmall.dtd cardMiddle.dtd
> emacs cardInternal.xml
> java dom.Counter -v cardInternal.xml
> emacs cardExternal.xml cardBig.dtd
> java dom.Counter -v cardInternal.xml
> emacs cardMixed.xml cardSmall.dtd
> java dom.Counter -v cardMixed.xml
```

Task: Make A Letter Document

- create a new directory

```
> mkdir course/tasks/letter
> cd course/tasks/letter
```

- create the smallest document with the letter root

```
<letter></letter>
```

- save it as `letter.xml`, parse

Task: Expand the Letter

- add the XML declaration, parse

```
<?xml version="1.0"?>
<letter></letter>
```

- add the DOCTYPE, parse

```
<?xml version="1.0"?>
<!DOCTYPE>
<letter></letter>
```

Task: Add the Document Type

- add the root element, parse

```
<?xml version="1.0"?>
<!DOCTYPE letter>
<letter></letter>
```

- add the empty internal subset, parse

```
<?xml version="1.0"?>
<!DOCTYPE letter []>
<letter></letter>
```

- validate, what happens?

Markup Declarations

Markup declarations include: element, attribute list, entity and notation. Processing instructions and comments are also allowed.

```
markupdecl ::=
    elementdecl |
    AttlistDecl |
    EntityDecl |
    NotationDecl |
    PI |
    Comment
```

Format of Markup Declarations

All markup declarations conform to similar syntax:

```
<!TYPE ...>
```

where `TYPE` is one of:

- `ELEMENT`
- `ATTLIST`
- `ENTITY`
- `NOTATION`

Element Type Declaration

Element type declaration contains the element's name and the specification of its content:

```
elementdecl ::=
  '<!ELEMENT' S Name S contentspec S? '>'
```

Four kinds of content specifications:

```
contentspec ::=
  'EMPTY' |
  'ANY' |
  Mixed |
  children
```

Empty Element Content

`EMPTY` means an element can neither contain:

- character data
- nor children elements

But it may contain attributes.

Any Element Content

`ANY` puts no constraints. The element may contain:

- elements,
- character data,
- a mixture of both, or none.

Useful during initial stages in the design of a DTD.

Example: Empty and Any Element Content

empty element declaration ->

```
<sex type="male"/>
```

```
<!ELEMENT sex EMPTY>
```

any element declaration ->

```
<test>
   <test1>...</test1>
   <test2>...</test2>
   <test3>...</test3>
   <test4>...</test4>
   <test5>...</test5>
</test>
```

```
<!ELEMENT test ANY>
<!ELEMENT test1 ...>
<!ELEMENT test2 ...>
<!ELEMENT test3 ...>
<!ELEMENT test4 ...>
<!ELEMENT test5 ...>
```

Task: Declare Letter as EMPTY

- declare `letter` as `EMPTY`, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter EMPTY>]>
<letter></letter>
```

- add some text, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter EMPTY>]>
<letter>this is a letter text</letter>
```

Task: Declare Letter as ANY

- declare `letter` as `ANY`, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter ANY>]>
<letter>this is a letter text</letter>
```

Mixed Element Content

The mixed model: element contains character data optionally interspersed with child elements:

```
Mixed ::=
  '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' |
  '(' S? '#PCDATA' S? ')'
```

The types of the child elements may be constrained, but not their order or their number of occurrences.

Example: Mixed Element Content

```
            text-only ->  <!ELEMENT name (#PCDATA)>
<name>Kowalski</name>

 text mixed with one ->  <!ELEMENT address (#PCDATA | zip)*>
<address>
  <zip>CV4 7AL</zip>
   Coventry, UK
</address>

 text mixed with two ->  <!ELEMENT address (#PCDATA|zip|country)*>
<address>
  <zip>CV4 7AL</zip>
   Coventry,
  <country>UK</country>
</address>
```

Task: Declare Letter as PCDATA

- declare letter as `PCDATA`, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter (#PCDATA)>]>
<letter>this is a letter text</letter>
```

Task: Expand The Letter Text

- expand the letter text, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter (#PCDATA)>]>
<letter>
  Dear Simon White,
  We are pleased to inform you that your
  credit card application has been approved.
  Sincerely, Steven Rod
</letter>
```

Task: Add Letter Markup

- add some letter markup, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [<!ELEMENT letter (#PCDATA)>]>
<letter>
  Dear <customer>Simon White</customer>,
  We are pleased to inform you that your
  <product>credit card</product> application
  has been approved. Sincerely, Steven Rod
</letter>
```

Task: Declare Markup

- declare markup, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
   <!ELEMENT letter (customer | product | #PCDATA)*>
   <!ELEMENT customer (#PCDATA)>
   <!ELEMENT product (#PCDATA)>
]>
<letter>...</letter>
```

Task: Correct Markup

- correct, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter (#PCDATA | customer | product)*>
  <!ELEMENT customer (#PCDATA)>
  <!ELEMENT product (#PCDATA)>
]>
<letter>...</letter>
```

Task: Modify Markup

- modify and validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
   <!ELEMENT customer (#PCDATA)>
   <!ELEMENT product (#PCDATA)>
   <!ELEMENT letter (#PCDATA | product | customer)*>
]>
<letter>...</letter>
```

Task: Add More Markup

- add more markup, declare, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [...]>
<letter>
  Dear <customer>Simon White</customer>,
  We are pleased to inform you that your
  <product>credit card</product> application
  has been approved. Sincerely,
  <manager>Steven Rod</manager>
</letter>
```

Element Content

An element must contain only child elements (no character data), optionally separated by white space.

Build with `choice` or `seq` and optional occurrence indicators:

```
children ::=
  (choice | seq) ('?' | '*' | '+')?
```

Occurrence Indicators

E	exactly one E
E?	zero or one E
E*	zero or more E
E+	one or more E

Choice Content

`choice` is a list of content particles `cp` separated by "|".

The list contains at least two particles, all in brackets:

```
choice ::=
  '(' S? cp ( S? '|' S? cp )+ S? ')'
```

Sequence Content

`seq` is a list of content particles `cp` separated by ",".

The list contains at least one particle, all in brackets:

```
seq ::=
  '(' S? cp ( S? ',' S? cp )* S? ')'
```

Content Particle

Content particle `cp` is one of:

- element name,
- `choice` or
- `seq`

with an optional occurrence indicator:

```
cp ::=
(Name | choice | seq) ('?' | '*' | '+')?
```

Example: Element and Mixed Content 1

(a)	a
(a?)	0,a
(a+)	a,aa,...
(a*)	0,a,aa,...
(a*)*	0,a,aa,...
(a+)?	0,a,aa,...
(a,b)	ab
(a,b)?	0,ab
(a,b?)	a,ab
(a?,b)	b,ab
(a?,b?)	0,a,b,ab
(a?,b?)?	0,a,b,ab
(a,b)*	0,ab,abab,...
(a,b*)	a,ab,abb,...
(a*,b)	b,ab,aab,...

Example: Element and Mixed Content 2

(a*,b*)	0,a,b,ab,aab,abb,...
(a*,b*)*	0,a,aa,b,bb,ab,abab,...
(a|b)	a,b
(a|b)?	0,a,b
(a|b?)	0,a,b
(a?|b?)	0,a,b
(a|b)+	a,b,aa,ab,ba,bb,...
(a?|b)+	a,b,aa,ab,ba,bb,...
(a+|b)+	a,b,aa,ab,ba,bb,...
((a,b)|c)	ab,c
(a|(b,c))	a,bc
(a,(b|c))	ab,ac
((a|b),c)	ac,bc
(#PCDATA|a)*	0,t,a,ta,at,tt,aa,...
(#PCDATA|a|b)*	0,t,a,b,ta,tb,at,bt,tt,ab,...

Demo: Element Content Models

```
> cd "demos/element content models"
> ls
test.xml
> java dom.Counter -v test.xml
```

Task: Divide the Letter

- divide the letter, declare, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
   <!ELEMENT letter (opening, body, closure)>
   <!ELEMENT opening (#PCDATA | customer)*>
   <!ELEMENT body (#PCDATA | product)*>
   <!ELEMENT closure (#PCDATA | manager)*> ...
]>
<letter>
   <opening>...</opening>
   <body>...</body>
   <closure>...</closure>
</letter>
```

Task: Make the Opening Optional

- make the opening optional, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
   <!ELEMENT letter (opening?, body, closure)>
   <!ELEMENT opening (#PCDATA | customer)*>
...
]>
<letter>
   <body>...</body>
   <closure>...</closure>
</letter>
```

Task: Add Enclosures

- add enclosures, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
   <!ELEMENT letter (opening?, body, closure, enclosure*)>
   <!ELEMENT enclosure (#PCDATA)>...
]>
<letter>
   ...
   <enclosure>...</enclosure>
   <enclosure>...</enclosure>
</letter>
```

Task: Add a Carbon Copy List

- add a carbon copy list, validate

```
<?xml version="1.0"?>
<!DOCTYPE letter [
   <!ELEMENT letter (..., cc*, enclosure*)>
   <!ELEMENT enclosure (#PCDATA)>...
]>
<letter>
   ...
   <cc>...</cc>
   <cc>...</cc>
   <cc>...</cc>
   <enclosure>...</enclosure>
</letter>
```

Next Declaration Type

Reminder:

```
markupdecl ::=
    elementdecl |
    AttlistDecl |
    EntityDecl |
    NotationDecl |
    PI | Comment
```

Attribute list declaration is next. . .

Attribute List Declaration

Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element.
- To establish type constraints for these attributes.
- To provide default values for attributes.

```
AttlistDecl ::=
  '<!ATTLIST' S Name AttDef* S? '>'
```

Attribute Definition

Attribute definition contains:

- the name of the attribute,
- its type and
- default declaration.

All are necessary.

```
AttDef ::= S Name S AttType S DefaultDecl
```

Attribute Types

Attributes are of three kinds:

- a string type
- one of tokenized types
- one of enumerated types

```
AttType ::=
    StringType |
    TokenizedType |
    EnumeratedType
```

String Type of Attributes

The string type may take any literal string as a value:

```
StringType ::= 'CDATA'
```

Tokenized Types of Attributes

There are seven kinds of tokenized types:

```
TokenizedType ::=
'ID' |
'IDREF' | 'IDREFS' |
'ENTITY' | 'ENTITIES' |
'NMTOKEN' | 'NMTOKENS'
```

Such types provide limited value-checking of attributes.

Tokenized Types of Attributes

Reminder:

```
Name ::= (Letter | '_' | ':') (NameChar)*
```

1. `ID`:
   - the value of this attribute must match `Name`
   - the value is unique in the whole document
   - only one such attribute exists for any element

2. `IDREF` – the value of this attribute must match the value of some ID attribute (in `Name`).

3. `ENTITY` – the value of this attribute must be the name of an unparsed entity declared in the DTD (in `Name`).

Tokenized Types of Attributes

Reminder:

```
Names ::= Name (S Name)*
```

4. `IDREFS` – the value must be in `Names`, each name must match the value of the ID attribute on some element in the document.

5. `ENTITIES` – the value must be in `Names`, each name is the name of an unparsed entity declared in the DTD.

Tokenized Types of Attributes

Reminder:

```
Nmtoken ::= (NameChar)+
Nmtokens ::= Nmtoken (S Nmtoken)*
```

6. `NMTOKEN` – the value must be in `Nmtoken`.

7. `NMTOKENS` – the value must be in `Nmtokens`.

Enumeration Types of Attributes

There are two kinds of enumerated types:

```
EnumeratedType ::=
  NotationType |
  Enumeration
```

Enumeration Type of Attributes

Values of the attribute of the `Enumeration` type must match one of the `Nmtoken` tokens in the declaration:

```
Enumeration ::=
  '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
```

The same `Nmtoken` should not occur more than once in the enumerated attribute types of a single element type.

Notation Type of Attributes

The value of the attribute of the `Notation` type is the name of a notation declared in the DTD.

```
NotationType ::=
  'NOTATION' S
  '(' S? Name (S? '|' S? Name)* S? ')'
```

Notations

Notations are used to interpret the content (usually non-XML) of the element to which this attribute is attached.

- all notation names in the declaration must be declared
- no element may have more than one `NOTATION` attribute

Default Declarations

Reminder:

```
AttDef ::= S Name S AttType S DefaultDecl
```

Default declaration informs:

- whether the attribute's presence is required
- if not, how an XML processor should react if a declared attribute is absent in a document.

Default Declarations

Syntax:

```
DefaultDecl ::=
   '#REQUIRED' | '#IMPLIED' |
   (('#FIXED' S)? AttValue)
```

Four possibilities:

1. `#REQUIRED` – the attribute must always be provided
2. `#IMPLIED` – the attribute is optional, no default is provided

Default Declarations

3. `#FIXED` – the attribute is optional. If present, its value must equal the default value `AttValue`:

```
AttValue ::=
   '"' ([^<&"] | Reference)* '"' |
   "'" ([^<&'] | Reference)* "'"
```

4. otherwise – the attribute is optional:

   - if present, its value is the one given
   - if absent, its value is given by the default `AttValue`

Example: Attribute Declarations

```
<document number="AB45673"/>
<document number="999 999"/>
```

```
<!ATTLIST document
   number CDATA #REQUIRED>
```

```
<document/>
<document number="c999999"/>
```

```
<!ATTLIST document number ID #IMPLIED>
```

```
<document/>
<document type="passport"/>
```

```
<!ATTLIST document type
   (passport | id) "id">
```

```
<document/>
<document type="id"/>
```

```
<!ATTLIST document type
   (passport | id) #FIXED "id">
```

```
<document number="AB45673"
children="CD32567 GH12658"/>
```

```
<!ATTLIST document number
   ID #REQUIRED children IDREFS "">
```

Demo: Attribute Declarations

```
> cd "demos/attribute declarations"
> ls
documents.xml
> java dom.Counter -v documents.xml
```

Task: Add Product's Name Attribute

```
<!DOCTYPE letter [ ...
   <!ELEMENT product EMPTY>
   <!ATTLIST product name CDATA #REQUIRED>
]>
<letter>
   ...
   <body>
    We are pleased to inform you that your
    <product name="credit card"/>
    application has been approved.
   </body>
</letter>
```

Task: Add Product's ID Attribute

```
<!DOCTYPE letter [ ...
   <!ATTLIST product
    name CDATA #REQUIRED
    id ID #REQUIRED>
]>
<letter>
   ...
   <body>
    We are pleased to inform you that your
    <product name="credit card" id="ab2345"/>
    application has been approved.
   </body>
</letter>
```

Task: Add a Reference to Product's ID

```
<!DOCTYPE letter [ ...
   <!ELEMENT enclosure (#PCDATA)>
   <!ATTLIST enclosure idref IDREF #IMPLIED>
]>
<letter>
   ...
   <body>
    ... <product name="credit card" id="ab2345"/> ...
   </body>
   <enclosure idref="ab2345">credit card</enclosure>
</letter>
```

Task: Add the Officer's Level

```
<!DOCTYPE letter [ ...
   <!ELEMENT officer (#PCDATA)>
   <!ATTLIST officer level
    (clerk | assistant | manager) #IMPLIED "clerk">
]>
<letter>
   ...
   <closure>
    Sincerely,
    <officer level="manager">Steven Rod</officer>
   </closure>
</letter>
```

Next Declaration Type

Reminder:

```
markupdecl ::=
  elementdecl |
  AttlistDecl |
  EntityDecl |
  NotationDecl |
  PI | Comment
```

Entity declaration is next. . .

Entities

Entities are a convenient way to represent information that:

- either occurs repeatedly or
- is expected to change

The information can reside in both XML instances and DTDs.

Role of Entities

An entity may represent:

1. a block of repeated text
2. a special character
3. a constant (fixed) string
4. an entire XML document
5. part of an XML document
6. part of a larger DTD
7. a content model in a DTD
8. a set of attributes in a DTD
9. a binary image
10. etc.

Pre-defined Entities

We have already seen two kinds of entities:

- pre-defined entities: `amp`, `apos`, `quot`, `gt`, `lt` to represent markup characters `&`, `'`, `"`, `>` and `<`.

- character entities: `#x0144` (hexadecimal) and `#324` (decimal) to refer to the Unicode characters, in this case to `ń`.

They are a special kind of pre-defined entities.

Entity Declaration

Other entities have to be declared in a DTD.

Even though may be uninterested in validation, you need a DTD to have user-defined entities!

If the same entity is declared more than once, the first declaration encountered is binding. XML processor may then issue a warning.

Types of Entities

Two kinds of entities exist:

1. general entity – defined in DTD but used in XML
2. parameter entity – defined and used in DTD

```
EntityDecl ::= GEDecl | PEDecl
```

A parameter entity and a general entity with the same name are two distinct entities; they use different namespaces.

General Entities

They are declared in a DTD (internal or external subset) as follows:

```
GEDecl ::=
  '<!ENTITY' S Name S EntityDef S? '>'
```

General entities are further divided into internal and external:

```
EntityDef ::=
  EntityValue | (ExternalID NDataDecl?)
```

Internal Entities

Internal entities: replacement text defined within a document and referenced from one or more locations in this document

- declared like this:

```
EntityDef ::= EntityValue
EntityValue ::=
  '"' ([^%&"] | PEReference | Reference)* '"' |
  "'" ([^%&'] | PEReference | Reference)* "'"
```

- referred like this: `&name;`

`PEReference` is a reference to parameter entity.

Example: Declaration of Internal Entities

```
   text replacement ->  <!ENTITY type "XML editor">
 markup replacement ->  <!ENTITY product "
' used instead of " ->  <product type='&type;'>
                           XML Typewriter
                        </product>
replacement text with:  ">
 1.entity reference ->  <!ENTITY disclaimer
                           "&company; accepts no responsibility
                           for damages due to corruption or loss
                           of your data caused by &product;."
                        >
2.predefined entity ->  <!ENTITY company "First&amp;Best">
 3.character entity ->  <!ENTITY copyright "Copyright &#xA9;
                           2003">
```

Example: Replacement of Internal Entities

```
&product; is the best
&type; available on
the market. It helps
writing XML without
worrying about those
silly syntax rules.

However, &disclaimer.

&copyright
```

```
<product type="XML editor"> XML
Typewriter </product> is the best
XML editor available on the market.
It helps writing XML without worrying
about those silly syntax rules.

However, First&Best accepts no
responsibility for damages due to
corruption or loss of your data
caused by <product type="XML editor">
XML Typewriter </product>.

Copyright © 2003
```

Example: Validation of Internal Entities

```
                                <!DOCTYPE advert [
entity declarations ->           <!ENTITY type "...">
                                 <!ENTITY product "...">
                                 <!ENTITY disclaimer "...">
                                 <!ENTITY copyright "...">
element declarations ->          <!ELEMENT advert (#PCDATA | product)*>
                                 <!ELEMENT product (#PCDATA)>
                                 <!ATTLIST product type CDATA #REQUIRED>
                                ]>
root element ->                 <advert>
                                   &product; is the best &type; available
                                   on the market. It helps writing XML
                                   without worrying about those silly
                                   syntax rules.
                                   However, &disclaimer;. &copyright;
                                </advert>
```

Demo: Internal Entities

```
> cd "demos/internal entities"
> ls
advert.xml
> emacs advert.xml
> java dom.Counter -v advert.xml
```

Well-Formed Internal Entities

An internal general parsed entity is well-formed if its replacement text matches content:

```
content ::= CharData?
  ( (element | Reference | CDSect | PI | Comment)
  CharData?
  )*
```

The replacement text is the content of the entity, after replacement of character references and parameter-entity references.

Example: Well-Formed Internal Entities

```
   parameter entity ->  <!ENTITY % pub "O'Reilly &amp; Associates" >
     general entity ->  <!ENTITY rights "All rights reserved" >
     general entity ->  <!ENTITY book
                           "Learning XML: Erik T. Ray,
character reference ->     &#xA9; 1947
parameter reference ->     %pub;.
  general reference ->     &rights;" >

  replacement text for
the entity ``book'' ->  Learning XML: Erik T.Ray, © 2001
                        O'Reilly & Associates. &rights;
```

Pre-Defined Entities Revisited

Character entities are instances of internal general entities, so are the predefined entities `amp`, `quot`, `apos`, `lt` and `gt`.

All XML processors must recognize them whether they are declared or not. If declared, the following should be used:

```
<!ENTITY lt "&#38;#60;">
<!ENTITY gt "&#62;">
<!ENTITY amp "&#38;#38;">
<!ENTITY apos "&#39;">
<!ENTITY quot "&#34;">
```

Notice the double-escape for `lt` and `amp`:

```
lt -> &#38;#60; -> &#60; -> <
```

Task: Customize Decision – Approved

```
<!DOCTYPE letter [ ...
   <!ENTITY polite "are pleased">
   <!ENTITY decision "approved">
]>
<letter decision="reject">
   ...
   <body>
    We &polite; to inform you that your
    <product>credit card</product>
    application has been &decision;.
   </body>
</letter>
```

Task: Customize Decision – Rejected

```
<!DOCTYPE letter [ ...
   <!ENTITY polite "regret">
   <!ENTITY decision "rejected">
]>
<letter decision="reject">
   ...
   <body>
    We &polite; to inform you that your
    <product>credit card</product>
    application has been &decision;.
   </body>
</letter>
```

External Entities

Reminder:

```
EntityDef ::= EntityValue | (ExternalID
  NDataDecl?)
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
```

External entities match `ExternalID NDataDecl?`.

Types of External Entities

They refer to the data (XML or non-XML) stored in an external file:

- external parsed entity – without `NDataDecl`

```
EntityDef ::= ExternalID
```

- external unparsed entity – with `NDataDecl`

```
EntityDef ::= ExternalID NDataDecl?
```

External Parsed Entities

External parsed entity is used to include fragments of XML from an external file into the current XML document.

Those fragments:

- need not be well-formed
- cannot contain document type declarations

A useful technique for constructing large XML documents from a set of smaller files and sharing content in many XML documents.

Well-Formed External Parsed Entities

An external parsed entity is well-formed if the content of its replacement file matches the production `extParsedEnt`:

```
extParsedEnt ::= TextDecl? Content
```

Such files may begin with a text declaration:

```
TextDecl ::=
   '<?xml' VersionInfo? EncodingDecl S? '?>'
```

Similar to the XML declaration but the version declaration is optional and the standalone declaration is forbidden.

Example: External Parsed Entities

```
                    <?xml version="1.0"?>
                    <!DOCTYPE card SYSTEM "card.dtd" [
      external ->     <!ENTITY visitor SYSTEM "visitor.xml">
        parsed ->     <!ENTITY document SYSTEM "document.xml">
        entity ->     <!ENTITY addresses SYSTEM "addresses.xml">
  declarations ->     <!ENTITY travel SYSTEM "travel.xml">
                    ]>
                    <card type="arrival">
      external ->     &visitor;
        parsed ->     &document;
        entity ->     &addresses;
    references ->     &travel;
                      <signature sigfile="mysig"/>
                    </card>
```

Example: External Parsed Entities Fragments

visitor.xml -> well-formed text declaration: UTF-8 encoding

```
<?xml encoding="UTF-8"?>
<visitor>
  <name type="surname">Kowalski</name>
  <name type="given">Jan</name>
  ...
</visitor>
```

addresses.xml -> ill-formed text declaration: UTF-16 encoding

```
<?xml encoding="UTF-16"?>
<address where="home">
  Morska 24B, 24-650 Gda&#x0144;sk,
  Poland
</address>
<address where="Macao">
  Flower Garden 12B, Taipa
</address>
```

Demo: External Parsed Entities

```
> cd "demos/external parsed entities"
> ls
card.xml card.dtd visitor.xml
document.xml addresses.xml travel.xml
> emacs *.xml
> java dom.Counter -v card.xml
> java dom.Counter visitor.xml
> java dom.Counter document.xml
> java dom.Counter addresses.xml
> java dom.Counter travel.xml
```

Task: Letter Body as External Parsed Entity

```
<!DOCTYPE letter [
   ...
   <!ELEMENT body (#PCDATA | product)*>
   <!ENTITY body SYSTEM "body.xml">
]>
<letter>
   <opening>...</opening>
   &body;
   <closure>...</closure>
</letter>
```

Next Declaration Type

Before we introduce unparsed entities a detour to XML Notations.

Reminder:

```
markupdecl ::=
    elementdecl |
    AttlistDecl |
    EntityDecl |
    NotationDecl |
    PI | Comment
```

Notations

Notations identify by name:

- the format of unparsed entities,
- the format of elements which bear a notation attribute, or
- the application to which a processing instruction is addressed

Notation Declarations

Notations are declared as follows:

```
NotationDecl ::=
  '<!NOTATION' S
  Name S (ExternalID | PublicID) S? '>'
```

Notation Declarations

System literal (typically URL or MIME type), public literal (`-//org//desc//lang`), or both:

```
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
PublicID ::= 'PUBLIC' S PubidLiteral
```

Only one notation can be defined for a given name.

Notations and XML Processors

XML processors:

- must provide applications with the name and external identifiers of any notation used
- may resolve the external identifier to find a processor for data in the notation described

It is not an error if an XML document declares a notation for which no notation-specific processor is available on the system.

Example: Notation Declarations

1. MIME type for jpg

```
<!NOTATION jpeg SYSTEM "image/jpg">
```

2. MIME type for troff-encoded text

```
<!NOTATION troff
   SYSTEM "application/x-troff">
```

3. application to process data

```
<!NOTATION gif
   SYSTEM "/usr/local/bin/xv">
```

4. URL to a technical document about formats

```
<!NOTATION date
   SYSTEM "http://www.w3.org/TR/date.pdf">
```

5. A formal public identifier to an online resource

```
<!NOTATION date
   PUBLIC "-//W3C//Date Format//EN"
   SYSTEM "http://www.w3.org/TR/date">
```

External Unparsed Entities

Reminder:

```
EntityDef ::= EntityValue | (ExternalID
  NDataDecl?)
ExternalID ::=
  'SYSTEM' S SystemLiteral |
  'PUBLIC' S PubidLiteral S SystemLiteral
```

External unparsed entity declaration contains `ExternalID` as well as `NDataDecl`, where `Name` is the declared name of a notation:

```
NDataDecl ::= S 'NDATA' S Name
```

External Unparsed Entities: Declaration

External unparsed entity is used to incorporate non-XML, typically binary data into the XML document:

```
<!ENTITY view SYSTEM "view.jpg" NDATA jpeg>
```

It is declared in conjunction with a declared notation:

```
<!NOTATION jpeg SYSTEM "image/jpg">
```

External Unparsed Entities: Use

It is used as the value of the attribute of type `ENTITY` or `ENTITIES`, not via the `&view;` reference:

```
<!ELEMENT product (#PCDATA)>
<!ATTLIST product
   img ENTITY #IMPLIED>
```

```
<product img='view' ...>
   XML Typewriter
</product>
```

Example: External Unparsed Entities

```
                                <!DOCTYPE advert [
          root element ->          <!ELEMENT advert (#PCDATA | product)*>
       'product' child ->          <!ELEMENT product (#PCDATA)>
   attributes of product:          <!ATTLIST product
        text attribute ->                type CDATA #REQUIRED
      entity attribute ->                img ENTITY #IMPLIED>
  notation declaration ->          <!NOTATION jpeg SYSTEM "image/jpg">
       unparsed entity ->          <!ENTITY view SYSTEM
                                          "view.jpg" NDATA jpeg>
                                   ...
          unparsed entity       ]>
       reference as value       <advert>
    of 'img' attribute ->        <product img='view' ...>
                                  XML Typewriter
                                 </product> is the best &type; ...
                                </advert>
```

Demo: External Unparsed Entities

```
> cd "demos/external unparsed entities"
> ls
advert.xml
> emacs advert.xml
> java dom.Counter -v advert.xml
```

Task: Signature as External Unparsed Entity

```
<!DOCTYPE letter [ ...
   <!ATTLIST officer
        level (clerk | assistant | manager) "clerk"
        signature ENTITY #IMPLIED>
   <!ENTITY rod SYSTEM "rod.jpeg" NDATA jpeg>
   <!NOTATION jpeg SYSTEM "image/jpg">
]>
<letter> ...
   Sincerely,
   <officer level="manager" signature="rod">
   Steven Rod
   </officer> ...
</letter>
```

Parameter Entities

Reminder:

```
EntityDecl ::= GEDecl | PEDecl
```

Parameter entities provide a reuse mechanism for building a DTD.

Parameter Entities

They are declared and used in DTD only:

- declaration:

```
PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
PEDef ::= EntityValue | ExternalID
```

- reference: `%name;`

Parameter-entity replacement text must be properly nested with markup declarations (and parenthesized groups).

Parameter versus General Entities

Compare with general entities:

1. declaration of parameter entity uses `%` after `ENTITY`
2. declaration of parameter entity has no `NDataDecl`: there are no unparsed external parameter entities
3. parameter entities are references `%name`; not `&name`;
4. parameter entity reference is recognized in DTD only

Internal and External Parameter Entities

- internal entities – to define internally a replacement text for building a DTD from reusable pieces

```
PEDecl ::=
  '<!ENTITY' S '%' S Name S EntityValue S? '>'
```

- external entities – bring DTD content from an outside file

```
PEDecl ::=
  '<!ENTITY' S '%' S Name S ExternalID S? '>'
```

Internal Parameter Entity Usage

Common usage scenarios:

1. to serve as documentation of an intended specific datatype for an attribute that is actually declared as `CDATA` or `NMToken`:

```
<!ENTITY % Margin "CDATA">
<!ENTITY % Color "CDATA">
<!ENTITY % URI "CDATA">
```

Internal Parameter Entity Usage

2. to specify an element content model that is common in a DTD:

```
<!ENTITY % fontstyle "tt | i | b | big | small">
<!ENTITY % phrase "em | strong | code | cite |
   code">
<!ENTITY % inline "a | %fontstyle; | %phrase;">
```

Internal Parameter Entity Usage

3. to collect a group of related attribute declarations that are used repeatedly in various content models in a DTD:

```
<!ATTLIST frame %standard; %color; %margins;>
<!ENTITY % standard "
   id ID #IMPLIED
   xml:base %URI; #IMPLIED">
<!ENTITY % color "
   background %Color; #IMPLIED
   foreground %Color; #IMPLIED">
<!ENTITY % margins "
   left %Margin; #IMPLIED
   right %Margin; #IMPLIED">
```

Parameter Entity References

In entity declarations, as entity value:

```
EntityValue ::=
  '"' ([^%&"] | PEReference | Reference)* '"' |
  "'" ([^%&'] | PEReference | Reference)* "'"
```

This case is only allowed in the external DTD subset (`ExternalID`), not the internal subset (`markupdecl`):

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

Parameter Entity References

In document type declarations, as markup separators:

```
doctypedecl ::=
  '<!DOCTYPE' S Name (S ExternalID)? S?
  '[' (markupdecl | DeclSep)* ']' S?)? '>'
DeclSep ::= PEReference | S
```

This case is allowed in both internal and external DTD subsets.

Example: External Parameter Entities

external general ->
entity references ->
->
->

```
<?xml version="1.0"?>
<!DOCTYPE card SYSTEM "card.dtd">
<card type="arrival">
   &visitor;
   &document;
   &addresses;
   &travel;
   <signature sigfile="mysig"/>
</card>
```

Example: External Parameter Entities

```
internal parameter entity
            declaration ->  <!ENTITY % details "document, address, ...">
              reference ->  <!ELEMENT card (visitor, %details;, ...)>
external parameter entity
            declaration ->  <!ENTITY % visitor SYSTEM "visitor.dtd">
                        ->  <!ENTITY % document SYSTEM "document.dtd">
                        ->  <!ENTITY % address SYSTEM "address.dtd">
                        ->  <!ENTITY % travel SYSTEM "travel.dtd">
              reference ->  %visitor;
                        ->  %document;
                        ->  %address;
                        ->  %travel;
  external general entity
            declaration ->  <!ENTITY visitor SYSTEM "visitor.xml">
                        ->  <!ENTITY document SYSTEM "document.xml">
                        ->  <!ENTITY addresses SYSTEM "addresses.xml">
                            <!ENTITY travel SYSTEM "travel.xml">
```

Demo: Parameter Entities

```
> cd "demos/parameter entities"
> ls
card.xml visitor.xml document.xml
addresses.xml travel.xml
card.dtd date.dtd visitor.dtd document.dtd
address.dtd travel.dtd signature.dtd
> java dom.Counter -v card.xml
```

Task: Officer Level as Parameter Entity

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ENTITY % levels "clerk | assistant | manager">
  <!ATTLIST officer level (%levels;) "clerk">
]>
<letter>...</letter>
```

Doesn’t work?

Task: Officer Level as Parameter Entity

XML document:

```
<?xml version="1.0"?>
<!DOCTYPE letter SYSTEM "letter.dtd">
<letter>...</letter>
```

DTD document:

```
...
<!ENTITY % levels "clerk | assistant | manager">
<!ATTLIST officer level (%levels;) "clerk">
```

Task: Entity Declarations in External File

DTD document:

```
<!ENTITY % entities SYSTEM "entities.dtd">
%entities;
...
```

Another DTD document:

```
<!ENTITY % levels "clerk | assistant | manager">
<!ENTITY polite "are sorry">
<!ENTITY decision "rejected">
```

Well-Formed Entities

- internal general entity – replacement text (text after expansion of character and parameter references) matches content

```
content ::=
  CharData?
  (
   ( element |
      Reference |
      CDSect |
      PI |
      Comment)
   CharData?
  )*
```

More Well-Formed Entities

- external general entity – it matches `extParsedEnt`

```
extParsedEnt ::= TextDecl? Content
```

- parameter entities are well-formed by definition

Nesting of Logical and Physical Structures

The consequence of well-formedness is that the logical and physical structures in an XML document are properly nested.

None of:

start-tag	comment
end-tag	processing instruction
empty-element tag	character reference
element	entity reference

can begin in one entity and end in another.

External DTD Fragments

External DTD fragments are linked from XML document in two ways:

- As replacement text of an external parameter entity reference, occurring as the declaration separator (`DeclSep`), matching:

```
extSubsetDecl ::=
    (markupdecl | conditionalSect | DeclSep)*
```

A list of markup declarations, conditional sections and declaration separators.

External DTD Fragments

- as `ExternalID` in the document type declaration:

```
doctypedecl ::=
 '<!DOCTYPE' S Name (S ExternalID)? S?
 ('[' (markupdecl | DeclSep)* ']' S?)? '>'
 DeclSep ::= PEReference | S
```

This is called the external subset. It has to match:

```
extSubset ::= TextDecl? extSubsetDecl
```

A replacement text for external parameter references, with an optional text declaration.

Conditional Sections

Portions of the document type declaration external subset which are included in, or excluded from, the DTD:

```
conditionalSect ::=
  '<![' S? conditionTest S? '[' extSubsetDecl
  ']]>'
```

Conditional Sections Tests

The test is one of:

- `INCLUDE`
- `IGNORE` or
- a parameter entity reference who may be either of them

```
conditionTest ::=
    'INCLUDE' |
    'IGNORE' |
    PEReference
```

Example: Conditional Sections

always include ->

```
<![INCLUDE[
   <!ELEMENT travel (place, flight?)>
   <!ATTLIST travel type (from) #IMPLIED>
]]>
```

always ignore ->

```
<![IGNORE[
   <!ELEMENT travel (place, flight?)>
    <!ATTLIST travel type (to) #IMPLIED>
]]>
```

parameter entity ->

```
<!ENTITY % direction "IGNORE">
```

include or ignore ->

```
<![%direction;[
   <!ELEMENT travel (place, flight?)>
   <!ATTLIST travel type (to) #IMPLIED>
]]>
```

Task: Conditional Acceptance/Rejection

```
<!ENTITY % accept "INCLUDE">
<!ENTITY % reject "IGNORE">

<![%accept;[
   <!ENTITY polite "are pleased">
   <!ENTITY decision "approved">
]]>

<![%reject;[
   <!ENTITY polite "are sorry">
   <!ENTITY decision "rejected">
]]>
```

XML Processors

XML processors: validating and non-validating.

Both must report violations of well-formedness encountered in the main document and any other parsed entity that they read.

- validating processors must report violations of the constraints expressed by the declarations in the DTD. They must read and process the entire DTD and all external parsed entities referenced in the document.

- non-validating processors have to check the main document, including the entire internal DTD subset, for well-formedness.

Namespaces

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

Mixing Vocabularies

Suppose we want to embed SVG image in a DocBook document.

Both SVG and DocBook have the `title` elements.

How to distinguish them?

```
<book>
 <title>SVG Manual</title>
 <chapter>
  ...
  <mediaobject>
   <imageobject>
    <svg>
     <title>SVG Example</title>
     ...
    </svg>
   </imageobject>
  </mediaobject>
 </chapter>
</book>
```

In general, how to mix different vocabularies in an XML document to avoid name conflicts?

Name Disambiguation with URI references

Solution: leverage the uniqueness of Universal Resource Identifiers (URI) ensured by the DNS (Domain Name System):

- SVG – http://www.w3.org/TR/SVG
- DocBook – http://www.oasis-open.org/docbook
- XML Schema - http://www.w3.org/2001/XMLSchema
- etc.

Apply URI as a prefix to element/attribute names.

URI as Name Prefixes

Consider this:

```
<book>
 <http://www.oasis-open.org/docbook:title>
  SVG Manual
 </http://www.oasis-open.org/docbook:title>
 ...
 <http://www.w3.org/TR/SVG:title>
  SVG Example
 </http://www.w3.org/TR/SVG:title>
</book>
```

Two problems with this solution:

- element names become very long
- URIs include characters not allowed in XML names

XML Namespaces

Instead, associate URI with a short name – namespace – then use this namespace as a prefix to qualify element/attribute names.

```
<book
   xmlns:docbook="http://www.oasis-open.org/docbook"
   xmlns:svg="http://www.w3.org/TR/SVG">
   ...
   <docbook:title>SVG Manual</docbook:title>
   ...
   <svg:title>SVG Example</svg:title>
   ...
</book>
```

Namespace Declaration

A namespace is declared using a family of reserved attributes. Such attribute names have `xmlns` or `xmlns:` as a prefix.

```
NSAttName ::= PrefixedAttName | DefaultAttName
```

Two kinds of namespace declarations:

- regular – include a prefix

  ```
  PrefixedAttName ::= 'xmlns:' NCName
  ```

- default – without a prefix

  ```
  DefaultAttName ::= 'xmlns'
  ```

Namespace Prefix

`NCName` is the namespace prefix - any legal XML name without a colon:

```
NCName ::= (Letter | '_') (NCNameChar)*
NCNameChar ::=
  Letter | Digit | '.' | '-' | '_' |
  CombiningChar | Extender
```

Prefixes beginning with the sequence `xml` in any case combination, are reserved by XML and XML-related specification.

Namespace Value

The value of the namespace attribute is the namespace name:

- this should be a URI reference
- empty string is also allowed, but only for a default namespace
- the URI may, but need not, point to an existing address
- typically, the namespace points to formal description of a vocabulary: W3C Recommendation, DTD or XML Schema

Example: Namespace declarations

```
xmlns:docbook="http://www.oasis-open.org/docbook"
xmlns:svg="http://www.w3.org/TR/SVG"
xmlns="http://www.w3.org/TR/REC-xml-names"
xmlns=""
```

Example: Namespace Declarations

```
xmlns:docbook="http://www.oasis-open.org/docbook"

xmlns:svg="http://www.w3.org/TR/SVG"

xmlns="http://www.w3.org/TR/REC-xml-names"

xmlns=""
```

Qualified Names

A qualified name is an XML name without “:”, or with one “:” separating the prefix and the local part:

```
QName ::= (Prefix ':')? LocalPart
Prefix ::= NCName
LocalPart ::= NCName
```

The prefix must be declared in a namespace declaration.

Example: Qualified Names

```
<book xmlns:docbook="http://www.oasis-open.org/docbook">
        <docbook:title>
                ...
        </docbook:title>
        ...
</book>
```

Element and Attributes Revisited

Start tags, end tags and empty element tags with qualified names:

```
STag ::= '<' QName (S Attribute)* S? '>'
ETag ::= '</' QName S? '>'
EmptyElemTag ::= '<' QName (S Attribute)* S? '/>'
```

Attributes are either namespace declarations or their names are given as qualified names:

```
Attribute ::=
  NSAttName Eq AttValue | QName Eq AttValue
```

Namespace Constraint

The namespace prefix used in element- and attribute-names:

- `xmlns` is used only for namespace binding and is not itself bound to any namespace name
- `xml` is by definition bound to `http://www.w3.org/XML/1998/namespace`
- any other prefix must have been declared:
  - in the start-tag of the element where the prefix is used, or
  - in an ancestor element

DTDs Revisited 1

Qualified names appear in:

1) document type declarations, to refer to the root element:

```
doctypedecl ::= '<!DOCTYPE' S
 QName (S ExternalID)? S?
 ('[' (markupdecl | PEReference | S)* ']' S?)? '>'
```

DTDs Revisited 2

2) in element declarations, content particles and mixed models:

```
elementdecl ::=
   '<!ELEMENT' S QName S contentspec S? '>'

cp ::= (QName | choice | seq) ('?' | '*' | '+')?

Mixed ::=
   '(' S? '#PCDATA' S? ')' |
   '(' S? '#PCDATA' (S? '|' S? QName)* S? ')*'
```

DTDs Revisited 3

3) in attribute list declarations and attribute definitions:

```
AttlistDecl ::=
   '<!ATTLIST' S QName AttDef* S? '>'

AttDef ::=
   S (QName | NSAttName) S AttType S DefaultDecl
```

Location of Namespace Declarations

Due to namespace declaration constraints on XML documents, namespace declarations should be provided:

- directly in the XML document
- via a default attribute declared in the internal DTD subset

Namespace Scoping

The namespace declaration applies to:

- the element where it is specified and
- to descendants of this element, unless overridded by another namespace declaration with the same `NSAttName` part.

Multiple namespace can be declared within a single element.

Example: Namespace Scoping

```
http://www.nul1.org ->  <nul:book xmlns:nul="http://www.nul1.org">
                            …
                  none ->       <title>
                                    …
                                </title>
                  none ->       <chap xmlns:nul="http://www.nul2.org">
                                    …
http://www.nul2.org ->              <nul:title>
                                        …
                                    </nul:title>
                                </chap>
http://www.nul1.org ->          <nul:chap>
                                    …
                                </nul:chap>
                        </nul:book>
```

Example: Multiple Namespaces

```
                  none ->  <book
                            xmlns:ns1="http://www.nul1.org"
                            xmlns:ns2="http://www.nul2.org">
http://www.nul1.org ->     <ns1:title>...</ns1:title>
http://www.nul2.org ->     <ns2:title>...</ns2:title>
http://www.nul2.org ->     <ns3:chap
                                  xmlns:ns3="http://www.nul2.org">
                            <title
                  ERROR ->   ns2:status=...
                        ->   ns3:status=... >
                             ...
                            </title>
                           </ns3:chap>
                          </book>
```

Default Namespace

A default namespace applies to:

- the element where it is declared, provided it has no prefix
- to all its descendants with no prefix

If the URI reference is empty, then un-prefixed elements within its scope are considered not to be in any namespace.

Default namespaces do not apply to attributes.

Example: Default Namespace

```
 http://www.nul.org ->  <book
                          xmlns="http://www.nul.org"
                          xmlns:ns1="http://www.nul1.org"
                          xmlns:ns2="http://www.nul2.org">
 http://www.nul.org ->    <title>...</title>
http://www.nul1.org ->    <ns1:title>...</ns1:title>
http://www.nul2.org ->    <ns2:chap xmlns="">
                 none ->     <title>...</title>
                             ...
                          </ns2:chap>
                        </book>
```

Task: Arrival Card with Namespaces

Retype a simple arrival card:

```
<?xml version="1.0"?>
<card>
   <visitor>
    <name>Jan Kowalski</name>
   </visitor>
   <document>passport</document>
</card>
```

1. Declare the default namespace `http://www.macau.gov.mo` for all elements of the document.
2. Declare the namespace `http://www.kowalski.org` for the `visitor` element.
3. Make sure that `name` does not belong to any namespace.

XML Namespace Recommendation

Namespaces in XML:

- W3C Recommendation published in January 1999
- Editors: Tim Bray (Textuality), Dave Hollander (Hewlett-Packard), Andrew Layman (Microsoft)
- abstract:

  XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

XML Namespace Conformance

Conformance to the XML namespace specification:

- element and attribute names contain either zero or one colon
- no entity name, PI target or notation name contains any colon
- values of the attributes of types `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES` and `NOTATION` should contain no colon

XML Schema

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

DTD Limitations 1-2

1. document-centric – DTDs are a simplification of SGML DTDs, which themselves are document-focused.

   As a result, DTDs are more suited to describe the content of documents, and less to describe structured data.

2. no meta-data access - applications cannot access the content of the DTD, once the document is processed by an XML parser, e.g. via DOM.

   Grouping, sharing and reuse of markup declarations and all metadata information (data about data) in a DTD are lost.

DTD Limitations 3

3. limited datatyping – DTDs provide very limited datatyping

elements	attributes	
EMPTY	CDATA	
PCDATA	ID	NOTATION
element	IDREF	IDREFS
mixed	NMTOKEN	NMTOKENS
ANY	ENTITY	ENTITIES

DTD Limitations 4-5

4. ranges or sets are hard to define - DTD enable enumeration of legal values for attributes and this is helpful for only very small sets – but not element content.

```
<!ATTLIST date dayofweek
 (monday | tuesday | wednesday | thursday |
 friday | saturday | sunday)
 #IMPLIED
>
```

5. no subclassing - DTD does not permit describing common data structures in a class definition, and capturing all variations in subclasses.

DTD Limitations 6-7

6. order of children is too rigid - DTD require us either:
   - list all children elements (optional or not) in the order in which they must occur

     ```
     <!ELEMENT P (A, B+, C?)>
     ```

   - or use a mixed model where no order constraints are imposed

     ```
     <!ELEMENT P (#PCDATA| A, B, C)*>
     ```

7. no namespace support – to check validity, we must keep prefixes in XML in synch with the DTD: if you change one, you have to change the other.

DTD Limitations 8-9

8. limited ways to express repetitions

   element B must occur exactly 15 times:

```
<!ELEMENT P (A,B,B,B,B,B,B,B,B,B,B,B,B,B,B,C)>
```

   element B may occur between 13 and 15 times:

```
<!ELEMENT P
 (A,B,B,B,B,B,B,B,B,B,B,B,B,B,C)
 (A,B,B,B,B,B,B,B,B,B,B,B,B,B,B,C)
 (A,B,B,B,B,B,B,B,B,B,B,B,B,B,B,C)>
```

9. DTDs are written in native, non-XML syntax - XML tools cannot be used to process DTD documents.

XML Schema – W3C Recommendation 1

W3C Recommendation May 2001.

Three parts:

1. Primer - A non-normative document intended to provide an easily readable description of the XML Schema language. XML Schema Part 1: Structures and XML Schema Part 2: Datatypes provide the complete normative description of the XML Schema language.

XML Schema – W3C Recommendation 2

2. Structures - XML Schema language offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespaces. The schema language, which is itself represented in XML 1.0 and uses namespaces, considerably extends the capabilities found in DTDs.

3. Datatypes - Defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 DTDs for specifying datatypes on elements and attributes.

Example: XML versus DTD

XML instance ->
document ->

```
->  <?xml version="1.0"?>
->  <!DOCTYPE date SYSTEM "date.dtd">
->  <date>
->   <day>14</day>
->   <month>September</month>
->   <year>2003</year>
->   <weekday>Sunday</weekday>
->  </date>
```

Document Type Definition ->

```
->  <!ELEMENT date (day, month, year, weekday?)>
->  <!ELEMENT day (#PCDATA)>
->  <!ELEMENT month (#PCDATA)>
->  <!ELEMENT year (#PCDATA)>
->  <!ELEMENT weekday (#PCDATA)>
```

Example: XML Schema

```
                    <?xml version="1.0"?>
  schema element -> <xsd:schema
schema namespace ->  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    date element ->  <xsd:element name="date">
    complex type ->   <xsd:complexType>
     sequence of ->    <xsd:sequence>
             day ->     <xsd:element name="day" type="xsd:string"/>
           month ->     <xsd:element name="month" type="xsd:string"/>
            year ->     <xsd:element name="year" type="xsd:string"/>
         weekday ->     <xsd:element name="weekday"
     is optional ->       minOccur="0" type="xsd:string"/>
                    </xsd:sequence>
                   </xsd:complexType>
                  </xsd:element>
                 </xsd:schema>
```

Example: XML Referring Schema

Location of the schema document -> Namespace -> declaration for XML instance documents

```
<?xml version="1.0"?>
<date
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="date.xsd"
>
  <day>14</day>
  <month>September</month>
  <year>2003</year>
  <weekday>Sunday</weekday>
</date>
```

Example: XML Referring Schema and DTD

```
                    <?xml version="1.0"?>
external DTD ->     <!DOCTYPE date SYSTEM "date.dtd"
internal DTD ->     [
                     <!ATTLIST date
   attribute ->      xsi:noNamespaceSchemaLocation CDATA #IMPLIED
declarations ->      xmlns:xsi CDATA #FIXED "...XMLSchema-instance">
                    ]>
                    <date
schema location      xsi:noNamespaceSchemaLocation="date.xsd"
              ->     xmlns:xsi="...XMLSchema-instance">
   namespace ->      <day>14</day>
    declaration      <month>September</month>
                     <year>2003</year>
                     <weekday>Monday</weekday>
                    </date>
```

Demo: DTD versus Schema Validation

```
> cd "demos/dtd versus schema validation"
> ls
date.xml date.dtd date.xsd
dateDTD.xml dateSchema.xml dateDTDSchema.xml
> java dom.Counter date.xml
> java dom.Counter -v dateDTD.xml
> java dom.Counter -s dateSchema.xml
> java dom.Counter -v dateDTDSchema.xml
> java dom.Counter -v -s dateDTDSchema.xml
```

What Have Been Gained?

So we replace a 5-line DTD with a 14-line XML Schema.

What have been gained?

A solid foundation to build a better schema:

- more stringent datatyping
- reuse of data structures
- more expressive content model
- XML syntax reused
- XML tools reapplied
- self-description and -validation
- etc.

Schema Document Structure

root element ->	<schema …>
any number of ->	<include …/>
	<import>…</import>
	<redefine>…</redefine>
	<annotation>…</annotation>
any number of ->	<simpleType>…</simpleType>
	<complexType>…</complexType>
	<element>…</element>
	<attribute …/>
	<attributeGroup>…</attributeGroup>
	<group>…</group>
	<annotation>…</annotation>
	</schema>

Schema Namespace

XML schema standards is namespace-sensitive:

1. it can describe documents with elements and attributes that belong to different namespaces
2. namespaces distinguish between references to built-in data types and other types defined by the schema author

Schema belongs to `http://www.w3.org/2001/XMLSchema` namespace:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
   ...
</schema>
```

Referring to Schema: No Namespace 1

XML instance document with no namespace:

```
<elem
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="schemaNoNamespace.xsd">
   this is text
</elem>
```

The attribute `noNamespaceSchemaLocation` determines the schema location for those elements that do not belong to any namespace.

The attribute belongs itself to the namespace `http://www.w3.org/2001/XMLSchema-instance`.

Referring to Schema: No Namespace 2

XML Schema file `schemaNoNamespace.xsd`:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="elem" type="xsd:string"/>
</xsd:schema>
```

Demo: No Namespace Validation

```
> cd "demos/schema no namespace"
> dir
schemaNoNamespace.xsd noNamespace.xml
> java dom.Counter -v -s noNamespace.xml
```

Comments

In addition to the normal XML comments, annotation elements distinguish between human- and software-aimed comments:

```
<annotation>
    <documentation source="…">
        this is documentation
    </documentation>
    <appinfo source="…">processing instruction</appinfo>
</annotation>
```

The source element includes a URL to the document with further information about the issue.

Element Declarations

Element declaration requires that there exists an element in the instance document:

- whose name is given by the `name` attribute and
- whose content is of the type specified

```
<xsd:element name="date">
  <xsd:complexType>...</xsd:complexType>
</xsd:element>
```

Named and Anonymous Types

The type of the component may be given:

- as anonymous type embedded directly inside `element`:

```
<xsd:element name="date">
  <xsd:complexType>...</xsd:complexType>
</xsd:element>
```

- as the named type referred by `element` via its `type` attribute

```
<xsd:element name="date" type="Date"/>
<xsd:complexType name="Data">
  ...
</xsd:complexType>
```

Named Type Reuse

Named types can be reused by several element declarations:

```
<xsd:element name="date" type="Date"/>
<xsd:element name="mydate" type="Date"/>

<xsd:complexType name="Data">
  ...
</xsd:complexType>
```

Anonymous types are only used within one element declaration.

Top-Level Elements

Element declaration can occur at:

- top level – element must exist in the instance document

```
<xsd:element name="date" type="Date"/>
```

When several top-level element declaration are given, one of the elements must exist in the document.

Local-Level Elements

- local level – part of the type's definition

```
<xsd:complexType name="Data">
  ...
  <xsd:element name="day" type="xsd:string">
  ...
</xsd:complexType>
```

Task: Schema for Arrival Card

Return the arrival card to the no-namespace version:

```
<?xml version="1.0"?>
<card>
   <visitor>
    <name>Jan Kowalski</name>
   </visitor>
   <document>passport</document>
</card>
```

Design the schema for this card.

Refer to the schema from the XML document.

Schema-validate the document.

Element Repetition

Local-level elements can contain repetition attributes:

- `minOccurs` – minimal number of occurrences, default 1

```
<xsd:element name="elem" minOccurs="0"/>
```

- `maxOccurs` – maximum number of occurrences, default 1

```
<xsd:element name="elem"
  minOccurs="2" maxOccurs="unbounded"/>
```

Task: Repeated Names

In the schema allow the visitor to have from one to three names.

Modify the XML document.

Validate.

Element Reference

Top-level elements can be referred when declaring local-level elements; no `name` and `type` attributes are needed.

```
<xsd:element name="day" type="xsd:string"/>

<xsd:complexType name="Date">
  ...
  </xsd:element ref="day"/>
  ...
</xsd:complexType>
```

Task: Top-Level Visitor Element

In the schema define the visitor element on the top-level.

Refer to the visitor from the card type.

Validate.

Simple and Complex Types

Types can be simple or complex:

- simple types describe values of attributes, as well as elements that contain text and attributes (but no children)

```
<xsd:simpleType name="productNumber">
  ...
</xsd:simpleType>
```

- complex types describe elements with text, attributes and children

```
<xsd:complexType name="productSpecification">
  ...
</xsd:complexType>
```

Pre-Defined and User-Defined Types

XML Schema provides 44 pre-defined simple types, usually referred to with a prefix. Other types have to be declared:

- pre-defined type `xsd:string`

```
<xsd:element name="day" type="xsd:string"/>
```

- user-defined type `Date`

```
<xsd:element name="date" type="Date"/>
<xsd:complexType name="Data">
  ...
</xsd:complexType>
```

Pre-Defined Types: DTD and String

DTD type	Schema type	example
ID	string	a practical guide
IDREF	normalizedString	a practical guide
IDREFS	token	a practical guide
ENTITY	Name	my:book, book
ENTITIES	NCName	book
NMTOKEN	QName	my:book
NMTOKENS	language	de, en
NOTATION	anyURI	http://www.iist.unu.edu

Pre-Defined Types: Numeric

Schema type	Schema type
boolean	byte
float	short
double	int
decimal	long
integer	unsignedByte
nonNegativeInteger	unsignedShort
positiveInteger	unsignedInt
negativeInteger	unsignedLong
nonPositiveInteger	base64Binary
	hexBinary

Pre-Defined Types: Date and Time

Schema type	example
duration	P2Y4M7DT10H30M17.5S
date	2003-09-15
time	15:07:01
dateTime	2003-09-15T15:07:00
gYear	2003
gMonth	--02
gYearMonth	2003-09
gDay	---15
gMonthDay	--09-15

Task: Predefined Versus User-Defined Date

In the schema for the arrival card, define date of birth as:

1. pre-defined type
2. use-defined type with separate elements for:

    a) day
    b) month
    c) year

User-Defined Simple Types: Restriction

Given pre-defined simple types, we can derive new simple types.

This can be done by:

- restriction

```
<xsd:simpleType name="myType">
  <xsd:restriction base="simpleType">
    <facet1 value="..."/>
    <facet2 value="..."/>
    <facet3 value="..."/>
  </xsd:restriction>
</xsd:simpleType>
```

User-Defined Simple Types: List and Union

- list

```
<xsd:simpleType name="myType">
  <xsd:list itemType="simpleType">
</xsd:simpleType>
```

- union

```
<xsd:simpleType name="myType">
  <xsd:union memberTypes="simpleType1   simpleType2">
</xsd:simpleType>
```

Facets for Simple Type Restriction

XML Schema defines 12 constraining facets:

string types	numeric types
`length`	`minInclusive`
`minLength`	`maxInclusive`
`maxLength`	`minExclusive`
`pattern`	`maxExclusive`
`enumeration`	`totalDigits`
`whiteSpace`	`fractionDigits`

Facets: MinLength and MaxLength

Strings of 5 to 10 characters:

```
<xsd:simpleType name="string5to10">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="5"/>
    <xsd:maxLength value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Task: Restricted String Types

In the schema for the arrival card, define the travel document element with the number element.

Define the number element as the string of exactly 10 characters.

Facets: MinInclusive and MaxInclusive

Numbers in the range from 1 to 31:

```
<xsd:simpleType name="days">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="31"/>
  </xsd:restriction>
</xsd:simpleType>
```

Task: Restricted Numerical Types

In the schema for the arrival card, define

a) day
b) month
c) year

as restricted numerical types.

Facets: Enumeration

```
<xsd:simpleType name="weekDays">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Monday"/>
        <xsd:enumeration value="Tuesday"/>
        <xsd:enumeration value="Wednesday"/>
        <xsd:enumeration value="Thursday"/>
        <xsd:enumeration value="Friday"/>
        <xsd:enumeration value="Saturday"/>
        <xsd:enumeration value="Sunday"/>
    </xsd:restriction>
</xsd:simpleType>
```

Only `enumeration` and `pattern` can appear many times.

Task: Enumerated String Types

Inside schema for the arrival card, define the sex element inside the visitor element using string enumeration.

Facets: TotalDigits and FractionDigits

Price type with 2 fraction digits and 8 in total:

```
<xsd:simpleType name="Price">
    <xsd:restriction base="xsd:float">
        <xsd:totalDigits value="8"/>
        <xsd:fractionDigits value="2"/>
    </xsd:restriction>
</xsd:simpleType>
```

Facets: Pattern

Strings of upper-case or lower-case letters (not mixed):

```
<xsd:simpleType name="myName">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]+"/>
      <xsd:pattern value="[a-z]+"/>
   </xsd:restriction>
</xsd:simpleType>
```

The value of a pattern is a regular expression.

Facets: Regular Expressions for Patterns

`*`	zero or more
`+`	one or more
`?`	zero or one
`.`	any character
`(a|b)`	a or b
`(abc)`	sequence of a, b and c
`[abc]`	any or a, b or c
`[^a-z]`	not a letter in the range
`(expr){n}`	expr repeated exactly n times
`(expr){m,n}`	expr repeated from m to n times
`\p{X}`	one character from the Unicode character class X

Task: Patterns for String Types

Inside schema for the arrival card, define the travel document identifier following three constraints:

1. the identifier consists of 10 characters
2. the first two characters are letters
3. the last eight characters are digits

Declarations with Simple Types

- elements can have simple or complex types:

```
<xsd:element name="price" type="Price"/>
<xsd:simpleType name="Price">
    ...
</xsd:simpleType>
```

- attributes always have simple types:

```
<xsd:attribute name="price">
    <xsd:simpleType>
    ...
    </xsd:simpleType>
</xsd:attribute>
```

Attribute Declarations

Similar to element declarations, with `name`, `type` and `ref` attributes.

Attribute declaration is related to the parent element, it must follow the declarations of the element's children:

```
<xsd:complexType name="Collection">
    ...
    <xsd:element ref="book"/>
    <xsd:element ref="CD"/>
    <xsd:attribute name="version" type="xsd:string"/>
</xsd:complexType>
```

Only elements with complex types may contain attributes!

Top- and Local-Level Attributes

Like elements, attributes can be defined at top and local levels:

- top level attribute – referred by local-level attributes

```
<xsd:attribute name="version" type="xsd:string"/>
```

- local level attribute – content of its parent element:

```
<xsd:complexType name="Collection">
    ...
    <xsd:element ref="CD"/>
    <xsd:attribute ref="version"/>
</xsd:complexType>
```

Attributes and Types

- attribute with named type

```
<xsd:attribute name="version" type="xsd:string"/>
```

- attribute with anonymous type

```
<xsd:attribute name="versions">
    <xsd:simpleType>
        <xsd:list itemType="Version"/>
    </xsd:simpleType>
</xsd:attribute>
```

- local-level attribute referring to top-level attribute

```
<xsd:attribute ref="version"/>
```

Attribute Occurrence

The `use` attribute determines the attribute's occurrence:

- attribute is required:

```
<xsd:attribute name="version" use="required" .../>
```

- attribute is optional (default):

```
<xsd:attribute name="version" use="optional" .../>
```

- attribute is prohibited:

```
<xsd:attribute name="version" use="prohibited" .../>
```

Attribute Default and Fixed Values

Two more attributes:

- `fixed` - fixed value for the attribute:

```
<xsd:attribute name="version" fixed="1.0" .../>
```

- `default` - default value for the attribute:

```
<xsd:attribute name="version" default="1.0" .../>
```

If an element is missing, the defaults for its attributes are not supplied.

Complex Type Definition

Consists of element declarations/references embedded inside a content model, and attribute declarations:

```
<xsd:complexType name="...">
    <xsd:sequence>
        <xsd:element .../>
        <xsd:element .../>
    </xsd:sequence>
    <xsd:attribute .../>
    <xsd:attribute .../>
</xsd:complexType>
```

Content Models

Three content models:

- `sequence` – similar to `(a,b)` in DTD
- `choice` – similar to `a|b` in DTD
- `all` – does not occur in DTD

They can be nested within each other to any level, and may have `minOccurs` and `maxOccurs` attributes.

Sequence Content Model

Elements must occur exactly in the order indicated, and they all must occur (unless `minOccurs="0"` for individual elements):

```
<xsd:complexType>
    <xsd:sequence>
        <xsd:element name="day" type="day"/>
        <xsd:element name="month" type="month"/>
        <xsd:element name="year" type="xsd:integer"/>
        <xsd:element name="weekeday"
          minOccurs="0" type="weekDay"/>
    </xsd:sequence>
</xsd:complexType>
```

Choice Content Model

Exactly one of elements may occur; elements are mutually exclusive:

```
<xsd:complexType name="address">
    <xsd:sequence>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:choice minOccurs="0">
            <xsd:element name="state" type="xsd:string"/>
            <xsd:element name="province" type="xsd:string"/>
        </xsd:choice>
        <xsd:element ref="zip"/>
    </xsd:sequence>
</xsd:complexType>
```

All Content Model

This model does not occur in DTD, but it does occur in SGML.

All of the element may appear, in any order.

Limitations:

- each element may occur no more than once: `minOccurs` is zero or one, `maxOccurs` is one
- cannot contain `sequence` and `choice` models
- must occur as the only immediate child at the beginning of the content model, and occur no more than once

All Content Model

Address must contain: street, city, country and zip, and may contain state. Any order of elements is allowed:

```
<xsd:complexType name="address">
    <xsd:all>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="state"
            minOccurs="0" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string"/>
        <xsd:element ref="zip"/>
    </xsd:all>
</xsd:complexType>
```

Complex Types Again

Formally, there are two ways to define complex types:

- `simpleContent` – permits character data and attributes

```
<xsd:complexType>
    <xsd:simpleContent>...</xsd:simpleContent>
</xsd:complexType>
```

- `complexContent` – permits children elements and attributes; this is the default, so may be omitted

```
<xsd:complexType>
    <xsd:complexContent>...</xsd:complexContent>
</xsd:complexType>
```

Derivation of Complex Types

Four ways to derive a complex type from another type:

1. extension of any simple type
2. extension of another complex type
3. restriction of another complex type
4. restriction of the generic `anyType`

Adding Attributes

Adding `version` and `status` attributes (both `string`) to an element with simple content (`integer`):

```
<xsd:complexType>
    <xsd:simpleContent>
        <xsd:extension base="xsd:integer">
          <xsd:attribute name="version" type="xsd:string"/>
          <xsd:attribute name="status" type="xsd:string"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Adding Elements

Adding `new` and `more` elements to an existing `myType`, at the end of its children list:

```
<xsd:complexType>
    <xsd:complexContent>
        <xsd:extension base="myType">
            <xsd:sequence>
                <xsd:element name="new" type="xsd:string"/>
                <xsd:element name="more" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Removing Children Elements

When removing elements of an existing `myType`, we have to repeat all its children (modified or not), except those that are being removed:

```
<xsd:complexType>
    <xsd:complexContent>
        <xsd:restriction base="myType">
         <xsd:sequence>
          <xsd:element name="old" type="xsd:string"/>
          <xsd:element name="modified" type="xsd:integer"/>
         </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Empty Elements

- element without content/attributes:

```
<xsd:complexType name="empty"/>
```

- element without content but with attributes:

```
<xsd:complexType name="empty">
    <xsd:attribute name="price" type="xsd:integer"/>
    <xsd:attribute name="version" type="xsd:string"/>
</xsd:complexType>
```

Limiting Derivation

We can limit the derivation of types with the `final` attribute:

- extended but not restricted

```
<xsd:complexType final="extension">...<xsd:complexType>
```

- restricted but not extended

```
<xsd:complexType final="restriction">...<xsd:complexType>
```

- cannot be derived at all

```
<xsd:complexType final="#all">...<xsd:complexType>
```

Mixed Content Model

- element content consists of either sub-elements or character data, but not both (default):

```
<xsd:complexType mixed="false">...<xsd:complexType>
```

- element content is a mixture of sub-elements and character data:

```
<xsd:complexType mixed="true">...<xsd:complexType>
```

The order and number of elements in the mixed model is constrained by the schema, like in the non-mixed case; not possible for DTD.

Any Content Model

All simple and complex types are derived from `xsd:anyType`.

Restriction of `anyType` with complex content is the default:

```
<xsd:complexType>
    <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
            <xsd:sequence>
                <xsd:element name="..." .../>
                <xsd:element name="..." .../>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Element Groups: Declaration

Element group – a set of elements defined with a name:

```
<xsd:group name="myGroup">
    <xsd:sequence>
        <xsd:element ref="name"/>
        <xsd:element ref="scope"/>
    </xsd:sequence>
</xsd:group>
```

Must be the immediate child of `schema`, and may contain `sequence`, `choice` or `all` only.

Element Groups: Reference

Element group referenced in the complex type definition:

```
<xsd:complexType name="myType">
    <xsd:sequence>
        <xsd:group ref="myGroup"/>
        <xsd:element ref="price"/>
    </xsd:sequence>
</xsd:complexType>
```

Element groups play a similar role as parameter entities in DTD.

Attribute Groups: Declaration

Grouping attributes with `attributeGroup`:

```
<xsd:attributeGroup name="margins">
    <xsd:attribute name="top" type="xsd:float"/>
    <xsd:attribute name="bottom" type="xsd:float"/>
    <xsd:attribute name="left" type="xsd:float"/>
    <xsd:attribute name="right" type="xsd:float"/>
</xsd:attributeGroup>
```

Like `group`, `attributeGroup` is an immediate child of `schema`.

Attribute Groups: Reference

Referencing an attribute group by its name:

```
<xsd:complexType name="myType">
    <xsd:sequence>
        <xsd:group ref="myGroup"/>
        <xsd:element ref="price"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="margins"/>
</xsd:complexType>
```

They also play the role similar to DTD's parameter entities.

Referring to Schema: One Namespace 1

XML instance document with the default namespace `http://www.w3c.org` for all its elements:

```
<elem
  xmlns="http://www.w3c.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3c.org schemaNamespace.xsd">
  text
</elem>
```

The attribute `SchemaLocation` determines the schema location for the elements that belong to the specified namespace:

```
xsi:schemaLocation="namespace schema"
```

Referring to Schema: One Namespace 2

XML Schema file `schemaNamespace.xsd`:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3c.org">
  <xsd:element name="elem" type="xsd:string"/>
</xsd:schema>
```

The `TargetNamespace` attribute of `schema` determines the namespace this schema is used to validate.

Every schema is used to validate a single namespace!

Demo: One Namespace Validation

```
> cd "demos/schema one namespace"
> dir
schemaNamespace.xsd namespace.xml
> java dom.Counter -v -s namespace.xml
```

Referring to Schema: Two Namespaces 1

XML instance document with two namespaces
http://www.w3c.org/1 and http://www.w3c.org/2:

```
<ns1:outside
  xmlns:ns1="http://www.w3c.org/1"
  xmlns:ns2="http://www.w3c.org/2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
   http://www.w3c.org/1 schemaNamespace1.xsd
   http://www.w3c.org/2 schemaNamespace2.xsd">

  <ns2:inside>text</ns2:inside>

</ns1:outside>
```

Referring to Schema: Two Namespaces 2

The attribute `SchemaLocation` determines the schema locations for the elements that belong to specified namespaces:

```
xsi:schemaLocation="
    namespace1 schema1
    namespace2 schema2
    namespace3 schema3…"
```

Referring to Schema: Two Namespaces 3

The file `schemaNamespace1.xsd` used to validate the elements from the namespace `http://www.w3c.org/1`:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.iist.unu.edu/1">

  <xsd:element name="outside">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="http://www.iist.unu.edu/2"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Referring to Schema: Two Namespaces 4

The file `schemaNamespace2.xsd` used to validate the elements from the namespace `http://www.w3c.org/2`:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.iist.unu.edu/2">

  <xsd:element name="inside" type="xsd:string"/>

</xsd:schema>
```

Demo: Two Namespace Validation

```
> cd "demos/schema two namespaces"
> dir
schemaNamespace1.xsd schemaNamespace1.xsd
namespaces.xml
> java dom.Counter -v -s namespaces.xml
```

Instance Documents and Qualification

Should elements/attributes in an instance document be qualified?

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.iist.unu.edu/xml"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <xsd:element name="top">...</xsd:element>
    ...
</xsd:schema>
```

- `qualified` – they must be in the target namespace
- `unqualified` – they should not be in any namespace; default

Schema Modularisation

Two top-level elements permit the inclusion of an external schema:

- `import` – allows access to elements and type definitions from different namespaces

- `include` – the target namespace of the included schema must be the same as the target namespace of the including schema

They must occur prior to any other definitions in a schema.

DTD for XML Schema

As XML Schema documents apply XML syntax, they can be checked for validity. Fragment of the official XML Schema DTD:

```
          namespace prefix -> | <!ENTITY % p 'xs:'>
           namespace sufix -> | <!ENTITY % s ':xs'>
      namespace declaration -> | <!ENTITY % nds 'xmlns%s;'>
      schema qualified name -> | <!ENTITY % schema "%p;schema">
schema element declaration -> | <!ELEMENT %schema;
                           -> |  (%simpleType; | %complexType; |
                           -> |   %element; | %attribute; | ...)*>
   schema attributes         | <!ATTLIST %schema;
                    version -> |   version CDATA #IMPLIED
           schema namespace -> |   %nds; %URIref; #FIXED '...XMLSchema'
         default namespace -> |   xmlns CDATA #IMPLIED
         unique identifier -> |   id ID #IMPLIED
          other attributes -> |   %schemaAttrs;>
```

DTD Validation of XML Schema Documents

Validation statement:

> Any XML document which is not valid per this DTD given redefinitions in its internal subset of the ’p’ and ’s’ parameter entities appropriate to its namespace declaration of the XML Schema namespace is almost certainly not a valid schema.

XPath

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

XPath

A language to address parts of the XML document.

Apart from this main goal, it serves additional goals:

1. manipulation of text, numbers and logical values
2. formulating patterns for document tree nodes
3. checking if a node satisfies a pattern

Intended for the use by XSLT and XPointer.

W3C Recommendation November 1999.

XPath Features

XPath provides a compact non-XML notation.

XPath operates on the logical tree structures of an XML document.

Its name originates from the path notation to navigate the hierarchical tree structures of XML documents.

Evaluating XPath Expressions

Evaluating an XPath expression returns one of four kinds of objects:

1. set of nodes
2. boolean (true or false)
3. number (floating point)
4. string (list of UCS characters)

Evaluation Context

Expression is evaluated with respect to the context:

1. context node
2. a pair of positive integers: position and size of the context
3. values of the variables; one of the four types
4. function library with arguments are results of the four types
5. namespace declarations

During evaluation:

- variables, functions and namespaces remain unchanged
- context node, position and size may change

Task: Create XML Document

Create `collection.xml`:

```
<?xml version="1.0"?>
<collection>
  <book>
    <title>Learning XML</title>
    <review>4</review>
  </book>
  <book id="abc">
    <title>Java and XML</title>
  </book>
  <article>
    <title>XML and Semantic Web</title>
    <journal><title>CACM</title></journal>
  </article>
</collection>
```

Task: Create XSLT Document

Create `collection.xsl`:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:for-each select="CONTEXT">
    <xsl:value-of select="XPATH"/>
    <xsl:text> </xsl:text>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Task: XPath Experiments

Use the values of `CONTEXT` and `XPATH` to test XPath expressions.

For instance:

```
CONTEXT = /
XPATH   = .
```

Run:

```
> java org.apache.xalan.xslt.Process
   -in collection.xml
   -xsl collection.xsl
```

Relative Paths

The simplest path is the reference to an element name.

Let:

```
CONTEXT = collection
XPATH   = .
```

Multiple Steps

A path with two steps:

```
collection/article
```

Let:

```
CONTEXT = collection/article
XPATH   = .
```

Wildcard Steps

Any element may occur between `collection` and `title`:

```
collection/*/title
```

Let:

```
CONTEXT = collection/*/title
XPATH   = .
```

Descendants Selection

Any number of elements may occur between `collection` and `title`:

```
collection//title
```

Let:

```
CONTEXT = collection//title
XPATH   = .
```

Self, Parents and Grandparents

Self node: `.`

Parent node: `..`

Grandparents: `../..`

Let:

```
CONTEXT = collection/article
XPATH   = .
XPATH   = ../book
XPATH   = ../..
```

Absolute Path

`/collection/article` selects `article` from every context node.

Let:

```
CONTEXT = collection/book
XPATH   = /collection/article
```

`//title` selects all `title` elements in the document:

Let:

```
CONTEXT = //title
XPATH   = .
```

Predicates

Location paths are indiscriminate in node selection.

Predicate:

```
book[…]/title
```

can be used to quality any step in the path.

Position Tests

Selects the second book:

```
para[position()=2]
```

Let:

```
CONTEXT = collection/book[position()=2]
XPATH   = .
```

Presence Tests

Selects a books with a review element:

```
collection/book[review]
```

Let:

```
CONTEXT = collection/book[review]
XPATH   = .
```

Value Tests

Selects a books with a given title:

```
collection/book[title='Java and XML']
```

Let:

```
CONTEXT = collection/book[title='Java and XML']
XPATH   = .
```

Attribute Presence Test

Selects `book` if it contains an attribute `id`:

```
book[@id]
```

Let:

```
CONTEXT = //book[@id]
XPATH   = .
```

Attribute Value Test

Selects `book` if it contains an attribute `id` which has the value `"abc"`:

```
book[@id="abc"]
```

Let:

```
CONTEXT = //book[@id="abc"]
XPATH   = .
```

Boolean Tests: Negation

Selects `book` if it does not contain an attribute `id`:

```
book[not(@id)]
```

Let:

```
CONTEXT = //book[not(@id)]
XPATH   = .
```

Boolean Tests: Disjunction

Selects the element `book` if it does not contain an attribute `id` or occurs on the last position:

```
book[not(@id) or position()=2]
```

Let:

```
CONTEXT = //book[not(@id) or position()=last()]
XPATH   = .
```

String Calculation: Contains

Selects the element `book` which `title` contains `'Java'`:

```
//book/title[contains(text(),'Java')]
```

Let:

```
CONTEXT = //book/title[contains(text(),'Java')]
XPATH   = .
```

String Calculation: Substring

Selects the element `book` which `title` contains `'Java'`:

```
//book/title[contains(text(),'Java')]
```

Displays the first four characters of the title:

```
substring(.,1,4)
```

Let:

```
CONTEXT = //book/title[contains(text(),'Java')]
XPATH   = substring(.,1,4)
```

Multiple Predicates

Predicates can be combined.

First select all books with the `id` attribute, then among them all books with the "`Java and XML`" title:

```
//book[@id][title='Java and XML']
```

Let:

```
CONTEXT = //book[@id][title='Java and XML']
XPATH   = .
```

XSLT

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) tree-based programming (DOM)
   b) event-based programming (SAX)
   c) rule-based programming (XSLT)

XSL

XSL = eXtensible Stylesheet Langugage

Consists of three parts:

- XSLT – an XML language for transforming XML documents
- XSL-FO – an XML language for formatting semantics
- XPath – a non-XML syntax for addressing parts of an XML document. Also used in XLink, XPointer and XQuery.

XSLT

- XSLT = eXtensible Stylesheet Language Transformations
- XSLT is a fully-fledged declarative programming language specializing in XML transformations
- XSLT programs are themselves written in XML

XSLT

- XSLT = eXtensible Stylesheet Language Transformations
- XSLT is a fully-fledged declarative programming language specializing in XML transformations
- XSLT programs are themselves written in XML

Example: XML Input

```
<?xml version="1.0"?>
<greeting>
   Hello World!
</greeting>
```

Example: XSLT Program

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>

    <xsl:template match="/">
        <xsl:apply-templates select="greeting"/>
    </xsl:template>

    <xsl:template match="greeting">
        <html>
            <body>
                <h1><xsl:value-of select="."/></h1>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Example: Invocation

```
java org.apache.xalan.xslt.Process
    -in greeting.xml
    -xsl greeting.xsl
    -out greeting.html
```

Example: HTML Output

```
<html>
    <body>
        <h1>
            Hello World!
        </h1>
    </body>
</html>
```

XSLT Document Structure

Root element and namespace declaration:

```
<xsl:stylesheet
    version="1.0"
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
    …
</xsl:stylesheet>
```

Output Format 1

What is the format of the output document?

```
<xsl:stylesheet version="1.0"
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>

    <xsl:output method="html"/>

    …
</xsl:stylesheet>
```

Possibilities: text, HTML, XML.

Output Format 2

Each output method provides its own formatting attributes.

For instance:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>

    <xsl:output method="html" ident="yes"/>

    …
</xsl:stylesheet>
```

XSLT Processing

XSLT program describe a set of transformation rules from the input document to the output document.

Transformation is defined with:

1. templates matched against the input elements
2. patterns describing fragments of the output document

XSLT Templates

General format of templates:

```
<xsl:template match="xpath-expr">
    …
</xsl:template>
```

The value of the match attribute is the XPath expression.

The template matching the root element only:

```
<xsl:template match="/">
    …
</xsl:template>
```

Applying XSLT Templates

The template calling recursively other templates (perhaps itself) given `greeting` as the current element:

```
<xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
</xsl:template>
```

Templates with Text

The only matching template is this:

```
<xsl:template match="greeting">
    <html>
        <body>
            <h1>
                <xsl:value-of select="."/>
            </h1>
        </body>
    </html>
</xsl:template>
```

Produces text on output (HTML markup).

Referring to the Current Element

The content of the current element (greeting) is sent to output:

```
<xsl:template match="greeting">
    <html>
        <body>
            <h1>
                <xsl:value-of select="."/>
            </h1>
        </body>
    </html>
</xsl:template>
```

Template Processing

Let the current context contain the root element.

1) are there any nodes to process in the current context?

2) for every node in the context:

   a) are there any templates matching the node?
      i. if there are several templates choose the most specific
      ii. if there is no template, choose the default template

   b) invoke the template recursively for the next context

Default Templates 1

Applied when lacking specific templates:

1. ensures continued processing even if there is no template for a given element:

```
<xsl:template  match="*|/">
    <xsl:apply-templates/>
</xsl:template>
```

Default Templates 2

2. the elements are processed regardless of the mode:

```
<xsl:template match="*|/" mode="x">
   <xsl:apply-templates mode="x"/>
</xsl:template>
```

3. text and attributes of selected elements are send to output:

```
<xsl:template match="text()|@*">
   <xsl:value-of select="."/>
</xsl:template>
```

Default Templates 3

4. comment and processing instruction nodes are not processed:

```
<xsl:template
    match="comment() |
    processing-instruction()"/>
```

5. namespace nodes are not processed:

```
<xsl:template match="namespace()"/>
```

Template Attributes

The template must contain one of the following attributes:

1. `match` – the template invoked when matching elements are found:

```
<xsl:template match="...">...</xsl:template>
```

2. `name` – the template is invoked by name:

```
<xsl:template name="...">...</xsl:template>
```

Calling Template by Name

Definition of the named template:

```
<xsl:template name="myName">...</xsl:template>
```

Calling the template:

```
<xsl:template ...>
   <xsl:call-template name="myName"/>
</xsl:template>
```

Example: Named Template

```
<xsl:template match="root">
  <xsl:call-template name="myName"/>
  <xsl:call-template name="myName"/>
</xsl:template>

<xsl:template name="myName">
  <xsl:text>text</xsl:text>
</xsl:template>
```

Matched Template

Definition of the matched template:

```
<xsl:template match="elem">...</xsl:template>
```

Invoking the template:

1. applies to all children of the current node:

```
<xsl:apply-templates/>
```

2. selects the nodes to which the template applies:

```
<xsl:apply-templates select="..."/>
```

Example: Matched Template

```
<xsl:template match="root">
  <xsl:apply-templates/>
  <xsl:apply-templates select="@att"/>
</xsl:template>

<xsl:template match="elem">
  <xsl:text>elementy</xsl:text>
</xsl:template>

<xsl:template name="@att">
  <xsl:text>atrybuty</xsl:text>
</xsl:template>
```

Templates with Mode

The template may contain the mode attribute that allows to process the same set of nodes several times.

Definition of the template with mode:

```
<xsl:template match="..." mode="...">
    ...
</xsl:template>
```

Invoking the mode template:

```
<xsl:apply-templates select="..." mode="...">
```

Example: Templates with Mode

```
<xsl:template match="root">
  <xsl:apply-templates mode="mode1"/>
  <xsl:apply-templates mode="mode2"/>
</xsl:template>

<xsl:template match="elem" mode="mode1">
  <xsl:text>mode1:</xsl:text>
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="elem" mode="mode2">
  <xsl:text>mode2:</xsl:text>
  <xsl:value-of select="."/>
</xsl:template>
```

Task: Simplify Credit Card Letter

From document-oriented to data-oriented XML:

```
<?xml version="1.0"?>
<letter>
    <customer>Simon White</customer>
    <product>credit card</product>
    <officer level="manager">Steven Rod</officer>
    <enclosure>credit card</enclosure>
    <enclosure>initial PIN</enclosure>
</letter>
```

Task: Create the Transformation Stylesheet

Transformation to generate the acceptance letter:

```
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:template match="letter">
        Dear <xsl:value-of select="customer"/>,
        We are pleased to inform you that your
        <xsl:value-of select="product"/> application
        has been accepted.
        Sincerely, <xsl:value-of select="officer"/>
    </xsl:template>
</xsl:stylesheet>
```

Task: Run the Transformation

Execute `xalan`:

```
> xalan letter.xml letter.xsl output
```

What is the result?

Task: Change Default Output Format

```
<?xml version="1.0"?>
<xsl:stylesheet ...>

    <xsl:output method="text"/>

    <xsl:template match="letter">...</xsl:template>

</xsl:stylesheet>
```

Run `xalan`. Note the change.

Task: Modify the Stylesheet

Letter template:

```
<xsl:template match="letter">
   Dear <xsl:value-of select="customer"/>,
   <xsl:apply-templates select="product"/>
   Sincerely, <xsl:value-of select="officer"/>
</xsl:template>
```

Product template:

```
<xsl:template match="product">
   We are pleased to inform you that your
   <xsl:value-of select="."/> application
   has been accepted.
</xsl:template>
```

Task: Run the Transformation

Execute `xalan`:

```
> xalan letter.xml letter.xsl output
```

Any change?

Task: Add Officer's Level

Refer to the officer's level attribute:

```
<xsl:template match="letter">
    Dear <xsl:value-of select="customer"/>,
    <xsl:apply-templates select="product"/>
    Sincerely, <xsl:value-of select="officer"/>
    (<xsl:value-of select="officer/@level"/>)
</xsl:template>
```

Run `xalan`.

Which Template?

There is no template for a given context:

```
<xsl:template match="root">...</xsl:template>
<xsl:template match="sub">...</xsl:template>

<root att=“value">
    <elem><sub>text1</sub></elem>
</root>
```

Apply the default template.

Which Template?

There is exactly one template for a given context:

```
<xsl:template match="root">...</xsl:template>
<xsl:template match="elem">...</xsl:template>

<root att="value">
    <elem><sub>text1</sub></elem>
</root>
```

Apply this template.

Which Template?

There are two templates for a given context.

```
<xsl:template match="root">...</xsl:template>
<xsl:template match="sub">...</xsl:template>
<xsl:template match="elem/sub">...</xsl:template>

<root att="value">
   <elem><sub>text1</sub></elem>
</root>
```

Apply the more specific template.

Which Template?

There are two equally-specific templates for a given context:

```
<xsl:template match="elem" priority="1">
  <xsl:text>first</xsl:text>
</xsl:template>

<xsl:template match="elem" priority="0">
  <xsl:text>second</xsl:text>
</xsl:template>
```

Apply the template with the higher priority.

Which Template?

There are two equally-specific template for a given context and with equal priorities:

```
<xsl:template match="elem" priority="0">
  <xsl:text>pierwszy</xsl:text>
</xsl:template>

<xsl:template match="elem" priority="0">
  <xsl:text>drugi</xsl:text>
</xsl:template>
```

Apply the template that occurs later in the stylesheet.

Template with Parameters

Template definition with two parameters:

```
<xsl:template name="area">
  <xsl:param name="height"/>
  <xsl:param name="width"/>
  <xsl:value-of select="$height * $width"/>
</xsl:template>
```

Calling the template with concrete arguments:

```
<xsl:call-template name="area">
  <xsl:with-param name="height" select="10"/>
  <xsl:with-param name="width" select="20"/>
</xsl:call-template>
```

Default Parameters

Template with default values of the parameters:

```
<xsl:template name="area">
  <xsl:param name="height">10</xsl:param>
  <xsl:param name="width">20</xsl:param>
  <xsl:value-of select="$height * $width"/>
</xsl:template>
```

Calling the template with one parameter:

```
<xsl:call-template name="area">
  <xsl:with-param name="height" select="30"/>
</xsl:call-template>
```

Default Parameters with Choice

Template with parameters which default values are associated with the input document:

```
<xsl:template name="area">
  <xsl:param name="height" select="hi"/>
  <xsl:param name="width" select="wi"/>
  <xsl:value-of select="$height * $width"/>
</xsl:template>
```

Calling the template without parameters:

```
<xsl:call-template name="area"/>
```

Global Parameters

Parameters which scope expands the whole stylesheet.

They occur as the children of the `stylesheet` element:

```
<xsl:stylesheet ...>
    <xsl:param name="depth">30</xsl:param>
    ...
</xsl:stylesheet>
```

Default values like for local parameters.

Use of Global Parameters

Using the global parameter in the template:

```
<xsl:template name="volume">
  <xsl:param name="height" select="height"/>
  <xsl:param name="width" select="width"/>
  <xsl:value-of select="$height * $width* $depth"/>
</xsl:template>
```

Global Parameter Assignment

Global parameters can be assigned value in the command line.

For instance:

```
java org.apache.xalan.xslt.Process
    -in file.xml
    -xsl file.xsl
    -out file.out
    -param depth 50
```

For the depth parameter:

```
<xsl:param name="gleb"/>
```

Variables

Variable declarations:

1. name `x` and empty value:

```
<xsl:variable name="x"/>
```

2. name `x` and value `test`:

```
<xsl:variable name="x" select="'test'"/>
```

3. name `x` and value of the `test` element:

```
<xsl:variable name="x" select="test"/>
```

Example: Variables

The value of the `depth` variable depends on the choice expression that refers to the `depth` element:

```
<xsl:template name="depth">
  <xsl:param name="heigth" select="height"/>
  <xsl:param name="width" select="width"/>
  <xsl:variable name="depth">
    <xsl:choose>
      <xsl:when test="depth = 'one'">1</xsl:when>
      <xsl:when test="depth = 'two'">2</xsl:when>
      <xsl:otherwise>3</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:value-of select="$height * $width * $depth"/>
</xsl:template>
```

Variable Scope

Visible in the element in which it is declared:

1. local variable

```
<xsl:template name="...">
  <xsl:variable name="...">...</xsl:variable>
</xsl:template>
```

2. global variable

```
<xsl:stylesheet ...>
   <xsl:variable name="...">...</xsl:variable>
   ...
</xsl:template>
```

Variable or Constant?

Constant.

The values of variables cannot be modified.

Conditional Execution

Element `if`:

```
<xsl:if test="...">
   ...
</xsl:if>
```

If the calculated value of the test attribute is:

1. `true` – the content of the element is processed
2. `false` – the content of the element is ignored

Value of the test Attribute

The value is converted to `boolean`:

1. number – if zero or NaN (non-number) then false, otherwise true
2. node-set – if empty then false, otherwise true
3. string – if empty then false, otherwise true

Example: test Attributes

```
<xsl:if test="count(elem) &gt;=2">
<xsl:if test="$x">
<xsl:if test="true()">
<xsl:if test="true">
<xsl:if test="'true'">
<xsl:if test="'false'">
<xsl:if test="not(3)">
<xsl:if test="section/section">
```

Example: Conditional Execution

```
<xsl:if test="count(elem) = 2  and @att">
   <xsl:text>condition fulfilled</xsl:text>
</xsl:if>

<root att="wartosc">
   <elem>text1</elem>
   <elem>text2</elem>
</root>
```

Choice Execution

At least one `when`, optional `otherwise`:

```
<xsl:choose>
    <xsl:when test="...">...</xsl:when>
    ...
    <xsl:when test="...">...</xsl:when>
    <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

The first element with the `test` attribute returning true is processed.

If one does not exist, the content of `otherwise` is processed.

Example: Choice Execution

```
<xsl:choose>
    <xsl:when test="not(@att)">wybor 1</xsl:when>
    <xsl:when test="contains(elem[1],elem[2])">
        wybor 2
    </xsl:when>
    <xsl:otherwise>wybor 3</xsl:otherwise>
</xsl:choose>
```

```
<root att="wartosc">
    <elem>maly przyklad</elem>
    <elem>przyklad</elem>
</root>
```

Task: Make Decision Explicit

Modify XML:

```
<?xml version="1.0"?>
<letter decision="accepted">...</letter>
```

Task: One Template for Each Decision

Accept:

```
<xsl:template name="accepted">
    We are pleased to inform you that your
    <xsl:value-of select="product"/> application
    has been accepted.
</xsl:template>
```

Reject:

```
<xsl:template name="rejected">
    We are sorry to inform you that your
    <xsl:value-of select="product"/> application
    has been rejected.
</xsl:template>
```

Task: Conditionally Call Templates

Choose the template depending on the `decision` attribute:

```
<xsl:choose>
    <xsl:when test="@decision='accepted'">
        <xsl:call-template name="accepted"/>
    </xsl:when>
    <xsl:when test="@decision='rejected'">
        <xsl:call-template name="rejected"/>
    </xsl:when>
</xsl:choose>
```

Task: Run Xalan for Each Case

- for accept

```
<?xml version="1.0"?>
<letter decision="accepted">...</letter>
```

- for reject

```
<?xml version="1.0"?>
<letter decision="rejected">...</letter>
```

Task: Modify Decision Templates

Make the decision explicit:

- accepted template

```
<xsl:template name="accepted">
    We are pleased ... application has been
    <xsl:value-of select="@decision"/>.
</xsl:template>
```

- reject template

```
<xsl:template name="rejected">
    We are sorry ... application has been
    <xsl:value-of select="@decision"/>.
</xsl:template>
```

Task: Unify Decision Templates

Make a `body` template with `polite` parameter:

```
<xsl:template name="body">
    <xsl:param name="polite"/>
    We are <xsl:value-of select="polite"/>
    to inform you that your
    <xsl:value-of select="product"/> application
    has been <xsl:value-of select="@decision"/>.
</xsl:template>
```

Task: Call Template with Parameter

```
<xsl:choose>
    <xsl:when test="@decision='accepted'">
        <xsl:call-template name="body">
            <xsl:with-param name="polite">pleased</xsl:with-param>
        </xsl:call-template>
    </xsl:when>
    <xsl:when test="@decision='rejected'">
        <xsl:call-template name="body">
            <xsl:with-param name="polite">sorry</xsl:with-param>
        </xsl:call-template>
    </xsl:when>
</xsl:choose>
```

Run `xalan` in both cases.

Task: Remove Second Test

```
<xsl:choose>
    <xsl:when test="@decision='accepted'">
        ...
    </xsl:when>
    <xsl:otherwise>
        ...
    </xsl:otherwise>
</xsl:choose>
```

Run `xalan` in both cases.

Iterative Execution

Selects the set of node, then processes each of them in order selecting every time the new current node:

```
<xsl:for-each select="...">
    ...
</xsl:for-each>
```

It may initially contain several sort elements which are ordering the set before processing.

Example: Iterative Execution

```
<xsl:for-each select="elem">
   <xsl:value-of select="substring(.,1,5)"/>
</xsl:for-each>
```

```
<root att="wartosc">
    <elem>small example</elem>
    <elem>big example</elem>
</root>
```

Sorting

Sorting the set of node after selection by `apply-templates`:

```
<xsl:template match="root">
  <xsl:apply-templates select="elem">
    <xsl:sort/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="elem">
  <xsl:value-of select="."/>
</xsl:template>
```

Also a child of `for-each`.

Sorting Order

Inverse sorting order:

```
<xsl:template match="root">
  <xsl:apply-templates select="elem">
    <xsl:sort order="descending"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="elem">
  <xsl:value-of select="."/>
</xsl:template>
```

Task: Generate a List of Enclosures

Use a `for-each` loop to process each `enclosure` element:

```
<xsl:template match="letter">
    ...
    Enclosures:
    <xsl:for-each select="enclosure">
        <xsl:sort/>
        <xsl:value-of select="position()"/>.
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
    </xsl:for-each>
</xsl:template>
```

Run `xalan`.

Task: Sort the Enclosures

Use a `sort` element to sort the enclosures in ascending lexicographic order:

```
<xsl:template match="letter">
    ...
    Enclosures:
    <xsl:for-each select="enclosure">
        <xsl:sort/>
        ...
    </xsl:for-each>
</xsl:template>
```

Run `xalan`.

Task: Calculate the Number of Enclosures

Use the XPath `count` function:

```
<xsl:template match="letter">
    ...
    Attached, please see
    <xsl:value-of select="count(enclosure)"/>
    enclosures.
    ...
</xsl:template>
```

Run `xalan`.

Creating New Nodes

1. elements
2. attributes
3. text
4. processing instructions

Creating Elements 1

A template creating an element with name `elem` and the value equal that of the attribute `att`:

```
<xsl:template match="root">
  <xsl:element name="elem">
    <xsl:value-of select="@att"/>
  </xsl:element>
</xsl:template>
```

Creating Elements 2

A template creating an element with the name equal to the value of the attribute `att` and the value equal to the name of that attribute:

```
<xsl:template match="root">
  <xsl:element name="{@att}">
    <xsl:value-of select="name(@att)"/>
  </xsl:element>
</xsl:template>
```

Creating Attributes

A template creating an element `root` with attribute `att` which value is the content of the element `elem`:

```
<xsl:template match="root">
  <root>
    <xsl:attribute name="att">
      <xsl:value-of select="elem"/>
    </xsl:attribute>
  </root>
</xsl:template>
```

SAX

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) event-based programming (SAX)
   b) tree-based programming (DOM)
   c) rule-based programming (XSLT)

XML Programming Models

Three programming models:

- based on events – SAX
- based on trees – DOM
- based on templates – XSLT

How to program applications using XML syntax?

XML Processing Applications

The application has a built-in XML parser:

- the parser is rarely implemented, more often used off-the-shelf
- the parser processes XML syntax in various phases of the application's execution

The application and the parser are both using the same model (API) and representation of the input XML file.

Here we concentrate on the XML API for Java.

Java APIs for XML 1

Five different interfaces:

- JAXP – Java API for XML Processing

  Programming XML applications in Java using SAX, DOM and XSLT programming models.

2) JAXB – Java Architecture for XML Binding

   Writing Java objects in XML (marshalling), converting XML back to Java (unmarshalling)

3) JAXR – Java API for XML Registries

   Recording available services in an external registry, looking up the services in the registry.

Java APIs for XML 2

4) JAXM – Java API for XML Messaging

   Asynchronous exchange mechanism (send and forget) for XML messages exchanged between applications.

5) JAX-RPC – Java API for XML RPC

   Synchronous exchange mechanism (send and wait for reply) for XML messages exchanged between applications.

Here we describe JAXP – Java API for XML Processing.

Java API for XML Processing

JAXP is available in the package `javax.xml.parsers`.

Two abstract classes are contained in the package:

1) `SAXParserFactory` – enables the applications to create and configure a SAX parser
2) `DocumentBuilderFactory` – enables the applications to create and configure the DOM parser

Replacing API Implementations

Factory classes allow to replace parser implementations without the need to change the application's source code.

The implementation used depends on the setting of the properties:

1) `javax.xml.parsers.SAXParserFactory`
2) `javax.xml.parsers.DocumentBuilderFactory`

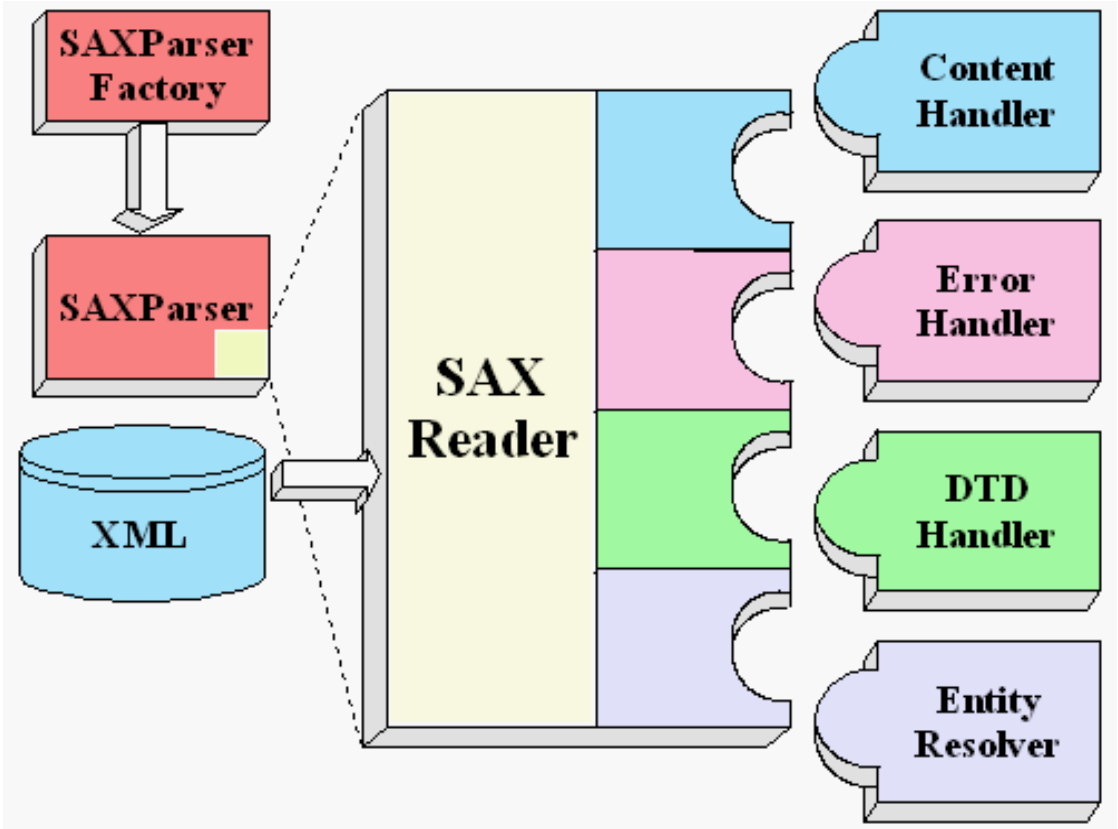
SAX API

Simple API for XML.

A mechanism for processing XML documents element after element - serial and event-driven.

Most often used in server applications which are subject to the high performance requirements.

SAX API Architecture



SAX API – Operation

1) the object of the factory class `SAXParserFactory` class creates the parser – object of the `SAXParser` class

2) the parser encapsulates the SAXReader object which is used to read the input XML document

3) during parsing, SAXReader invokes the methods that belong to the following four interfaces:
   a) `ContentHandler`
   b) `ErrorHandler`
   c) `DTDHandler`
   d) `EntityResolver`

4) Those methods are realized by the application.

Create a SAX Parser Factory

Create an object of the SAX parser factory class:

```
SAXParserFactory factory =
   SAXParserFactory.newInstance();
```

The `newInstance()` method is:

```
public static SAXParserFactory newInstance()
   throws FactoryConfigurationError
```

It raises an error when there is no implementation.

Create a SAX Parser

Create the SAX parser through the new factory object:

```
SAXParser saxParser = factory.newSAXParser();
```

according to the current parameters set for the factory object.

The `newSAXParser()` method is:

```
public abstract SAXParser newSAXParser()
    throws ParserConfigurationException, SAXException
```

The parser generator raises an exception if the factory does not support the required combination of the parser features.

Configuring the SAX Parser Factory

1. the parser handles namespaces

```
void setNamespaceAware(boolean awareness)
boolean isNamespaceAware()
```

2. the parser validates documents

```
void setValidating(boolean validating)
boolean isValidating()
```

Parsing XML Documents

The `SAXParser` class enables parsing of XML documents that originate from different sources:

```
void parse(java.io.File f, ...)
void parse(java.io.InputStream is, ...)
void parse(java.lang.String uri, ...)
void parse(InputSource is, ...)
```

`InputSource` helps decide how the XML document should be read by the parser: as character stream, byte stream or the URL-addressed file.

Handling SAX Events

The second argument of the parse method is the object for handling events generated during parsing:

```
void parse(..., DefaultHandler dh)
```

The `DefaultHandler` class includes default implementations for the event-handling methods, declared by the interfaces:

1. `EntityResolver`
2. `DTDHandler`
3. `ContentHandler`
4. `ErrorHandler`

Entity-Resolving Events

Interface `EntityResolver`.

The parser will invoke this method before opening any external entity:

```
public InputSource
   resolveEntity(String publicId, String systemId)
    throws SAXException, java.io.IOException
```

where:

1. `publicId` is the Formal Public Identifier of the external entity, if one exists, otherwise `null`
2. `systemId` – is the system identifier of the external entity

Error-Handling Events

Interface `ErrorHandler`. Reporting errors:

1. reporting a warning
   ```
   void warning(SAXParseException exc)
   ```

2. reporting a non-critical error
   ```
   void error(SAXParseException exc)
   ```

3. reporting a fatal error
   ```
   void fatalError(SAXParseException exc)
   ```

The SAX parser is required to use those interfaces for reporting errors or warnings related to XML document processing, not exceptions.

This limitation does not apply to applications.

DTD-Handling Events

Interface `DTDHandler`. Events related to DTD processing:

Encountering notation declaration:

```
void notationDecl(
   String name, String publicId, String systemId)
```

Encountering unparsed entity declaration:

```
void unparsedEntityDecl(
   String name, String publicId, String systemId,
   java.lang.String notationName)
```

Content-Handling Events

Interface `ContentHandler`.

Handling events informing about the logical content of the document.

The main interface implemented by SAX applications.

If an application wants to be informed about events generated during document parsing, then it:

1. implements this interface
2. registers the implementation with the SAX parser using the `setContentHandler` method

Types of Content-Handling Events

Events informing about various kinds of content:

- informing about the document beginning
- informing about the document end
- informing about character data
- informing about ignorable white characters
- informing about the start of an element
- informing about the end of an element
- informing about the entry to and exit from a new namespace
- informing about processing instructions
- informing about ignorable entity

Content-Handling: Document Start/End

Informing about the start of the document:

```
void startDocument()
```

Informing about the end of the document:

```
void endDocument()
```

Content-Handling: Character Data

Informing about encountered character data:

```
void characters(char[] ch, int start, int length)
```

where

1. `ch` – character data of the document
2. `start` – initial table index
3. `length` – table length

Content-Handling: Ignorable Whitespace

Informing about encountered ignorable whitespaces:

```
void ignorableWhitespace(
    char[] ch, int start, int length)
```

where

1. `ch` – character data of the document
2. `start` – initial table index
3. `length` – table length

Content-Handling: Element Start 1

Informing about the start of an element:

```
public void startElement(
   String namespaceURI, String localName,
   String qName, Attributes atts)
```

where

1. `String namespaceURI` – URI of the element's namespace
2. `String localName` – local name (without prefix)

   Required when `http://xml.org/sax/features/namespaces` (system property) is true; default case.

Content-Handling: Element Start 2

More parameters:

3. `String qName` – the element's qualified name (with prefix).

   Optional when `http://xml.org/sax/features/namespace-prefixes` (system property) is false; default case.

4. `atts` – the attributes of the element; only the attributes with values given directly (not `#IMPLIED`) are included.

   This includes namespace declarations (`xmlns:*`) given the system property `http://xml.org/sax/features/namespace-prefixes`.

Content-Handling: Element End

Informing about the end of an element:

```
void endElement(
   String namespaceURI, String localName, String qName)
```

The end-tag event is also called for the empty element.

Content-Handling: Namespace Begin/End

Informing about the beginning of the namespace, occurs just before the corresponding `startElement`:

```
public void startPrefixMapping(
    String prefix, String uri)
```

Informing about the end of the namespace, occurs just after the corresponding `endElement`:

```
public void startPrefixMapping(
    String prefix, String uri)
```

Content-Handling: Processing Instruction

Informing about processing instruction:

```
public void processingInstruction(
    String target, String data)
```

Content-Handling: Ignorable Entity

Informing about encountering an ignorable entity:

```
public void skippedEntity(String name)
```

Non-validating parsers are allowed to ignore entities when they did not see their declarations (e.g. in external DTD).

Both validating and non-validating parsers may ignore external entities depending on the system properties:

```
http://xml.org/sax/features/external-general-entities
http://xml.org/sax/features/external-parameter-entities
```

Registering Event Handlers 1

Methods for registering/retrieving event handlers:

- content handler:

```
ContentHandler getContentHandler()
void setContentHandler(ContentHandler handler)
```

2. DTD handler

```
DTDHandler getDTDHandler()
void setDTDHandler(DTDHandler handler)
```

Registering Event Handlers 2

3. entity resolver

```
EntityResolver getEntityResolver()
void setEntityResolver(EntityResolver resolver)
```

4. error handler

```
ErrorHandler  getErrorHandler()
void setErrorHandler(ErrorHandler handler)
```

An application may register a new event handler in the middle of parsing a document. The parser would switch immediately.

Packages for SAX

Where is this all located?

- input and output:

```
import java.io.*;
```

2. all interfaces of SAX parsers:

```
import org.xml.sax.*;
```

3. handling of parser-generated events:

```
import org.xml.sax.helpers.DefaultHandler;
```

Packages 2

4. creating the SAX parser:

```
import javax.xml.parsers.SAXParserFactory;
```

5. parser-generation exception:

```
import javax.xml.parsers.ParserConfigurationException;
```

6. the SAX parser:

```
import javax.xml.parsers.SAXParser
```

Application Skeleton

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser

public class App {
   public static void main(String argv[]) {
       …
   }
}
```

Event Handling

The application must be able to catch and handle events issued by the XML parser about the encountered XML document content.

Implement all four interfaces `EntityResolver`, `DTDHandler`, `ErrorHandler`, `ContentHandler`? No.

Instead:

1. Inherit the `DefaultHandler` class that provides empty methods implementing all four interfaces.
2. Implement within `App` the handlers to those events that require a specific response.

Application Skeleton Revisited

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser

public class App extends DefaultHandler {
   public static void main(String argv[]) {…}
    …
    public void startElement(…){…}
    public void endElement(…){…}
    public void characters(…){…}
    …
   }
}
```

Main Method 1

```
public static void main(String argv[]) {
```

Check if there is a command-line argument:

```
if (argv.length != 1) {
 System.err.println("Usage: cmd filename");
 System.exit(1);
}
```

Main Method 2

The current class provides handling of the SAX events:

```
DefaultHandler handler = new App();
```

Create the SAX parser factory object:

```
SAXParserFactory factory =
    SAXParserFactory.newInstance();
```

Main Method 3

```
    try {
```

Obtain the parser from the factory object:

```
        SAXParser saxParser = factory.newSAXParser();
```

Invoke the parser passing on the input document and the object of the current class to handle the events:

```
        saxParser.parse(new File(argv[0]), handler);

    } catch (Throwable t) { t.printStackTrace(); }

    System.exit(0);
}
```

Demo: Empty SAX Application

```
> cd “sax empty”
> dir
date.xml App.java
> javac App.java
> java App date.xml
```

Example: Element Counter 1

```
public class App extends DefaultHandler {
```

Declare a counter variable:

```
    int counter;

    public static void main(String argv[]) {
        …
    }
```

Example: Element Counter 2

Increment the counter for every new element:

```
    public void startElement(
        String namespaceURI, String sName,
        String qName, Attributes attrs) throws SAXException {
        counter++;
    }
```

Print the counter when encountering the end of the document:

```
    public void endDocument() throws SAXException {
        System.out.println(counter);
    }
}
```

Demo: Element Counter

```
> cd “sax counter”
> dir
date.xml App.java
> javac App.java
> java App date.xml
```

Task: Document Depth

Design a SAX application to calculate the depth of an XML document, that is the longest nesting of element within each other.

Implement event handlers `startElement`, `endElement` and `endDocument`.

Attribute Interface

Reconsider:

```
public void startElement(
 String namespaceURI, String sName,
    String qName, Attributes attrs) throws SAXException {
 …
}
```

What is `Attributes`? An interface for a list of XML attributes.

1. `int getLength()`
2. `String getLocalName(int index)`
3. `String getValue(int index)`
4. etc.

Task: Document Statistics

Write a SAX application to print the names of all elements encountered and the number of attributes for each of them.

DOM

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) event-based programming (SAX)
   b) tree-based programming (DOM)
   c) rule-based programming (XSLT)

DOM

Document Object Model:

- Document – written with HTML, XML and others
- Object – representing parts of a document
- Model – document modeled as a tree

DOM Standard

A standard application programming interface (API) to access and update the structure of a document:

- standard method to access and update XML
- widely used in all major programming languages
- very useful in web browsers

DOM Components

What is DOM:

1) API
2) W3C Recommendation
3) document represented as a tree

DOM Components: API

DOM is a tree-based API - a set of interface definitions to access and update the tree representation of a document.

Overhead versus convenience:

- overhead - the document is loaded into memory
- convenience – documents can be randomly accessed

DOM Components: W3C Recommendation 1

DOM is the W3C Recommendation:

> …*a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents…*

DOM is an open standard.

Defined in IDL.

`http://www.w3.org/DOM`

DOM Components: W3C Recommendation 2

Evolution of DOM W3C Recommendation:

- level 1 – October 1998
  programmatic interface to manipulate XML and HTML

2) level 2 – November 2000
   multiple interfaces: core, views, events, CSS, traversal, range

- level 3 – April 2004
  support for information sets, XBase, attaching user information

DOM Components: Document Tree 1

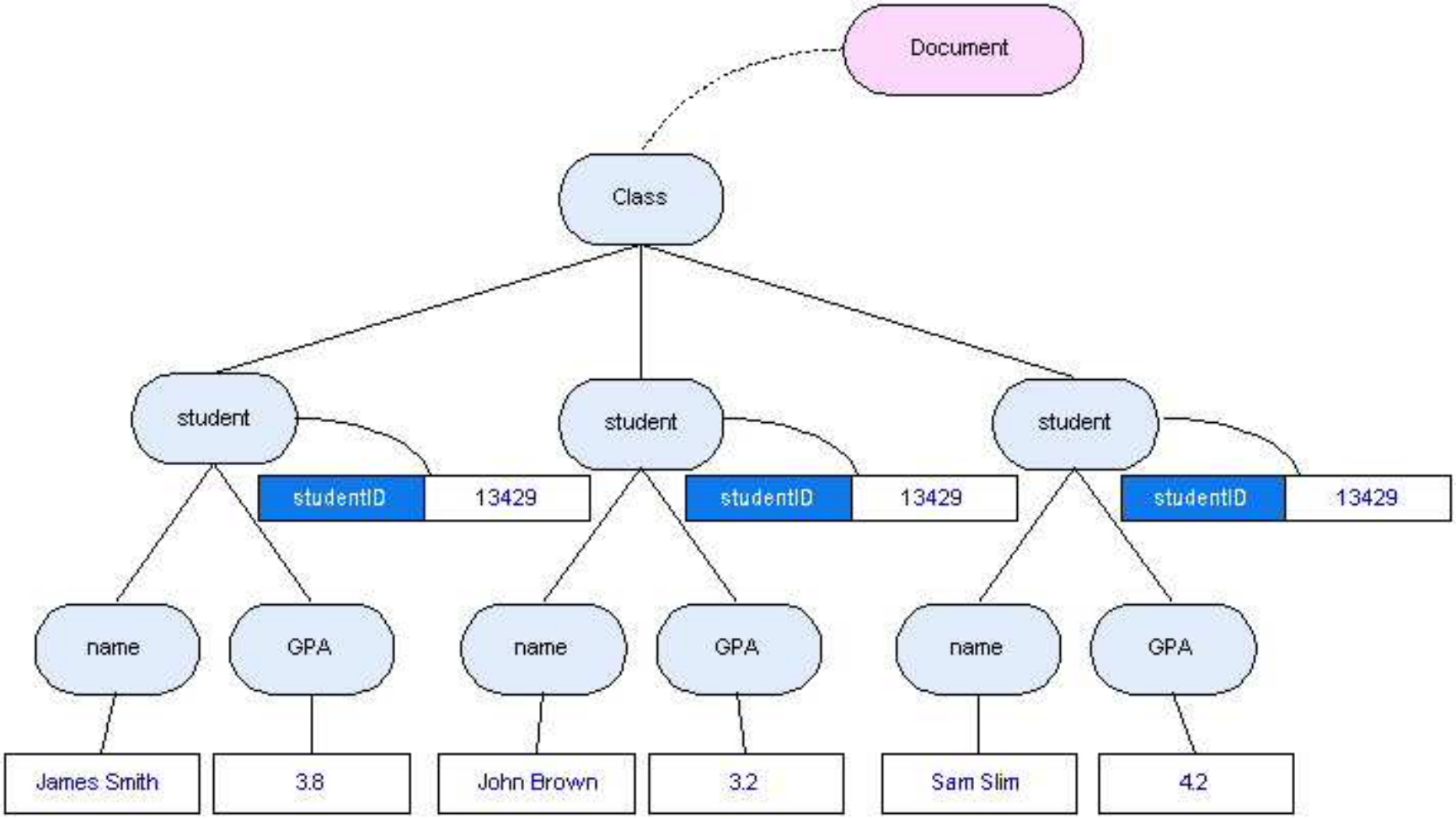
Document components represented as node objects.

Nodes related to each other through properties.

Nodes are of various types:

1) elements
2) attributes
3) comments
4) text
5) etc.

DOM Components: Document Tree 2



DOM Components: Document Tree 3

Why represent a document as a tree?

1. suitable for recursive processing
2. suitable for accessing the content randomly
3. natural fit
   - documents have parts, trees have parts
   - documents have hierarchies, trees have hierarchies

Usage of DOM

When is DOM typically used?

1. applications that require random access to parts of a document
2. applications that require a document to be modified

For example:

1. scripting HTML pages
2. XML authoring tools
3. SVG viewers

DOM Strengths and Weaknesses 1

DOM is build to address some classes of problems better than others.

DOM stores a document in memory:

- suitable for random access
- heavy memory usage

DOM is unsuitable for:

- large documents
- devices with limited memory

DOM Strengths and Weaknesses 2

DOM is language-independent.

This:

- encourages open standards and implementations
- no need to switch models when switching languages

but:

- choosing the lowest common denominator – one cannot take advantage of specific language support

DOM versus SAX

SAX (Simple API for XML):

1. does not store the document in memory
2. parses XML by triggering callbacks to an application

Both build to address different classes of problems:

1. DOM
   - randomly access a document
   - update a document
2. SAX
   - large documents
   - document processing

DOM Implementations

W3C defines the interface.

Many language bindings: Java, JavaScript, C++, Perl, C#, etc.

A wide variety of implementations:

1. open-source and commercial
2. many different languages
3. complete and incomplete

Java Example

Create a DOM from an SVG image file:

```
DOMParser parser = new DOMParser();
Document doc = parser.parse("hello_world.svg");
Element docEl = doc.getDocumentElement();
System.out.println(docEl.hasChildNodes());
```

1. instantiate a DOM Parser
2. load the image into a DOM
3. get the root element of the document
4. print out if the root element has children

DOM Read Application 1

Defines the API to obtain DOM Document instances from XML:

```
import javax.xml.parsers.DocumentBuilder;
```

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents:

```
import javax.xml.parsers.DocumentBuilderFactory;
```

Exception classes for parser configuration errors:

```
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
```

SAX parser exceptions:

```
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```

DOM Read Application 2

Required I/O classes:

```
import java.io.File;
import java.io.IOException;
```

The Document interface represents the entire HTML or XML document:

```
import org.w3c.dom.Document;
```

DOM exceptions are raised when an operation is impossible for logical reasons, lost data, or because the implementation became unstable:

```
import org.w3c.dom.DOMException;
```

DOM Read Application 3

DOM application class:

```
public class Dom {
```

Static document object:

```
   static Document document;
```

The main method:

```
   public static void main(String[] argv) {
    …
   }

}
```

DOM Read Application 4

The main method:

Checking the presence of a command-line argument:

```
if (argv.length != 1) {
System.err.println("Usage: java Dom filename");
System.exit(1);
 }
```

Creating a new DOM parser factory object:

```
DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
```

DOM Read Application 5

Creating the DOM builder from the factory object:

```
try {
    DocumentBuilder builder =
        factory.newDocumentBuilder();
```

Creating a DOM tree for the input document:

```
    document = builder.parse(new File(argv[0]));
```

DOM Read Application 6

Catching four different kinds of exceptions:

```
    } catch (SAXParseException spe) {
        System.out.println(spe.getMessage());
    } catch (SAXException sxe) {
        System.out.println(sxe.getMessage());
    } catch (ParserConfigurationException pce) {
        System.out.println(pce.getMessage());
    } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
    }
   }
}
```

Task: DOM Read Application

Type the DOM read application skeleton.

Compile.

Run.

Demo: DOM Read Application

```
> cd “dom read”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

DOM Read-Write Application 1

Previous DOM application could only read XML.

Lets write one that both reads and writes.

As in the read application:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.File;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
```

DOM Read-Write Application 2

Setting up the transformer and stream packages:

```
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
```

DOM Read-Write Application 3

As before:

```
public class App {

static Document document;

   public static void main(String[] argv) {
    if (argv.length != 1) {
       System.err.println("Usage: java App filename");
       System.exit(1);
    }

    DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
    try {
       DocumentBuilder builder =
          factory.newDocumentBuilder();
             document = builder.parse(new File(argv[0]));
```

DOM Read-Write Application 4

Creating the source, transformer and result:

```
TransformerFactory tFactory =
TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer();
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

Catching transformer exceptions:

```
} catch(TransformerConfigurationException tce){
   System.out.println(tce.getMessage());
} catch(TransformerException te){
   System.out.println(te.getMessage());
}
```

DOM Read-Write Application 5

As before:

```
  catch (SAXParseException spe) {
    System.out.println(spe.getMessage());
} catch (SAXException sxe) {
    System.out.println(sxe.getMessage());
} catch (ParserConfigurationException pce) {
    System.out.println(pce.getMessage());
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
  }
 }
}
```

Task: DOM Read-Write Application

Type the DOM read-write application by extending the read application.

Compile.

Run.

Demo: DOM Read-Write Application

```
> cd “dom read write”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

DOM Data Types

1. `Node` – the main DOM data type
2. `NodeList` – ordered collection of nodes
3. `NamedNodeMap` – name-indexed collection of nodes
4. `DOMString` – UTF16-encoded string
5. `DOMImplementation` – current implementation
6. `DOMException` – error codes
7. `DOMTimeStamp` – time value

DOM Interface

```
NodeList
NamedNodeMap
Node        |   Node
            |   Document
            |   DocumentFragment
            |   Element
            |   Attr
            |   CharacterData   |   Text    |   CDATASection
            |                   |   Comment
            |   Entity
            |   EntityReference
            |   ProcessingInstruction
DOMString
DOMTimeStamp
DOMImplementation
DOMException
```

DOM Interface: Node

The base interface for DOM.

Many different types of Nodes.

Defines common properties and methods.

Each of the specific node types inherit these.

Possible to completely access the content and structure.

DOM Interface: Node Types

Tree-building node types:

```
ELEMENT_NODE                1
ATTRIBUTE_NODE              2
ENTITY_REFERENCE_NODE       5
ENTITY_NODE                 6
DOCUMENT_NODE               9
DOCUMENT_FRAGMENT_NODE     11
```

Leaf node types:

```
TEXT_NODE                   3
CDATA_SECTION_NODE          4
PROCESSING_INSTRUCTION_NODE 7
COMMENT_NODE                8
DOCUMENT_TYPE_NODE         10
NOTATION_NODE              12
```

DOM Interface: Node Properties

1. `node.nodeName` - depends on `nodeType`
2. `node.nodeValue` - depends on `nodeType`
3. `node.nodeType` - one of defined types
4. `node.childNodes` - if any
5. `node.attributes` - if `nodeType` is `Element`

DOM Interface: Node Navigation

1. `node.ownerDocument` - if `nodeType` isn't `Document`
2. `node.parentNode` - if one exists
3. `node.firstChild` - if one exists
4. `node.lastChild` - if one exists
5. `node.nextSibling` - if one exists
6. `node.previousSibling` - if one exists

DOM Interface: Node Methods

1. `node.insertBefore(newChild,refChild)`
2. `node.replaceChild(newChild,oldChild)`
3. `node.removeChild(oldChild)`
4. `node.appendChild(newChild)`
5. `node.hasChildNodes()`
6. `node.cloneNode(deep)`
7. `node.hasAttributes()`
8. `node.isSupported()`
9. `node.normalize()`

DOM Interface: Element 1

`Element` extends `Node`

Most common node type in a document.

The only node type relevant for the `Attrs` interface.

Property: `Element.tagName` – the name of the element.

DOM Interface: Element 2

General methods:

```
element.getElementsByTagName(name)
element.hasAttribute(name)
```

Attribute node methods:

```
element.getAttributeNode(name)
element.setAttributeNode(newAttr)
element.removeAttributeNode(oldAttr)
```

Example: Inserting the Text Node 1

Consider the credit card application document:

```
<?xml version="1.0"?>
<letter decision="rejected">
    <customer>Simon White</customer>
    <product>credit card</product>
    <officer level="manager">Steven Rod</officer>
    <enclosure>credit card</enclosure>
    <enclosure>initial PIN</enclosure>
    <cc></cc>
</letter>
```

Suppose we would like to transform this document by adding to the element `cc` the text “to archives”.

Example: Inserting the Text Node 2

Add to the read-write application after:

```
document = builder.parse(new File(argv[0]));
```

Retrieve the root element from the document:

```
Element elem = document.getDocumentElement();
```

Create the next text node to hold the text “to archives”:

```
Text text = document.createTextNode("to archives");
```

Example: Inserting the Text Node 3

Get the last child of the root element:

```
Node cc = elem.getLastChild();
```

Append the new text node as the last child of the cc element:

```
cc.appendChild(text);
```

Demo: DOM Inserting Application

```
> cd “dom new child”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

Task: Element-Removing Application

Remove the second enclosure element from the credit card document.

Use:

1. `getElementByTagName`
2. `removeChild` of the `Element` interface

Example: Element-Removing Application

Add to the read-write application after:

```
document = builder.parse(new File(argv[0]));
```

Retrieve the root element:

```
Element elem = document.getDocumentElement();
```

Obtain the list of all `enclosure` children of the root:

```
NodeList nodes = elem.getElementsByTagName("enclosure");
```

Remove the second child from this list:

```
elem.removeChild(nodes.item(1));
```

Demo: DOM Element-Removing Application

```
> cd “dom remove child”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

Task: Attribute-Changing Application

Change the letter's `decision` attribute from `rejected` to `accepted`.

Use:

1. `getElementByTagName`
2. `removeChild` of the `Element` interface

Example: Attribute-Changing Application

Add to the read-write application after:

```
document = builder.parse(new File(argv[0]));
```

Retrieve the root element, change the attribute value:

```
Element elem = document.getDocumentElement();
elem.setAttribute("decision", "accepted");
```

Demo: DOM Attribute-Changing Application

```
> cd “dom change attribute”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

DOM Interface: Attributes

`Attr` interface extends `Node`.

Properties:

1. `attr.ownerElement` – the element this attribute is attached to
2. `attr.name` – the name of the attribute
3. `attr.specified` – true if value was specified in the document
4. `attr.value` – value of the attribute
   - if read – returned as string
   - if write – text node with a given string

However, attributes are not part of the tree.

Attr objects inherit the Node interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree.

DOM Interface: Character Data

`CharacterData` interface extends `Node`.

Used to access character data.

Properties:

1. `node.data` – character data of the node
2. `node.length` – number of characters

Methods:

1. `cData.substringData(offset, count)`
2. `cData.appendData(arg)`
3. `cData.insertData(offset,arg)`
4. `cData.deleteData(offset,count)`
5. `cData.replaceData(offset,count,arg)`

DOM Interface: Document 1

`Document` interface extends `Node`.

Represents the entire document.

Contains information about the document type, document element and implementation.

Provides methods for:

1. abstract factory for creating the document's components
2. finding specific components in the document

DOM Interface: Document 2

Properties:

1. `node.doctype` – the document type of the document
2. `node.implementation` – the document's DOM implementation
3. `node.documentElement` – convenient access to the root element

Search methods:

1. `document.getElementsByTagName(tagname)`
2. `document.getElementByID(tagname)`

DOM Interface: Document 3

Abstract factory methods:

1. `document.createElement(tagName)`
2. `document.createDocumentFragment()`
3. `document.createTextNode(data)`
4. `document.createComment(data)`
5. `document.createCDATASection(data)`
6. `document.createProcessingInstruction(target,data)`
7. `document.createAttribute(name)`
8. `document.createEntityReference(name)`

DOM Interface: DOM Implementation

Represents the current implementation of the DOM.

Methods:

1. `DOMImplementation.hasFeature(feature, version)`
2. `createDocumentType(qualifiedName,publicID,systemID)`
3. `createDocument(namespaceURI,qualifiedName,docType)`

DOM Interface: DOM Implementation Features

XML Module:	`XML`
HTML Module:	`HTML`
Views Module:	`Views`
StyleSheet Module:	`StyleSheets`
CSS Module:	`CSS`
CSS (extended) Module:	`CSS2`
Event Module:	`Events`
User Interface Events:	`UIEvents`
Mouse Events Moudule:	`MouseEvents`
Mutation Events Module:	`MutationEvents`
Traversal Module:	`Traversal`
Range Module	`Range`

DOM Interface: DOM Exception

Defines error codes for specific processing situations:

```
INDEX_SIZE_ERR                   1
DOMSTRING_SIZE_ERR               2
HIERARCHY_REQUEST_ERR            3
WRONG_DOCUMENT_ERR               4
INVALID_CHARACTER_ERR            5
NO_DATA_ALLOWED_ERR              6
NO_MODIFICATION_ALLOWED_ERR      7
NOT_FOUND_ERR                    8
NOT_SUPPORTED_ERR                9
INUSE_ATTRIBUTE_ERR             10
INVALID_STATE_ER        11
SYNTAX_ERR           12
INVALID_MODIFICATION_ERR       13
NAMESPACE_ERR           14
INVALID_ACCESS_ERR         15
```

DOM Validation with DTD 1

How to validate with the DOM parser?

Extend the credit card application with DTD declaration:

```
<?xml version="1.0"?>
<!DOCTYPE letter [
  <!ELEMENT letter
   (customer,product,officer,enclosure*)>
  <!ATTLIST letter decision CDATA #REQUIRED>
  <!ELEMENT customer (#PCDATA)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT officer (#PCDATA)>
  <!ATTLIST officer level CDATA #IMPLIED>
  <!ELEMENT enclosure (#PCDATA)>
  <!ELEMENT cc (#PCDATA)>
]>
<letter decision="rejected">
  …
</letter>
```

DOM Validation with DTD 2

How to validate with the DOM parser?

```
…
DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
```

Set the validating feature for the DOM factory objects:

```
factory.setValidating(true);
try {
   DocumentBuilder builder = factory.newDocumentBuilder();
```

Register an instance of the current class as the error handler:

```
   builder.setErrorHandler(new App());
   document = builder.parse(new File(argv[0]));
}
catch {…}
```

DOM Validation with DTD 3

Implement error handlers:

```
public void error(SAXParseException exception) {
   System.out.println("Error: " + exception.getMessage());
}

public void fatalError(SAXParseException exception) {
   System.out.println("Fatal Error: " +
    exception.getMessage());
}

public void warning(SAXParseException exception) {
   System.out.println("Warning :" +
    exception.getMessage());
}
```

Demo: DOM Validation with DTD

```
> cd “dom validate DTD”
> dir
App.java credit.xml
> javac App.java
> java App credit.xml
```

DOM Validation with Schema 1

How to validate with the Schema parser?

Packages as before plus `XMLConstants`:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import org.xml.sax.helpers.DefaultHandler;
import java.io.File;
import org.w3c.dom.*;
import org.w3c.dom.DOMException;
import javax.xml.validation.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;

import javax.xml.XMLConstants;
```

DOM Validation with Schema 2

```
public class App {

   static Document document;
```

Set the schema validation feature:

```
    static final String SCHEMA_VALIDATION_FEATURE_ID =
    "http://apache.org/xml/features/validation/schema";
```

Schema is referred to from the command line:

```
   public static void main(String[] argv) {
    if (argv.length != 2) {
                  System.err.println("Usage: java App
          xmlfile xmlschemafile");
                  System.exit(1);
          }
```

DOM Validation with Schema 3

```
try {
```

Parse an XML document into a DOM tree:

```
DocumentBuilder parser =
   DocumentBuilderFactory.newInstance().newDocumentBuilder(
   );
Document document = parser.parse(new File(argv[0]));
```

Create a `SchemaFactory` capable of understanding schemas:

```
SchemaFactory factory =
   SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS
   _URI);
```

Load a WXS schema, represented by a Schema instance:

```
Source schemaFile = new StreamSource(new File(argv[1]));
Schema schema = factory.newSchema(schemaFile);
```

DOM Validation with Schema 4

Load a schema:

```
Source schemaFile = new StreamSource(new File(argv[1]));
Schema schema = factory.newSchema(schemaFile);
```

Create a Validator instance to validate an instance document:

```
Validator validator = schema.newValidator();
```

Validate the DOM tree:

```
   validator.validate(new DOMSource(document));
} catch (Exception e) {
   System.out.println(e.getMessage());
}
}
}
```

Demo: DOM Validation with Schema

```
> cd “dom validate schema”
> dir
App.java noNamespace.xml schemaNoNamespace.xsd
> javac App.java
> java App noNamespace.xml schemaNoNamespace.xsd
```

Java and XSLT

Program

1) Introduction
   a) motivation
   b) overview
   c) origin
   d) W3C

2) XML Language
   a) Unicode
   b) XML
   c) DTD
   d) namespaces

3) XML Technologies
   a) validation (XML Schema)
   b) access (XPath)
   c) transformation (XSLT)

4) XML Java Processing
   a) event-based programming (SAX)
   b) tree-based programming (DOM)
   c) rule-based programming (XSLT)

XSLT and Java

XSLT – Extensible Stylesheet Language Transformation has been introduced before.

How to use XSLT from Java applications?

- formulate the transformation template in external XSLT file
- use the transformer class as in the read-write DOM application

Java XSLT Application 1

Import the packages as before:

```
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import java.io.FileOutputStream;
import javax.xml.XMLConstants;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
```

Java XSLT Application 2

The beginning as the read-write DOM application:

```
public class App {

    static Document document;

    public static void main(String[] argv) {
        if (argv.length != 3) {
            System.err.println("Usage:
                java App xmlfile xslfile outfile");
            System.exit(1);
        }
```

Java XSLT Application 3

Parser an XML document into a DOM tree:

```
try {
    DocumentBuilder parser =
    DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document document = parser.parse(new File(argv[0]));
```

Setup the transformer according to the external XSLT file:

```
    TransformerFactory tFactory =
       TransformerFactory.newInstance();
    Transformer transformer =
       tFactory.newTransformer(new StreamSource(argv[1]));
```

Java XSLT Application 4

Perform the transformation on the parser-generate document model, sending the output to the specified file:

```
    transformer.transform(
        new DOMSource(document),
        new StreamResult(new FileOutputStream(argv[2])));

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

DEMO: Java XSLT Application

```
> cd “dom xslt”
> dir
App.java farewell.xml farewell.xsl
> javac App.java
> java App farewell.xml farewell.xsl output
> type output
```

Acknowledgements

The author would like to thank Milton Chau, Elsa Estevez, Brian Iu, Adegboyega Ojo, Gabriel Oteniya and Frank Wong for their comments, code and support during the preparation and delivery of this course.

B. Assessment

B.1. Set 1

1. (8%)	Which line of the following xml document is wrong: 1 <foo:aaa xmlns:foo="ns1" xmlns="ns2"> 2 <foo:aaa> 3 <foo:bbb xmlns:foo=""> 4 <foo:bbb>abcd</foo:bbb> 5 </foo:bbb> 6 <ccc xmlns =""> 7 abcd 8 </ccc> 9 </foo:aaa>	
	A.	Line 1
Ans	B.	Line 3
	C.	Line 6
	D.	Line 7
2. (8%)	What will be the class name of a Java class which maps to the following XML schema, : <schema> <complexType name="Address"> <sequence> <element name="number" type="xsd:int"/> <element name="street" type="xsd:string"/> <element name="city" type="xsd:string"/> </sequence> </complexType> </schema>	
Ans	A.	Address
	B.	number
	C.	sequence
	D.	schema
3. (8%)	Which one of the following is not a part of a SOAP message?	
	A.	Envelope
Ans	B.	Port
	C.	Header
	D.	Body
4. (9%)	Which one of the following statement is wrong?	
	A.	SOAP is a protocol for exchanging information in a decentralized and distributed environment.

Ans	B.	SOAP deals with objects with remote object references.
	C.	SOAP clients do not hold any stateful references to remote objects.
	D.	SOAP uses XML to send and receive messages.
5. (9%)	Which one of the following statement about WSDL document is wrong?	
Ans	A.	The `binding` element has attributes: `name` and `portType`.
	B.	must contain a root element: `definitions`.
	C.	like Java interfaces, is a contract between the server and client developers.
	D.	The `service` element describes the location of the web service.
6. (8%)	Which one of the following statement is correct?	
	A.	UDDI specifications define registry service for web services only.
Ans	B.	UDDI uses the publsherAssertion structure to define the relationship between two businessEntities.
	C.	A web service provider can update its information through any UDDI Business Registry.
	D.	Answer A, B and C
7. (8%)	Which one of the following statement is wrong?	
	A.	X.509 is a kind of security token.
	B.	Digital signature can be used to determine whether the message was altered in transit.
	C.	Certificate authority (CA) is an entity which issues certificates.
Ans	D.	Exclusive XML Canonicalization is also known as XML Canonicalization.
8. (9%)	A JAX-RPC client cannot invoke the service using:	
	A.	Client side stubs automatically generated by a tool
Ans	B.	A client must invoke the service by looking up the Java interface in UDDI
	C.	Dynamic Invocation Interface (DII)
	D.	Dynamic proxies
9. (9%)	Canonical XML specification is used to	
Ans	A.	Determine if two XML documents are logically equivalent
	B.	Digitally encrypt an XML document
	C.	Digitally sign an XML documents
	D.	Register a web service

10. (8%)	SOAP with Attachments API for Java (SAAJ) is	
	A.	An API to for Java messaging.
	B.	An API for remote procedure calls.
Ans	C.	An API to produce, consume and manipulate the XML structure for the SOAP message programmatically.
	D.	An API for describing, discovering and integrating business services.
11. (8%)	In a WSDD file, which one is not a valid value for the `provider` attribute of the `service` element?	
	A.	Java:RPC
	B.	Java:MSG
Ans	C.	Java:Handler
	D.	Java:EJB
12. (8%)	In a WSDL file, `operation` element within the `portType` element may define:	
	A.	Input message
	B.	Output message
	C.	Fault message
Ans	D.	Answer A, B and C

B.2. Set 2

1. (9%)	Canonical XML specification is used to	
	A.	Register a web service
	B.	Digitally encrypt an XML document
	C.	Digitally sign an XML documents
Ans	D.	Determine if two XML documents are logically equivalent
2. (8%)	In a WSDD file, which one is not a valid value for the `provider` attribute of the `service` element?	
	A.	Java:RPC
	B.	Java:MSG
Ans	C.	Java:Handler
	D.	Java:EJB
3. (8%)	In a WSDL file, `operation` element within the `portType` element may define:	
	A.	Input message
	B.	Output message
	C.	Fault message
Ans	D.	Answer A, B and C
4. (9%)	Which one of the following statement is wrong?	
	A.	SOAP is a protocol for exchanging information in a decentralized and distributed environment.
Ans	B.	SOAP deals with objects with remote object references.
	C.	SOAP clients do not hold any stateful references to remote objects.
	D.	SOAP uses XML to send and receive messages.
5. (9%)	A JAX-RPC client cannot invoke the service using:	
	A.	Client side stubs automatically generated by a tool
Ans	B.	A client must invoke the service by looking up the Java interface in UDDI
	C.	Dynamic Invocation Interface (DII)
	D.	Dynamic proxies

6. (8%)	Which line of the following xml document is wrong: 1 <foo:aaa xmlns:foo="ns1" xmlns="ns2"> 2 <foo:aaa> 3 <foo:bbb xmlns:foo=""> 4 <foo:bbb>abcd</foo:bbb> 5 </foo:bbb> 6 <ccc xmlns =""> 7 abcd 8 </ccc> 9 </foo:aaa>	
	A.	Line 1
Ans	B.	Line 3
	C.	Line 6
	D.	Line 7
7. (8%)	What will be the class name of a Java class which maps to the following XML schema, : <schema> <complexType name="Address"> <sequence> <element name="number" type="xsd:int"/> <element name="street" type="xsd:string"/> <element name="city" type="xsd:string"/> </sequence> </complexType> </schema>	
Ans	A.	Address
	B.	number
	C.	sequence
	D.	schema
8. (8%)	Which one of the following is not a part of a SOAP message?	
	A.	Envelope
Ans	B.	Port
	C.	Header
	D.	Body
9. (9%)	Which one of the following statement about WSDL document is wrong?	
Ans	A.	The `binding` element has attributes: `name` and `portType`.
	B.	must contain a root element: `definitions`.
	C.	like Java interfaces, is a contract between the server and client developers.
	D.	The `service` element describes the location of the web service.

10. (8%)	Which one of the following statement is correct?	
	A.	UDDI specifications define registry service for web services only.
Ans	B.	UDDI uses the publsherAssertion structure to define the relationship between two businessEntities.
	C.	A web service provider can update its information through any UDDI Business Registry.
	D.	Answer A, B and C
11. (8%)	Which one of the following statement is wrong?	
	A.	X.509 is a kind of security token.
	B.	Digital signature can be used to determine whether the message was altered in transit.
	C.	Certificate authority (CA) is an entity which issues certificates.
Ans	D.	Exclusive XML Canonicalization is also known as XML Canonicalization.
12. (8%)	SOAP with Attachments API for Java (SAAJ) is	
	A.	An API to for Java messaging.
	B.	An API for remote procedure calls.
Ans	C.	An API to produce, consume and manipulate the XML structure for the SOAP message programmatically.
	D.	An API for describing, discovering and integrating business services.

Web Services and Java

Training Course

Elsa Estevez
Chau Keng Fong
Adegboyega Ojo
Gabriel Oteniya
Tomasz Janowski

e-Macao Report 23

Version 1.0, October 2005

Table of Contents

1. Overview

Web Services are software systems that are designed to support machine-to-machine interaction over a network or the Internet using standard internet protocols. Web Services represent a convergence of Service-Oriented Architecture and the Web, combining the benefits of traditional component-based development and the Web. Like components, Web Services offer functionalities that can be easily reused without requiring the knowledge of implementation details. At the same time, their availability through the Web makes them ubiquitous. Traditional interoperability challenges associated with distributed computing protocols like CORBA, DCOM or RMI due to vendor-specific implementations, different encoding schemes and heterogeneous platforms, are addressed by Web Services.

Currently, Web Services represent the standard means of providing interoperability between different software applications running on different platforms and frameworks. Web Services are characterised by machine-readable descriptions based on the Extensible Markup Language (XML). They can also be composed to achieve complex operations.

Web Services are associated with three basic standards: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description Discovery and Integration (UDDI). SOAP defines a standard communication protocol for Web Services. WSDL provides a standard mechanism to describe Web Services. UDDI provides a standard approach to register and discover Web Services. These standards are all based on XML.

This course - Web Services and Java presents a detailed exposition of Web Services. It starts with an introduction to Service-Oriented Architectures, describing the features and comparing Web Services with other distributed architectures. Next, it presents how Web Services communicate using SOAP messages. The various elements of WSDL are also discussed to show how Web Services are described using this language. A discussion on how Web Services can be published and discovered through public registries follows. The Apache AXIS implementation of SOAP is used in the course and is discussed after UDDI. Finally, security issues related to Web Services are presented.

The rest of this document explains the objectives, prerequisites and methodology for teaching the course in Sections 2, 3 and 4 respectively. The content of the course is introduced in Section 5. The assessment and organization of the course are explained in the final Sections 6 and 7. Following references, Appendix A includes the complete set of slides and Appendix B contains two sets of assessment questions with answers.

2. Objectives

This course aims to achieve three major objectives:

1) To provide students with a good understanding of Service-Oriented Architectures and how such architectures are realized through Web Services.
2) To introduce three XML-based technologies associated with Web Services:
   a. Simple Object Access Protocol (SOAP),
   b. Web Services Description Language (WSDL) and
   c. Universal Description Discovery and Integration (UDDI)
3) To equip students with skills in developing Java Web Services.
4) To show how Web Services can be used for delivering Electronic Services.

3. Prerequisites

The course requires that students already know how to write Java applications. It is also expected that the students have some knowledge of how to develop distributed applications in Java. In addition, the knowledge of XML technologies is essential.

4. Methodology

The course has been designed based on the following didactic principles:

- *Depth versus Breadth* - As foundation, an attempt has been made to cover the various aspects of Web Services without much loss of depth.
- *Academic Orientation* – A body of concepts is defined rigorously and incrementally to establish a foundation for proper understanding and use of technology.
- *From Definitions to Demonstrations* – All major concepts introduced are illustrated with small-size examples which are also demonstrated on the computer, whenever possible.
- *From Demonstrations to Assignments* – On the basis of demonstrations, students are asked to perform different tasks with increasing level of difficulty and independence.

5. Content

The course consists of 762 slides organized into six major sections: Introduction, SOAP, WDSL, AXIS, UDDI and Web Service Security. These sections are discussed in turn below.

5.1. Introduction

This section consists of the slides 1 through 118. It presents the general information on the course: objectives, outline and organization. Next, the section explains the basic concepts of Service-Oriented Architectures. Web Services are also introduced in the context of Service-Oriented Architectures. Subsequently, the Web Service Architecture stack is described. Finally, the section demonstrates how Apache AXIS may be installed and configured to implement and deploy Web Services.

5.2. SOAP

This section, comprising the slides 119 through 268 presents the general introduction to SOAP and discusses its various aspects: Messaging, Data Structures, Protocol Binding and Binary Data Handling. Overviews of subsequent subsections follow:

1) *Introduction* – This section begins with a brief history of SOAP, its definition and features. Next, it compares SOAP with ebXML and presents the concepts of the SOAP Toolbox and the SOAP Implementation model.

2) *Messaging* – SOAP messages are introduced. The content of a typical SOAP message is presented: Envelope, Header, Body, Fault, etc. The SOAP namespace is described. An envelope is explained as a SOAP message container. The use of headers for extending SOAP messaging is explained. Different types of SOAP faults are described. SOAP intermediaries are explained and how they may be associated with different roles. Finally, processing rules, error handling and various fault elements are presented.

3) *Data Structure* – The SOAP data model is presented. A description of the encoding rules for simple (e.g. Integer, String) and compound (e.g. Arrays) values is provided. The encoding rules for faults, requests, responses, Remote Procedure Call requests, Remote Procedure Call return values, Remote Procedure Call in-out parameters are also explained. Finally, two basic SOAP communication styles are explained: synchronous (Remote Procedure Call) and asynchronous (Document Exchange).

4) *Protocol Binding* – The SOAP protocol binding is explained. The use of SOAP features as an extension mechanism is presented. Two Message Exchange Patterns are described: Request-Response and SOAP Response. At the end, the SOAP HTTP binding is discussed.

5) *SOAP and Binary Data* – The mechanism for sending SOAP messages with binary attachment included is presented.

5.3. WSDL

This section consists of the slides 269 through 433. It presents an introduction to Web Service Description Language. The language, transmission primitives, language extensions and WSDL Java APIs are also described. The overview of subsequent sections follows:

1) *Introduction* - The importance of service descriptions is discussed. Functional and non-functional aspects of Web Services, including the layers of a typical service description are explained. The section is concluded with a discussion on the history of WSDL and a comparison between WSDL and IDL.

2) *Language* - The structural components of a WSDL specification are explained: abstract part - Port Types, Operations, Messages and Types, and concrete part – Interfaces Bindings, Ports and Services. The WSDL Information Model is presented to explain the content of a WSDL document. Different language elements are described: Definition, Type, Message, Part, Port Type, Binding, Port, Service, Documentation and Import.

3) *Transmission Primitives* - The four basic operation patterns or transmission primitives are presented: one-way, request-response, solicit-response and notification. Each of these primitives is described with examples.

4) WSDL Extensions - The mechanism for extending WSDL with elements from other namespaces is presented here. The standard functional extensions for HTTP GET/POST and MIME attachments are discussed. The use of WS-Policy and related specifications for describing non-functional features of Web Services like security requirements, transactional capabilities, logging features and auditing is also presented.

5) *WSDL and Java:* The conventions for mapping Port Types, Operations, Messages and Bindings to Java are presented.

5.4. AXIS

This section, comprising the slides 432 through 565 presents: the overview of AXIS, Service Invocation, AXIS Tools, AXIS Configuration, Service Deployment and Service Lifecycle. The sections are described as follows:

1) *Overview* – A brief history of the AXIS SOAP engine is presented. The AXIS architecture is explained together with the concepts of Handlers and Chains. Two categories of Chains are described: simple and targeted. The use of Message Context in passing objects to Handlers is discussed. Handler development and deployment are also covered. Lastly, Java API for Web Services (JAX) RPC is presented.

2) *Service Invocation* – The Message Context Class and the Message Type are explained. The details of how Envelop elements can be accessed and replaced are presented. The AXIS Client APIs are described for both static (stub generation) and dynamic invocation. The use of Service Objects is discussed. The techniques for calling Web Services are presented using End-Points and Call-and-Service Objects. Dynamic Binding and the use of Sessions are also explained.

3) *AXIS Tools* – Three basic AXIS tools are presented. First, it is explained how WSDL2Java generates Java code for client and server sides of an application. Second, the process of generating WSDL specifications from Java interfaces is presented. Third, the automatic generation of WSDL specifications for deployed services is discussed.

4) *AXIS Configuration* – The AXIS Web Service Deployment Descriptor (WSDD) document is described. Then, the global configuration of the AXIS engine, Handler declarations and Chain definitions are discussed. The mappings of Types and Beans are also covered.

5) *Service Deployment* – The AXIS WSDD is presented. Two major elements of the WSDD document are described: JAX-RPC Handler and Operation elements. Methods used for deploying and un-deploying Web Services are discussed as well.

6) *Service Lifecycle* – The issues relating lifecycle and scope of Web Services are presented: creation, deployment and usage.

5.5. UDDI

This section, consisting of the slides 566 through 664 starts with a general introduction to UDDI, followed by an overview of the major UDDI concepts. The UDDI data types followed by the UDDI Registry are also presented. Here is a brief overview of the sections:

1) *Introduction* – The motivation, goals and history of the Universal Description Discovery and Integration (UDDI) project are presented. The UDDI process is explained, from publishing services in the registry, to searching, to service invocation. UDDI services are explained, follows by major UDDI operators. The UDDI specification is discussed as well.

2) *Concepts* – The different UDDI identifier systems are presented: Universally Unique Identifier (UUID), D-U-N-S Identifier and Thomas Register Identifier. Different Classification Schemes or Taxonomies are described as well: United Nations Standard Products and Services Code (UNSPC), North American Industry Classification System (NAICS), and ISO 3166. UDDI models for technical specifications and type taxonomies for classification of technical specifications are discussed.

3) *Data Types* – The UDDI data model is presented. The various elements of the UDDI data types are described in detail: businessEntity, businessService, bindingTemplate, tModel, publisherAssertion, identifierBag and categoryBag.

4) *UDDI Registry* – The UDDI Business Registry is described. A description of how to publish services in UDDI registry is provided along with the necessary APIs. Four categories of APIs are provided for some of the data types.

5.6. Security

This section consists of the slides 665 through 759. It presents security basics, Web Service security and digital signatures. Each section is described briefly below:

1) *Security* – Basic security concepts are given: Confidentiality, Integrity, Authentication, Non-repudiation and Authorization. Cryptography and its use to build symmetric and asymmetric encryptions are discussed. Digital Signatures as well as the Public Key Infrastructure are presented. At the end, Digital Certificates, Kerberos and Security Domains are explained.

2) *Web Service Security* – Basic concepts related to the Web Service security model are explained: Security Token, Subject, Claim, Web Service End-Point Policy and Security Token Service. The Web Service Security Architecture and its specification (SOAP Message Security) are described, followed by the SOAP Message Security Model. Finally, the use of Message Protection, Rules, Security Headers, Security Tokens, User Name Tokens, Basic Security Tokens and XML Tokens is also discussed.

3) *Digital Signature* – The notion of a Signature is explained and Signature Algorithms are discussed. Following the XML Digital Signature Specification, examples of how XML messages are signed are given.

6. Assessment

The course ends with assessment. The assessment comprises 15 multiple-choice questions, covering all major sections and concepts taught.

Two sets of 15 assessment questions and answers are given in Appendices B.1 and B.2. The two sets are different permutation of the same collection of questions. The assessment complements the tasks provided in the various sections.

7. Organization

The course consists of lectures, demonstrations and assignments:

- *lectures* – The lectures aim to incrementally build a body of concepts to establish the foundation for proper understanding and the use of technology.
- *demonstrations* - Demonstrations illustrate the concepts introduced during the lectures with running code and small-size examples.
- *assignments* - During assignments, students are asked to perform a range of tasks with increasing level of difficulty and independence. They are encouraged to reuse the demonstrated code and examples in their assignments.

The full course has been taught for 7 days with 6 hours of lectures per day: (1) Introduction, (2) SOAP, (3) WSDL, (4) Apache AXIS, (5) UDDI, (6) Security and (7) Laboratory Work.

A shorter version has also been taught over 4 days: (1) Introduction and SOAP, (2) SOAP and WSDL, (3) WSDL and UDDI and (4) UDDI and Security.

References

1) Steve Graham, et. al., Building Web Services with Java, Making sense of XML, SOAP, WSDL, and UDDI (2nd ed.), Sams Publishing, 2004.
2) Robert J. Brunner et al., Java Web Services Unleashed, Sams Publishing, 2002
3) Gustavo Alonso, Fabio Casati, Harumi Kuno and Vijay Machiraju, Web Services Concepts, Architectures and Applications, Springer, 2004.
4) WebSphere Application Developer Web Services Handbook – http://publib-b.boulder.ibm.com/abstracts/sg246891.html
5) Systinet Corp., Web Services: A Practical Introduction, White Paper, 2003.
6) RogueWave Software, An Introduction to Web Services, White Paper, 2004.
7) Infravio, Web Services: Next Generation Application Architecture, White Paper, July 2003.
8) CapeClear Software Inc., Principles of SOA Design, White Paper, 2004

Appendix

A. Slides

A.1. Course Overview

Web Services and Java

Elsa Estevez, Tomasz Janowski
Milton Chau and Gabriel Oteniya

UNU-IIST, Macau

The Course

1) objectives - what do we intend to achieve?
2) outline - what content will be taught?
3) resources - what teaching resources will be available?
4) organization - duration, major activities, daily schedule

e-Macao-16-5-3

Course Objectives

1) explain the concept of web services
2) present three XML technologies comprising web services:
   a) Simple Object Access Protocol (SOAP)
   b) Web Services Description Language (WSDL)
   c) Universal Description Discovery and Integration (UDDI)
3) present the best practices in developing web services with Java
4) motivate the use of web services for e-government

e-Macao-16-5-4

Course Outline

1) Introduction
2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data
3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java
4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle
5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry
6) Security
   a) security basics
   b) web service security
   c) digital signatures

e-Macao-16-5-5

Introduction Outline

An overview of web services (WS).

Main points:

1) WS definition, components, process, properties
2) Service-Oriented Architectures (SOA)
3) WS and SOA
4) WS architecture stack and interoperability
5) WS implementation

e-Macao-16-5-6

SOAP Outline

SOAP = Simple Object Access Protocol

A protocol for exchanging XML messages in a distributed environment.

Main points:

1) introduction to SOAP
2) messaging framework: envelope and its components, processing rules
3) data structures and rules for encoding data and service requests
4) protocol binding framework
5) using SOAP to send binary data

e-Macao-16-5-7

WSDL Outline

WSDL = Web Services Description Language.

A language for describing web services with XML.

Main points:

1) introduction to WSDL
2) WSDL language structure
3) transmission primitives
4) WSDL extension mechanisms
5) WSDL and Java

e-Macao-16-5-8

AXIS Outline

Presentation of a particular SOAP engine - Apache AXIS.

Open source project.

Main points:

1) AXIS concepts and architecture
2) web service invocation
3) AXIS tools and configuration
4) web service deployment
5) service lifecycle

e-Macao-16-5-9

UDDI Outline

UDDI = Universal Description, Discovery and Integration

An open, platform-independent framework for describing, discovering and integrating business services.

Main points:

1) introduction
2) concepts
3) data types
4) registries

e-Macao-16-5-10

Security Outline

WS-Security describes enhancements to SOAP messaging in order to provide message integrity and confidentiality.

Main points:

1) basic security concepts
2) web service security
3) digital signatures

e-Macao-16-5-11

Course Resources - Books

1) Building Web Services with Java, Making sense of XML, SOAP, WSDL, and UDDI (2nd ed.) – Steve Graham, et. al. – Sams Publishing, 2004
2) Java Web Services Unleashed – Robert J. Brunner et al – Sams, 2002
3) Web Services Concepts, Architectures and Applications – Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju – Springer, 2004
4) WebSphere Application Developer Web Services Handbook – http://publib-b.boulder.ibm.com/abstracts/sg246891.html

e-Macao-16-5-12

Course Resources - Organizations

1) W3C – World Wide Web Consortium, http://www.w3.org
2) OASIS – Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org
3) Apache – Apache Software Foundation, http://www.apache.org

e-Macao-16-5-13

Course Resources - Specifications

1) SOAP - http://www.w3.org/TR/soap/
2) AXIS - http://ws.apache.org/axis/
3) WSDL - http://www.w3.org/TR/wsdl
4) UDDI - http://www.uddi.org/
5) WS-Security - http://www.oasis-open.org/

e-Macao-16-5-14

Course Logistics

1) duration - 42 hours
2) activities - lectures and development
3) timing
   a) Monday 9:00-13:00 14:30-17-45
   b) Tuesday 9:00-13:00 14:30-17-45
   c) Wednesday 9:00-13:00
   d) Thursday 9:00-13:00 14:30-17-45
   e) Friday 9:00-13:00
4) sessions - 7 morning, 5 afternoon
5) style - interactive and tutorial

e-Macao-16-5-15

Course Prerequisite

1) basic Java
2) distributed Java
3) XML, XML namespaces, XML Schema

A.2. Introduction

e-Macao-16-5-16

Introduction

e-Macao-16-5-17

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

e-Macao-16-5-18

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

e-Macao-16-5-19

Service Definition

Definition by W3C:

A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requestors entities.

To be used, a service must be realized by a concrete provider agent.

[Web Services Glossary
http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/]

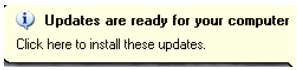
e-Macao-16-5-20

Service Concepts

A service:

1) is a resource and has an owner
2) is provided by a person or an organization
3) must be realized by a (software) provider agent
4) performs one or more tasks
5) is used by a requestor agent

Example: a service for updating software



e-Macao-16-5-21

Web Service (WS)

A web service is a software application that applies XML to exchange data with other applications on other computers.

Features of web services:

1) Web services operate over any network (the Internet or a private Intranet) to achieve specific tasks.
2) The tasks performed by a web service are methods or functions that other applications can invoke and use.
3) Web service requests/responses can be sent/received between different applications on different computers belonging to different businesses.

e-Macao-16-5-22

Web Service Definition

Definition by W3C:

A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network:

1) *It has an interface described in a machine-processable format (specifically WSDL).*

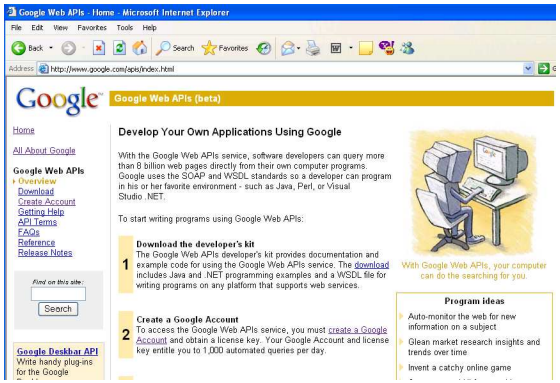
2) *Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web standards.*

[Web Services Glossary http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/]

e-Macao-16-5-23

Web Service Example

A Google web service for Internet search - http://www.google.com/apis:



e-Macao-16-5-24

Task 1: Google Search

Copy from the server in `\WebServices` the folder `demos` to your local PC, in drive E: under a folder with your own name.

Objective: automatic search in Google using a WS

```
1) cd demos\Ws\FirstExample

2) dir
googleapi.jar
GoogleApiDemo.class
GoogleSearch.wsdl

1) copy googleapi.jar \j2sdk1.4.2_04\jre\lib\ext

2) java -cp \demos\WS\FirstExample GoogleApiDemo Macao
```

e-Macao-16-5-25

Service Description

A service description is data describing the capabilities of a web service:

1) all information needed in order to invoke the web service

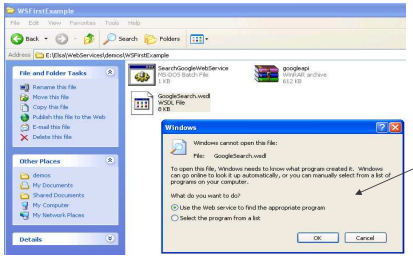
2) the key concept for Service-Oriented Architectures (SOA)

The standard for writing service descriptions is WSDL.

e-Macao-16-5-26

Task 2 : Google Description

1) `cd demos\Ws\FirstExample`
2) double-click `GoogleSearch.wsdl`
3) the following window appears:



4) open the file with a browser - WSDL document describing the service

e-Macao-16-5-27

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

e-Macao-16-5-28

SOA Definition

SOA = Service-Oriented Architecture

SOA is a software architecture where all software-implemented tasks and processes are designed as services to be consumed over a network.

Keywords:

1) architecture
2) service

e-Macao-16-5-29

SOA Approach

SOA approach:

The focus of design is the service interface.

A service:

1) has a well-defined interface
2) can be potentially invoked over a network
3) can be reused in multiple business contexts

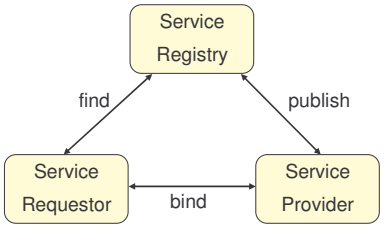
An application:

1) is integrated at the interface and not implementation level
2) is built to work with any implementation of a contract, resulting in a loosely coupled and more flexible system

e-Macao-16-5-30

SOA Components

1) roles
   a) service provider
   b) service requestor
   c) service registry
2) operations
   a) publish
   b) bind
   c) find



e-Macao-16-5-31

SOA Roles: Service Provider

What a service provider does?

1) creates a service description
2) deploys the service in a runtime environment to make it accessible to other entities over the network
3) publishes the service description to one or more services registries
4) receives messages invoking the service from service requestors

Any entity that hosts a network-available web service is a service provider.

e-Macao-16-5-32

SOA Roles: Service Requestor

What a service requestor does?

1) finds a service description published in a service registry
2) applies the service description to bind and invoke the web service hosted by a service provider

A service requestor can be any consumer of a web service.

e-Macao-16-5-33

SOA Roles: Service Registry

What a service registry does?

1) accepts request from service providers to publish and advertise web service descriptions
2) allows service requestors to search the collection of service descriptions contained within the service registry

The role of service registry is to enable match-making between service providers and service requestors.

Once the match has been found, the interactions are carried out directly between the service requestor and the service provider.

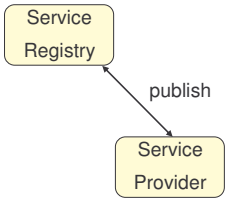
e-Macao-16-5-34

SOA Operations: Publish

The publish operation is an act of service registration or service advertisement.

When a service provider publishes its web service in a service registry, it is advertising the service to the whole community of potential service requestors.

The details of the publish operation depends on how the service registry is implemented.



e-Macao-16-5-35

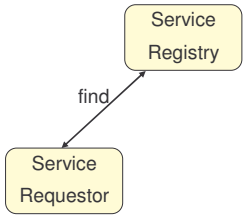
SOA Operations: Find

The find operation is an act of looking for a service satisfying certain conditions:

1) service requestor states a search criteria, such as: the type of the service, its quality, etc.
2) service registry matches the search criteria against the published web service descriptions

The result is a list of service descriptions that match the search criteria.

Details of the operation depend on the implementation of the service registry.



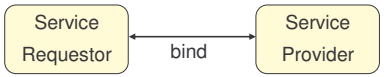
e-Macao-16-5-36

SOA Operations: Bind

The bind operation creates the client-server relationship between service requestor and service provider.

The operation can be:

1) dynamic - creating a client-side proxy on-the-fly based on the service description to invoke the web service
2) static - the developer hard-codes the way the client invokes the web service



e-Macao-16-5-37

SOA Properties 1

SOA is a form of distributed systems architecture.

It is characterized by:

1) logical view - a service is an abstraction is what actual programs, databases, businesses processes etc. are able to do.
2) message exchange – a service is defined in terms of the messages exchanged between provider and requestor agents and not in terms of the properties of the agents themselves

e-Macao-16-5-38

SOA Properties 2

3) abstraction – SOA hides the implementation details of the underlying languages, process and database structures, etc.
4) meta-data – a service is described by machine-processable meta-data
5) small number of operations – a service tends to rely on a small number of operations with relatively large and complex messages
6) network orientation - services are oriented to their use over a network
7) platform-neutral - messages are sent in a standardized format delivered through the interfaces. XML is typically used.

e-Macao-16-5-39

SOA Benefits 1

SOA enables the agents participating in the message exchange to be loosely coupled, which in turn allows for more flexibility:

1) a client is only coupled to a service, not to a server - the integration of the server takes place outside the scope of the client application
2) functional components and their interfaces are separated - new interfaces can be easily added
3) old and new functionality can be encapsulated as software components that provide and receive services

e-Macao-16-5-40

SOA Benefits 2

4) the control of business processes can be isolated:
   a) business-rule engine can control the workflow of a business process
   b) depending on the state, the engine invokes different services
5) services can be incorporated dynamically during runtime
6) service bindings are specified using configuration files and can be easily adapted to satisfy new needs

e-Macao-16-5-41

Service Description in SOA

The key to SOA is service description:

1) it is published by the service provider in the service registry
2) it is returned to the service requestor as a result of the search operation
3) it specifies to the service requestor:
   a) how to bind and invoke the web service
   b) what information is returned as a result of the invocation

e-Macao-16-5-42

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

e-Macao-16-5-43

WS Components

A web service includes three basic components:

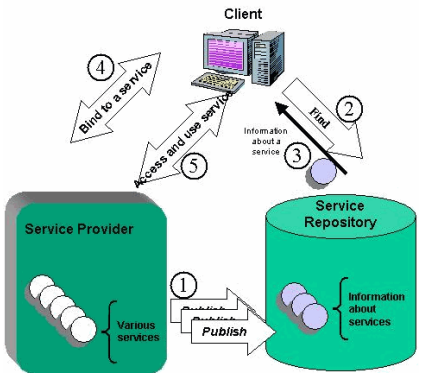
1) a mechanism to find and register interest in a service
2) a definition of the service's input and output parameters
3) a transport mechanism to access a service

Web services also include other technologies that can be used to provide additional features such as security, transaction processing and others.

e-Macao-16-5-44

WS Process

1) a service provider publishes a service to an external repository
2) a client looks up for a service in the repository
3) the repository returns information about the service:
   - call format
   - provider address
4) the client binds to the underlying service
5) the client calls and accesses the service



[courtesy Al Saganich]

e-Macao-16-5-45

WS and Others

Web services do not introduce new functionality.

Similar functionality is provided by:

1) Sun/RPC
2) DCOM
3) Enterprise Java Beans
4) etc.

The difference is how this functionality is provided.

e-Macao-16-5-45

WS and Others

Web services do not introduce new functionality.

Similar functionality is provided by:

1) Sun/RPC
2) DCOM
3) Enterprise Java Beans
4) etc.

The difference is how this functionality is provided.

e-Macao-16-5-46

CORBA Application

Recall the application developed in the Distributed Programming course.

A client requests a file from the server. The server sends the file to the client. When received, the client saves the file on the local machine.

The steps involved:

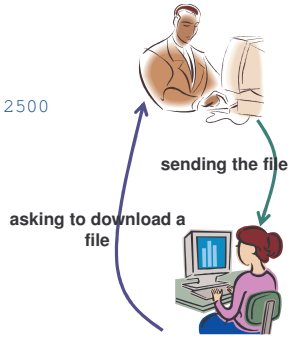
1) define a service interface in IDL
2) map the IDL interface to Java (done automatically)
3) implement the interface (`FileInterface.idl`)
4) develop the server (`FileServer.java`)
5) develop a client (`FileClient.java`)
6) run the naming service, the server, and the client

e-Macao-16-5-47

CORBA Example 1

Running the application:

1) the server is running on the auxiliary PC:
   - run the CORBA naming service:
     `tnameserv -ORBInitialPort 2500`
   - start the server:
     `java FileServer -ORBInitialPort 2500`
2) the client is running on the current PC:
   run
   `run_CORBA_Client.bat`

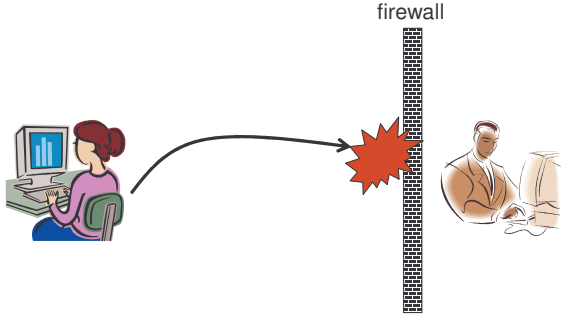


e-Macao-16-5-48

CORBA Example 2

What happens if we enable a firewall on the server side?

Let's try again to run the client application:



e-Macao-16-5-49

Web Service Application

Consider the same application, but built as a web service.

A client requests a file from the server. The server sends the file to the client. When received, the client saves the file on the local machine.

The steps involved:

1) setup the SOAP server
2) develop the server
3) develop the client
4) start the web server
5) deploy the server as a web service
6) run the client application

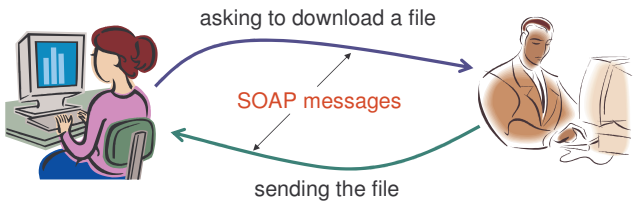
e-Macao-16-5-50

Web Service Example 1

Running the application:

1) the server is running on the auxiliary PC - start the web server
2) the client is running on the current PC

```
run_client.bat
```

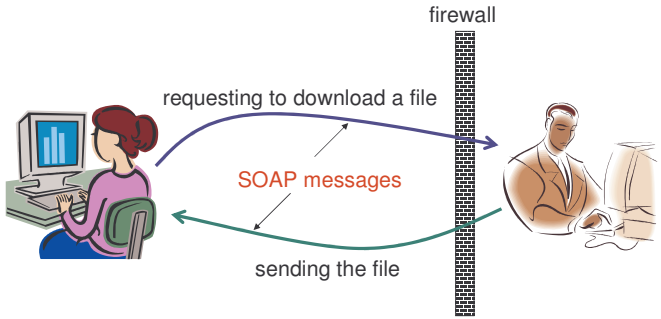


e-Macao-16-5-51

Web Service Example 2

What happens if we enable a firewall on the server side?

Let's try again to run the client application:



e-Macao-16-5-52

Comparison: Communication

What is the observable difference between CORBA and WS applications? With the firewall enabled, the CORBA application was unable to run.

One advantage of SOAP is its explicit definition of HTTP binding through the process of hiding another protocol inside HTTP messages.

This allows SOAP messages to pass through a firewall unimpeded.

Firewalls will usually allow HTTP protocol through port 80, while they will restrict the use of other protocols or ports.

e-Macao-16-5-53

Comparison: Functionality

The same functionality in CORBA and WS.

The difference is how WS provides this functionality:

1) data is formatted for transfer using XML
2) data is passed using standard communication protocols
3) the exposed service is well defined in an XML vocabulary
4) services are found in standard ways with XML vocabularies

WS provides more a flexible design than CORBA.

e-Macao-16-5-54

Comparison: Standards

The main difference with past Distributed Computing Environments is adopted standards and implementations:

1) a standard lookup service – UDDI
2) a standard definition mechanism – WSDL
3) a standard way for two parties to communicate – SOAP

The foundation technology for all three (and more) is XML.

e-Macao-16-5-55

Web Service: Request Message

A request message looks as follows:

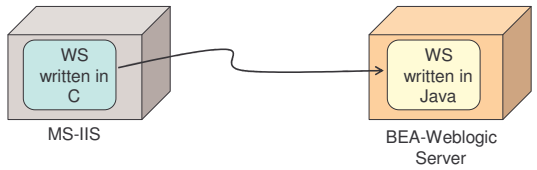
```
<soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:downloadFile
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://soapinterop.org/">
      <ns1:arg0
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string"> name_of_file
      </ns1:arg0>
    </ns1:downloadFile>
  </soapenv:Body>
</soapenv:Envelope>
```

e-Macao-16-5-56

Web Service Implementation

The standards used by web services are defined with little concern for the underlying implementation mechanism.

Therefore: a web service written in C and running on Microsoft IIS can access a web service written in Java, running on BEA WebLogic Server.



e-Macao-16-5-57

Web Service Environments

Several environments exist to build, deploy and access web services.

Best known:

1) Microsoft's .Net Platform
2) Sun's Java 2 Platform

We rely on Java Web Services.

e-Macao-16-5-58

Traditional Communication

Traditional system communication:

1) systems must be tightly bound
2) data must be transferred in such a way that two systems agree beforehand on the format
3) various "network normal forms" were created to decide how bytes, integers, etc. were to be encoded for transfer

e-Macao-16-5-59

XML-Based Communication

Before - no common data-definition mechanism.

With XML:

1) common, well-defined data and representation
2) well-defined set of validity and well-formedness rules

Web service communication relies on XML syntax to write messages.

e-Macao-16-5-60

WS - Business Perspective

Web services and business processes/goals:

1) a web service is an implementation of a business process or a step within such a process
2) a web service is made available over a network to internal and/or external business partners to achieve specific business goals

Web services promote integration of applications within an organization and between different business partners.

Key feature: to allow for rapid construction of business applications by combining web services built internally with those of business partners.

e-Macao-16-5-61

Web Service Usage

Two main scenarios of web service usage:

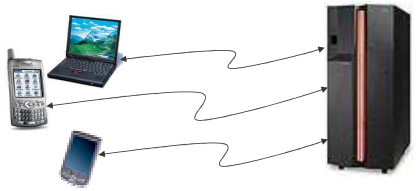
1) application integration
2) B2B partner integration over the Internet

e-Macao-16-5-62

WS Usage: Application Integration

Legacy systems can be wrapped as web services and made available for integration with other systems.

Applications exposed as web services are accessible by other applications running on different hardware platforms and written in different languages.

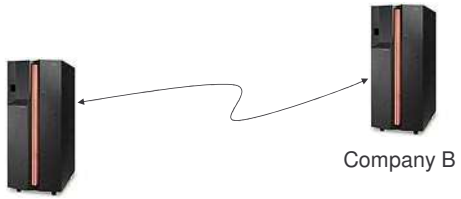


e-Macao-16-5-63

WS Usage: B2B Integration

Business-to-business (B2B) partner integration over the Internet.

B2B integrates business systems of two or more companies to support cross-enterprise business processes, e.g. supply chain management.



e-Macao-16-5-64

WS Properties 1

1) self-contained - no additional software is required for WS:
   a) client-side: a programming language with XML/HTML client support
   b) server-side: a web server and a SOAP server are needed
2) loosely coupled - client and server only knows about messages - a simple coordination level that allows for more flexible re-configuration
3) web-enabled – WS are published, located and invoked across the web, using established lightweight Internet standards
4) language-independent and interoperable - client and server may be implemented in different environments and different languages

e-Macao-16-5-65

WS Properties 2

5) composable - WS can be aggregated using workflow techniques to perform higher-level business functions
6) dynamically bound - with UDDI and WSDL, the discovery and binding of web services can be automated
7) programmatic access - the web services approach does not provide a graphical user interface but operates at the command level
8) wrap existing applications - stand-alone applications can easily be integrated by implementing a web service as an interface

e-Macao-16-5-66

Web Service Benefits

1) platform integration - the platform-neutrality of WS allows combining business systems using different devices (PDAs, cell phones, desktops) with service providers of all sizes and shapes
2) software integration - systems supporting new or modified business processes can be rapidly delivered by wrapping the existing functionality
3) standard technology - open standards enable developers to choose among different products, avoiding vendor-dependence
4) small businesses integration - the low cost of WS allows small businesses to deploy and participate in WS applications
5) easy integration - interface-based development using web service descriptions reduces the time to integrate applications

e-Macao-16-5-67

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

e-Macao-16-5-68

SOA and WS

Service-Oriented Architecture:

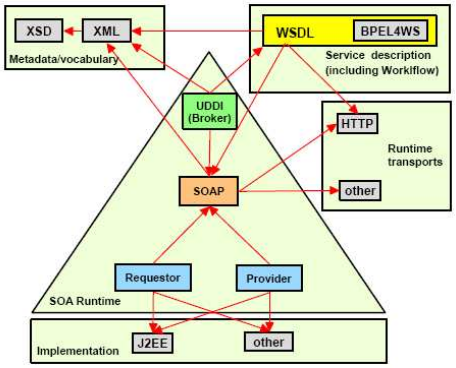
1) provides an approach for building systems focused on a loosely coupled set of components (services) that can be dynamically composed
2) promotes seamless software integration as a business benefit

Web Services:

1) one approach to building SOA
2) provide a standard for a particular set of XML-based technologies that can be used to build SOA systems

e-Macao-16-5-69

WS-Based Approach to SOA



[courtesy IBM]

e-Macao-16-5-70

Using SOA and WS

SOA and WS are most appropriate for the applications that:

1) can operate over the Internet, accepting that reliability and performance of communication cannot be guaranteed in this case
2) do not require that all service requestors and providers are upgraded at the same time
3) consist of the components running remotely on different execution platforms and vendor products
4) were designed using legacy technology but need to be exposed for use over a network, using a web service wrapping

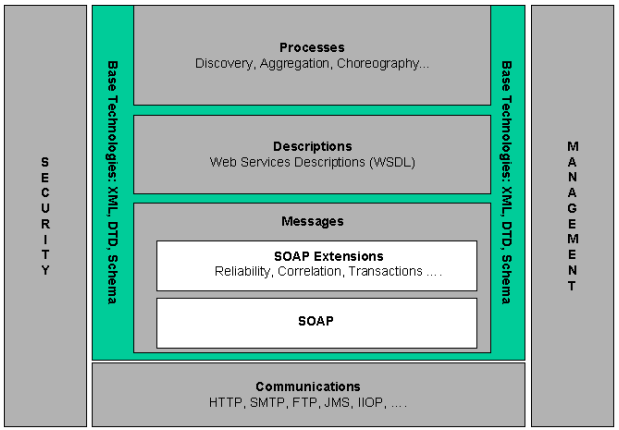
e-Macao-16-5-71

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

e-Macao-16-5-72

Web Services Architecture Stack



[courtesy W3C]

e-Macao-16-5-73

Communications Layer

Web Services are essentially transport-neutral.

A web service message can be transported using HTTP or HTTPS, as well as more specialized transport mechanisms, such as e.g. JMS.

Web services insulate the designer from most of the details and implications of the message transport layer.

e-Macao-16-5-74

Messaging Layer

SOAP = Simple Object Access Protocol

A protocol to exchange structured information in a distributed environment.

SOAP extensions:

1) WS-ReliableMessaging - a standard for web services messaging to guarantee the receipt of messages for WS requestors and providers

2) WS-Transactions - a series of standards related to WS invocations in transactions (atomicity, consistency, isolation and durability semantics)

e-Macao-16-5-75

Descriptions Layer

WSDL = Web Services Description Language

A language that allows a service provider to specify the functional characteristic of its web services.

WSDL extensions:

1) WS-Policy - augment WSDL with non-functional constraints on WS

2) WS-ResourceProperties – describes how to define and access properties of resources through WS

e-Macao-16-5-76

Processes Layer: Discovery

Discovery - locating a machine-processable description of a web service that may have been previously unknown and that meets certain criteria.

UDDI = Universal Description, Discovery and Integration

UDDI defines a way to store and retrieve information about web services.

e-Macao-16-5-77

Processes Layer: Choreography

Choreography - defines how multiple cooperating independent agents exchange messages in order to perform a task to achieve a given goal.

WS-CDL = WS Choreography Description Language

WS-CDL describes peer-to-peer collaborations where ordered message exchanges result in accomplishing a common business goal.

e-Macao-16-5-78

WS Interoperability

Web Services tackle the set of problems related to loosely coupled, dynamically configured heterogeneous distributed computing.

WS Specifications:

1) A series of smaller, purpose-focused specifications dealing with narrow problems (security, transactions, etc.) in isolation.
2) Each WS specification is designed to be composed with the others.
3) WS designers determines which specifications their system needs and implement them accordingly.

e-Macao-16-5-79

WS-I Organization

Web Services Interoperability organization (WS-I):

1) WS-I is to standardize combinations of WS specifications that can be used to increase the level of interoperability between web services.

2) WS-I promotes the Basic Profile - implementation guidelines for how non-proprietary WS specifications, such as SOAP, WSDL, UDDI, should be used together for best interoperability.

WS-I website - http://www.ws-i.org/

e-Macao-16-5-80

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

e-Macao-16-5-81

Apache Axis

Apache and Axis:

1) Apache is an open-source HTTP server - http://ws.apache.org/
2) Axis is an Open Source SOAP engine - http://ws.apache.org/axis/

Axis converts Java objects to SOAP data for sending/receiving messages.

e-Macao-16-5-82

Apache Axis - Modules

Axis implements the standard Java API for Web Services - JAX-RPC.

Axis:

1) is compiled in the JAR file `axis.jar`
2) implements the JAX-RPC API declared in:
   a) `jaxrpc.jar`
   b) `saaj.jar`

All these files can be packaged into a web application called `axis.war` that can be deployed in a servlet container.

Servlet - Java class that can respond to HTTP requests.

e-Macao-16-5-83

Apache Axis - Requirements

What is needed?

1) Java 1.4
2) Tomcat 4.x

New versions of Java and Tomcat are not supported yet.

e-Macao-16-5-84

Tomcat

What is Tomcat?

1) Servlet container used in the official Reference Implementation of the Java Servlet and JavaServer Pages technologies.
2) A free, open-source implementation.
3) Was developed under the Jakarta project at the Apache Software Foundation.
4) Tomcat reference - http://jakarta.apache.org/tomcat

e-Macao-16-5-85

Installing Apache Axis

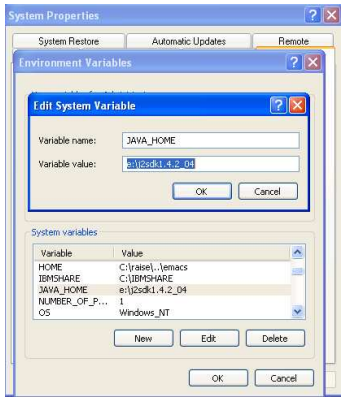
Steps for installing Apache Axis:

1) update the JAVA_HOME variable
2) install Tomcat
3) install Apache Axis
4) deploy Axis
5) validate the installation

e-Macao-16-5-86

Task 3: Update JAVA_HOME

1) pointing at My Computer press the right bottom and select Properties
2) select Advanced and Environment Variables
3) select System Variables and modify JAVA_HOME to contain the path to the j2sdk1.4 installation directory



e-Macao-16-5-87

Task 4: Installing Tomcat

1) visit http://jakarta.apache.org/tomcat
2) select Download - Binaries
3) select Download - Tomcat
4) select Tomcat 4
5) select Binary - 4.1.31.exe
6) save the file:
   jakarta-tomcat-4.1.31.exe
   on your own directory

e-Macao-16-5-88

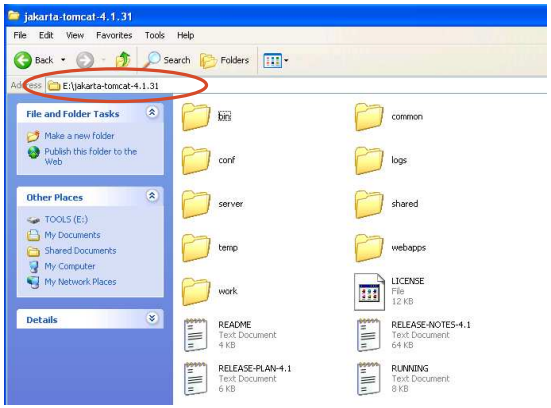
Task 5: Installing Tomcat

7) execute: jakarta-tomcat-4.1.31.exe

8) Answer the following:
   a) `Using Java Development Kit found in j2sdk1.4.2_04` → OK
   b) `To the window about Apache License` → I Agree
   c) `Setup Installation Options:`
      1) `Tomcat`
      2) `JSP Development Shell Extensions`
      3) `Tomcat Start Menu Group`
      4) `Documentation and Examples` → Next
   d) `Destination Folder:` → D:\Tomcat 4.1 or E:\Tomcat 4.1
      Install → Next
   e) `HTTP/1.1 Connector Port:` 8080
      `User name:` admin → Finish

e-Macao-16-5-89

Task 6: Verifying Installation

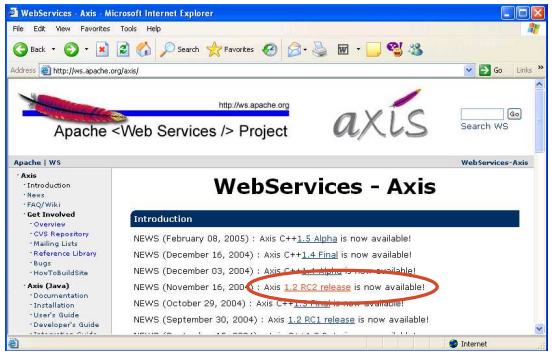


e-Macao-16-5-90

Task 7: Installing Axis 1

1) visit http://ws.apache.org/axis/

2) select Axis 1.2 RC2 release

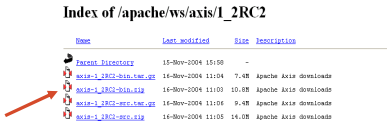


e-Macao-16-5-91

Task 8: Installing Axis 2

3) select the Apache download mirrors

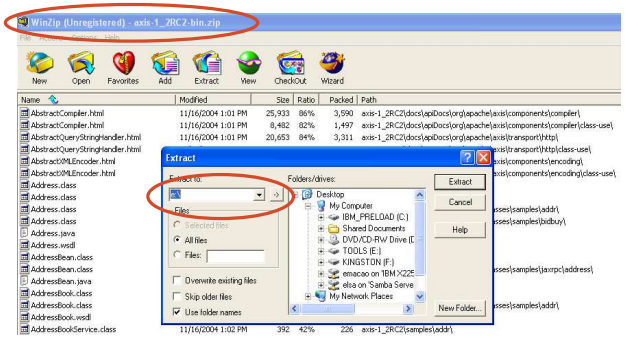
4) download the file: axis-1_2RC2-bin.zip to your own directory



e-Macao-16-5-92

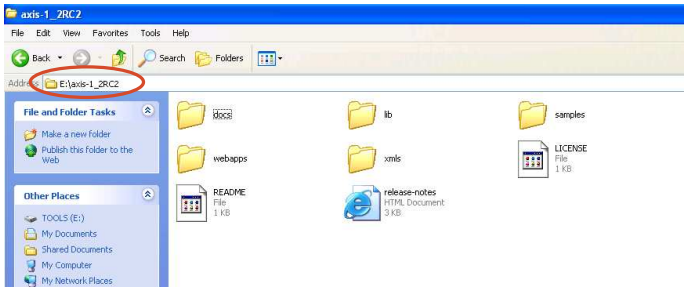
Task 9: Installing Axis 3

5) uncompress: axis-1_2RC2-bin



e-Macao-16-5-93

Task 10: Verify Installation

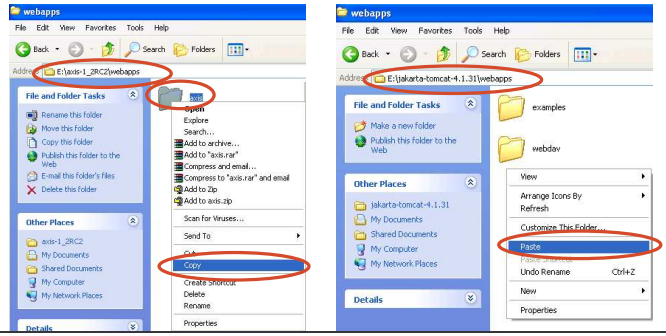


e-Macao-16-5-94

Task 11: Deploy Axis

In order to deploy Apache Axis in Tomcat:

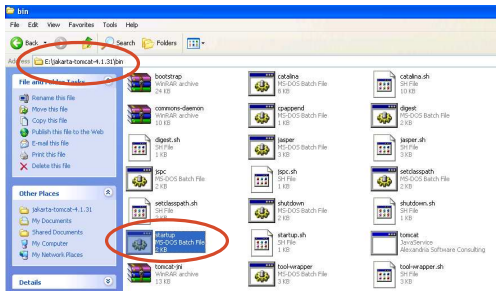
1) in E:\axis-1_2RC2\webapps, copy the folder: axis
2) in E:\jakarta-tomcat-4.1.31\webapps, paste the folder: axis



e-Macao-16-5-95

Task 12: Validate Installation 1

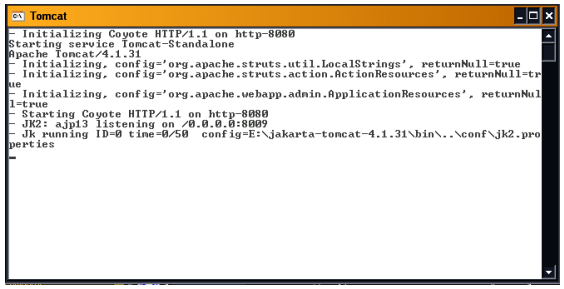
Start Tomcat - double-click E:\jakarta-tomcat-4.1.31\bin\startup.bat



e-Macao-16-5-96

Task 13: Validate Installation 2

Starting Tomcat will cause the following window to appear:

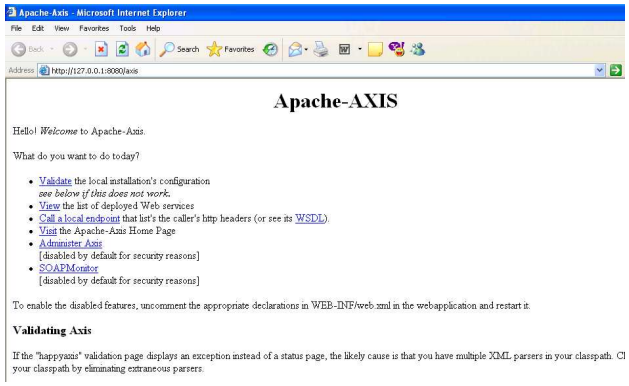


(colors are inverted)

e-Macao-16-5-97

Task 14: Validate Installation 3

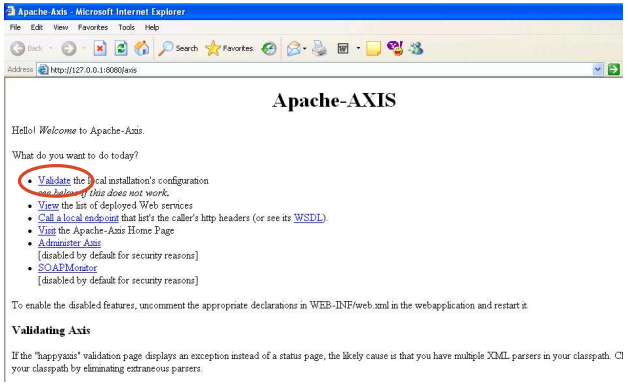
Navigate to the start page of the webapp http://localhost:8080/axis



e-Macao-16-5-98

Task 15: Validate Installation 4

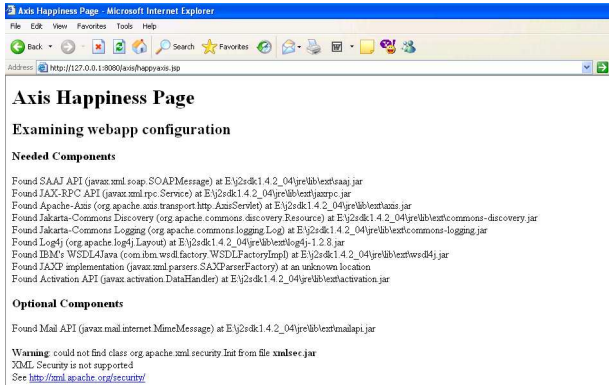
Validate Axis installation - follow the link Validate



e-Macao-16-5-99

Task 16: Validate Installation 5

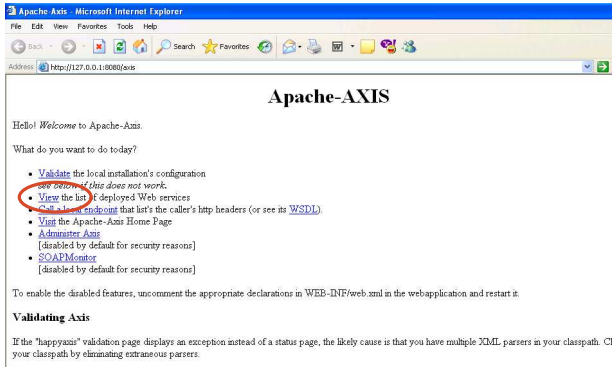
If the installation was successful then the following page is displayed:



e-Macao-16-5-100

Task 17: Executing WS 1

Navigate to the start page of http://localhost:8080/axis and click on View to list the deployed web services:



e-Macao-16-5-101

Task 18: Executing WS 2

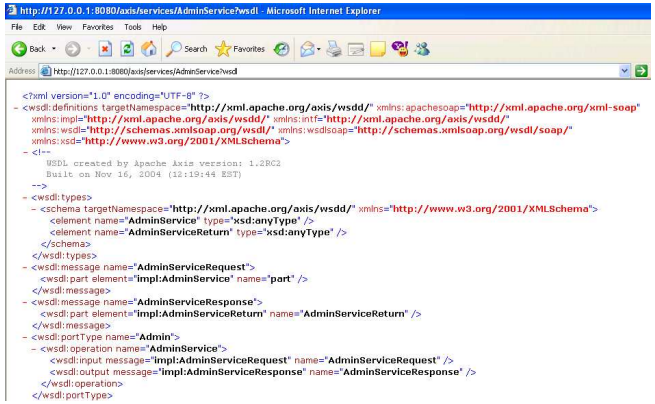
This page is displayed:

Click on AdminService (*wsdl*).

e-Macao-16-5-102

Task 19: Executing WS 3

Here is a web service description in WSDL:



e-Macao-16-5-103

Task 20: Testing WS 1

We are going to invoke the getVersion service which returns a message with the version number of the Axis installation.

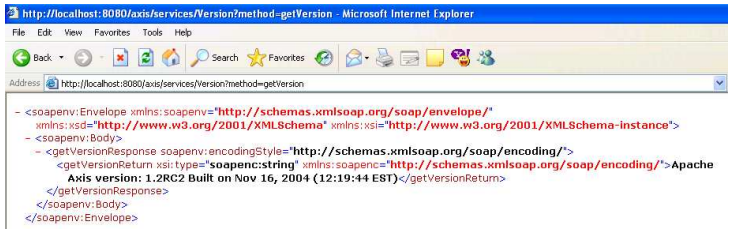
Open the browser at

http://localhost:8080/axis/services/Version?method=getVersion

e-Macao-16-5-104

Task 21: Testing WS 2

In response, the following message is obtained:



A SOAP envelope!

e-Macao-16-5-105

Deploying a Web Service

Axis uses a deployment descriptor to deploy a web service.

A deployment descriptor is an Axis-specific XML file that tells Axis how to deploy (or undeploy) a web service, and how to configure Axis itself.

Deploying a web service:

1) copy the class that is being deployed as a web service to

```
\Tomcat 4.1\webapps\axis\WEB-INF\classes
```

2) write the deployment descriptor

3) run `AdminClient`

e-Macao-16-5-106

WS Deployment: Descriptor 1

To deploy a web service, the root of the XML deployment descriptor document must be the tag `<deployment>`.

The mandatory child of the `<deployment>` element is:

```
<service name="name" provider="provider">
        …
</service>
```

It is used to deploy/undeploy an Axis Service, where:

1) `name` - name of the web service

2) `provider` - specifies the particular provider of the web service such as: Java-RPC, Java-EJB, etc.

e-Macao-16-5-107

WS Deployment: Descriptor 2

The different options of the service may be specified as follows :

```
<parameter name="name" value="value"/>
```

and common ones include:

1) `className` - the backend implementation class

2) `allowedMethods` - each provider can determine which methods are allowed to be exposed as web services

e-Macao-16-5-108

WS Deployment: Descriptor 3

```
<deployment
   xmlns="http://xml.apache.org/axis/wsdd/"
   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="FileDownloadService" provider="java:RPC">
   <parameter name="className" value="FileDownload"/>
   <parameter name="allowedMethods" value="*"/>
 </service>

</deployment>
```

e-Macao-16-5-109

WS Deployment: AdminClient

With Tomcat running, execute AdminClient:

```
java org.apache.axis.client.AdminClient deploy.wsdd
```

where:

1) `deploy.wsdd` is the name of the deployment descriptor.

2) AdminClient is a tool that comes with Axis that allows to deploy/undeploy web services and to configure the Axis engine.

e-Macao-16-5-110

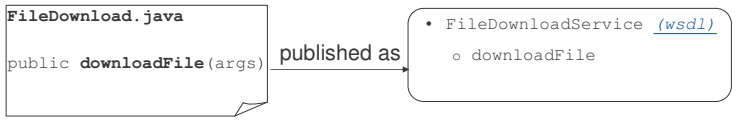
WS Deployment Example

We developed a web service that allows to download a file:

1) Java class is FileDownload.

2) The class has one method downloadFile that transmits a file.

3) The name of the web service is FileDownloadService.



e-Macao-16-5-111

Task 22: WS Deployment 1

1) change and check directory

```
> cd demos\WS\deployWebService
> dir
      deploymentDescriptor.wsdd
deployService.bat
FileDownload.class
```

2) Copy the Java class
`demos\WS\deployWebService\FileDownload.class`
to `\Tomcat 4.1\webapps\axis\WEB-INF\classes`

e-Macao-16-5-112

Task 23: WS Deployment 2

3) Edit the file `deploymentDescriptor.wsdd`:

- changing `name_of_the_service` to `FileDownloadService`
- changing `name_of_the_class` to `FileDownload`
- and save it as `FileDownloadDescriptor.wsdd`

4) Edit the file `deployService.bat`

- changing `deploymentdescriptor.wsdd` into `FileDownloadDescriptor.wsdd`
- and save it as `deployFileDownload.bat`

5) Execute `deployFileDownload.bat`.

e-Macao-16-5-113

Task 24: WS Testing

1) view the list of the deployed web services at http://localhost:8080/axis
2) select View
3) click on the wsdl corresponding to `FileDownloadService`
4) this is the service description prepared by Axis for your service
5) in the address of the browser, remove `?wsdl` at the end of the line
6) this is the execution of your web service

e-Macao-16-5-114

Introduction Outline

e-Macao-16-5-115

Introduction Summary 1

SOA is a Service-Oriented Architecture where software business processes are defined as services to be consumed over a network.

SOA defines three roles for agents: service provider, service requestor and service registry, and three operations: publish, find and bind.

Web Services (WS) is one approach to implementing SOA.

WS-based SOA produce a loosely coupled and flexible applications.

e-Macao-16-5-116

Introduction Summary 2

Web Services are software applications that use XML to exchange data with other applications.

Web Services provide a seamless integration framework of software, execution platforms and businesses.

Web Services use open standards technologies such as HTTP and SOAP for communication, WSDL for definition and UDDI for identification.

e-Macao-16-5-117

Introduction Summary 3

WS are build on three technologies:

1) SOAP - a protocol for exchanging structured information in a distributed system based on XML
2) WSDL - a language for describing web services
3) UDDI - a specification that defines a way to store and retrieve information about web services

Other purpose-focused specifications are available, such as WS-Security, WS-Reliable Messaging and others.

A system designer has to determine which specifications are needed for the system and implement or deploy them accordingly.

e-Macao-16-5-118

Introduction Summary 4

Axis is a SOAP engine, a free and open-source product from Apache.

Axis runs as an application of the Tomcat web server.

Axis provides a deployment descriptor that allows to deploy web services.

A.3. SOAP

e-Macao-16-5-119

SOAP

e-Macao-16-5-120

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

A.3.1. Introduction

e-Macao-16-5-121

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

e-Macao-16-5-122

SOAP History 1

1997	Microsoft considers supporting XML-based distributed computing consisting of applications communicating via RPC using standard data types on top of XML/HTTP.
1998	DevelopMentor (a Microsoft ally) and Userland (a private company) joined the discussion inventing the SOAP name. Within Microsoft, the process was stalled: some people promoted the DCOM wire protocol via HTTP tunneling, instead of pursuing XML.
1998	Userland publishes a version of the SOAP specification as XML-RPC.
1999	SOAP 1.0 appeares, entirely based on HTTP.

e-Macao-16-5-123

SOAP History 2

2000	SOAP 1.1 is submitted as a note to W3C with IBM as a coauthor – a more generic version including other protocols: 1) IBM immediately releases a Java SOAP implementation that was donated to the Apache XML Project for open source development. 2) Sun voices support to SOAP and started working on integrating Web Services into J2EE platform. 3) Many vendors also begin working on WS implementations.
2001	First draft of SOAP 1.2 is presented.
2003	SOAP 1.2 becomes a W3C recommendation.

e-Macao-16-5-124

SOAP Definition

W3C (World Wide Web Consortium) definition:

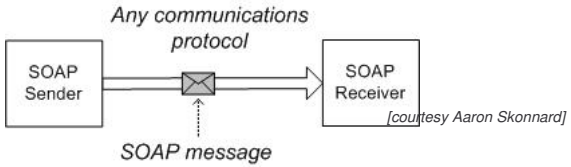
- SOAP is a lightweight protocol intend for exchanging structured information in a decentralized, distributed environment.
- SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols.
- The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

[SOAP Version 1.2 Part1:Messaging Framework http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/]

e-Macao-16-5-125

SOAP Features

SOAP defines a way to move XML messages from point A to point B:



[courtesy Aaron Skonnard]

It does this by providing an XML-based messaging framework that is:

1) extensible
2) usable over a variety of underlying networking protocols
3) independent of programming models

e-Macao-16-5-126

Features: Extensible

1) SOAP is simple by design
2) SOAP lacks various distributed system features:
   - security
   - routing
   - transactions
   - etc.
3) SOAP defines a communication framework that allows additional features to be added as layered extensions.

e-Macao-16-5-127

Features: Protocol-Independent

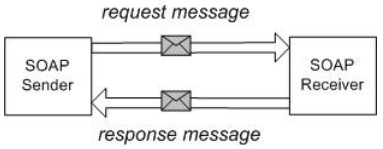
4) SOAP can be used over any protocol:
   - TCP
   - HTTP
   - SMTP
   - etc.
5) SOAP provides a flexible framework for defining bindings to arbitrary protocols to maintain interoperability.
6) SOAP provides an explicit binding for HTTP.

e-Macao-16-5-128

Features: Model-Independent

SOAP is independent of any distributed programming model:

7) allows for any programming model not tied to RPC
8) defines a model for processing individual, one-way messages, or combine multiple messages into an overall message exchange



[courtesy Aaron Skonnard]

9) allows for any number of message exchange patterns: request/response, solicit/response, notifications, peer-to-peer

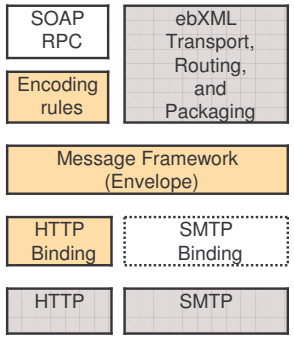
e-Macao-16-5-129

SOAP and ebXML

SOAP:
1) messaging framework
2) encoding rules
3) binding to HTTP protocol

ebXML:
1) messaging framework
2) SOAP bindings
3) own encoding rules

e-Commerce solutions with XML.



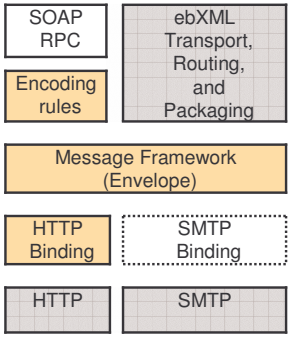
e-Macao-16-5-130

SOAP Toolbox

SOAP is like a toolbox.

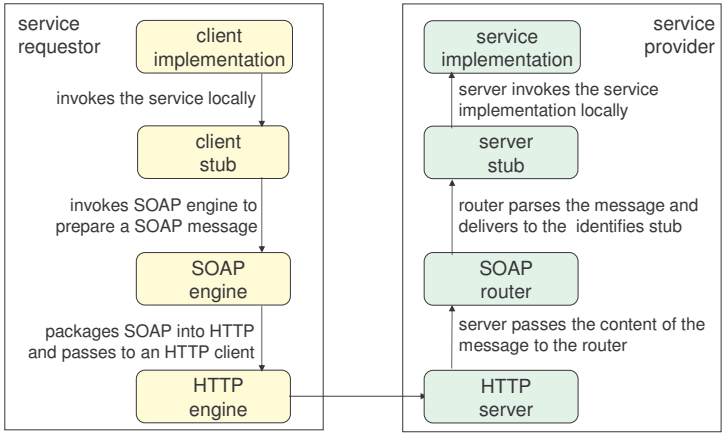
SOAP requests could be made:

- with the SOAP envelope, HTTP binding and some encoding or
- with the SOAP envelope, SOAP encoding and SMTP or
- any other combinations



e-Macao-16-5-131

SOAP Implementation Model



A.3.2. Messaging

e-Macao-16-5-132

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

e-Macao-16-5-133

SOAP Message

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <getVersionResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <getVersionReturn xsi:type="soapenc:string"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
         Apache Axis version: 1.2RC2 Built on Nov 16, 2004 …
      </getVersionReturn>
   </getVersionResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

An XML document!

e-Macao-16-5-134

SOAP Messaging

The SOAP messaging framework defines a suite of XML elements for "packaging" arbitrary XML messages for transport between systems:

1) envelope
2) header
3) body
4) fault
5) etc.

e-Macao-16-5-135

SOAP Namespaces

All XML elements belong to the following namespaces:

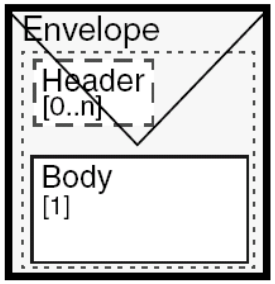
1) SOAP 1.1 - http://schemas.xmlsoap.org/soap/envelope
2) SOAP 1.2 - http://www.w3.org/2003/05/soap-envelope

e-Macao-16-5-136

SOAP Envelope 1

A SOAP message is an envelope with zero or more headers and one body:

1) envelope is a container for control information, recipient address and the message itself
2) headers contain control information
3) body contains the message information



[courtesy IBM]

e-Macao-16-5-137

SOAP Envelope 2

`Envelop` is always the root element of a SOAP message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope /">
  <soap:Header>...</soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

The namespace is specified in the envelope for:

1) defining the envelope elements
2) controlling the SOAP version

Additional namespaces may be defined as well.

e-Macao-16-5-138

SOAP Header

`Header` is a generic place-holder for application independent information.

A header:

1) provides a mechanism for extending SOAP messages in a decentralized and modular way

2) allows to pass control information to the receiving SOAP server

e-Macao-16-5-139

SOAP Header: Example

This header introduces a namespace and two elements:

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

  <soap:Header>
       <t:transaction xmlns:t="http://example.org/transac">
              <t:loginTime>10:20:00</t:loginTime>
         <t:logoutTime>10:21:00</t:logoutTime>
     </t:transaction>
  </soap:Header>

 ...

</soap:Envelope>
```

e-Macao-16-5-140

SOAP Header Attributes

SOAP 1.2 provides mechanisms to specify who should deal with headers and what to do with them.

For this purpose it includes attributes:

1) `role`
2) `mustUnderstand`
3) `relay`

Also it is possible to define:

4) `encodingStyle`

SOAP 1.1 has `actor` attribute instead of `role`, with the same semantic.

e-Macao-16-5-141

Mandatory/Optional Headers

Headers may be mandatory or optional.

If a header is mandatory:

1) the receiver must process the header

2) if the receiver is unable to process the header, it must fail

`mustUnderstand` attribute indicates if a header is mandatory or optional.

e-Macao-16-5-142

SOAP Body

The SOAP `Body` element represents a mechanism for exchanging information intended for the ultimate recipient of the message.

`Body` represents the message payload – a generic container that includes any number of elements from any namespace.

In the simplest case the body of a SOAP message includes:

- message name
- reference to a service instance
- parameters with values and optional type references

e-Macao-16-5-143

SOAP Body: Request Example

Request message to transfer funds between bank accounts:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <x:TransferFunds xmlns:x="urn:examples-org:banking">
      <x:from>983-23456</x:from>
      <x:to>672-24806</x:to>
      <x:amount>1000.00</x:amount>
    </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-144

SOAP Body: Response Example

Response message send back to the sender:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <x:TransferFundsResponse xmlns:x="urn:examples-org:banking">
      <x:balances>
        <x:account>
          <x:id>983-23456</x:id>
          <x:balance>34.98</x:balance>
        </x:account>
        <x:account>
          <x:id>672-24806</x:id>
          <x:balance>1267.14</x:balance>
        </x:account>
      </x:balances>
    </x:TransferFundsResponse>
  </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-145

Task 25: Sending Request 1

Objective:

Send a SOAP message to `FileDownloadService` deployed in the Introduction, asking to download a file.

The request message contains:

1) a header - contains user name and password
2) body - contains the method invocation

The client application has two command-line parameters:

1) the path to the downloaded file
2) the name to save this file on the client machine

e-Macao-16-5-146

Task 26: Sending Request 2

```
> cd demos\SOAP\Request

> dir
FileTransferRequest.class
MacaoNews.txt

> mkdir E:\WebServices

> copy MacaoNews.txt to \WebServices

> java -cp \demos\SOAP\Request FileTransferRequest
     E:\WebServices\MacaoNews.txt
     news.txt
```

e-Macao-16-5-147

Task 27: Sending Request 3

Based on the request message:

1) what is the SOAP version?
2) what is the structure of the header?
3) what is the user name?
4) what is the password?
5) what is the structure of the body?

```
> dir
```

e-Macao-16-5-148

Task 28: Receiving Response

Objective: receive a response to the message sent.

```
> cd demos\SOAP\Response

> dir
FileTransferResponse.class

> java -cp \demos\SOAP\Response FileTransferResponse
     \WebServices\MacaoNews.txt
     news.txt
```

e-Macao-16-5-149

SOAP Fault

The `Fault` element is used to represent errors:

1) processing errors
2) errors understanding a mandatory header
3) all abnormal situations

Faults are specified within the body of a SOAP message.

e-Macao-16-5-150

SOAP Fault: Example

Response message with the `Insufficient funds` error:

```
<soap:Envelope
      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
            <soap:Fault>
            <soap:Code>
                  <soap:Value>soap:Sender</soap:Value>
            </soap:Code>
            <soap:Reason>Insufficient funds</soap:Reason>
            <soap:Detail>
            <x:TransferError xmlns:x="urn:examples-
 org:banking">
                  <x:sourceAccount>22-342439</x:sourceAccount>
                  <x:transferAmount>100.00</x:transferAmount>
                  <x:currentBalance>89.23</x:currentBalance>
            </x:TransferError>
            </soap:Detail>
      </soap:Fault>
      </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-151

Task 29: Generating a Fault 1

Objective:

Generate a fault message when looking for the service “FileService” instead of “FileDownloadService”.

```
> cd demos\SOAP\Fault

> dir
FileTransferResponse.class

> java -cp \demos\Soap\Fault FileTransferResponse
           \WebServices\Macao.txt
           news.txt
```

e-Macao-16-5-152

Task 30: Generating a Fault 2

Based on the response message:

1) Where is the fault element located?
2) What is the structure of the fault element?

e-Macao-16-5-153

Extending SOAP: Wrong Way

Suppose we want to add authentication information to the message:

```
<soap:Envelope>
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
     <x:TransferFunds xmlns:x="urn:examples-org:banking">
        <x:from>983-23456</x:from>
        <x:to>672-24806</x:to>
        <x:amount>1000.00</x:amount>
        <credentials>
                  <username>dave</username>
                  <password>evad</password>
            </credentials>
     </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

Not the right way: other applications in need of security must develop their own solutions to the problem. Ultimately, interoperability suffers.

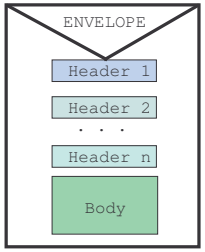
e-Macao-16-5-154

Extending SOAP: Right Way

An envelope wraps whatever XML content is sent in a message.

The header is used to insert message extensions without modifying its body.

Each individual header represents one piece of extensibility information that travels with the message.



e-Macao-16-5-155

Use of Headers

Headers can contain any kind of data.

They are generally used to:

1) extend the messaging infrastructure:
   a) infrastructure headers are processed by middleware
   b) the application does not see the headers, only their effects
   c) examples: security credentials, reliable messaging, etc.

2) define additional data:
   a) these headers are defined by the application
   b) called: vertical extensibility
   c) for instance extra-data to accompany non-extensible schemas

e-Macao-16-5-156

Headers for Extensions

For common needs such as security, it makes more sense to define standard SOAP headers that everyone agrees on.

Then, vendors can build support for the extended functionality into their generic SOAP infrastructure and everyone wins.

```
<soap:Envelope
       xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
       <s:credentials xmlns:s="urn:examples-org:security">
              <s:username>dave</s:username>
              <s:password>evad</s:password>
       </s:credentials>
  </soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

e-Macao-16-5-157

mustUnderstand Attribute

A global SOAP attribute `mustUnderstand` indicates whether or not a receiver is required to understand the header block before processing.

For example:

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
     <s:credentials
              xmlns:s="urn:examples-org:security"
        soap:mustUnderstand="1">
              ...
     </s:credentials>
       </soap:Header>
 ...
</soap:Envelope>
```

e-Macao-16-5-158

mustUnderstand Values

Two values:

1) `mustUnderstand="1"` – if a receiver cannot support the header, a fault should be returned with `soap:mustUnderstand` status code.

2) `mustUnderstand="0"` or `mustUnderstand` attribute is absent, the receiver can ignore those headers and continue processing.

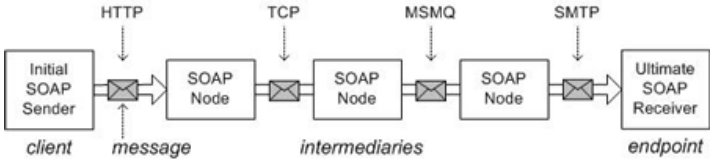
It may also have values `false` (`0`) and `true` (`1`).

e-Macao-16-5-159

SOAP Processing Model

SOAP defines a processing model that outlines rules for processing a SOAP message as it travels from a SOAP sender to a SOAP receiver.

The model allows for architectures with multiple intermediary nodes:



[courtesy Aaron Skonnard]

e-Macao-16-5-160

SOAP Nodes

A SOAP node can be:

1) initial SOAP sender
2) ultimate SOAP receiver
3) SOAP intermediary

e-Macao-16-5-161

SOAP Intermediaries

SOAP intermediaries:

1) they are applications that can process parts of a SOAP message as it travels from its origination point to its final destination point

2) can accept and forward SOAP messages, and usually they do carry out some form of message processing

The route taken by a SOAP message, including all intermediaries it passes through, is called the SOAP message path.

e-Macao-16-5-162

Reasons for Intermediaries 1

There are three major reasons for using intermediaries:

1) crossing-trust domains

   nodes that allow some requests to cross the trust domain boundary and deny access to others

2) ensuring scalability

   a) nodes that provide flexible buffering and routing of messages based on message parameters
   b) nodes that provide information about network traffic and the availability and load of network nodes

e-Macao-16-5-163

Reasons for Intermediaries 2

3) providing special services:

   a) encrypting and digitally signing a message, or decrypting and checking the digital signature

   b) making a persistent copy of the request message, providing a token that can be used to reference the transaction in the future (notarization or non-repudiation)

   c) enabling to find out the path that the message has followed, with arrival and departure times to and from intermediaries

e-Macao-16-5-164

Intermediaries: Sender-View

Message senders may or may not be aware of intermediaries:

1) transparent intermediary - the client knows nothing about it, it believes the message is sent to the service end-point.

   The security intermediary would likely be transparent.

2) explicit intermediary – it involves specific knowledge on the part of the client. The client knows the message will pass through the intermediary.

   The notarization intermediary would likely be explicit.

e-Macao-16-5-165

Intermediaries: Process-View

Intermediaries also differ with respect to processing:

1) forwarding intermediaries - nodes doing specific processing based on the contents of the incoming message.

   For instance a notarization node making a copy of the message based on what is defined on a particular header.

2) active intermediaries - nodes doing processing and eventually modifying the message in the ways not defined by the message contents.

   For instance a node at a boundary of a company to the outside world adding digital signatures to all outbound messages.

e-Macao-16-5-166

Nodes and Roles

While processing a message a SOAP node assumes one or more roles that influence how the headers are processed.

A SOAP node has its role declared.

When it receives a message for processing:

1) it must process all mandatory headers targeted at one of its roles
2) it may process any optional headers targeted at one of its roles

e-Macao-16-5-167

Role Attribute

The `role` attribute is defined optionally in the header element:

1) The value of `role` is a URI that identifies the name the intermediary who should handle the header entry.
2) The URI might mean:
   a) a particular node - the server `XY` or
   b) a class of nodes - any cache manager along the message path
3) A node can play multiple roles, e.g. `XY` server as a cache manager.

e-Macao-16-5-168

Predefined Roles

SOAP defines three special values for `role`:

1) `http://www/w3.org/2002/06/soap-envelope/role/next`

   Each SOAP intermediary and the ultimate SOAP receiver must act in this role and MAY additionally assume zero or more other roles.

2) `http://www/w3.org/2002/06/soap-envelope/role/ultimateReceiver`

   The final recipient of the SOAP message - processes the body. The end-receiver must act in this role. Intermediaries must not act in this role.

3) `http://www/w3.org/2002/06/soap-envelope/role/none`

   SOAP nodes must not act on this role. Headers addressed to this role should never be processed. They are used to carry data.

e-Macao-16-5-169

Roles: Example 1

A SOAP message with roles:

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

Mandatory header targeted at a SOAP node that plays the `http://example.org/security` role:

```
   <a:Security
      xmlns:a="http://example.com"
      soap:role="http://example.com/security"
      soap:mustUnderstand="true" >
      …
   </a:Security>
```

e-Macao-16-5-170

Roles: Example 2

Optional header targeted at the next node in the message path:

```
<b:NextExample xmlns:b="http://example.com"
    soap:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    soap:mustUnderstand="false" >
    …
</b:NextExample>
```

Header targeted at a SOAP node with the `ultimateReceiver` role:

```
<c:NoRoleDef xmlns:c="http://example.com">
    …
</c:NoRoleDef>

</soap:Header>

…
</soap:Envelope>
```

e-Macao-16-5-171

Processing Rules

The contract implied by a header is between the sender and the first node satisfying the role at which it is targeted.

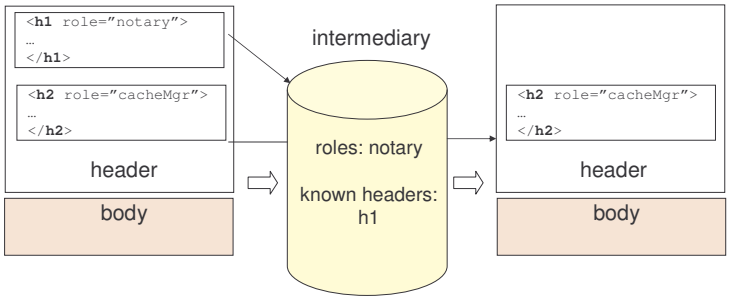
Two rules:

1) if a SOAP node successfully processes a header, it is required to remove the header from the message
2) if the SOAP node happens to be the ultimate receiver, it must also process the SOAP body.

SOAP nodes are allowed to reinsert headers, but doing so changes the contract parties – it's now between the current and the next node.

e-Macao-16-5-172

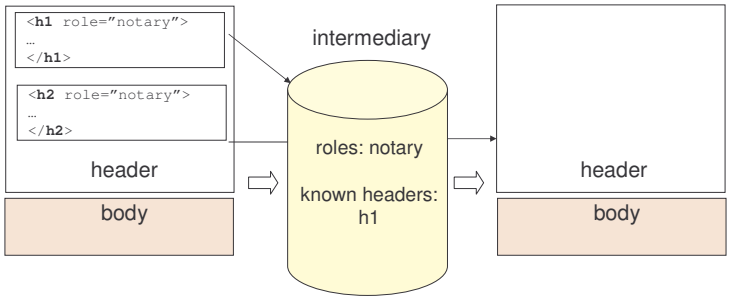
Processing Rules: Example 1



- h1 and h2 are optional headers
- h1 is processed and removed
- h2 is forwarded untouched

e-Macao-16-5-173

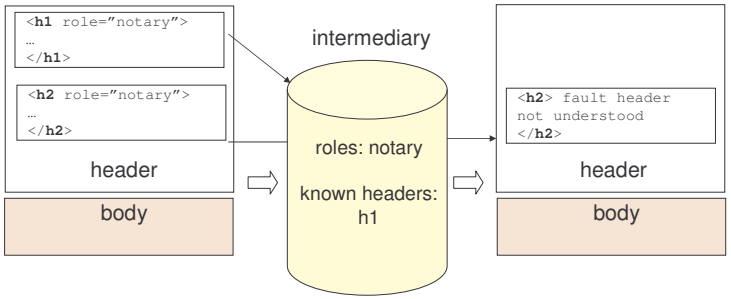
Processing Rules: Example 2



- h1 and h2 are optional headers
- h1 is processed and removed
- h2 is not understood and removed

e-Macao-16-5-174

Processing Rules: Example 3



- h1 and h2 are mandatory headers
- h1 is processed and removed
- h2 is not understood and a fault is generated

e-Macao-16-5-175

Relay Attribute

The `relay` attribute is used to indicate to the intermediaries that:

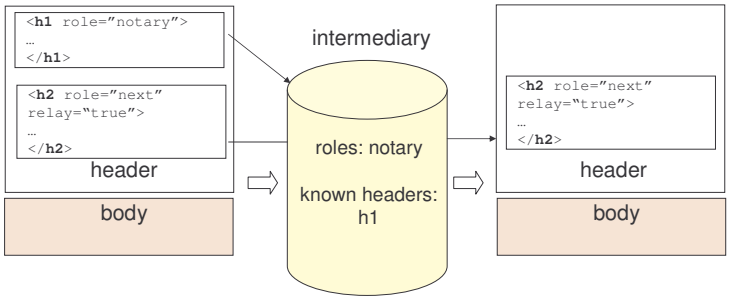
- if a header they do not understand is targeted at them
- then this header should still be passed through

The attribute may equal `true` or `false`.

When a header targeted at a given intermediary has `relay="true"`, it forwards the header regardless of whether it understands it.

e-Macao-16-5-176

Relay Attribute: Example



- h1 is processed and removed
- h2 is forwarded due to the relay attribute

e-Macao-16-5-177

Versioning

SOAP applies XML namespaces to define the protocol version.

The SOAP version is the URI of the SOAP envelope namespace:

1) `http://schemas.xmlsoap.org/soap/envelope` for SOAP 1.1
2) `http://www.w3.org/2003/05/soap-envelope` for SOAP 1.2

An engine supporting a later SOAP version should know previous versions.

The SOAP specification defines processing rules related to SOAP versions.

e-Macao-16-5-178

Versioning: Processing Rules

Rules to follow by the SOAP engine:

- If the message version is the same as a version the engine knows, it should process the message.
- If the message version is older than the one the engine knows, the engine should generate a `VersionMismatch` fault and attempt to negotiate the protocol version with the client.
- If the message version is newer than the one the engine knows, the engine must generate a `VersionMismatch` fault.

e-Macao-16-5-179

Error Handling

A SOAP fault message is a normal SOAP message with a single `Fault` element inside the body.

Components of the `Fault` element include:

1) `Code` - mandatory
2) `Subcodes` - optional
3) `Reason` - mandatory
4) `Node` - optional
5) `Role` - optional
6) `Details` - optional

e-Macao-16-5-180

Fault Elements: Code

The `Code` element includes two sub-elements: a mandatory `Value` element and an optional `Subcode` element.

1) `Value` specifies the type of a fault
2) `Subcode` specifies additional information

e-Macao-16-5-181

Fault Elements: Value 1

Here are the possible values of the Value sub-element:

1) `VersionMismatch`

   The namespace of the received SOAP envelope is not compatible with the SOAP version of the receiver.

2) `mustUnderstand`

   The node does not recognize the block that includes the `mustUnderstand` attribute.

3) `Sender`

   The node cannot process the message because of incorrect or missing data from the sender, e. g. the message is not properly formatted.

e-Macao-16-5-182

Fault Elements: Value 2

4) `Receiver`

The error is not due to the message itself but rather to the state in which the server was when processing the message.

5) `DataEncodingUnknown`

The node does not understand the encoding style.

For example:

```
<soap:Fault>
   <soap:Code>
       <soap:Value>soap:Sender</soap:Value>
   <soap:Code>
   . . .
</soap:Fault>
```

e-Macao-16-5-183

Fault Elements: Subcode 1

SOAP allows developers to specify an arbitrary hierarchy of fault subcodes for providing further details about the fault cause.

The `Subcode` element contains:

1) a mandatory `Value` element and

2) an optional `Subcode` sub-element

Each subcode may contain another subcode, to whatever level of nesting.

e-Macao-16-5-184

Fault Elements: Subcode 2

For example:

```
<soap:Body>
  <soap:Fault>
   <soap:Code>
       <soap:Value>soap:Sender</soap:Value>
       <soap:Subcode>
         <soap:Value>bk:InvalidAccount</soap:Value>
       </soap:Subcode>
   </soap:Code>
  </soap:Fault>
</soap:Body>
```

e-Macao-16-5-185

Fault Elements: Reason

The `Reason` element contains human-readable descriptions of the fault.

It contains the `Text` sub-element which includes the fault description.

`Text` may appear several times inside `Reason`.

```
<soap:Fault>

  <soap:Code>…</soap:Code>

  <soap:Reason>
    <soap:Text xml:lang="en">Processing Error</soap:Text>
    <soap:Text xml:lang="cn">处理错误</soap:Text>
    <soap:Text xml:lang="es">Error de Procesamiento</soap:Text>
  </soap:Reason>

</soap:Fault>
```

e-Macao-16-5-186

Fault Elements: Node and Role

Two subelements of `Fault`:

1) The `Node` element specifies which SOAP node was processing the message when the fault has occurred.

   It contains a URI.

2) The `Role` element specifies which role the node was playing when the fault has occurred.

   `Role` behaves in the same way as the headers' `role` attribute.

e-Macao-16-5-187

Fault Elements: Detail

The `Detail` element includes machine-readable data related to the fault.

```
<soap:Envelope
 xmlns:soap="http:www.w3.org/2003/05/soap-envelope"
 xmlns:ac="http://www.example.com">
 <soap:Body>
       <soap:Fault>
             <soap:Code> . . . </soap:Code>
             <soap:Reason>
                   <soap:Text xml:lang="en">
                         Invalid account!
                   </soap:Text>
             </soap:Reason>
             <soap:Detail>
             <ac:LineNumber>10</ac:LineNumber>
             <ac:ColumnNumber>57</ac:ColumnNumber>
             </soap:Detail>
       </soap:Fault>
 </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-188

Task 31: Generate Sender Fault

Objective:

Send a SOAP message that will generate a fault caused by the wrong operation name specified - "downloadf" instead of "downloadFile".

```
> cd demos\Soap\SenderFault

> dir
FileTransferSenderFault.class

> java -cp \demos\Soap\SenderFault
            FileTransferSenderFault
            \WebServices\MacaoNews.txt news.txt
```

e-Macao-16-5-189

Task 32: Generate Receiver Fault

Objective:

Send a SOAP message that will generate a fault because the server could not find the service "FileService".

```
> cd demos\SOAP\ReceiverFault

> dir
 FileTransferReceiverFault.class

> java -cp \demos\Soap\ReceiverFault
            FileTransferReceiverFault
            \WebServices\MacaoNews.txt news.txt
```

e-Macao-16-5-190

Faults in Headers

Since a fault is a SOAP message, it can also carry headers.

Problem:

1) a message may contain several mandatory headers
2) one node fails to understand one header
3) how can the header causing the fault be identified?

Solution: SOAP introduces the `NotUnderstood` header:

1) `NotUnderstood` is included for each header in the original message that was not understood.
2) The `qname` attribute of `NotUnderstood` specifies the name of the header that was not understood.

e-Macao-16-5-191

Faults in Headers: Example 1

Suppose a SOAP node receives the following message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Header>
      <a:Header1
         xmlns:a="http://example.com/header1"
         soap:mustUnderstand="true"/>
      <b:Header2
         xmlns:b="http://example.com/header2"
         soap:mustUnderstand="true"/>
   </soap:Header>
   <soap:Body>…</soap:Body>
</soap:Envelope>
```

If the SOAP node does not understand `Header2`,
it would return a message as follows…

e-Macao-16-5-192

Faults in Headers: Example 2

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xml="http://www.w3.org/XML/1998/namespace">
   <soap:Header>
      <soap:NotUnderstood
         qname= "b:Header2"
         xmlns:b="http://example.com/header2"/>
   </soap:Header>
   <soap:Body>
      <soap:Fault>
         <soap:Code>
            <soap:Value>soap:mustUnderstand</soap:Value>
         </soap:Code>
         <soap:Reason>
            <soap:Text xml:lang="en">
               One or more mandatory headers not understood!
            </soap:Text>
         </soap:Reason>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-193

Upgrade Header

SOAP provides a standard mechanism to indicate which versions of SOAP are supported by a node when generating a `VersionMismatch` fault.

1) An `Upgrade` header is used when a version mismatch fault occurs, to specify which SOAP versions are supported by the node.

2) The different supported version are specified in the `SupportedEnvelope` sub-element of `Upgrade`.

3) The `SupportedEnvelope` elements are ordered by preference, from the most preferred to the least.

e-Macao-16-5-194

Upgrade Header: Example

```
<soap:Envelope
       xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
       xmlns:xml="http://www/w3.org/XML/1998/namespace">
       <soap:Header>
              <soap:Upgrade>
                     <soap:SupportedEnvelope qname="ns1:Envelope"

       xmlns:ns1="http://www.w3.org/2003/05/soap-envelope" />
                     <soap:SupportedEnvelope qname="ns2:Envelope"

       xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/" />
              </soap:Upgrade>
       </soap:Header>
       <soap:Body>
              <soap:Fault>
                     <soap:Code>
                            <soap:Value>VersionMismatch</soap:Value>
         </soap:Code>
         <soap:Reason>
            <soap:Text xml:lang="en">Version Mismatch </soap:Text>
         </soap:Reason>
       </soap:Fault>
```

A.3.2. Data Structures

e-Macao-16-5-195

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

e-Macao-16-5-196

Data Model and Encoding

In order to be able to send Java and others programming language objects inside SOAP envelopes, SOAP defines:

1) SOAP Data Model - an abstract representation of the data structures such as the ones handled by Java or C#

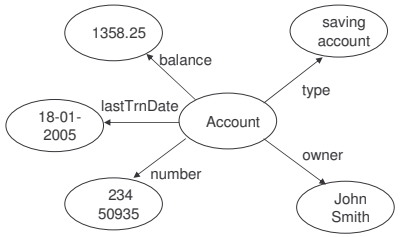
2) SOAP Encoding - a set or rules to map the data model into XML for sending the data inside SOAP envelopes

e-Macao-16-5-197

Data Model

The SOAP data model represents data structures as connected graphs, where nodes represent values and edges represent labels.

SOAP data model



Java object

```
class Account {
   int number;
   String owner;
   String type;
   double balance;
   int lastTrnDate;
}
```

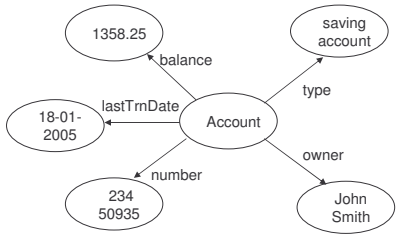
e-Macao-16-5-198

Simple Values

Simple values are nodes with only incoming edges.

They correspond to basic data types found in most programming languages, such as `int`, `string`, etc.

For instance `type`, `balance`, `lastTrnDate`, `number`, or `owner` below are all simple values:



e-Macao-16-5-199

Compound Values

Compound values are nodes with outgoing edges.

There are two types of compound values:

1) structures
2) arrays

e-Macao-16-5-200

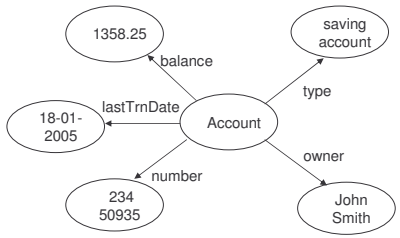
Compound Values: Structures

Structures are compound values where the outgoing edges have names.

They correspond to the named aggregated types.

Each element has a unique name called accessor, which is an XML tag.

`Account` below is a structure:

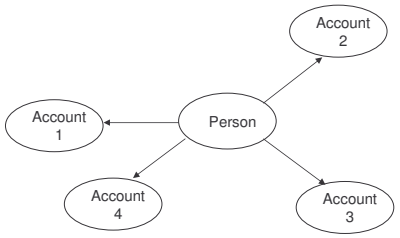


e-Macao-16-5-201

Compound Values: Arrays

Arrays are compound values where the outgoing edges are only distinguished by their position (first edge, second edge, etc.).

For instance:

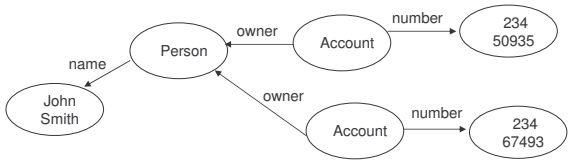


e-Macao-16-5-202

Multirefs

Multirefs is a value which is referred from more than one value.

For instance: John Smith is the owner of two different accounts below



e-Macao-16-5-203

Encoding

SOAP encoding describes how the SOAP data model is written with XML.

SOAP encoding is identified by the URI
`http://www.w3.org/2003/05/soap-encoding`.

When serializing XML using encoding rules, processors should use the `encodingStyle` attribute to indicate the SOAP encoding in use.

The `encodingStyle` attribute can appear in:

1) message headers
2) message bodies
3) `Detail` sub-element of `Fault`

or any of their children.

e-Macao-16-5-204

Encoding Example

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>

      <ns1:downloadFileResponse
         soapenv:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:ns1="http://soapinterop.org/">
         <downloadFileReturn
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
            xsi:type="soapenc:base64">
            TW9..QogDQo=
         </downloadFileReturn>
      </ns1:downloadFileResponse>

   </soapenv:Body>
</soapenv:Envelope>
```

e-Macao-16-5-205

Task 32: Encoding

1) browse: http://www.w3.org/2003/05/soap-encoding

2) Based on the encoding rule definitions:

   a) what are the different values of a node type?
   b) what are the possible attributes for an array?
   c) what is base64?

e-Macao-16-5-206

Encoding Rule

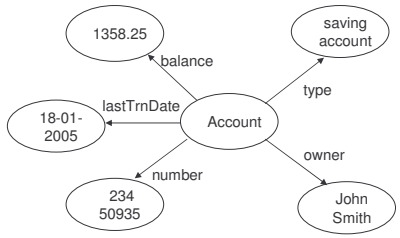
Each outgoing edge becomes an XML element which contains:

1) a text value, if the edge points to a terminal node
2) further sub-elements, if the edge points to a node which itself has outgoing edges.

e-Macao-16-5-207

Encoding Rule Example

```
<account
  soapenv:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  <number>23450935</number>
  <owner>John Smith</owner>
  <type>saving account</type>
  <balance>1358.25</balance>
  <lastTrnDate>20050118</lastTrnDate>
</account>
```



e-Macao-16-5-208

Encoding a Simple Value

For example, in a SOAP message containing:

```
<arg0 xsi:type="xsd:string">
   c:\WebServices\MacaoNews.txt
</arg0>
```

1) `xsi:type` means that `<arg0>` will take string values

2) `xsd:string` is the XML schema string type

In general, all encoded elements provide the `xsi:type` attribute to help recipients decode a message.

e-Macao-16-5-209

Simple Value Example

Java:

```
float balance=1358.25;
String owner="John Smith";
```

SOAP Encoding:

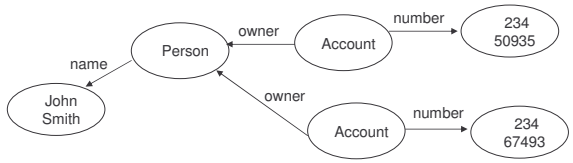
```
<balance xsi:type="xsd:float">
   1358.25
</balance>
<owner xsi:type="xsd:string">
   John Smith
</owner>
```

e-Macao-16-5-210

Encoding Multirefs

An ID attribute is used to identify objects that are referred to elsewhere.

```
<person id="1"
   soapenv:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
   <name>John Smith</name>
   <account id="2">
      <number>23450935</number>
      <owner>ref="1"</owner>
   </account>
   <account id="3">
      <number>23467493</number>
      <owner>ref="1"</owner>
   </account>
</person>
```



e-Macao-16-5-211

Encoding Arrays

An array in the SOAP object model is encoded in XML using a compound element with two attributes:

1) `itemType` - specifies the data type of the array elements
2) `arraySize` - specifies how many elements are in the array

For example:

```
<myAccounts
   soapenc:itemType="xsd:integer"
   sopaenc:arraySize="3">
   <item>23450935</item>
   <item>23467493</item>
   <item>23426741</item>
</myAccounts>
```

e-Macao-16-5-212

Encoding Multidimensional Arrays

Multidimensional arrays are supported by listing each dimension in the `arraySize` attribute separated by spaces.

The values are serialized as a single list of items in row-major order:

```
<myArray
    soapenc:itemType="xsd:string"
    sopaenc:arraySize="2 3">
    <item>blue</item>
    <item>yellow</item>
    <item>white</item>
    <item>red</item>
    <item>pink</item>
    <item>green</item>
</myArray>
```

blue	yellow	white
red	pink	green

e-Macao-16-5-213

Task 33: Encoding 1

Objective:

Send a request to receive the creation date, contents, size and name of a given file. Accept an object of the `FileAttribute` class in response.

Serialize and encode the `FileAttribute` object before sending.

```
> cd SOAP\Encoding

> dir
deployFileDownloadEncodedService.wsdd
deployService.bat
FileAttribute.class
FileDownloadEncoded.class
FileTransferRequestEncoding.class
FileTransferResponseEncoding.class
responseFormatted.txt
```

e-Macao-16-5-214

Task 34: Encoding 2

Deploy the web service:

```
> copy FileAttribute.class
             Tomcat 4.1\webapps\axis\WEB-INF\classes

> copy FileDownloadEncoded.class
             Tomcat 4.1\webapps\axis\WEB-INF\classes
```

Double-click `deployService.bat`

Test the web service: browse `http://localhost:8080/axis` and `View.`

e-Macao-16-5-215

Task 35: Encoding 3

Run the request and send the output to a log file (request.txt):

```
> java -cp \demos\SOAP\Encoding
             FileTransferRequestEncoding
             \WebServices\MacaoNews.txt > request.txt
> notepad request.txt
```

Run the response and send the output to a log file (response.txt):

```
> java -cp \demos\SOAP\Encoding
             FileTransferResponseEncoding
             \WebServices\MacaoNews.txt > response.txt
> notepad response.txt
```

Analize the response:

```
> notepad responseFormatted.txt
```

e-Macao-16-5-216

Encoding a Request

A SOAP request message is modeled as a structure with an accessor element for each input and output parameter:

```
<getBalance
   encodingStyle="http://www/w3.org/2003/05/soap-encoding">
   <accountNumber xsi:type="xsd:int">
      23450935
   </accountNumber>
</getBalance>
```

1) the only accessor is `accountNumber`
2) accessor names correspond to the names of parameters, their `type` attributes correspond to the programming language data types
3) parameters must appear in the same order as in the method signature
4) the name of the structure element is the procedure or method name

e-Macao-16-5-217

Encoding Specific Faults

SOAP defines some fault codes specifically for encoding problems.

These are recommended values to be sent in the `Subcode` value when the `Sender` code is used.

They all relate to problems with the sender's data serialization.

1) `MissingID` – generated when a `ref` attribute in the received message does not correspond to any of the `id` attributes in the message

2) `DuplicateId` – generated when more than one element in the message has the same `id` attribute value

3) `UntypedValue` – generated optionally to indicate that a type in the received message could not be determined by the receiver

e-Macao-16-5-218

Encoding RPC Request

Using a SOAP structure to represent a method call:

```
public float getBalance(int arg)
```

A request message representing a call to this method in SOAP is:

```
<soap:Envelope>
        <soap:Body>
                <orgNS:getBalance
                        xmlns:orgNS="http://myOrganization.com/"

        soap:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
                        <arg0 xsi:type="xsd:int">23450935</arg>
                </orgNS:getBalance>
        </soap:Body>
</soap:Envelope>
```

The structure contains one accessor for each argument.

The content of the `arg` element is the value for the argument.

e-Macao-16-5-219

Encoding RPC Response

The response is also modeled as a structure which name is the method name with `Response` element appended.

Here is a possible response to the previous message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
       <orgNS:getBalanceResponse
           xmlns:orgNS="http://myOrganization.com/"
           xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
           soap:encodingStyle=
               "http://www/w3.org/2003/05/soap-encoding">
           <rpc:result>ret</rpc:result>
           <ret xsi:type="xsd:decimal">1358.25</ret>
       </orgNS:getBalanceResponse>
   </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-220

Encoding RPC Return Values

SOAP specifies that an RPC response structure containing a return value must contain an accessor element called `result`.

The value of this element specifies the accessor's name containing the return value for the invocation.

```
<soap:Body>
        <orgNS:getBalanceResponse
                xmlns:orgNS="http://myOrganization.com/"
                xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
                soap:encodingStyle="http://www/w3.org/2003/05/soap-
 encoding">
                <rpc:result>ret</rpc:result>
                <ret xsi:type="xsd:decimal">1358.25</ret>
        </orgNS:getBalanceResponse>
</soap:Body>
```

e-Macao-16-5-221

Encoding RPC Out Parameters

Suppose we have this Java method:

```
public float getBalance(int arg, out String status)
```

The method now returns the account balance and the status.

The request message is as the one shown previously.

e-Macao-16-5-222

Encoding RPC Out Parameters

The response looks as follows:

```
<soap:Envelope
     xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <orgNS:getBalanceResponse
          xmlns:orgNS="http://myOrganization.com/"
          xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
          soap:encodingStyle=
              "http://www/w3.org/2003/05/soap-encoding">
          <rpc:result>ret</rpc:result>
          <ret xsi:type="xsd:decimal">1358.25</ret>
          <status xsi:type="xsd:string">active</status>
      </orgNS:getBalanceResponse>
   </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-223

RPC In-Out Parameters

In web services, all parameters are passed by value.

Therefore the notion of `in-out` and `out` parameters does not involve passing objects by reference, but exchanging copies of the data.

The client code should create the perception that the state of the object that has been passed to the method has been modified.

e-Macao-16-5-224

Communication Styles

SOAP enables two communication styles:

1) document-style

   The message has no fixed structure, so the interacting applications must agree beforehand on this structure.

2) RPC-style

   Synchronous method invocation - pre-defined message structure.

e-Macao-16-5-225

Document Style

Also known as a message-oriented style:

1) a request is an XML document
2) an optional response is also an XML document

Two interacting applications agree beforehands upon the structure of the documents exchanged, then use SOAP messages to transport them.

Very flexible communication style that provides the best interoperability, using synchronous or asynchronous communication.

e-Macao-16-5-226

Document Style Example

The response message in document-style:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Body>
  <orgNS:returnBalance
      xmlns:orgNS="http://myOrganization.com/"
      soap:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
    <orgNS:balance orgNS:type="xsd:float">1235.95
    </orgNS:balance>
  </orgNS:returnBalance>
 </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-227

Task 36: Document Style 1

Objective:

Generate a SOAP message in a document style, with a purchase order for two XML books with ISB-12345 and the total price 125.12.

```
1) cd demos\SOAP\DocumentStyle

2) dir
deployDocumentStyleService.wsdd
deployService.bat
DocumentStyleClient.class
DocumentStyleService.class
```

e-Macao-16-5-228

Task 37: Document Style 2

3) deploy DocumentStyleService:

```
a) copy documentStyleService.class
   to   Tomcat/webapps/axis/WEB-INF/classes
b)      execute deployService
```

4) test the service: http://localhost:8080/axis → View

5) what is the difference with the previous service deployed?
   a) methods?
   b) WSDL?

e-Macao-16-5-229

Task 38: Document Style 3

6) execute the client sending the output to a log file:

```
java -cp \demos\SOAP\DocumentStyle DocumentStyleClient
   > log.txt
```

```
7)  notepad  log.txt
```

e-Macao-16-5-230

RPC Style

RPC-style is a synchronous invocation of an operation returning a result:

1) One SOAP message encapsulates the request.

   The body of the request message contains the actual call including the name of the procedure being invoked and the input parameters.

2) Another SOAP message encapsulates the response.

   The body of the response contains the result and output parameters.

The two interacting applications agree upon the RPC method signature.

e-Macao-16-5-231

RPC Style Example

The response message in RPC-style:

```
<soap:Envelope
     xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
    <orgNS:getBalanceResponse
         xmlns:orgNS="http://myOrganization.com/"
         xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
         soap:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
       <rpc:result>ret</rpc:result>
       <ret xsi:type="xsd:decimal">1358.25</ret>
       <status xsi:type="xsd:string">active</status>
    </orgNS:getBalanceResponse>
 </soap:Body>
</soap:Envelope>
```

e-Macao-16-5-232

Task 39: RPC Request

Objective: generate a SOAP message in the RPC-style.

```
1)  cd demos\SOAP\RPCStyle

2)  dir
FileTransferRequest

3)  Java -cp demos\SOAP\RPCStyle FileTransferRequest
    e:\webservices\macaonews.txt news.txt > log.txt

4)  notepad log.txt
```

A.3.3. Protocol Binding

e-Macao-16-5-233

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

e-Macao-16-5-234

Protocol Binding Framework

SOAP enables exchange of messages using a variety of protocols.

The set of rules for carrying a SOAP message within or on top of another protocol for the purpose of exchange is called binding.

SOAP protocol binding framework:

1) provides general rules for the specification of protocol bindings

2) describes the relationship between bindings and SOAP nodes that implement those bindings

e-Macao-16-5-235

Binding and Transfer

For instance, SOAP HTTP binding describes how to take a SOAP infoset at one node and serialize it across an HTTP connection to another node.

The job of the binding is to move the infoset from one node to another. The way the infoset is represented in the “wire” is up to the binding author.

Bindings have the freedom to specify custom serializations in order to improve efficiency, security, etc.

e-Macao-16-5-236

Binding URI

Communicating parties must agree on what binding to use.

Thus, bindings are named with URIs.

SOAP HTTP binding URI:

http://www.w3.org/2003/05/soap/bindings/HTTP

e-Macao-16-5-237

SOAP Features

A feature extends SOAP with some specific functionality.

SOAP poses no constraints on the potential scope of features.

A feature description is identified by a URI, so that all applications referencing it are assured the same semantics.

e-Macao-16-5-238

SOAP Feature Examples

Examples of features:

1) reliability
2) security
3) routing
4) Message Exchange Patterns (MEPs):
   a) request/response
   b) one-way
   c) peer-to-peer conversations

e-Macao-16-5-239

Expressing Features

The SOAP extensibility model provides two mechanisms through which features can be expressed:

1) SOAP Processing Model
2) SOAP Protocol Binding Framework

e-Macao-16-5-240

Features with SOAP Processing

Describes the behavior of a single SOAP node with respect to the processing of an individual message

Characteristics:

1) features are expressed by modules
2) a module is a way to perform functions using the SOAP processing model via headers

e-Macao-16-5-241

Features with Protocol Binding

Mediates the act of sending and receiving SOAP messages by a SOAP node via an underlying protocol.

Characteristics:

1) features are expressed by bindings
2) a binding is a way to perform functions below the SOAP processing model

e-Macao-16-5-242

Features Method Comparison

Processing Model:

1) enables SOAP nodes, that are able to implement features to express them within the SOAP envelope as SOAP headers
2) header can be intended for any SOAP node along the SOAP message path

Protocol Binding Framework:

1) a protocol binding operates between two adjacent SOAP nodes along the message path
2) different protocols can be used along the path
3) some protocols are equipped, either directly or through an extension, with mechanisms for providing certain features

e-Macao-16-5-243

SOAP Modules

SOAP modules define the syntax and semantics of the extensions provided by headers, including constraints, rules, preconditions, and data formats .

A SOAP module realizes zero or more SOAP features.

SOAP modules are named with URIs so they can be referenced, versioned, and reasoned about.

e-Macao-16-5-244

Feature Specification

The specification of a feature must include:

1) a URI used to name the feature
2) the information required at each node to implement the feature
3) processing required at each node to implement the feature including handling of communication failures that might occur
4) the information to be transmitted from node to node

e-Macao-16-5-245

Feature Example

Suppose an application requires a “secure channel” feature.

The URI for this feature is http://www.myOrganization.com/secureChannel

The abstract feature describes that messages must travel from node to node in an unsnoopable fashion (reasonable level of security).

Alternatives:

1) Since HTTPS meets the security requirement specified by the feature, the feature would be satisfied by this protocol binding.
2) We can use a SOAP module (e.g. WS-Security) that provides encryption and signing of SOAP messages across any binding.

We can decide in some situations to engage the SOAP module, and not to do so in others (e.g. when using the HTTPS binding).

e-Macao-16-5-246

Message Exchange Patterns

MEP is a common type of feature.

A MEP specifies:

1) how many messages are involved in interaction
2) where the messages originate
3) where they end up

Each binding must support one or more MEP.

SOAP specifies two standard MEPs:

1) request-response
2) SOAP-response

e-Macao-16-5-247

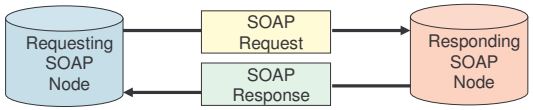
Request-Response MEP

Request-Response MEP involves two nodes:

1) requesting node sends a SOAP message to the responding node
2) responding node replies with a SOAP message that returns to the requesting node

Important features:

1) the response message is correlated to the request message
2) if a fault is generated at the responding node, the fault is delivered as part of the response message



e-Macao-16-5-248

SOAP Response MEP

SOAP Response MEP involves two nodes:

1) the requesting message is not a SOAP message
2) the responding node replies to the request with a SOAP message

Important features:

1) the request does not trigger the execution of the SOAP processing model on the receiving node
2) it allows a request to be something as simple as an HTTP GET

e-Macao-16-5-249

Request-Response MEP Example

One alternative is to implement the request-response MEP by an HTTP binding. Another one is using SOAP headers:

```
<soap:Envelope
 xmlns:saop="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
       <r:replyTo
              soap:mustUnderstand="true"
              xmlns:r="http://myOrganization/requestResponse">

       <destination>udp://anotherHost.com:6777</destination>
       </r:replyTo>
       <r:correlationID
              soap:mustUnderstand="true"
              xmlns:r="http://myOrganization/requestResponse">
              1202
       </reqresp:correlationID>
 </soap:Header>

 <soap:Body>…</soap:Body>
</soap:Envelope>
```

e-Macao-16-5-250

MEPs with HTTP Binding

The SOAP HTTP binding supports both MEPs.

1) request/response MEP with HTTP binding:
   - ✓ SOAP request message → HTTP request
   - ✓ SOAP response message → HTTP response

2) SOAP response MEP with HTTP binding:
   - ✓ non-SOAP request → HTTP GET request
   - ✓ SOAP response message → HTTP response

HTTP binding also specifies how to map faults to particular HTTP status codes and status codes to the web services invocations.

e-Macao-16-5-251

SOAP HTTP Protocol Binding

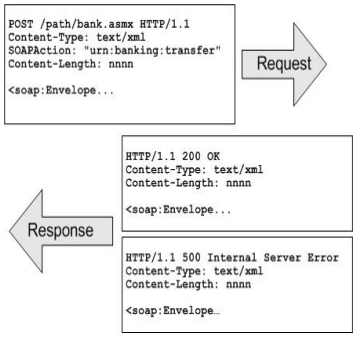
Example rules:

1) SOAP request/response maps naturally to the HTTP model

2) content-type header for HTTP request/response messages must be set to application/soap+xml

3) request messages must use POST and the URI identifying the SOAP processor

4) HTTP response should use 200 status if no errors occurred or 500 if the body contains a fault



[courtesy Aaron Skonnard]

e-Macao-16-5-252

SOAP HTTP Binding Details

HTTP GET request does not have a payload area and therefore cannot be used to carry SOAP messages.

The SOAP HTTP binding also implements two features:

1) SOAP action feature
2) web method feature

e-Macao-16-5-253

SOAP 1.1 Action Feature

In SOAP 1.1 HTTP binding, a custom header `SOAPAction` is required to:

1) let know the receiver that the content of the message is SOAP
2) convey the intent of the message via a URI

The decision to use a value for the `SOAPAction` header field is up to the web service designer.

Many implementations use this URI dispatching a particular piece of code on the backend, especially for document-style communication.

For instance, the same body content may be sent to two different methods, and the `SOAPAction` URI may be used to differentiate between them.

e-Macao-16-5-254

SOAP 1.2 Action Feature

SOAP 1.2 uses the `application/soap+xml` media type. The `SOAPAction` header is no longer needed to identify SOAP messages.

This media type specifies an optional `action` parameter used in SOAP 1.2, instead of an HTTP-specific header to carry the `SOAPAction` URI.

The binding implementing this feature must place the value of the URI in the action parameter. The message looks like:

```
POST /axis/TheService.jws
Content-Type: application/soap+xml; charset=utf-8
Action="http:// myOrganization.com/specificAction"
...
```

e-Macao-16-5-255

Web Method Feature

The web method feature was defined to integrate the semantics of SOAP with the semantic of HTTP.

Bindings to HTTP should use this feature to give control to applications over the web methods (GET, POST, PUT, …) used sending SOAP message.

When sending a message, the HTTP binding will use the verb specified in this property instead of the default POST.

A.3.4. Binary Data

e-Macao-16-5-256

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

e-Macao-16-5-257

SOAP and Binary Data

Suppose that in the bank application we would like to send an image of the account statement.

Since XML cannot encode binary data, a solution might be to use the XML Schema type `base64binary` and encode images as `base64` text:

```
<soap:Envelope xmlns:soap="..." xmlns:xsi="...">
 <soap:Body>
  <accountData>
   <number>23450935</number>
   <owner>John Smith</owner>
   <statement imageType="jpg" xsi:type="base64binary">
      4f3t68j …
   </statement>
  </accountData>
 </soap/Body>
</soap:Envelope>
```

Not efficient! E-mail is using the MIME standard.

e-Macao-16-5-258

SOAP with Attachments 1

In 2000, HP and Microsoft released a specification SOAP with Attachments.

SwA describes a simple way to use the multiref encoding in SOAP 1.1 to reference MIME-encoded attachment parts.

Here is the previous example in SwA:

```
MIME-Version:1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
type=application/soap+xml;start="<account@myOrganization.com>"

--MIME_boundary
Content-Type: application/soap+xml: charset=UTF-8
Content-Transfer-Encoding: 8 bit
Content-ID:<account@myOrganization.com>
```

e-Macao-16-5-259

SOAP with Attachments 2

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 <soap:Body>
        <accountData>
                <number>23450935</number>
                <owner>John Smith</owner>
                <statement href="cid:statemet@myOrganization.com"/
        </accountData>
 </soap/Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID:<statement@myOrganization.com>
. . . Binary JPG image . . .
--MIME_boundary
```

The problem is that this approach introduces a data structure that is explicitly outside the realm of the XML data model.

e-Macao-16-5-260

Task 40: SOAP with Attachments

Objective: send a request for uploading an attached file.

```
1)  cd \demos\SOAP\WithAttachments

2) dir
deployFileUploadService.wsdd
deployService.bat
FileUploadRequest.class
FileUploadService.class
```

e-Macao-16-5-261

Task 41: SOAP with Attachments

Deploy the web service:

```
3) copy FileUploadService.class
   to  Tomcat 4.1\webapps\axis\WEB-INF\classes
```

4) double-click: `deployService.bat`

Test the web service:

5) http://localhost:8080/axis --> View

e-Macao-16-5-262

Task 42: SOAP with Attachments

Run the request and send the output to a log file (`request.txt`):

```
6) java -cp \demos\SOAP\WithAttachments  FileUploadRequest
       \WebServices\MacaoNews.txt macao.txt > request.txt

7) notepad request.txt
```

A.3.5. Summary

e-Macao-16-5-263

SOAP Outline

e-Macao-16-5-264

SOAP Summary 1

SOAP is a lightweight protocol that allows to move XML messages in a distributed environment.

SOAP provides:

1) a messaging framework
2) data model and encoding rules
3) bindings to various communication protocols

SOAP is extensible.

SOAP is independent of any protocol or programming language.

e-Macao-16-5-265

SOAP Summary 2

1) XML messages are packed in envelopes for transmission.
2) A SOAP envelope contains zero or more headers and a body.
3) A header is a container for control information or application data sent to a SOAP server. It provides a mechanism to extend SOAP messages.
4) A body is a container to exchange information.
5) A fault is a structure to inform a sender that something went wrong while processing the message by the receiver.

e-Macao-16-5-266

SOAP Summary 3

1) A SOAP message can visit several intermediaries before it reaches the ultimate receiver. All these nodes constitute the message path.
2) SOAP defines a processing model that outlines rules for processing a message through the message path.
3) Headers may be addressed to specific intermediaries.
4) Headers may include four attributes:
   a) `role`: the name of the intermediary who should handle the header
   b) `mustUnderstand`: is processing the header mandatory?
   c) `relay`: should the header be passed to the next node?
   d) `encodingStyle`: a URI for the SOAP encoding

e-Macao-16-5-267

SOAP Summary 4

1) SOAP data model defines an abstract representation of the common data structures handled by programming languages.
2) SOAP encoding provides a set of rules to map the data model to XML.
3) SOAP also provides the rules for encoding requests, faults and RPC.
4) SOAP supports two communication styles for invoking a service:
   a) document style: interacting applications must agree upon the structure of the documents exchanged
   b) RPC style: synchronous service invocation, defined message structure

e-Macao-16-5-268

SOAP Summary 5

1) SOAP protocol binding framework provides the general rules for specification of protocol bindings.
2) A SOAP protocol binding describes how to take a SOAP infoset at one node and serialize it across the protocol connection to another node.
3) A feature is a unit of SOAP extension, implemented via SOAP modules or bindings. Message Exchange Pattern (MEP) is a common binding.
4) Two standard MEPs: request-response and SOAP response.
5) SOAP HTTP binding satisfies both standard MEPs.
6) Binary data may be sent by SOAP using a text encoding or using SOAP with attachments.

A.4. WSDL

WSDL

e-Macao-16-5-270

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

A.4.1. Introduction

e-Macao-16-5-271

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

e-Macao-16-5-272

Service Description

A client needs to use a web service to exchange SOAP messages, but:

1) what to include on the body of the message?

2) is any security SOAP header required?

3) what is the format of the response message?

4) what protocol is required?

5) where to send the message?

e-Macao-16-5-273

Service Description Need

A service description is needed.

Service descriptions are needed for the three SOA operations:

1) publish
2) find
3) bind

e-Macao-16-5-274

Service Description Components

A service description has two major components:

1) functional description - defines details of how the service is invoked, where is invoked, etc.
2) non-functional description - provides other details that are secondary to the message but instructs the requestor's runtime environment to add SOAP headers, such as: security policy

e-Macao-16-5-275

Functional Description

The functional description describes the operations available in the web service and the syntax of the messages required to invoke them.

The functional description is composed of:

1) service interface definition - describes:
   a) what messages must be sent
   b) how to use the various messaging protocols
   c) which encoding schemes must be used in order to format messages acceptable by the service provider
2) service implementation definition - describes where the service is located

Both definitions use the Web Service Description Language (WSDL).

e-Macao-16-5-276

Non-Functional Description

The non-functional description adds more information about the service:

1) why a service requestor should invoke the service - what business function the web service addresses and how it fits into a broader business process
2) who is the service provider, if the service provider carries out auditing, ensures privacy, etc.
3) specific aspects of the service not dependent on the domain, such as security

Currently, the most widely adopted approach to describing non-functional requirements is the combination of: WS-Policy and WS-PolicyAttachment.

e-Macao-16-5-277

Service Description Layers

A web service is described using a combination of techniques.

These are the questions that a service description should answer and which layer is providing this information:

who ?	non-functional description
what ?	service interface
where ?	service implementation
why ?	non-functional description
how ?	service interface and non-functional description

e-Macao-16-5-278

WSDL History

Two prior IDL languages for web services:

1) IBM's Network Accessible Service Specification Language (NASSL)
2) Microsoft's SOAP Contract Language (SCL)

WSDL is the result of merging NASSL and SCL.

e-Macao-16-5-279

WSDL Recommendation

IBM, Microsoft and other companies submitted WSDL 1.1 to the W3C for standardization in March 2001.

The specification is available at: http://www.w3.org/TR/wsdl

On 2004, the Web Services Description Working Group has released the First Public Working Draft of WSDL 2.0.

Still, not a recommendation.

e-Macao-16-5-280

WSDL Service Description

A WSDL service description is an XML document conformant to the WSDL schema definition.

This document, without any extensions, is not a complete service description, since it only covers the functional part.

WSDL is "the IDL for Web Services" describing:

1) what a service does - the operations (methods) the service provides, and the data (arguments and returns) needed to invoke them
2) how a service is accessed - details about data formats and protocols necessary to access the service operations
3) where a service is located - details of the protocol-specific network address, such as a URL

e-Macao-16-5-281

WSDL and IDLs

As WSDL describes service interfaces, it has a role and purpose similar to that of an IDL in conventional middleware platforms, but:

1) IDL only specifies a service interface: name and signature	1) WSDL also defines the mechanisms to access the service
2) the location of the requested object is transparent and unknown to the client	2) the WS middleware at the client site should be able to identify the location of the service
3) describes a single entry point (single RPC interaction)	3) involves the exchange of several asynchronous messages
4) objects are accessed through a concrete middleware platform	4) services are accessed using different protocols

e-Macao-16-5-282

Repositories of WSDL Documents

Public repositories of WSDL documents:

1) http://www.salcentral.com
2) http://www.xmethods.com
3) http://www.grandcentral.com/

A.4.2. Language

e-Macao-16-5-283

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

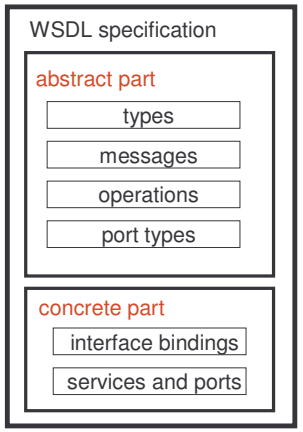
e-Macao-16-5-284

WSDL Structure

WSDL specifications include:

1) abstract part - conceptually analogous to conventional IDL

2) concrete part - defines protocol binding and other information

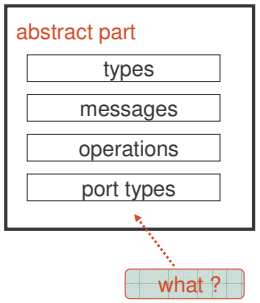


e-Macao-16-5-285

WSDL Structure - Abstract

The abstract part includes:

1) port type - logical collection of related operations
2) operation - abstract description of an action supported by the service
3) message - data exchanged in a single logical transmission
4) types - data structures that will be exchanged as parts of messages



There is no:

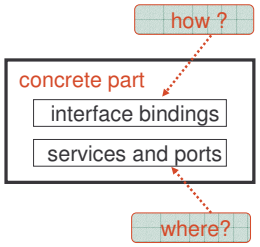
1) concrete binding
2) encoding specified
3) service implementing the set of ports

e-Macao-16-5-286

WSDL Structure - Concrete

The concrete part includes:

1) interface bindings - message encoding and protocol binding for all operations and messages defined in a given port-type
2) ports - combine the interface binding information with a network address specified by a URI
3) services - are logical groupings of ports

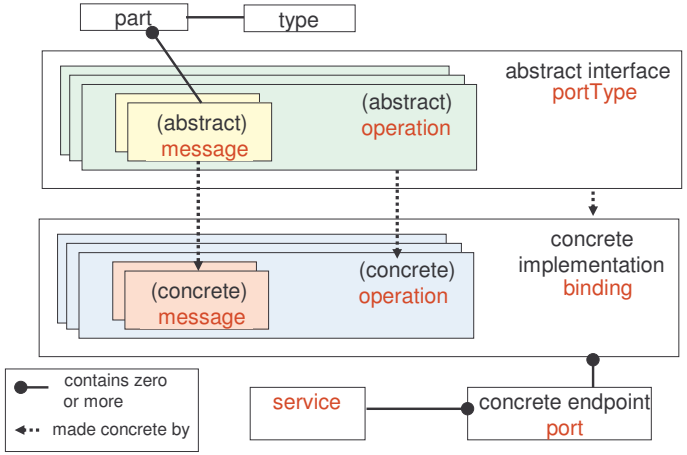


These allow:

1) a specific web service at different web addresses (different servers)
2) different ports (interface bindings) for the same port type, allowing the same functionality to be accessible via multiple transport protocols and interaction styles

e-Macao-16-5-287

WSDL Information Model



e-Macao-16-5-288

WSDL Document Example 1

```
<?xml version="1.0"?>
<definitions
   name= "Orders"
   targetNamespace="http://www.example.com/orders"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:tns=http://www.example.com/orders">
```

abstract part

```
<message name="OrderMsg">
   <part name="productId" type="xsd:string"/>
   <part name="quantity"  type="xsd:integer"/>
</message>
```

messages

```
<portType name="OrderPortType">
  <operation name="orderProductRequest">
    <input message = "OrderMsg"/>
  </operation>
</portType>
```

operation and port type

e-Macao-16-5-289

WSDL Document Example 2

e-Macao-16-5-290

WSDL Document Structure

The root element of a WSDL document is a `definitions` element.

This element can contain:

1) an optional `types` element
2) zero or more `message` elements
3) zero or more `portType` elements (usually one)
4) zero or more `binding` elements (usually one)
5) zero or more `service` elements (usually one)
6) zero or more `documentation` elements
7) zero or more `import` elements

A WSDL document must conform to the XML Schema defined at:
http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd

e-Macao-16-5-291

Definitions Element

The `definitions` element contains:

1) `name` attribute - corresponds to the name of the web service. It is only for documentation and is optional
2) `targetNamespace` attribute - a URI for the entire WSDL file
3) default namespace - all elements without a namespace prefix, such as `message` or `portType`, are assumed to be part of the default WSDL namespace: `http://schemas.xmlsoap.org/wsdl/`
4) other XML namespace declarations

e-Macao-16-5-292

Definitions Example

A Web Service that converts temperature in Fahrenheit to Celsius.

The service supports a single operation FahrenheitToCelsius, deployed using the SOAP protocol over HTTP.

```
<definitions
 name= "TemperatureConverterService"
 targetNamespace="http://www.converter.com/TemperatureConverter"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns=http://www.converter.com/TemperatureConverter"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">

 ...
</definitions>
```

e-Macao-16-5-293

Task 43: Definitions Element

Objective: Write a WSDL file describing a web service that will send an email with a custom message and error description to a developer or help desk. Two bindings will be provided: SOAP and HTTP-GET

1) `cd \demos\WSDL\Example1`

2) create a document "Myexample.wsdl" and add:

   a) the XML declaration

```
<?xml version="1.0" encoding="utf-8" ?>
```

   b) the root element for the WSDL document

```
<wsdl:definitions
```

e-Macao-16-5-294

Task 44: Definitions Element

3) add the definitions of namespaces for:

```
targetNamespace="http://example.com/ErrorMailer"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

and:

```
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://example.org/ErrorMailer"
```

4) save the file

e-Macao-16-5-295

Types Element

The `types` element encloses the definitions of user-defined XML types and elements for later use in the contents of the message.

A WSDL document can have at most one `types` element, and when present, it typically contains a single schema definition.

XML Schema is the predominant type system used, although `types` allows to describe other type systems.

In order to build interoperable web services, WSDL should only use the datatypes defined with XML Schema.

e-Macao-16-5-296

Types Example

```
<types>
   <xsd:schema
     targetNamespace="http://www.converter.com/TemperatureConverter">
        <xsd:element name="tempCelsius" type="xsd:float" />
        <xsd:element name="tempFahrenheit" type="xsd:float" />
   </xsd:schema>
</types>
```

Using the `schema` element to define XML datatypes and elements requires to include a `targetNamespace` attribute.

Many designers use the same value as the one used in WSDL `definitions` element.

e-Macao-16-5-297

Task 45: Types

Objective: Add the types definition to “MyExample.wsdl”

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” and add:

   a) the type and the schema elements:

```
<wsdl:types>
  <s:schema  targetNamespace="http://example.com/ErrorMailer">
```

e-Macao-16-5-298

Task 46: Types

b) element `SendError` defined as a complex type with two elements `LicenseKey` and `ErrorMessage`, both of type string:

```
<s:element name="SendError">
     <s:complexType>
        <s:sequence>
             <s:element name="LicenseKey" type="s:string" />
             <s:element name="ErrorMessage" type="s:string" />
        </s:sequence>
     </s:complexType>
</s:element>
```

e-Macao-16-5-299

Task 47: Types

c) element `SendErrorResponse` defined as a complex type, with an element `SendErrorResult` of type string:

```
<s:element name="SendErrorResponse">
   <s:complexType>
     <s:sequence>
        <s:element name="SendErrorResult
                   type="s:string" />
     </s:sequence>
   </s:complexType>
</s:element>
```

3) no more type definitions are needed. Add:

```
    </s:schema>
</wsdl:types>
```

4) save the file

e-Macao-16-5-300

Message Element

A `message` is the construct that describes the abstract form of an input, output or a fault message.

A message describes the data being communicated.

Each message has a unique name within the WSDL document and contains a collection of `parts`.

```
<message name="FahrenheitToCelsiusRequest">
      <part name="tempFahrenheit" type="xsd:float" />
</message>

<message name="FahrenheitToCelsiusResponse">
      <part name="tempCelsius" type="xsd:float" />
</message>
```

A `message` may have several `parts`.

A `part` may belong to several `messages`.

e-Macao-16-5-301

Part Element

Parts provide a flexible mechanism for describing the logical content of messages.

A `part` element has two properties:

1) name - represented by the `name` attribute, which must be unique among all the `part` elements of the `message` element

2) kind - defined as either a `type` or an `element` attribute:
   a) `element` - the payload of the message on the wire is precisely the XML element
   b) `type` - any element conforming to the type

e-Macao-16-5-302

Message Part Example

These are messages for the operation asking for information about weather:

```
<message name="WeatherRequest">
   <part name="userID" type="xsd:string" />
   <part name="city"   type="xsd:string' />
</message>

<message name="WeatherResponse">
  <part name="weather"  element="tns:weatherData" />
</message>
```

e-Macao-16-5-303

Task 48: Message Parts

Objective: Add the message definitions to "MyExample.wsdl". Four messages are needed to formulate a request and response for both bindings.

1) `cd \demos\WSDL\Example1`

2) edit "MyExample.wsdl" and add:

   a) two messages `SendErrorSoapIn` and `SendErrorSoapOut`. Each message has a part `parameters`. Both parts are elements of type "SendError"and "SendErrorResponse" respectively.

   b) two messages `SendErrorHttpGetIn` and `SendErrorHttpGetOut`. The first has two parts: `LicenseKey` and `ErrorMessage`, both of type string. The other message has only one part `Body` of type string.

3) save the file

e-Macao-16-5-304

PortType Element

`portType` is a collection of one or more related operations describing the interface of a web service.

`portType` definition is a collection of `operation` elements.

Generally, WSDL documents contain only one `portType` element, because different web service interface definitions are written with different documents.

`portType` has a single `name` attribute.

The name of `portType` together with the namespace of the WSDL document define a unique name for the `portType`.

e-Macao-16-5-305

Operation Element

`operation` defines a method of a web service, including the name of the method, input parameters, and the output or return type of the method.

All operations in a `portType` must have different names.

Each `operation` may define:

1) input message
2) output message
3) fault message

An `operation` in WSDL is the equivalent of a method signature in Java.

e-Macao-16-5-305

Operation Element

`operation` defines a method of a web service, including the name of the method, input parameters, and the output or return type of the method.

All operations in a `portType` must have different names.

Each `operation` may define:

1) input message
2) output message
3) fault message

An `operation` in WSDL is the equivalent of a method signature in Java.

e-Macao-16-5-307

Task 49: PortTypes and Operations

Objective: Add two portType definitions to "MyExample.wsdl", one for each binding. Both contain the operation `sendError` with the corresponding messages.

1) `cd \demos\WSDL\Example1`
2) edit "MyExample.wsdl" and add:
   a) one portType `ErrorMailerSoap`
   b) other portType `ErrorMailerHttpGet`
3) save the file

e-Macao-16-5-308

Abstract – Concrete Definitions

We already defined:

1) types
2) messages
3) portTypes
4) operations

} abstract, reusable portions of a WSDL definition

We didn't define yet how to relate these definitions with SOAP headers, SOAP bodies or SOAP encodings:

1) is this service invoked using a SOAP message or a simple HTTP POST of an XML payload?
2) is the service invoked with an RPC or a document style?

These aspects relate to the concrete implementation and are defined using the `binding` element.

e-Macao-16-5-309

Binding Element 1

The `binding` element specifies how to format messages in a protocol-specific manner:

1) message encoding
2) protocol binding

for all operations and messages defined in a given port type.

Each `portType` can have several `binding` elements associated with it.

Each binding specifies how to invoke operations using particular transport protocols. For instance: SOAP over HTTP, SOAP over SMTP, etc.

e-Macao-16-5-310

Binding Element 2

The `binding` element has two attributes:

1) `name` - must be unique among all binding elements defined in the WSDL document

2) `type` - identifies which portType the binding describes

e-Macao-16-5-311

Binding Example

```
<binding
     name="TemperatureConverter_ServiceSOAPBinding"
     type="TemperatureConverter_Service"
     . . .
</binding>
```

relates the binding with the portType

Conventions:

1) the name of the binding is combined with:
   a) the `portType` name (e.g.TemperatureConverter_Service),
   b) the name of the protocol to which the binding maps (e.g.SOAP)
   c) the word "Binding"
2) most WSDL documents contain only a single binding

e-Macao-16-5-312

Binding Protocol

To which protocol is the portType mapped by the binding?
We need to inspect the definitions inside the `binding` element.

The definitions inside the `binding` element are standard WSDL extensions that depends on the binding. WSDL specification describes extensions for:

1) SOAP/HTTP
2) HTTP GET/POST
3) SOAP with MIME attachments

Once the transport protocol is selected, find the WSDL convention that corresponds to the pair and fill in the details.

Most web services define at least a SOAP binding.

e-Macao-16-5-313

SOAP Binding Protocol

The `soap:binding` element has two attributes:

1) `style` - specifies the communication style. The values include:
   a) `document` – operation is document-oriented; messages carry documents that are agreed upon by the two applications
   b) `rpc` – operation is RPC-oriented; messages carry the input parameters and return values of the procedure call

2) `transport`- specifies the communication protocol that is used to transport the messages. The values include:
   1) http://schemas.xmlsoap.org/soap/http
   2) http://schemas.xmlsoap.org/soap/SMTP
   3) http://schemas.xmlsoap.org/soap/ftp
   4) other URI

e-Macao-16-5-314

SOAP Binding Protocol Example

This binding defines that the included operations will use a document-oriented style and HTTP as the communication protocol.

```
<binding
     name="TemperatureConverter_ServiceSOAPBinding"
     type= "TemperatureConverter_Service" >
     <soap:binding style="document"
               transport="http://schemas.xmlsoap.org/soap/http" />
   . . .
</binding>
```

this declaration applies to the whole binding, specifying that all operations defined will be SOAP messages

e-Macao-16-5-315

Different Styles and Restrictions

1) if a `portType` references messages whose `parts` use the `element` attribute, it should only use `style="document"`

2) if a `portType` references messages whose `parts` use the `type` attribute, only a `style="RPC"` should be used.

e-Macao-16-5-316

Binding Protocol Operations

An `operation` element within a binding specifies the binding information for that operation:

```
<operation name="FahrenheitToCelsius">
  <soap:operation soapAction:"urn:temperatureconverter-service"/>
 . . .
</operation>
```

The `soap:operation` element provides information for the operation:

1) `soapAction` attribute specifies the value of the soapAction in the HTTP header for this operation.

e-Macao-16-5-317

Binding Protocol Encoding Rules

The binding also specifies the encoding rules used in serializing parts of a message into XML:

1) literal encoding - takes the WSDL types defined in XML Schema and "literally" uses those definitions to represent the XML content of messages. Abstract WSDL types becomes concrete types

2) SOAP encoding - considers the XML Schema definitions as abstract entities and translates them into XML using SOAP encoding rules

Literal encoding is used for document style interactions.

SOAP encoding is used for RPC style interactions.

One part of the message can be encoded literally in the header and other part can use the SOAP encoding in the body.

e-Macao-16-5-318

Operation Encoding Example

```
<binding
  name="TemperatureConverter_ServiceSOAPBinding"
  type= "TemperatureConverter_Service">
  <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="FarenheitToCelsius">
    <soap:operation soapAction="urn:temperatureconverter-service" />
    <input>
       <soap:body use="literal"/>
    </input>
    <output>
       <soap:body use="literal" />
    </output>
   </operation>
</binding>
```

The content of the input and output messages are sent in the body of the message. The content of the body is literally an XML element.

Only one part was defined for the input and output messages.

e-Macao-16-5-319

Binding Protocol Body Example

The input message that takes a data value 98.0, in the body of the message is described below:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi:"http:// www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <tempFarenheit xmlns="http://www/converter.com/TermperatureConverter">
   98.0</tempFarenheit>
 </soapenv:Body>
</soapenv:Envelope>
```

e-Macao-16-5-320

Task 50: SOAP Binding

Objective: Add the SOAP binding to "MyExample.wsdl". The communication style is "document" and messages are encoded literally.

1) `cd \demos\WSDL\Example1`
2) edit "MyExample.wsdl" and add the binding:

```
 <wsdl:binding name="ErrorMailerSoap" type="tns:ErrorMailerSoap" >
    <soap:binding
       transport="http://schemas.xmlsoap.org/soap/http"
       style = "document" />
    <wsdl:operation name="SendError">
       <soap:operation
           soapAction="http://example.com.ErrorMailer/SendError"
           style = "document" />
       <wsdl:input>  <soap:body use="literal" /> </wsdl:input>
       <wsdl:output> <soap:body use="literal" /> </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
```

3) save the file

e-Macao-16-5-321

Port Element

The only purpose of the `port` element is to specify the network address of the end-point hosting the web service.

`port` is a single end-point defined as a combination of a binding and a network address.

There can be many ports for a binding, just like many implementations for the same interface.

The `soap:address` element is used to give a port an address.

e-Macao-16-5-322

Port Example

```
<port
  name="TemperatureConverter_ServicePort"
  binding="TemperatureConverter_ServiceSOAPBinding"

    <soap:address
          location://localhost:8080/soap/servlet/TempConverter" />
</port>
```

reference to the binding

network address of the web service

The name identifies the port.

e-Macao-16-5-323

Service Element

A `service` is a collection of `ports`.

Although a WSDL document can contain a collection of `service` elements, by convention a WSDL document contains a single service.

Usage: group the ports that are related to the same service interface (`portType`) but expressed by different protocols (`binding`).

e-Macao-16-5-324

Service Example

```
<service name="TemperatureConverter_Service">
  <port
    binding="TemperatureConverter_ServiceSOAPBinding"
    name="TemperatureConverter_ServicePort">
    <soap:address
          location://localhost:8080/soap/servlet/TempConverter" />
  </port>
  <port>… </port>
</service>
```

e-Macao-16-5-325

Task 51: Service and Ports

Objective: Add the service and port definitions to "MyExample.wsdl". The address of the service is: http://www.example.com/ErrorMailer/Errormail.

```
1) cd \demos\WSDL\Example1
```

2) edit "MyExample.wsdl" adding the service and port definitions. The location is ="http://www.example.com/ErrorMailer/Errormail"

3) save the file

e-Macao-16-5-326

Documentation Element

The `documentation` element is used to provide useful, human-readable information about the web service description.

Any WSDL element can contain a `documentation` element, usually as its first child.

One conventional use is declaring that the WSDL file is an interoperable description: it is compliant with the WS-I basic profile. This use of the `documentation` element appears in the `service` element.

e-Macao-16-5-327

Documentation Example

```
<service name="TemperatureConverter_Service">
 <port binding="TemperatureConverter_ServiceSOAPBinding"
       name="TemperatureConverter_ServicePort">
   <documentation>
      <wsi:Claim
        conformsTo="http://ws-i.org/profiles/basic/1.0" />
   </documentation>
 <soap:address location://localhost:8080/soap/servlet/TempConverter" />
 </port>
</service>
```

e-Macao-16-5-328

Task 52: Documentation

Objective: Add documentation comments to "MyExample.wsdl".

```
1) cd \demos\WSDL\Example1
```

2) edit "MyExample.wsdl" adding a documentation element inside the service definition.

3) save the file

e-Macao-16-5-329

Import Element

The `import` element is used to include other WSDL documents or XML Schemas into a WSDL document.

The use of the `import` element allows to:

1) separate the different elements of a service definition into independent documents
2) import these documents as needed

It helps writing clearer WSDL descriptions by separating the definitions according to their level of abstraction and maximizing reusability.

For example, data structures modeled as XML Schemas can be imported by several WSDL documents defining different services.

e-Macao-16-5-330

Import Example

```
<definitions
    name= "TemperatureConverterService"
    targetNamespace="http://www.converter.com/TemperatureConverter"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns=http://www.converter.com/TemperatureConverter”
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <types>
     <xsd:schema>
               <import namespace=“http://www.converter.com/schemas”
                 location=“http://converter.com/temperature.xsd"/>
    </xsd:schema>
    </types>

    ...
</definitions>
```

e-Macao-16-5-331

Import Conventional Use

Many designers split their WSDL design into two parts:

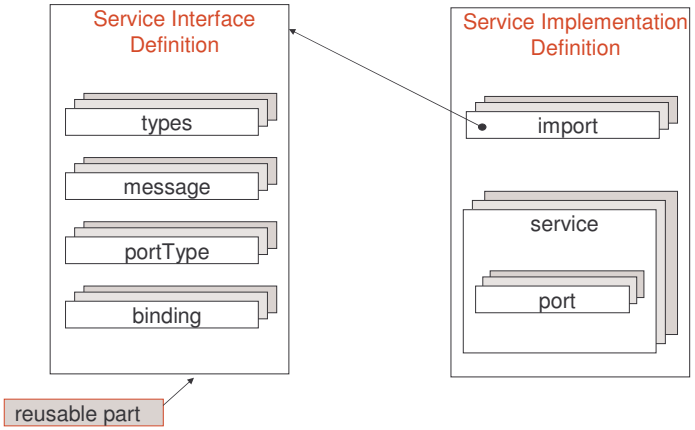
1) service interface definition
2) service implementation definition

Interface definition contains: `types`, `message`, `portType` and `binding` elements and encapsulates the reusable components of a service description.

Each organization wanting to implement a web service conformant to this interface definition would describe an implementation definition containing the `port` and `service` elements.

e-Macao-16-5-332

Interface versus Implementation



A.4.3. Transmission Primitives

e-Macao-16-5-333

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

e-Macao-16-5-334

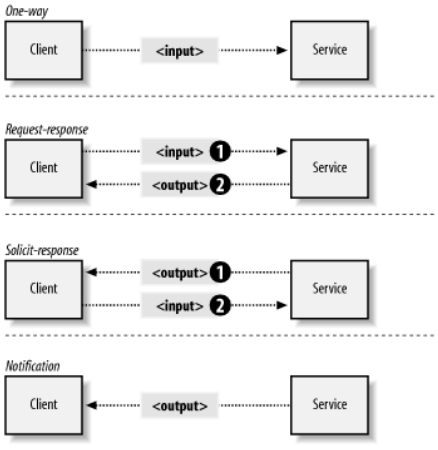
Transmission Primitives 1

WSDL supports four basic operation patterns called transmission primitives:

One way	The service receives a message. The operation has a single input element.
Request-Response	The service receives a message and sends a response. The operation has one input and one output element. To encapsulate errors fault elements can be specified.
Solicit-Response	The service sends a message and receives a response. The operation has one output element and one input element. To encapsulate errors fault element can also be specified.
Notification	The service sends a message. The operation has a single output element.

e-Macao-16-5-335

Transmission Primitives 2



[Courtesy Ethan Cerami]

e-Macao-16-5-336

Web Service Examples

Four different examples will be explained, one for each transmission primitive.

All examples are related to a virtual organization providing informational services about weather to its subscribed users.

e-Macao-16-5-337

WS Example One Way

1) `CancelUser` - a message is received asking to cancel the subscription service for a user.

   a) input message contains user identification and password

e-Macao-16-5-338

WS Example Request-Response

2) `AskData` - a user request for information related to the weather in a particular city and for a date, and receives a response.

   a) input message contains user identification, city and date

   b) output message contains temperature and humidity

e-Macao-16-5-339

WS Example Solicit-Response

3) `ServiceInterruption` - the service provider notifies a user that the service will be interrupted and waits for a response

a) output message contains notification

b) input message contains acknowledgement

e-Macao-16-5-340

WS Example Notification

4) `NotifyBadWeather` - the service provider sends a notification to a user when bad weather is forecast in a city where the user lives

a) output message contains temperature, humidity and notification

e-Macao-16-5-341

WS Example Definitions

WSDL document structure:

```
<?xml version="1.0"?>
<definitions name="WeatherServices"
    targetNamespace="http://www.example.com/WeatherService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.com/WeatherService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  >

  <types>
   ...
  </types>
   ...
   ...
</definitions>
```

It will be reused for all four examples.

e-Macao-16-5-342

WS Example Types 1

Types for all four weather examples:

```
<types>
  <xsd:schema
    targetNamespace="http://www.example.com/WeatherService"
     <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"
      schemaLocation= ="http://schemas.xmlsoap.org/soap/encoding/" />

     <xsd:element name="notification"    type="xsd:string" />
     <xsd:element name="acknowledge"     type="xsd:string" />
     <xsd:element name="errorString"     type="xsd:string" />

     <xsd:complexType name="userData">
       <xsd:sequence>
         <xsd:element name="userId"      type="xsd:string" />
         <xsd:element name="password"    type="xsd:string" />
       </xsd:sequence>
     </xsd:complexType>
```

e-Macao-16-5-343

WS Example Types 2

```
    <xsd:complexType name="dataRequest">
      <xsd:sequence>
        <xsd:element name="city"        type="xsd:string" />
        <xsd:element name="date"        type="xsd:dateTime" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="weatherData">
      <xsd:sequence>
        <xsd:element name="temperature" type="xsd:float" />
        <xsd:element name="humidity"    type="xsd:float" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
...
</definitions>
```

e-Macao-16-5-344

One-Way Operations

A one-way operation has only an input message. It acts like a data sink.

No response message (output or fault) going back to the requestor.

Basic functionality - change the state of the service provider

Many one-way messages at this level end up being request-response messages at the network transport level (response is the HTTP-level acknowledgement).

For a one-way operation, the HTTP response must not contain a SOAP envelope. Most clients will ignore it if it does appear.

e-Macao-16-5-345

One Way Example 1

Service Example: `CancelUser` - a message is received asking to cancel the subscription service for a user.

```
<message name="cancelUser">
  <part name="userCancel" type="tns:userData" />
</message>

<portType name="cancelUserPortType">
   <operation name="cancelUser">
     <input message="tns:cancelUser" />
   </operation>
</portType>


<binding name="CancelUserSOAPBinding"
    type="tns:cancelUserPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />
```

e-Macao-16-5-346

One Way Example 2

```
<operation name="cancellation">
   <soap:operation
      soapAction= "http://www.example.com/WeatherService/cancel" />
   <input>
     <soap:body use="encoded"
       namespace="http://www.example.com/WeatherService"
       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
</operation>
</binding>


<service name="CancelUser">
   <port name="CancelUser"
        binding="CancelUserSOAPBinding">
      <soap:address
        location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

e-Macao-16-5-347

Request-Response Operations

The most common form of operation because many web services are deployed using SOAP over HTTP.

Defines:

1) input message (request)
2) output message (response)
3) optional collection of fault messages

Basic functionality:

1) retrieve information about a web service object
2) change the state of the service provider
3) include information about the new state in the response

Like input and output elements, the fault element refers to a message which describes the data contents of the fault.

e-Macao-16-5-348

Request Response Example 1

Service Example: `AskData` - a user request for information related to the weather in a particular city and for a date, and receives a response.

```
<message name="askDataRequest">
  <part name="userIdent"        element="tns:userID" />
  <part name="userRequestData"  type="tns:dataRequest" />
</message>

<message name="askDataResponse">
   <part name="cityDateWeatherData" type="tns:weatherData" />
</message>

<message name="askDataLoginError">
   <part name="errorString" element="xsd:string" />
</message>

<message name="askDataDataError">
   <part name="errorString" element="xsd:string" />
</message>
```

fault messages must have a single part

e-Macao-16-5-349

Request Response Example 2

```
<portType name="askDataPortType">
   <operation name="askData">
     <input message="tns:askDataRequest" />
     <output message="tns:askDataResponse" />
     <fault message="tns:askDataLoginError"
            name="HeaderErrorMessage" />
     <fault message="tns:askDataDataError"
            name="BodyErrorMessage" />
   </operation>
</portType>
```

e-Macao-16-5-350

Request Response Example 3

```
<binding name="AskDataSOAPBinding"
    type="tns:askDataPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="askData">
   <soap:operation
      soapAction= "http://www.example.com/WeatherService/askData" />
   <input>
     <soap:header message="tns:askDataRequest" part="userIdent"
                  use="literal" >
     <soap:headerfault message="tns:HeaderErrorMessage"
          part="errorString" use="literal" />
     </soap:header>
     <soap:body parts="userRequestData" use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
```

a part of the input message on the header

best practice

e-Macao-16-5-351

Request Response Example 4

```
   <output>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </output>
   <fault name="BodyErrorMessage">
     <soap:fault name="BodyErrorMessage"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </fault>
</operation>
</binding>

<service name="AskData">
   <port name="AskData"
         binding="AskDataSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

e-Macao-16-5-352

Solicit-Response Operations

A solicit-response operation models a push operation similar to a notification. It expects an input (response) from the service requestor.

Defines:

1) output message (solicit)
2) optional fault messages (solicit)
3) input message (response)

Basic functionality:

1) notify the service requestor about the result of some event by the service provider
2) waits for an answer from the service requestor

e-Macao-16-5-353

Solicit Response Example 1

Service Example: `ServiceInterruption` - the service provider notifies a user that the service will be interrupted and waits for a response

```
<message name="serviceInterruptionSolicit">
  <part name="interruptionComment" element="tns:notification" />
</message>

<message name="serviceInterruptionResponse">
   <part name="userAcknowledge"    element="tns:acknowledge" />
</message>

<portType name="serviceInterruptionPortType">
   <operation name="serviceInterruption">
     <output message="tns:serviceInterruptionSolicit" />
     <input message="tns:serviceInterruptionResponse" />
</operation>
</portType>
```

e-Macao-16-5-354

Solicit Response Example 2

```
<binding name="ServiceInterruptionSOAPBinding"
    type="tns:serviceInterruptionPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="serviceInterruption">
   <soap:operation
    soapAction="http://www.example.com/WeatherService/Interruption"/>
   <input>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
   <output>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </ouput>
   </operation>
</binding>
```

e-Macao-16-5-355

Solicit Response Example 3

```
<service name="ServiceInterruption">
   <port name="ServiceInterruption"
         binding="ServiceInterruptionSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

e-Macao-16-5-356

Notification Operations

A notification operation is like a one-way operation, but the message is pushed by the service provider.

Output messages are pushed to the service requestor as the result of an event occurring on the service provider side, such as: time-out or operation completion.

Basic functionality - notify the service requestor about an event.

The notification style of interaction is commonly used in systems built around asynchronous messaging.

It is not possible to describe the semantic of these operations in WSDL (where to push the messages) without extensions, except by text comments.

To ensure interoperability of web services, the use of notifications is not recommended.

e-Macao-16-5-357

WS-Notification Example 1

Service Example: `NotifyBadWeather` - the service provider sends a notification to a user when bad weather is forecast in a city where the user lives

```
<message name="notifyBadWeather">
  <part name="notifyWeatherData"  type="tns:weatherData" />
  <part name="notifyCommnent"     element="tns:notification" />
</message>

<portType name="notifyBadWeatherPortType">
   <operation name="notifyBadWeather">
     <output message="tns:notifyBadWeather" />
   </operation>
</portType>

<binding name="NotifyBadWeatherSOAPBinding"
         type="tns:notifyBadWeatherPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />
```

e-Macao-16-5-358

WS-Notification Example 2

```
<operation name="notifyBadWeather">
   <output>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </output>
</operation>
</binding>

<service name="NotifyBadWeather">
   <port name="NotifyBadWeather"
         binding="NotifyBadWeatherSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

e-Macao-16-5-359

Operation Default Names

Default names for input and output elements for an operation OP:

	Input	Output
One-Way	OP	not applicable
Request-Response	OPRequest	OPResponse
Notification	not applicable	OP
Solicit-Response	OPResponse	OPSolicit

Fault elements require a name, because several fault elements can be associated with any operation and the fault name is used to distinguish among them.

e-Macao-16-5-360

Operation Parameter Order

Operations using an RPC-binding can specify a list of parameter names via the `parameterOrder` attribute.

The value of this attribute is a space-separated list of message part names with the following rules:

1) the order reflects the order of parameters in the RPC signature
2) the return value is not present in the list
3) if a part name appears in both input and output messages, it is an input/output parameter
4) if a part name appears in only the input message, it is an input parameter
5) if a part name appears in only the output message, it is an output parameter

e-Macao-16-5-361

Parameter Order Example

```
<message name="input">
     <part name="A" element="xsd:int"/>
     <part name="B" element="xsd:long"/>
</message>

<message name="output">
     <part name="A" type="xsd:int"/>
</message>

<portType name="servicePortType">
   <operation name="example" parameterOrder="B A">
      <input message="tns:input"/>
      <output message="tns:output"/>
   </operation>
</portType>
```

A.4.4. WSDL Extensions

e-Macao-16-5-362

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

e-Macao-16-5-363

Functional Extensions

The WSDL language allows most of the WSDL elements to be extended with elements from other namespaces.

The language specification defines standard extensions for:

1) SOAP
2) HTTP GET/POST operations
3) MIME attachments

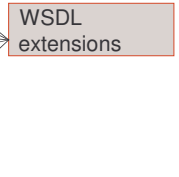
SOAP was already explained.

e-Macao-16-5-364

WSDL HTTP Extension

The HTTP binding extends WSDL with the following elements:

```
<binding .... >
    <http:binding verb="nmtoken"/>
    <operation .... >
      <http:operation location="uri"/>
      <input .... >
          <-- mime elements -->
      </input>
      <output .... >
          <-- mime elements -->
      </output>
     </operation>
    </binding>

    <port .... >
       <http:address location="uri"/>
    </port>
```



e-Macao-16-5-365

HTTP Binding

The `http:binding` element indicates that this binding uses HTTP.

```
<definitions .... >
...
    <binding .... >
        <http:binding verb="nmtoken"/>
    </binding>
...
</definitions>
```

The value of the required `verb` attribute may be GET or POST, or others HTTP requests.

e-Macao-16-5-366

HTTP Operation

An `operation` element within a binding specifies the binding information for that operation.

The `location` attribute specifies a relative URI for the operation.

This URI is combined with the URI specified in the `http:address` element (port definition) to form the full URI for the HTTP request.

The URI value must be a relative URI.

```
<binding .... >
   <operation .... >
       <http:operation location="uri"/>
   </operation>
</binding>
```

e-Macao-16-5-367

HTTP urlEncoded

The `urlEncoded` element indicates that all message parts are encoded into the HTTP request URI using the standard URI-encoding rules.

The names of parameters correspond to the names of the message parts.

Each value contributed by the part is encoded using a `name="value"` pair.

This may be used with GET to specify URL encoding.

For GET, "?" character is automatically appended as necessary.

```
<http:urlEncoded/>

http://www.example.com/WeatherService/askData?userId=2289193
```

e-Macao-16-5-368

HTTP urlReplacement

The `urlReplacement` element indicates that all message parts are encoded into the HTTP request URI using a replacement algorithm:

1) The relative URI value of `http:operation` is searched for a set of search patterns.
2) The search occurs before the value of the `http:operation` is combined with the value of the location attribute from `http:address`.
3) There is one search pattern for each message part. The search pattern string is the name of the message part surrounded with parenthesis.
4) For each match, the value of the corresponding message part is substituted for the match at the location of the match.
5) Matches are performed before any values are replaced; replaced values do not trigger additional matches.

Message parts MUST NOT have repeating values.

```
<http:urlReplacement/>
```

e-Macao-16-5-369

HTTP mime:content

The `mime:content` element's attribute `type="valid_type"` indicates that the message will appear in the HTTP code as the `valid_type`.

Examples:

1) the message is XML text in the HTTP response:

```
<output>
   <mime:content type="text/xml" />
</output>
```

2) the message is send as a gif file:

```
<output>
     <mime:content type="image/gif"/>
</output>
```

e-Macao-16-5-370

HTTP Address

The `location` attribute of the `http:address` element specifies the base URI for the port.

The value of the attribute is combined with the values of the `location` attribute of the `http:operation` binding element.

```
<port name="...">
  <http:address location="URI" />
</port>
```

e-Macao-16-5-371

HTTP Binding Example

```
<binding name="AskDataHTTPBinding"
         type="tns:askDataPortType">
    <http:binding verb="GET"/>
    <operation name="askData">
     <http:operation location= "askData" />
     <input>
        <http:urlEncoded/>
     </input>
     <output>
        <mime:content type="text/xml"/>
     </output>
   </operation>
</binding>

<service name="AskData">
   <port name="AskData"
         binding="AskDataHTTPBinding">
      <http:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
```

e-Macao-16-5-372

Task 53: HTTP/GET Binding

Objective: Add the HTTP/GET binding to "MyExample.wsdl". The location is "`/SendError`", the verb is "`GET`" and the message parts in the HTTP request are encoded.

1) `cd \demos\WSDL\Example1`

2) edit "MyExample.wsdl" and add the `ErrorMailerHttpGet` binding

3) save the file

e-Macao-16-5-373

Task 54: Port for HTTP Binding

Objective: Add the port definition of the service for the HTTP/GET binding, to "MyExample.wsdl". The location is:
"`http://www.example.com/ErrorMailer/Errormail`".

1) `cd \demos\WSDL\Example1`

2) edit "MyExample.wsdl" and add the port definition and the final tag of the `definitions` element

3) save the file

e-Macao-16-5-374

Functional Extensions

The WSDL language allows most of the WSDL elements to be extended with elements from other namespaces.

The language specification defines standard extensions for:

1) SOAP
2) HTTP GET/POST operations
3) MIME attachments

SOAP and HTTP were already explained.

e-Macao-16-5-375

WSDL MIME Extension

WSDL also supports a standard extension to describe message parts as MIME.

This extension could be used to include a GIF image as part of a message.

Example: we would like to add a map in a graphical file when sending data about the weather.

The response message must include the new part:

```
<message name="askDataResponse">
   <part name="cityDateWeatherData" type="tns:weatherData" />
   <part name="weatherpicture"      type="xsd:binary" />
</message>
```

e-Macao-16-5-376

WSDL with MIME 1

MIME extensions are only for the binding, indicating that the output is modeled as multipart MIME.

no changes

```
<binding name="AskDataSOAPBinding"
         type="tns:askDataPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="askData">
   <soap:operation
      soapAction= "http://www.example.com/WeatherService/askData" />
   <input>
     <soap:header message="tns:askDataRequest" part="userIdent"
                  use="literal" >
        <soap:headerfault message="tns:HeaderErrorMessage"
           part="errorString" use="literal" />
     </soap:header>
     <soap:body parts="userRequestData" use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
```

e-Macao-16-5-377

WSDL with MIME 2

changes

```
<output>
    <mime:multipartRelated>
     <mime:part>
     <soap:body use="encoded" />
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </mime:part>
      <mime:part>
        <mime:content part="weatherpicture" type="image/gif" />
        <mime:content part="weatherpicture" type="image/jpeg" />
      </mime:part>
    </mime:multipartRelated>
   </output>
   <fault name="BodyErrorMessage">
     <soap:fault name="BodyErrorMessage"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </fault>
</output>
</binding>
```

e-Macao-16-5-378

Task 55: WSDL Example 1

Objective: Access to a complete WSDL document published on the web.

1) browse: http://www.errormail.net/EM/ErrorMailer.asmx?wsdl
2) analyze the document

e-Macao-16-5-379

Task 56: WSDL Example 2

Objective: Access the WSDL document of `FileDownloadService`.

1) start Tomcat
2) browse: http://localhost:8080/axis
3) access: view → FileDownloadService (wsdl)
4) analyze the document

e-Macao-16-5-380

Non-Functional Descriptions

How to describe:

a) security requirements

b) transactional capabilities

c) logging features for the invocation of the service

d) auditing realized by the service provider

Non-functional characteristics of a web service can be described using WS-Policy and related specifications.

e-Macao-16-5-381

WS-Policy

WS-Policy version 1.0 was originally published in December 2002, by BEA, IBM, Microsoft, and SAP.

In May 2003, version 1.1 was published.

The WS-Policy family of specifications has three major components:

1) framework

2) assertions

3) attachment

The basic component of the policy framework is a policy assertion.

e-Macao-16-5-382

Policy Assertion

Policy assertion is a concrete statement about requirements, preferences, capabilities and other characteristic of a web service or its operating environment.

Policy assertions describe certain qualities of service such as reliability of messaging or security aspects.

A policy assertion may be a simply statement of fact:

```
<wsrm:DeliveryAssurance Value="wsrm:ExactlyOnce"/>
```

Also, a policy assertion may be a complicated statement, indicating possible sets of requestor-specifiable parameters, etc.

Policy assertions are grouped together to form a policy.

e-Macao-16-5-383

Policy Assertion Example

Two policy assertions are specified:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
```

1) the subject requires a Kerberos V5 service ticket token

```
   <wsse:SecurityToken wsp:Usage="wsp:Required">
       <wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
   </wsse:SecurityToken>
```

2) an XML digital signature is required

```
   <wsse:Integrity wsp:Usage="wsp:Required">
      <wsse:Algorithm Type="wsse:AlgSignature"
            URI="http://www.w3.org/2000/09/xmlenc#aes" />
   </wsse:Integrity>
</wsp:Policy>
```

e-Macao-16-5-384

Policy and Policy Subject

A policy forms a named collection of policy assertions that can be referenced using standard XML mechanisms, by other XML and Web service components such as a WSDL definition.

One mechanism to reference a policy is to associate a policy with a policy subject.

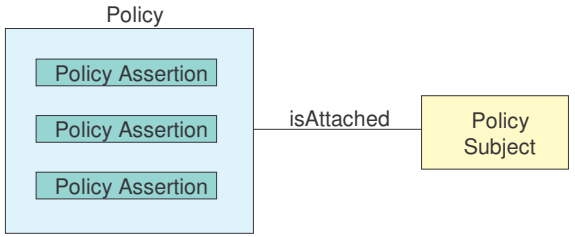
A policy subject can be:

1) web service
2) component of a web service description
3) part of a web service's operating environment
4) other entities related to a web service

WS-PolicyAttachments specification describes how to associate policies with policy subjects.

e-Macao-16-5-385

Policy and Policy Subjects



e-Macao-16-5-386

WS-Policy Framework

WS-Policy defines how to group policy assertions into a named collection that can be referenced by other components.

WS-Policy framework consists of:

1) an XML element to act as a container for one or more policy assertions
2) a set of XML elements that describe how the policy assertions grouped by the container are to be combined
3) a set of standard XML attributes that may be associated with policy assertions

e-Macao-16-5-387

Policy Container

General Form:

```
<wsp:Policy  ((name="..." TargetNamespace= "...") | Id="...") >
   <policy-specific assertion>
                ...
   <policy-specific assertion>
   <policy-specific security>
</wsp:Policy>
```

Defines:

1) the policy name
2) policy assertions
3) security policy assertions specific to this policy element

It is assumed that policy assertions are completely independent

Policy operators are needed to provide semantics to policy assertions

e-Macao-16-5-388

Policy Name

Two standard mechanisms to name a policy:

1) by XML QName

```
<wsp:Policy  name="..." TargetNamespace= "..."  >
```

2) by URI – combined with the XML base of the document

```
<wsp:Policy xml:base="http://example.com" Id="Pol1">
```

The URI will be: `http://example.com#Pol1`

One mechanism should be chosen and followed through all the policy work.

e-Macao-16-5-389

Policy Operators

Four operators exists to describe different combinations of policy assertions:

1) All
2) ExactlyOne
3) OneOrMore
4) the basic policy element

e-Macao-16-5-390

Policy Operators Example 1

The following example:

1) defines a policy named Example1 in http://example.com namespace
2) states that all assertions A,B, and C are in effect

```
<wsp:Policy
     name="Example1"
     TargetNamespace="http://example.com" >
<wsp:All>
   <Assertion A/>
   <Assertion B/>
      <Assertion C/>
</wsp:All>
</wsp:Policy>
```

For a policy assertion to be in effect is entirely dependent on the domain of each policy assertion and the policy subject to which the policy is attached

e-Macao-16-5-391

Policy Operators Example 2

The following example:

1) defines a policy named Example2 in http://example.com namespace
2) states that exactly one of the assertions A,B, and C is in effect

```
<wsp:Policy
     name="Example2"
     TargetNamespace="http://example.com" >
<wsp:ExactlyOne>
   <Assertion A/>
   <Assertion B/>
      <Assertion C/>
</wsp:ExactlyOne>
</wsp:Policy>
```

Operators can be nested – any of the assertion elements can be replaced by an operator.

e-Macao-16-5-392

Policy Attributes

WS-Policy framework provides a pair of global XML attributes : `Usage` and `Preference`

These attributes can be added to the various policy assertions.

e-Macao-16-5-393

Usage Attribute

`Usage` describes how the policy assertion is to be interpreted in the context of the policy. Possible values:

1) `Required`: the assertion must apply or an error occurs
2) `Rejected`: the assertion must not apply or an error occurs
3) `Optional`: the assertion may apply or may not apply
4) `Observed`: let the requestor know that a particular assertion will be applied
5) `Ignored`: tell the requestor that if something happens to cause the policy assertion to be in effect then no error message will be emitted

e-Macao-16-5-394

Usage Attribute Example

This policy specifies that assertion A must apply, assertion B must not apply, and assertion C may or may not apply.

```
<wsp:Policy
     name="Example3"
     TargetNamespace="http://example.com" >
  <wsp:ExactlyOne>
      <Assertion A wsp:Usage="Required"/>
      <Assertion B wsp:Usage="Rejected" />
       <Assertion C wsp:Usage="Optional" />
  </wsp:ExactlyOne>
</wsp:Policy>
```

e-Macao-16-5-395

Preference Attribute

This attribute is used in conjunction with the `ExactlyOne` operator.

If there is a choice between a set of policy assertions, this value acts as a hint to the requestor.

The value is integer. The higher the number, the stronger the preference.

e-Macao-16-5-396

Preference Attribute Example

This policy specifies that the requestor has a choice of assertions A, B, and C, and that the service provider would much prefer the requestor to choice assertion A.

```
<wsp:Policy
     name="Example3"
     TargetNamespace="http://example.com" >
  <wsp:ExactlyOne>
      <Assertion A wsp:Preference="100"/>
      <Assertion B wsp:Preference="50" />
       <Assertion C wsp:Preference="10" />
  </wsp:ExactlyOne>
</wsp:Policy>
```

e-Macao-16-5-397

Policy Example

This policy indicates that the subject requires exactly one security token, either a UsernameToken, x509 security token or Kerberos.

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
 <wsp:ExactlyOne wsp:Usage="Required">
    <wsse:SecurityToken>
       <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Preference="10">
       <wsse:TokenType>wsse:x509v3</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Preference="1">
       <wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
    </wsse:SecurityToken>
 </wsp:ExactlyOne>
</wsp:Policy>
```

The preference values indicate that the preferred token type is x509, followed by Kerberos, followed by UsernameToken.

e-Macao-16-5-398

Referencing Policies

WS-Policy defines `PolicyReference` elements that allows to include the contents of one policy in another.

The `PolicyReference` element can appear anywhere a policy assertion can appear, and it refers to another policy.

The meaning is that the contents of the included policy element are wrapped with an `All` operator element and inserted in the place of the reference.

e-Macao-16-5-399

Referencing Policies Example 1

Consider this policy specification:

```
<wsp:Policy
     name="Example4"
     TargetNamespace="http://example.com"
     xmlns:tns="http://www.example.com/policies" >
  <wsp:ExactlyOne>
      <wsp:PolicyReference Ref="tns:Example2" />
      <Assertion X />
      <Assertion Y />
   </wsp:ExactlyOne>
</wsp:Policy>
```

e-Macao-16-5-400

Referencing Policies Example 2

It is equivalent to:

```
<wsp:Policy
     name="Example4"
     TargetNamespace="http://example.com"
     xmlns:tns="http://www.example.com/policies" >
  <wsp:ExactlyOne>
     <wsp:All>
        <wsp:ExactlyOne>
             <Assertion A/>
             <Assertion B/>
              <Assertion C/>
        </wsp:ExactlyOne>
     </wsp:All>
      <Assertion X />
      <Assertion Y />
   </wsp:ExactlyOne>
</wsp:Policy>
```

```
<wsp:Policy name="Example2" ...>
  <wsp:ExactlyOne>
     <Assertion A/>
     <Assertion B/>
     <Assertion C/>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Only one of assertions A, B, C, X or Y must be in effect.

e-Macao-16-5-401

Different Policy Assertions

Discipline-specific - policy assertions selected, configured and combined into a policy document, such as:

1) security policy assertions,
2) reliability policy assertions,
3) etc.

Generic assertions - four standard policy assertions defined by WS-Policy in a separate specification called WS-PolicyAssertions

e-Macao-16-5-402

Generic Policy Assertions

Four generic policy assertions:

1) text encoding - declares which character set is used for text that appears in web service messages

2) language assertion - declares the human language expected in messages

3) spec assertion - declares which version of a particular technical specification a web service is compliant with

4) message predicate - declares the exact content of a message going into or coming out of a web service. The contents are defined using the XPath language.

e-Macao-16-5-403

Message Predicate Example

This policy requires:

1) exactly one **wsse:Security** header element, and
2) exactly one child element within the **soap:Body** element

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
  <wsp:MessagePredicate wsp:Usage="wsp:Required">
    count(wsp:GetHeader(.)/wsse:Security) = 1
  </wsp:MessagePredicate>
  <wsp:MessagePredicate wsp:Usage="wsp:Required">
    count(wsp:GetBody(.)/*) = 1
  </wsp:MessagePredicate>
  ...
</wsp:Policy>
```

e-Macao-16-5-404

Task 57: Policy

Objective: Write a policy "MyPolicy" specifying that messages must be sent using UTF-8 encoding, SOAP 1.2, and optionally a digital signature.

1) `cd \demos\WSDL\Example1`

2) create MyPolicy.xml for the specified requirements, using the following namespaces:

```
xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
```

e-Macao-16-5-405

Policy Attachments

A policy can be attached to a policy subject, such as WSDL `portType`, WSDL `message`, UDDI elements or others.

A policy can be attached to a policy subject in two ways:

1) as part of the subject's definition

2) external to the subject's definition

e-Macao-16-5-406

Policy Attachment Example 1

Suppose we want to declare that the language expected for a web service can be English, Chinese or Spanish:

```
<wsp:Policy name = "WeatherLanguages"
            TargetNamespace = "http://www.example.com/policies" >
  <wsp:OneOrMore>
     <wsp:Language Language="en" />
     <wsp:Language Language="cn" />
     <wsp:Language Language="es" />
  </wsp:OneOrMore>
</wsp:Policy>
```

e-Macao-16-5-407

Policy Attachment Example 2

The policy can be referenced from within the `AskData` service declaration, using the `PolicyRefs` attribute:

```
<service name="AskData"
   wsp:PolicyRefs="pol:WeatherLanguages"
   xmlns:wsp="http://schemas.xmloap.org/ws/2002/12/policy"
   xmlns:pol="http://www.examples.com/policies">
   <port name="AskData" binding="AskDataSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

The `policyRefs` attribute takes a list of QNames allowing to associate a collection of policies to any policy subject.

The `policyURIs` attribute provides the same functionality allowing to associate a collection of policies identified by URI with any policy subject.

e-Macao-16-5-408

Task 58: Policy Attachment

Objective: Attach "MyPolicy" to the Error Mail Service.

1) `cd \demos\WSDL\Example1`

2) edit MyExample.wsdl and below the service definition, add the reference to the policy

3) save the file

e-Macao-16-5-409

Effective Policy

A policy attached to a WSDL element can be inherited by its child elements.

For instance, a policy attached to a `portType` would be inherited by its `input`, `output` and `fault` child elements.

Effective policy is the policy associated with a WSDL element. It can be an inherited policy or a policy directly attached to the element.

e-Macao-16-5-410

Policy Inheritance in WSDL 1

WSDL Element	Effective Policy
`message`	policy associated with `message`
`message/part`	policy associated with `part`, merged with the effective policy of the `part`'s `message` parent
`portType`	policy associated with `portType`
`portType/operation`	policy associated with `operation`, merged with the effective policy of the `operation`'s `portType` parent
`portType/operation/input`	policy associated with `input`, merged with the effective policy of the `input`'s `operation` parent and with the effective policy of the `message` associated with the `input` element
`portType/operation/output`	similar to input
`portType/operation/fault`	similar to input

e-Macao-16-5-411

Policy Inheritance in WSDL 2

WSDL Element	Effective Policy
`binding`	policy associated with the `binding` merged with the effective policy of the associated `portType`
`binding/operation`	policy associated with the `operation` merged with the effective policy of the `operation`'s `binding` parent and merged with the effective policy of the `portType operation`
`binding/operation/input`	policy associated with the `input` merged with the effective policy of the `input`'s `operation` parent and merged with the effective policy of the corresponding `portType/operation/input`
`binding/operation/output`	similar to input
`portType/operation/fault`	similar to input
`service`	policy associated with the `service`
`service/port`	policy associated with the `port` merged with the effective policy of the `port`'s `service` parent

e-Macao-16-5-412

External Policy Attachment 1

Allows policies to be associated with a policy subject independent of that subject's definition and/or representation through the use of a `PolicyAttachment` element.

The `PolicyAttachment` element has three components:

1) the policy scope of the attachment
2) the policy expressions being bound
3) security information (optional)

e-Macao-16-5-413

External Policy Attachment Example

```
<wsp:PolicyAttachment ... >
  <wsp:AppliesTo>
    <x:DomainExpression/> +
  </wsp:AppliesTo>
 ( <wsp:Policy>...</wsp:Policy> |
    <wsp:PolicyReference>...</wsp:PolicyReference> ) +
    <wsse:Security>...</wsse:Security> ?
       ...
</wsp:PolicyAttachment>
```

policy scope

policy expressions

security information

```
<wsp:PolicyAttachment>
 <wsp:AppliesTo>
    <wsa:EndpointReference
           xmlns:ad="http://www.example.com/WeatherService" >
      <wsa:Address>http://www.example.com/WeatherService/askData
      </wsa:Address>
      <wsa:PortType>ad:askDataPortType</wsa:PortType>
      <wsa:ServiceName>ad:AskData</wsa:ServiceName>
    </wsa:EndpointReference>
 </wsp:AppliesTo>
 <wsp:PolicyReference URI="http://www.example.com/policies#ASKDATAPOL"/>
</wsp:PolicyAttachment>
```

A.4.5. WSDL and Java

e-Macao-16-5-414

WSDL: Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

e-Macao-16-5-415

WSDL Mapping to Java

Many tools maps WSDL to Java, both sides:

1) service requestor
2) service provider.

For instance: the Axis WSDL2Java tool.

Some conventions are defined for mapping:

1) portTypes
2) operations
3) messages
4) bindings

e-Macao-16-5-416

portType to Java

1) the `portType` naturally maps into a Java Interface
2) the name of the interface typically takes the name of the `portType`
3) the file is declared in a package named from the targetNamespace URI of the WSDL `definitions` element containing the `portType`

e-Macao-16-5-417

portType to Java Example

The WeatherService.wsdl where the `portType` `AskData` was defined, include:

```
<?xml version="1.0"?>
<definitions name="WeatherServices"
 targetNamespace=http://www.example.com/WeatherService
```

Thus a piece of the Java Interface for the `AskData PortType` is:

```
Package com.example.www.WeatherService.AskData;

Public interface AskDataPortType extends java.rmi.Remote {
```

e-Macao-16-5-418

Operation to Java

1) for each of the `portType`'s operations, a public method is declared as part of the interface
2) the signature of the method is built from:
   a) the name of the operation
   b) the input and output values defined in the operation
   c) any fault element associated with the operation are included as exceptions thrown by the method.

e-Macao-16-5-419

Operation to Java Example

The method signature generated from the `askData` operation defined in `askDataPortType` is:

```
public com.example.www.WeatherService.cityDateWeatherData
  askData (com.example.www.WeatherService.userID uid,
           com.example.www.WeatherService.dataRequest uRequestData)
  throws java.rmi.RemoteException;
```

e-Macao-16-5-420

Message to Java

1) for those messages that are referenced by `input` and `output` elements, a class is generated for `complexTypes` referenced by the parts of those messages
2) a class is generated for those messages referenced in `fault` elements.
3) the name of the class is taken from the name of the type or element
4) the package from the class is taken from the targetNamespace URI of the XML schema that defines the type or element.

These type-based classes are used as part of the mechanism to serialize and deserialize XML to and from Java.

e-Macao-16-5-421

Message to Java Example

Input element:

```
<input>
  <soap:header message="tns:askDataRequest" part="userIdent"
               use="literal" >
  <soap:headerfault message="tns:HeaderErrorMessage"
        part="errorString" use="literal" />
  </soap:header>
  <soap:body parts="userRequestData" use="encoded"
     namespace="http://www.example.com/WeatherService"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
```

Classes in Java:

1) `userId`
2) `dataRequest`
3) `fault`

defined in package: com.example.www.WeatherService

e-Macao-16-5-422

Binding to Java

1) each binding is generated as a stub class
2) the name of the class is the name of the binding
3) the `targetNamespace` of the `definitions` element is used to define the name of the package
4) this class implements the interface defined by `portType`

This class is a proxy to the service – encapsulates the implementation details associated with how a given `portType` is made concrete by the binding.

e-Macao-16-5-423

Binding to Java Example

```
<binding name="AskDataSOAPBinding"
    type="tns:askDataPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="askData">
    . . .
```

A class called `AskDataSOAPBindingImpl` is defined.

Implements the interface defined by the `portType askDataPortType`.

e-Macao-16-5-424

Service to Java

The WSDL `service` also generates an interface and class.

They encapsulate details of invoking the service from the client application.

A.4.6. Summary

e-Macao-16-5-425

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

e-Macao-16-5-426

WSDL Summary 1

WSDL is an XML-based language for describing and accessing web services.

WSDL describes four pieces of critical data:

1) data type declarations for all requests and response messages

2) interfaces describing available functions

3) binding information about the transport protocol

4) address information for locating the service

e-Macao-16-5-427

WSDL Summary 2

The service description is an XML document where the root element called `definitions` may include the following elements:

1) `types` - user-defined types and elements used in messages
2) `message` - composed of `parts`, describes data exchanged
3) `portType` - collection of related operations describing a WS interface
4) `binding` - specification of how to format messages in a protocol-specific manner
5) `service` - a collection of ports specifying network addresses of the end-points hosting the web service
6) `documentation` - human-readable information about the WS
7) `import` - including other WSDL documents or XML Schemas documents

e-Macao-16-5-428

WSDL Summary 3

Conventional use of WSDL includes:

1) Service interface definition:
   a) types
   b) message
   c) portType
   d) binding
2) Service implementation definition:
   a) service and port

e-Macao-16-5-429

WSDL Summary 4

WSDL supports four patterns of operation called transmission primitives:

1) one way - only an input message
2) request-Response - input-output messages and optional faults
3) notification - only an output message
4) solicit-Response - output-input messages and optional faults

e-Macao-16-5-430

WSDL Summary 5

WSDL provides functional extensions for:

1) SOAP
2) HTTP GET/POST operations
3) MIME attachments

e-Macao-16-5-431

WSDL Summary 6

Non-functional aspects of a web service can be specified using WS-Policy and related specifications:

1) A policy assertion describes certain aspects of a service quality: reliability, security, etc.

2) Policy assertions are grouped to form a policy.

3) WS-Policy also specifies how policies can be referenced by other components.

4) A policy can be attached to a subject, such as: WSDL `portType` and `message`, UDDI elements, or others in two ways:
    a) as part of the subject definition
    b) external to the subject definition using WS-PolicyAttachments

A.5. AXIS

AXIS

e-Macao-16-5-433

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

A.5.1. Overview

e-Macao-16-5-434

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-435

Axis Overview

Axis is a SOAP engine.

Axis consists of several subsystems working together for processing messages.

The Axis engine invokes in sequence a series of handlers to process messages.

The order in which handlers are invoked is determined by:

1) deployment configuration
2) whether the engine is a client or a server

e-Macao-16-5-436

Axis History

IBM contributed with an early implementation of the SOAP protocol to Apache in 1999, known as Apache SOAP.

This implementation was based on SOAP4J and was written in a monolithic style.

Apache Community decided to make re-engineering of this code, and Apache SOAP 2.1 emerged.

The development team started a major refactoring and redesign of the codebase and it was supposed to be called Apache SOAP 3.0.

Axis (Apache eXtensible Interaction System) was chosen instead of Apache SOAP 3.0.

e-Macao-16-5-437

Axis Architecture

A chain of message-processing components that can be developed separately and assembled at deployment time.

These components are called handlers.

Axis replaced the Apache SOAP's DOM-based XML processing used in predecessors, to faster SAX system.

e-Macao-16-5-438

Axis Handlers

Axis handlers tell the engine how to deal with messages that need to be processed.

Handlers can be:

1) built-in the engine
2) included in a module defined by the user

Handlers may:

1) send a request and receive a response
2) process a request and produce a response - called pivot point of the sequence of messages

e-Macao-16-5-439

Handlers Implementations

Handlers are Java classes based around a simple abstract class:

```
apache.axis.handlers.BasicHandler
```

A handler implementation, overrides the following method:

```
void invoke (MessageContext context) throws AxisFault;
```

When a handler is invoked, it may execute different functions:

1) reading and writing pieces of SOAP message,
2) logging information to a database,
3) checking user's identification credentials,
4) . . .

e-Macao-16-5-440

Different Types of Handlers

Handlers are:

1) transport-specific
2) service-specific
3) global

e-Macao-16-5-441

Handlers and Chains

Handlers can be combined into chains.

A chain is a pre-defined ordered collection of handlers.

The overall sequence of handlers comprises three chains:

1) transport
2) global
3) service

e-Macao-16-5-442

Chains

The Axis engine processes chains in the same way as handlers.

A chain class groups handlers together.

The chain class also implements the `Handler` interface.

When a chain's `invoke` method is called, it calls the `invoke` method on each of its constituent handlers, which themselves might be chains.

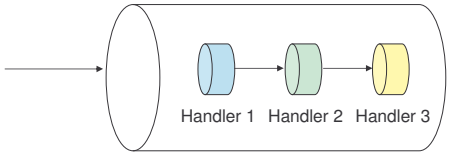
Axis uses two types of chains:

1) simple chains
2) targeted chains

e-Macao-16-5-443

Simple Chains

A simple chain is a list of handlers that should be invoked in order.

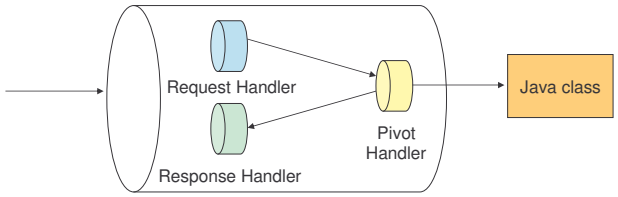


e-Macao-16-5-444

Targeted Chains

A targeted chain has exactly three handlers:

1) request hander: does the pre-processing work
2) pivot handler: the place where real work is done
3) response handler: does the post-processing work



Deployed services in Axis are invoked by targeted chains. The pivot handler calls the Java class that is exposed as a web service.

e-Macao-16-5-445

Message Context

The object passed to each handler is called `MessageContext`.

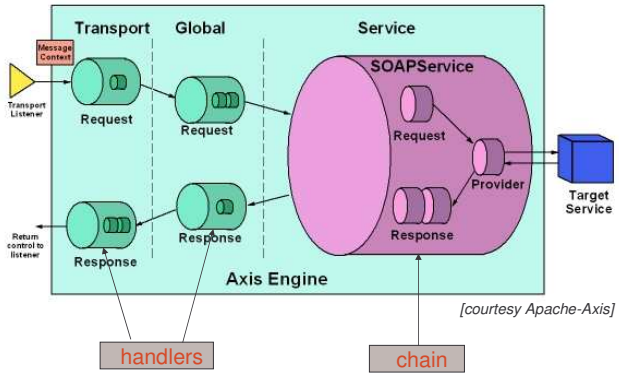
`MessageContext` is a structure which contains:

1) a request message
2) a response message
3) several properties
4) other fields

Axis processing framework passes the `MessageContext` through the set of handlers that are configured in the engine.

e-Macao-16-5-446

Message Path on the Server



[courtesy Apache-Axis]

e-Macao-16-5-447

Server Side – Listener Request

A message arrives at a transport listener - any software that can take input messages and handle them to Axis.

A transport listener:

– implements a particular SOAP binding

1) packages protocol-specific data into a `Message` object and puts it into a `MessageContext`

2) sets some properties in the `MessageContext`:
   a) `http.SOAPAction` - with the value defined in the HTTP header
   b) `transportName` - "http"
   c) other general properties such as reception-time, etc.

3) hands the `MessageContext` to the AxisEngine

A built-in HTTP listener:

`org.apache.axis.transport.http.AxisServlet`

e-Macao-16-5-448

Server Side – Transport Level

The AxisEngine:

a) looks for a transport chain whose name matches the `transportName` in the `MessageContext`

b) if a transport chain exists, the engine invokes the request handler of this chain passing it the `MessageContext`

This allows the server to implement transport-specific processing.

Transport-specific processing consists of any work that closely relates to the transport over which the message was received.

Example: any process dealing with HTTP headers for an HTTP transport, for instance HTTP authentication.

e-Macao-16-5-449

Server Side – Global Level

After the transport-specific request processing completes without error, the server passes the `MessageContext` to the global request chain.

This chain contains handlers that process all incoming messages, regardless the transport.

The global chain may be used to implement common features to all messages, such as: security or logging.

e-Macao-16-5-450

Server Side – Service Level

After the global chain is finished, the server calls the service handler.

The service handler is a special kind of wrapper called a SOAPService.

The SOAPService class is a targeted chain, including:

1) request chain – allows to insert pre-processing handlers specific for the service invoked

2) provider handler – is the pivot handler that calls the service class

3) response chain – allows to insert post-processing handlers specific for the service that is invoked

e-Macao-16-5-451

Server Side – Listener Response

After the Axis engine processes the message, the control is passed again to the listener.

The listener:

1) takes the message out from the `MessageContext`

2) sends it back to the client as an HTTP response

e-Macao-16-5-452

Deploying Global Handlers Example

Deployment descriptor to deploy a global handler :

```
<deployment xmlns="..."
  <globalConfiguration>
       <requestFlow>
           <handler type="requestHandler"/>
       </requestFlow>
       <responseFlow>
           <handler type="responseHandler"/>
       </responseFlow>
  </globalConfiguration>

 <handler name="requestHandler" type="java:MyRequestHandler">
    ...
 </handler>

 <handler name="responseHandler" type="java:MyResponseHandler">
   ...
 </handler>
</deployment>
```

e-Macao-16-5-453

Task 59: Handlers' Development

Objective: deploy two global handlers on the server side - request and response handlers. Both handlers write the envelope of the messages they process to a file.

```
1)  cd \demos\Axis\Handlers

2)  dir
deployService.bat
FileTransferRequest.class
MyHandlers.wsdd
MyRequestHandler.class
MyResponseHandler.class
```

Analyze the deployment file:

```
3) edit MyHandlers.wsdd
```

e-Macao-16-5-454

Task 60: Deploy Global Handlers

Deploy the handlers:

```
4) copy MyRequestHandler.class  MyResponseHandler.class
   to  Tomcat 4.1\webapps\axis\WEB-INF\classes
5) deployService.bat
6) browse: http://localhost:8080/axis --> View
```

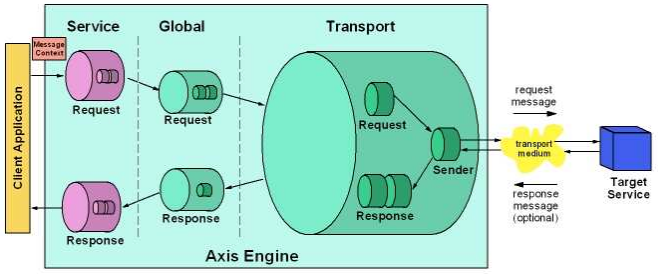
7) Why they do not appear?

Test the handlers:

```
8)  java -cp \demos\Axis\Handlers
      FileTransferRequest \WebServices\MacaoNews.txt Macao.txt
8)  cd Tomcat 4.1\bin
9)  dir
10) edit SOAPRequest.log SOAPResponse.log
```

e-Macao-16-5-455

Message Path on the Client



[courtesy Apache-Axis]

e-Macao-16-5-456

Client Side Message Processing

The `AxisClient` is the class handling the message flow through the various components on the client side.

The `Call` object (`org.apache.axis.client.Call`) is the main client-side entry point to Axis.

Inside the `AxisClient`, the message flows in a reverse order than in the server.

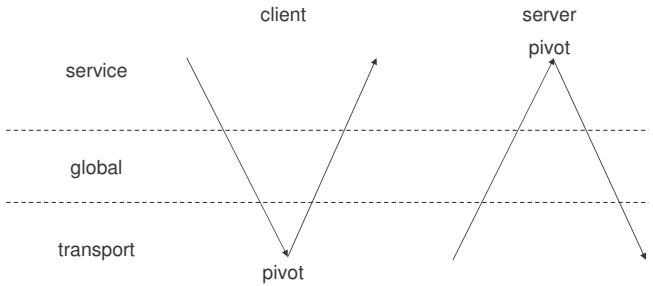
The last chain, is the transport-specific chain.

The transport chain has a pivot handler called the `sender`.

The sender is responsible for taking the request message out of the `MessageContext` and sending it across the wire in a protocol-specific way.

e-Macao-16-5-457

Sequence of Handlers



e-Macao-16-5-458

JAX-RPC Overview

JAX-RPC: Java API for XML-based RPC.

The fundamental purpose of JAX-RPC is to make communications between Java and non-Java platforms easier:

1) using Web services technologies like XML, SOAP and WSDL
2) providing a simple object-oriented API that Java developers can use to communicate with other technologies

It is possible to use JAX-RPC to:

1) access web services that run in non-Java environments
2) host Java web services, so that non-java applications can access them

e-Macao-16-5-459

JAX-RPC

1) JAX-RPC is designed as a Java API for web services.

2) incorporates XML-based RPC functionality according to the SOAP 1.1 specification

3) requires support for:

   a) SOAP and WSDL
   b) RPC encoded messaging
   c) SOAP with Attachments

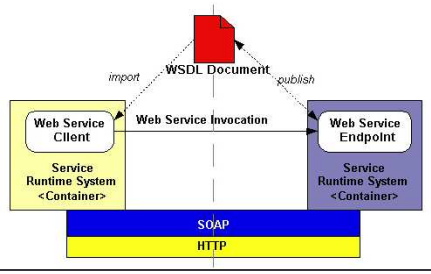
e-Macao-16-5-460

WS and JAX-RPC

A Web service endpoint is deployed on either the Web container or EJB container based on the corresponding component model.

These endpoints are described using a WSDL document.

A client uses this WSDL document and invokes the Web service endpoint.



JAX-RPC requires SOAP over HTTP for interoperability.

e-Macao-16-5-461

JAX-RPC and SOAP

JAX-RPC provides support for SOAP message processing model through the SOAP message handler functionality.

JAX-RPC uses SAAJ API (SOAP with Attachments API for Java).

SAAJ provides a standard Java API for constructing and manipulating SOAP messages with attachments.

e-Macao-16-5-462

JAX-RPC Extensions

1) Message Handlers: they allow to manipulate SOAP header blocks as they flow in and out of JAX-RPC endpoint and the client applications.

2) Mappings from WSDL and XML to Java describing how:

a) Java endpoint interfaces used by JAX-RPC web services are converted into WSDL
b) WSDL documents are converted into JAX-RPC generated stubs and dynamic proxies
c) method calls to JAX-RPC client APIs are converted into SOAP messages
d) SOAP messages are mapped to JAX-RPC service endpoint methods

A.5.2. Service Invocation

e-Macao-16-5-463

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-464

Message Context Class

`MessageContext` is the Axis implementation of the `SOAPMessageContext` class.

Is the core class for processing messages in handlers and other parts of the system.

This class also contains constants for accessing some well-known properties.

Some methods include:

- `getAllPropertyNames()`
- `getMessage()`
- `getService()`
- `getRequestMessage()`
- `getProperty()`
- `getOperation()`
- `getSOAPActionURI()`
- `getResponseMessage()`

e-Macao-16-5-465

Message Type

Asking the MessageContext for the request or response message will return a message of type `org.apache.axis.Message`.

The `Message` class extends `SOAPMessage`.

The message contains:

1) a `SOAPPart`
2) zero or more `AttachmentsParts`

Messages conforming to the simple SOAP HTTP binding will have only a `SOAPPart` and no attachments.

e-Macao-16-5-466

SOAPPart

The `SOAPPart` allows to get the `SOAPEnvelope`.

A client can access the `SOAPPart` object of a `SOAPMessage` object like:

```
void invoke(MessageContext context) ...

   SOAPMessage message = context.getMessage();
    SOAPPart soapPart = message.getSOAPPart();
```

`SOAPPart` object contains a `SOAPEnvelope` object, which in turn contains a `SOAPBody` object and a `SOAPHeader` object.

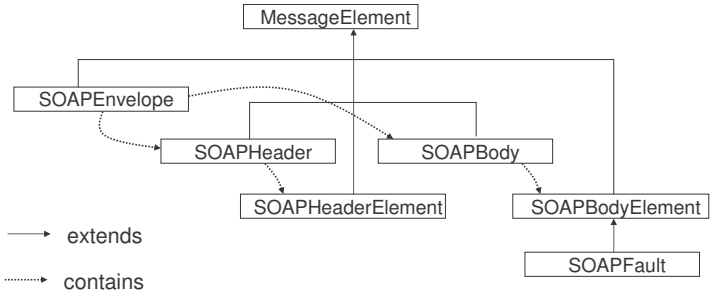
The `SOAPPart` method `getEnvelope` can be used to retrieve the `SOAPEnvelope` object.

```
      SOAPEnvelope env = soapPart.getEnvelope();
```

e-Macao-16-5-467

Accesing the Envelope Elements

Some important classes involving the SOAP envelope:



`MessageElement` is the base type of nodes of the SOAP message parse tree.

e-Macao-16-5-468

Accesing to Envelope Elements

Once, obtained the object `SOAPEnvelope` (env), it is possible to access `SOAPHeader` and `SOAPBody` objects:

```
void invoke(MessageContext context) ...
   SOAPMessage message = context.getMessage();
    SOAPPart soapPart = message.getSOAPPart();

       SOAPEnvelope env = soapPart.getEnvelope();

   SOAPHeader sh = env.getHeaders();
   SOAPBody sb = env.getBody();
```

e-Macao-16-5-469

Envelope Elements Example

Count the headers on a SOAP Envelope:

```
void invoke(MessageContext mc) ...

   /* Get the SOAP Envelope from the request message */
   Message requestMsg = mc.getRequestMessage();
   SOAPEnvelope env = requestMsg.getSOAPEnvelope();

   /* Obtain the headers */
   Vector headers = env.getHeaders();
```

e-Macao-16-5-470

Looking for a Fault Example

Analyze if the content of the body is a fault:

```
void invoke(MessageContext mc) ...

   /* Get the SOAP Envelope from the response message */
   Message responseMsg = mc.getResponseMessage();
   env = responseMsg.getSOAPEnvelope();
   SOAPBodyElement body = env.getFirstBody();

   /* controls whether the body contains a fault */
   if (body instanceof SOAPFault) {
     System.out.println ("First body element is a fault code,
       code =" + ((SOAPFault)body).getFaultCode().toString() );
```

e-Macao-16-5-471

Task 61: Envelope

Objective: Write the EnvelopeManager Java class which provides the invoke method. This method receives as argument an object of type `MessageContext`. The method provides the following functionality:

1) prints a message specifying how many headers has the request message
2) prints a message specifying if the body element of the response message contains a fault and what is the code of the fault

The EnvelopeManager class is used by:

1) envelopeExample1: the same example as FileTransferRequest
2) envelopeExample2: the same example as FileTransferSenderFault

These classes include the following code:

```
MessageContext mc = call.getMessageContext();
EnvelopeManager em = new EnvelopeManager();
em.invoke(mc);
```

e-Macao-16-5-472

Task 62: Envelope

```
1) cd demos\axis\envelope

2) dir
EnvelopeExample1.class
EnvelopeExample2.class
EnvelopeManagerTemplate.java
```

3) edit the class using: `EnvelopeManagerTemplate.java`
4) add the lines of code to get the headers of the envelope
5) add the lines of code to get the envelope
6) save the file as `EnvelopeManager,java`

e-Macao-16-5-473

Task 63: Envelope

Compile the class:

```
7)  javac EnvelopeManager.java
```

Execute the examples:

```
8)  java -cp \demos\Axis\Envelope envelopeExample1
          c:\WebServices\MacaoNews.txt macao.txt
9)  java -cp \demos\Axis\Envelope envelopeExample2
          c:\WebServices\MacaoNews.txt macao.txt
```

e-Macao-16-5-474

Replacing the Envelope Elements

It is possible to change the body or header of a SOAPEnvelope object by retrieving the current one, deleting it, and then adding a new body or header.

The javax.xml.soap.Node method detachNode detaches the XML element (node) on which it is called.

For example, to create a new header:

```
env.getHeader().detachNode();
SOAPHeader sh = env.addHeader();
```

e-Macao-16-5-475

Type Mapping

Axis has the ability to map XML to Java and vice versa.

A type mapping consists of:

1) an XML type (a QName)
2) a Java type (a class)
3) a serializer (to write Java to XML)
4) a deserializer (to write Java from XML)

To map types on the client side, use:

```
call.registerTypeMapping (Class javaType,
                          QName xmlType,
                          Class serializerFactoryClass,
                          Class deserializerFactoryClass)
```

e-Macao-16-5-476

Type Mapping Example

Recall the example that downloads the attributes of the file.

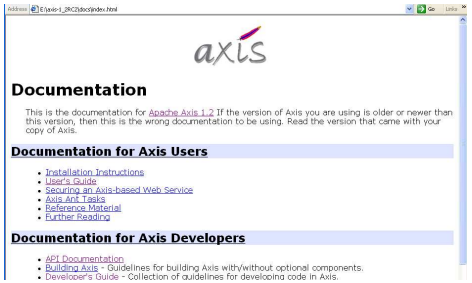
The code on the client, looks like:

```
QName qn = new QName("urn:FileAttribute","FileAttribute");

 call.registerTypeMapping(FileAttribute.class, qn,
   new org.apache.axis.encoding.ser.BeanSerializerFactory(
                                    FileAttribute.class, qn),
    new org.apache.axis.encoding.ser.BeanDeserializerFactory(
                                     FileAttribute.class, qn));
```

e-Macao-16-5-477

Task 64: Accessing Documentation

```
1) cd axis-1_2RC2\docs

2) open index.html
```



3) access API Documentation

e-Macao-16-5-478

Axis Client APIs

The client APIs can divided in two categories:

1) dynamic invocation: only pre-existing Java classes are used to do the work
2) stub generation: a tool generates code from the WSDL description

In order to invoke a web service, the client needs to use the Dynamic Invocation Interface (DII).

e-Macao-16-5-479

Service Object

The Service object acts as a factory for Call objects and it also stores meta-data about the service:

1) is the object representing the AxisClient instance that processes the client invocations
2) is where the type-mappings XML-Java are stored

The Service object can generate many Call objects.

Each Call object represents a single invocation of a service.

Since all these Call objects will talk to the same Web service, all the meta-data related to it is stored in the Service object.

e-Macao-16-5-480

Service Object and WSDL

A Service may or may not be associated with a WSDL description.

If it is related, it is possible to request:

1) a generic Call object,
2) a Call object that has been preconfigured with all the meta-data from the WSDL

The Service API as defined by JAX-RPC has no direct means to access WSDL documents for dynamic invocation.

The Service object has two constructors that allows the association with a WSDL document.

e-Macao-16-5-481

Service Object: WSDL Association

Two constructors for building a Service object with meta-data initialized from the WSDL:

1) `Service(URL wsdlLocation, QName serviceName)`: the WSDL is located at the specified URL.

2) `Service(String wsdlLocation, QName serviceName)`: the WSDLocation is a String that may be a URL or may also be a filename on the local filesystem, relative to the current directory.

The Axis Service object also has a no-argument constructor for use without a WSDL:

3) `Service()`

e-Macao-16-5-482

Call Object

Call objects are generated with the `createCall()` method of the Service object.

```
import org.apache.axis.client.Service;
import javax.xml.rpc.Call;

Service service = new Service();
Call call = service.createCall();
```

The no-arguments service constructor creates a blank service.

The `createCall()` factory method without arguments, creates a generic JAX-RPC Call object.

e-Macao-16-5-483

Setting Properties on the Call

The Call class has a setProperty() API:

```
void setProperty(String name, Object value)
```

This method allows to set properties on the Call object.

All the properties that are set on the Call will be available to every MessageContext that is created as a result of using the Call.

It is possible to use a Call object to make multiple invocations to a given service. Each time a new MessageContext will be created.

e-Macao-16-5-484

Generic Calls for WS

The development of the Web service client includes:

1) creating objects to manage the call and the service

2) defining the endpoint URL of the web service

3) defining the operation name of the web service

4) invoking the desired service passing the corresponding parameters

e-Macao-16-5-485

Creating Objects: Call and Service

Let's have a look at one of our Web Service client: FileTransferRequest:

```
public class FileTransferRequest {
   public static void main(String [] args) {
       . . .
      Service  service = new Service();
      Call     call    = (Call) service.createCall();
```

Two objects are created: Service and Call.

e-Macao-16-5-486

Defining the Endpoint

A variable endpoint is defined and initialized with the URL address of the desired web service.

```
String endpoint =
 "http://localhost:8080/axis/services/FileDownloadService";
```

This address containing the final destination of the SOAP message is passed to the newly created Call object:

```
call.setTargetEndpointAddress( new java.net.URL(endpoint) );
```

Axis also provides two constructors that allow to create a Call, pointing to a particular Web service endpoint:

```
1)  Call (String endpoint)
2)  Call(URL endpoint)
```

e-Macao-16-5-487

Defining the Operation Name

The name of the operation that the Call object is invoking is defined as:

```
call.setOperationName(new QName("http://soapinterop.org/",
 "downloadFile")
```

e-Macao-16-5-488

Invoking the Desired Service

The invoke() method allows to invoke a web service. It has several different forms.

The data for any given invocation is generally handed to the invoke method as an array of Java objects:

1) for RPC-Style services, these objects are parameters for a remote method call and each one maps to an XML element
2) for document-style services is generally a single object in the array and maps to the entire SOAP body for the operation

In our example, we have:

```
byte[] ret = (byte[])call.invoke( new Object[] { args[0] } );
```

args[0] is the name of the file to download, that is sent as a parameter

e-Macao-16-5-489

Invoke Method: Different Forms

`invoke()` Invokes this Call with its established MessageContext
`invoke(Message msg)` Invokes the service with a custom Message.
`invoke (java.lang.Object[] params)` Invokes the operation associated with this Call object using the passed in parameters as the arguments to the method.
`invoke (QName operationName, java.lang.Object[] params)` Invokes a specific operation using a synchronous request-response interaction mode.
`invoke(RPCElement body)` Invokes an RPC service with a pre-constructed RPCElement.
`invoke(SOAPEnvelope env)` Invokes the service with a custom SOAPEnvelope.

and more…

e-Macao-16-5-490

Task 65: Invoking the WS

Objective: Write the FileTransferClient Java class invoking the method downloadfile of the FileDownloadService. Use the FileTransferTemplate.java.

```
1) cd demos\Axis\Client

2) dir
FileTransferTemplate.java

3) edit FileTransferTemplate.java to generate
 FileTransferClient.java adding the corresponding lines of code

4) javac FileTransferClient.java

5) java -cp \demos\Axis\Client FileTransferClient
        c:\WebServices\MacaoNews.txt Macao.txt

6) dir
```

e-Macao-16-5-491

Naming Parameters

Axis automatically names the XML-encoded arguments in the SOAP message as “arg0”, “arg1”, etc.

For changing these names, we need to add the call `addParameter` for each parameter, and `setReturnType` for the return, before the invoke:

```
call.addParameter ("testParam",
                   org.apache.axis.Constants.XSD_String,
                   javax.xnl.rpc.ParameterMode.IN);
call.setReturnType (org.apache.axis.Constants.XSD_String);
```

The testParam will be the first parameter on the invoke call. It also defines the type of the parameter and whether it is an input, output or inout parameter.

If names are added to the parameters, it is needed to add the type of the result.

e-Macao-16-5-492

Defining SOAP Version

It is possible to define the SOAP version in the Call object:

```
import org.apache.axis.soap.SOAPConstants;
. . .
. . .
call.setSOAPVersion(SOAPConstants.SOAP12_CONSTANTS);
```

The default version of SOAP is 1.1.

e-Macao-16-5-493

Task 66: Parameters and Version

Objective: Add to the FileTransferClient the name "fileName" to the argument of the method. The name of the result should be: "outputFile". Generate a SOAP 1.2 envelope.

```
1) cd demos\Axis\ParametersVersion

2) copy:    \demos\Axis\Client\FileTransferClient.java
    to :    \demos\Axis\ParametersVersion
```

e-Macao-16-5-494

Task 67: Parameters and Version

3) edit FileTransferClient.java, adding:

```
   import org.apache.axis.soap.SOAPConstants;

  /* Naming Parameters and Result */
  call.addParameter("fileName",
       org.apache.axis.Constanst.XSD_STRING,
                    javax.xml.rpc.ParameterMode.IN);
  call.setReturnType (org.apache.axis.Constants.XSD_BASE64);

   /* Defining SOAP Version */
  call.setSOAPVersion(SOAPConstants.SOAP12_CONSTANTS);
```

```
4) javac FileTransferClient.java

5) java -cp \demos\Axis\ParametersVersion
        FileTransferClient
        c:\WebServices\MacaoNews.txt Macao.txt
```

e-Macao-16-5-495

Inserting a Header Example

Let us remember the header of our example:

```
<soapenv:Header>
 <ns1:authentication
   soapenv:actor="http://manager"
   soapenv:mustUnderstand="0"
    xmlns:n1="http://localhost:8080/axis/services/FileDownloadService">
   <ns1:username>admin</ns1:username>
   <ns1:password>admin</ns1:password>
 </ns1:authentication>
</soapenv:Header>
```

1) one header: authentication
2) two attributes: actor and mustUnderstand
3) two sub-elements: username and password

e-Macao-16-5-496

Implementing Headers 1

1) define a SOAP Header Element with the name of the header:

```
String nameSpace =
  "http://localhost:8080/axis/services/FileDownloadService";
SOAPHeaderElement she = new
   SOAPHeaderElement(XMLUtils.StringToElement(nameSpace,
                    "authentication", ""));
```

2) define the attributes:

```
she.setRole("http://manager");
she.setMustUnderstand(false);
```

e-Macao-16-5-497

Implementing Headers 2

3) define the sub-elements and its contents:

```
String nameSpace =
   "http://localhost:8080/axis/services/FileDownloadService";
MessageElement username = new
                MessageElement(nameSpace,"username");
username.addTextNode("admin");

MessageElement password = new
                 MessageElement(nameSpace,"password");
password.addTextNode("admin");
```

4) add the childs to the header element:

```
she.addChild(username);
she.addChild(password);
```

5) create the header:

```
call.addHeader(she);
```

e-Macao-16-5-498

Task 68: Inserting a Header

Objective: add a header to the request message.

1) `cd demos\Axis\Headers`

2) `copy: \demos\Axis\ParametersVersion\FileTransferClient.java`
   `to : \demos\Axis\Headers`

3) edit `FileTransferClient` and add the lines of code to generate the header

4) `javac FileTransferClient.java`

5) `java -cp \demos\Axis\ParametersVersion`
   `FileTransferClient`
   `c:\WebServices\MacaoNews.txt Macao.txt`

e-Macao-16-5-499

Dynamic Binding

Using one of the WSDL-aware Service() constructors, Axis extracts the meta-data (endpoint, parameters and return types) from the WSDL file and prepares the Call.

The only missing information is the port and the operation. Some possible formats include:

`service.createCall (QName portName)` Returns a Call that has been initialized with the endpoint address referred to by the named port.
`service.createCall (QName portName, QName operation)` Returns a Call that has been initialized with the endpoint address and also with all the parameters and return types.
`service.createCall (QName portName, String operation)` The same as the previous format, but it accepts the unqualified operation name.

e-Macao-16-5-499

Dynamic Binding

Using one of the WSDL-aware Service() constructors, Axis extracts the meta-data (endpoint, parameters and return types) from the WSDL file and prepares the Call.

The only missing information is the port and the operation. Some possible formats include:

`service.createCall (QName portName)` Returns a Call that has been initialized with the endpoint address referred to by the named port.
`service.createCall (QName portName, QName operation)` Returns a Call that has been initialized with the endpoint address and also with all the parameters and return types.
`service.createCall (QName portName, String operation)` The same as the previous format, but it accepts the unqualified operation name.

e-Macao-16-5-501

Dynamic Binding Example 2

Using the following method:

```
createCall (QName portName, java.lang.String operationName)
```

The example looks like:

```
QName portName = new QName(nameSpace, "FileDownloadService");
String operationName = "downloadFile";
Call call = (Call)service.createCall( portName, operationName);
```

```
- <wsdl:service name="FileDownloadService">
  - <wsdl:port binding="impl:FileDownloadServiceSoapBinding" name="FileDownloadService">


- <wsdl:binding name="FileDownloadServiceSoapBinding" type="impl:FileDownload">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="downloadFile">
```

e-Macao-16-5-502

Task 69: Dynamic Binding

Objective: Develop the FileTransferClient using a WSDL-aware Service() constructor. Use FileTransferTemplate.java

```
1) cd demos\Axis\Dynamic

2) dir
FileTransferTemplate.java
```

3) edit `FileTransferTemplate.java` and save it as `FileTransferClient.java`

e-Macao-16-5-503

Task 70: Dynamic Binding

4) add the following instructions:

```
String wsdlLocation =
"http://localhost:8080/axis/services/FileDownloadService?wsdl";
String nameSpace =
    "http://localhost:8080/axis/services/FileDownloadService";
QName serviceName = new QName(nameSpace, "FileDownloadService");
Service service = new Service(wsdlLocation, serviceName);

QName portName = new QName(nameSpace, "FileDownloadService");
String operationName = "downloadFile";
Call call = (Call)service.createCall( portName, operationName);

5)  javac FileTransferClient.java

6)  java -cp \demos\Axis\Dynamic FileTransferClient
        c:\WebServices\MacaoNews.txt Macao.txt
```

e-Macao-16-5-504

Using Sessions

In some cases, it is useful if the server is able to “remember” data related to various invocations of the same client.

Some kind of session is needed to associate some data with a given client.

Axis has some simple APIs for session support.

e-Macao-16-5-505

Session Implementations

Axis provides two built-in ways to maintain sessions across web services connections, using:

1) standard HTTP session mechanisms
2) SOAP headers
3) WS-Resource Framework

e-Macao-16-5-506

Sessions with HTTP

Uses HTTP cookies to store session state:

1) the server transmits to the client some kind of cookie
2) the server accepts the same cookie from the client on subsequent requests
3) the server realizes that the new requests are associated with the same client

The actual session data is kept by the servlet framework.

It is needed:

1) to use HTTP
2) to call `setMaintainSession(true)` on either the Call or the Service object

e-Macao-16-5-507

Sessions with SOAP Headers

To use this approach, the `SimpleSessionHandler` must be deployed.

The handler `org.apache.axis.handlers.SimpleSessionHandler` is included with Axis.

In order to work, this handler must be deployed in the global request and response flows of the client in order to work.

e-Macao-16-5-508

Sessions with WS-Resource

WS-Resource Framework consists of five specifications:

1) WS-ResourceProperties
2) WS-ResourceLifetime
3) WS-ServiceGroup
4) WS-RenewableReferences
5) WS-BaseFaults

and an approach to modeling statefull resource using web services.

The WS-Resource Framework was developed by Computer Associates, Fujitsu, Globus, Hewlett-Packard and IBM.

It was submitted to OASIS.

A.5.3. Tools and Configuration

e-Macao-16-5-509

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-510

Axis Tools

Axis comes with the following tools:

1) WSDL2Java: based on WSDL descriptions generates Java code for the client and server side
2) Java2WSDL: is a command-line tool for taking Java interfaces and generating WSDL
3) generation of WSDL: at runtime, the Axis engine automatically generates WSDL for the deployed services

e-Macao-16-5-511

WSDL2Java

WSDL2Java automatically builds stubs.

A Stub is a Java class with a Java-friendly API that closely matches the Web service interface defined in a given WSDL document.

The tool is executed from the command line:

```
java org.apache.axis.wsdl.WSDL2Java  WSDL_Document_URL
```

For instance:

```
java org.apache.axis.wsdl.WSDL2Java
  http://localhost:8080/axis/services/FileDownloadService?wsdl
```

e-Macao-16-5-512

WSDL2Java Example 1

Executing WSDL2Java for our example, the following Java classes are generated:

1) FileDownload.java: this is the service interface . In JAX-RPC is known as Service Endpoint Interface (SEI).

```
/**
 * FileDownload.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST)
   WSDL2Java emitter.
 */

package localhost.axis.services.FileDownloadService;

public interface FileDownload extends java.rmi.Remote {
    public byte[] downloadFile(java.lang.String in0) throws
    java.rmi.RemoteException;
}
```

e-Macao-16-5-513

WSDL2Java Example 2

2) FileDownloadService: this interface allows type-safe access to the SEI from the locator class. Includes two methods:

   a) `getFileDownloadService()`: gets an implementation of the FileDownload interface that will call the endpoint specified in the WSDL
   b) `getFileDownloadService(URL url)`: gets a FileDownload stub that uses the same WSDL interface but points at a different endpoint

3) FileDownloadServiceLocator.java: this class implements the FileDownloadService interface and acts as the factory for stub instances

4) FileDownloadServiceSOAPBindingStub.java: the class that implements the FileDownload interface – the core of the client

e-Macao-16-5-514

Task 71: Execute WSDL2Java

Objective: execute WSDL2Java for the FileDownloadService

```
1) cd demos\Axis\WSDL2Java

2) java org.apache.axis.wsdl.WSDL2Java
  http://localhost:8080/axis/services/FileDownloadService?wsdl

3) cd localhost\axis\services\FileDownloadService

4) dir
FileDownload.java
FileDownloadService.java
FileDownloadServiceLocator.java
FileDownloadServiceSOAPBindingStub.java

5) notepad FileDownload*.java
```

e-Macao-16-5-515

Tools for Invoking a Service

Two ways for invoking a service from the client:

1) using a generic stub: instantiating a service and a call object

2) using a specific stub: invoking the interfaces generated by WSDL2Java based on the WSDL file

e-Macao-16-5-516

Task 72: Using a Specific Stub

Objective: testing the service with a client that uses the interfaces generated by WSDL2Java.

1) `cd \demos\Axis\TestStubs`

2) `dir`
`FileTransferTestTemplate.java`

3) generate the stubs:
```
java org.apache.axis.wsdl.WSDL2Java
http://localhost:8080/axis/services/FileDownloadService?wsdl
```

4) compile the generated classes:
```
cd localhost\axis\services\FileDownloadService
javac *.java
```

e-Macao-16-5-517

Task 73: Using a Specific Stub

5) `cd \demos\Axis\TestStubs`

6) edit `FileTransferTestTemplate.java`, adding:

```
import localhost.axis.services.FileDownloadService.*;

/* Get the stub from the locator object */
FileDownloadServiceLocator locator = new
  ileDownloadServiceLocator();
FileDownload stub = locator.getFileDownloadService();

/* Call the web service
 byte[] ret = stub.downloadFile(fileName);
```

save it as FileTransferTest.java

e-Macao-16-5-518

Task 74: Using a Specific Stub

Compile:
```
7) javac -classpath \demos\Axis\TestStubs
            FileTransferTest.java
```

Execute:
```
8) java -cp \demos\Axis\TestStubs FileTransferTest
          c:\WebServices\MacaoNews.txt Macao.txt
```

9) `dir`

e-Macao-16-5-519

Using WSDL2Java for Services 1

It is possible to take a WSDL description of a Web Service and create a skeleton implementation of the service described by the WSDL.

Specifying the option `-s` to WSDL2Java, in addition to the client and data classes, will generate for the FileDownloadService:

1) FileDownloadServiceSOAPBindingImpl.java: the framework implementation of the service
2) deploy.wsdd: a pre-built deployment file for use with the AdminClient
3) undeploy.wsdd: a pre-built undeployment file

e-Macao-16-5-520

Using WSDL2Java for Services 2

The FileDownloadServiceSOAPBindingImpl.java looks like:

```
/**
 * FileDownloadServiceSoapBindingImpl.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST)... */

package localhost.axis.services.FileDownloadService;

public class FileDownloadServiceSoapBindingImpl implements
 localhost.axis.services.FileDownloadService.FileDownload{
    public byte[] downloadFile(java.lang.String in0) throws
 java.rmi.RemoteException {
        return null;
    }

}
```

e-Macao-16-5-521

Task 75: Using WSDL2Java More

Objective: execute WSDL2Java for the FileDownloadService with –s option

```
1)  cd demos\Axis\WSDL2JavaWS

2)  java org.apache.axis.wsdl.WSDL2Java -s
   http://localhost:8080/axis/services/FileDownloadService?wsdl

3) cd localhost\axis\services\FileDownloadService

4) dir
deploy.wsdd
FileDownload.java
FileDownloadService.java
FileDownloadServiceLocator.java
FileDownloadServiceSOAPBindingImpl.java
FileDownloadServiceSOAPBindingStub.java
undeploy.wsdd

5) notepad deploy.wsdd, undeploy.wsdd,
          FileDownloadServiceSOAPBindingImpl.java
```

e-Macao-16-5-522

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-523

WSDD Overview

WSDD is an XML format that Axis uses to store its configuration and deployment information.

The Axis server keeps its configuration in a file: `server-config.wsdd`.

The Axis client has an equivalent file: `client-config.wsdd`.

These files have default versions stored in `axis.jar`

e-Macao-16-5-524

WSDD Structure

In order to deploy web services the root element of WSDD is `deployment`.

Inside the root element is a `<globalConfiguration>` element which contains options for the Axis engine plus definitions for the global request and response chains.

```
<globalConfiguration>
  <parameter name="defaultSOAPVersion" value="1.2" />
      <requestFlow>
          <handler type="requestHandler"/>
      </requestFlow>
      <responseFlow>
          <handler type="responseHandler"/>
      </responseFlow>
</globalConfiguration>
```

e-Macao-16-5-525

Global Configuration

The `parameter` declarations inside the `globalConfiguration` set options on the AxisEngine. For instance: SOAP version.

The `<requestFlow>` and `<responseFlow>` elements define the request and response global chains. They can contain either `<handler>` elements or `<chain>` elements.

The components inside `<requestFlow>` and `<responseFlow>` are invoked in exactly the same order as they are declared in the XML file.

e-Macao-16-5-526

Handler Declarations

Handler declarations tell Axis that a given Java class is a handler, allowing:

1) configure it with a set of options
2) name the configuraion, so it is possible to refer to it

```
<handler [name="name"] type="type">
   <parameter name="name" value="value />
</handler>
```

For example:

```
<handler name="requestHandler" type="java:MyRequestHandler">
    <parameter name="filename" value="SOAPRequest.log"/>
</handler>
```

This handler can be referenced:

```
<handler type="requestHandler"/>
```

e-Macao-16-5-527

Chain Definitions

It is possible to group a series of handlers into a chain.

```
<chain [name="name"] >
   <handler type="type">
     <parameter name="name" value="value" />
   </handler>
</chain>
```

For instance:

```
<chain name="logAndNotify" >
   <handler type="java:LogHandler" />
   <handler type="java:NotifyHandler" />
</chain>
```

This chain can be referenced as:

```
<requestFlow>
   <handler type="logAndNotify" />
</requestFlow>
```

e-Macao-16-5-528

Transport Declarations

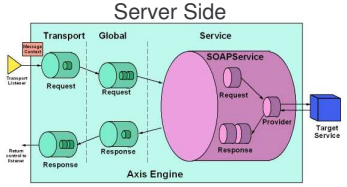
Transport declarations define a named, targeted chain, which has:

1) a requestFlow,
2) a responseFlow,
3) a pivot handler (only on the client)

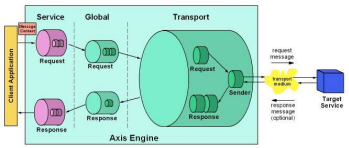
On the client, the pivot handler is the sender of the message.

On the server there is no need for a pivot handler.

Server Side



Client Side



e-Macao-16-5-529

Transport Handler: Client Example

Example from the client-config.wsdd:

```
<transport name="http"
   pivot="java:org.apache.axis.transport.http.HTTPSender" />
```

A pivot handler is defined using the `pivot` attribute.

e-Macao-16-5-530

Transport Handler: Server Example

Example from the server-config.wsdd:

```
<transport name="http">
 <requestFlow>
  <handler type="URLMapper"/>
  <handler
     type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
 </requestFlow>
...
```

This transport is named http and contains two-transport specific handlers:

1) URLMapper: sets the Axis service name in the MessageContext based on the HTTP URL
2) HTTPAuthHandler: takes the username and password out of the HTTP Basic authentication header and puts them in the MessageContext

e-Macao-16-5-531

Type Mapping

Type mappings control the mapping between Java classes and XML structures.

It is possible to tell the engine to map a particular Java class to a particular XML type, and even customize the serializer and deserializer classes.

```
<typeMapping qname="typeQName"
                        type="java:classname"
                        serializer="Serializer"
                        deserializer="DeserializerFactory"
                        encodingStyle="uri" />
```

e-Macao-16-5-532

Bean Mapping

The <beanMapping> tag is a shorthand for a <typeMapping>, which uses BeanSerializer, and BeanDeserializer classes to do the Axis's default data-mapping algorithms.

```
<beanMapping qname="typeQName"
             languageSpecificType="java:classname"
             encodingStyle ="url"/>
```

For instance, this mapping was used in the example of SOAP encoding, where the web service was returning the attributes of the file as an object:

```
<beanMapping qname="ns:FileAttribute"
                   xmlns:ns="urn:FileAttribute"
             languageSpecificType="java:FileAttribute"/>
```

A.5.4. Service Deployment

e-Macao-16-5-533

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-534

WSDD for Services

The entire configuration of the Axis server is contained in: server-config.wsdd

Each deployed service in the server, has a `<service>` element in the WSDD file with the following syntax:

```
<service name="name"
                [style="rpc | wrapped | document | message"]
               [use="literal | encoded"]
               [provider="provider"] >
   <operation>*
   <typeMapping>* | <beanMapping>*
   <namespace>uri</namespace>*
   <wsdlFile>absolute-filename</wsdlFile>
   <endpointURL>uri</endpointURL>
   <handlerInfoChain>
   <parameter name="name" value="value" />
</service>
```

* means that the element may appear zero or more times.

e-Macao-16-5-535

Attributes 1

The `name` attribute contains the name of the service.

The `style` attribute specifies one of several different ways that Axis can map SOAP messages to and from Java method calls: rpc, document, wrapped or messsage.

If a style is specified, there is no need to specify the `use` or `provider` attribute, since they will have a default value based on the style chosen.

The `use` attribute specifies encoded or literal use. The default value is based on the `style` attribute.

e-Macao-16-5-536

Attributes 2

The `provider` attribute allows to specify a QName representing the particular provider:

1) Java:RPC: used for rpc, document, and wrapped styles
   a) this provider is automatically selected if the style is rpc, document or wrapped
   b) using RPC provider, the class name and the methods allowed must be specified:

```
<parameter name="className" value="class_name"/>
<parameter name="allowedMethods" value="m1 m2" />
```

2) Java:MSG: used for message style
   a) the message provider will dispatch raw XML to the service
3) Java:EJB: allows to use an Enterprise JavaBean as a web service
4) Handler: lets specify a user-defined handler for a particular service

e-Macao-16-5-537

Some Elements

If `<typeMapping>` or `<beanMapping>` are defined inside a service deployment, those XML/Java mappings will hold only for the service.

If the `<namespace>` element is present, the first one is the default namespace for the service.

The `<wsdlFile>` element allows to specify a custom WSDL file that the engine will return when asked about the WSDL for the service.

e-Macao-16-5-538

JAX-RPC Handlers Element

The `<handlerInfoChain>` element is used to enable deploying JAX-RPC style handlers:

```
<handlerInfoChain>
   <handlerInfo class="className" >
      <parameter name="name" value="value"/>
      <header qname="qname" />*
      <role SOAPActorName="uri" />
   </handlerInfo>*
</handlerInfoChain>
```

JAX-RPC handlers specified in this chain will run after the global chain, but before the request flow of the service.

JAX-RPC handlers have two methods: `handleRequest()` and `handleResponse()`, while Axis handlers only have `invoke()`.

Each `<handlerInfo>` defines a single JAX-RPC handler.

e-Macao-16-5-539

Operation Element 1

A service can contain zero or more `<operation>` elements.

The `<operation>` element is used when more fine-grained control of the options of a particular operation is desirable.

It handles the mapping from arbitrary XML QNames in the SOAP body to arbitrary Java methods, controlling:

1) how parameters to those methods map to XML elements
2) how the exceptions thrown by the Java methods are map to and from SOAP faults

e-Macao-16-5-540

Operation Element 2

```
<operation name="name" [qname="qname"] [returnQName="qname"]
        [returnType="qname"] [returnHeader="true" | "false"]>
 <parameter [qname="qname" | name="name"]
            [mode="in | out | inout"]
            type="qname"
            inHeader="true | false" outHeader="true | false" />*
 <fault name="name" qname="qname"
            class="classname" type="qname" />*
</operation>
```

The operation `name` is the name of the Java method this web service operation will invoke.

The `qname` is the QName of the XML element that will map to this operation.

e-Macao-16-5-541

Operation Element 3

Inside the `operation` element are zero or more `parameter` elements, each represents a parameter of the operation.

If the `inheader` or `outheader` attribute is specified for the parameter, then the serialization of the parameter will be in the SOAP header or in the SOAP body respectively.

The data related to the faults thrown by the service and specified in the `fault` element, will be serialized inside a fault class as a `<detail>` element with the specified QName and XML type.

e-Macao-16-5-542

Deploying Services

Two ways:

1) directly edit the server-config.wsdd
2) use the AdminClient tool

e-Macao-16-5-543

Admin Client

```
> java org.apache.axis.client.AdminClient
            [-u {username}] [-w {password}]
            [-p {port}] [-l {service-url}] {wsdd-file}
```

Executing this from the command line:
1) reads the WSDD file
2) attempts to deploy the service to the Axis engine

If authentication is required the `username` and `password` arguments are used.

If the WSDD has `<deployment>` as the root element, all the components in the WSDD are deployed.

All the classes referred in the WSDD must be available on the server's classpath before doing the deployment.

e-Macao-16-5-544

Undeploying Web Services

If the WSDD document has `<undeployment>` as its root element, all the referenced components will be removed from the running server.

For undeploying services, only the name of the components to undeploy are specified.

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/>
    <handler name="MyHandler">
    <service name="MyService">
</undeployment>
```

e-Macao-16-5-545

Task 76: Undeploying a Service

Objective: undeploy the FileUploadService service.

1) Make a copy of the file server-config.wsdd:
```
copy: \Tomcat 4.1\webapps\axis\WEB-INF\server-config.wsdd
  to: server-configbup.wsdd
```

2) Undeploy the service:
```
a) cd \demos\Axis\Undeploy
```

b) write the undeploy.wsdd file to undeploy the service:
```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
    <service name="FileUploadService" />
</undeployment>
```

```
c) java org.apache.axis.client.AdminClient
            undeploy.wsdd

d) browse: http://localhost:8080/Axis --> View
```

e-Macao-16-5-546

Task 77: Return to Previous State

Objective: return to the previous state (including the FileUploadService).

```
1) cd \Tomcat 4.1\webapps\axis\WEB-INF

2) java org.apache.axis.client.AdminClient
3)             server-configbup.wsdd

4) browse: http://localhost:8080/Axis --> View
```

A.5.5. Service Lifecycle

e-Macao-16-5-547

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-548

Service Lifecycle and Scope

Some questions that should be answered during design include:

1) how is the web service object created?
2) how many web service objects will exist?
3) can the objects be shared across multiple threads at once?

These issues can be specified when deploying a web service, using the `scope` option.

```
<service name="service_name">
  . . .
  <parameter name="scope"
             value = "[application | request | session]" />
</service>
```

e-Macao-16-5-549

Scope

The valid values for the scope option are:

1) application:
   a) only a single instance of the service class for the entire Axis engine
   b) all methods must be thread-safe – many active requests in parallel for the same code
   c) any state of the service object will be shared across all invocations

2) request:
   a) a new service will be created for every SOAP request
   b) constructors should not have expensive initialization code
   c) default value

3) session:
   a) Are created once per client session
   b) data fields hold state on a per-session basis

e-Macao-16-5-550

Creation and Destruction

JAX-RPC provides an interface called: javax.xml.rpc.server.ServiceLifecycle

This interface contains two methods:

```
1) void init (Object context) throws ServiceException;
2) void destroy();
```

When a service is created or destroyed by the Axis runtime, the engine checks to see if the objects implements one of these interfaces.

Implementing these methods, initialization or cleanup may be done.

When a new service object is created, `init` is called with a context object allowing the service to access the MessageContext.

e-Macao-16-5-551

Sessions on the Server Side

Axis provides an abstraction to manage sessions in the server side in the org.apache.axis.session package.

A given interaction can optionally be associated with a session.

The MessageContext has a slot in it for the currently active session.

A `Session` object lets you to store values in a library indexed by `String` keys – like a map or hash-table.

Data stored in a `Session` during one interaction will be available again on the next interaction of the same client.

e-Macao-16-5-552

Accessing Sessions

There are two built-in ways for accessing sessions on the Axis server:

1) using the servlet HTTPSession: the servlet engine will handle time out
2) using SimpleSession: the SimpleSessionHandler will periodically read expired sessions

It is possible to set the timeout on a session with:

```
session.setTimeout(int)
```

The session implementations included in Axis are not persistent – data will be lost in case of a server crash or restart.

A.5.6. Summary

e-Macao-16-5-553

AXIS Outline

1) Overview
2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions
3) AXIS Tools
4) AXIS Configuration
5) Service Deployment
6) Service Lifecycle
7) Summary

e-Macao-16-5-554

AXIS Summary 1

1) Axis is an engine for processing SOAP messages
2) the Axis engine invokes a series of handlers to process messages
3) handlers are built-in the engine or can be included in a module defined by the user
4) handlers may:
   1) receive a request and send a response
   2) process a request and produce a response – pivot handlers
5) handlers are grouped in chains.

e-Macao-16-5-555

AXIS Summary 2

1) Axis defines different processing levels:
   a) on the server: transport – global – service
   b) on the client: service – global – transport
2) the object passed through the different handles in all these levels is the MessageContext
3) the MessageContext structure includes:
   a) the request message
   b) the response message
   c) several properties
   d) other fields

e-Macao-16-5-556

AXIS Summary 3

1) JAX-RPC is Java API for XML-based RPC
2) JAX-RPC facilitates communication between Java and non-Java platforms
3) JAX-RPC is designed as a Java API for web services
4) JAX-RPC supports:
   a) SOAP and WSDL
   b) RPC encoded messaging
   c) SOAP with Attachments

e-Macao-16-5-557

AXIS Summary 4

1) Axis provides APIs for implementing SOAP classes.
2) it is possible to access, add and delete SOAP Envelope elements
3) the client APIs can be divided in:
   a) dynamic invocation: using pre-existing Java classes
   b) stub generation: using code generated by WSDL2Java tool
4) dynamic invocation: uses a Service and a Call object, where the endpoint address and the operation name of the service are defined
5) the endpoint address and the operation name can be:
   a) directly hard-coded in the Java program
   b) extracted automatically by Axis from the WSDL file, using WSDL-aware service constructors

e-Macao-16-5-558

AXIS Summary 5

1) It is possible to manage sessions on web services, by:
   a) standard HTTP session mechanisms
   b) SOAP headers
   c) WS-Resource Framework
2) Axis provides several tools:
   a) WSDL2Java: generates Java code for the client and the server based on WSDL
   b) Java2WSDL: generates WSDL based on Java interfaces
   c) generation of WSDL: at runtime when deploying services

e-Macao-16-5-559

AXIS Summary 6

1) Axis uses an XML file called deployment descriptor to:
   a) deploy services
   b) undeploy services
   c) customize the engine

2) the scope parameter when deploying the service allows to define its lifecycle:
   a) application
   b) request
   c) session

3) JAX-RPC provides two methods that can be invoked when the service object is created or destroyed

4) Axis engine looks if these methods are implemented by the service, and invoke them accordingly.

e-Macao-16-5-560

Axis Summary 7

Accessing the envelope:

```
Message requestMsg = mc.getRequestMessage();
SOAPEnvelope env = requestMsg.getSOAPEnvelope();
Vector headers = env.getHeaders();
System.out.println("There are " + headers.size() + " headers.\n" );

Message responseMsg = mc.getResponseMessage();
env = responseMsg.getSOAPEnvelope();
SOAPBodyElement body = env.getFirstBody();
```

Example:
demos\SourceFiles\Axis\Envelope

e-Macao-16-5-561

Axis Summary 8

Adding a Header:

```
String nameSpace =  "http://localhost:8080/axis/services/FileDownloadService";
SOAPHeaderElement she = new
     SOAPHeaderElement(XMLUtils.StringToElement(nameSpace,"authentication", ""));

she.setRole("http://manager");
she.setMustUnderstand(false);

MessageElement username = new MessageElement(nameSpace,"username");
username.addTextNode("admin");

MessageElement password = new MessageElement(nameSpace,"password");
password.addTextNode("admin");

she.addChild(username);
she.addChild(password);

call.addHeader(she);
```

Example: demos\SourceFiles\Axis\Header

e-Macao-16-5-562

Axis Summary 9

Invoking a Service without referencing to the WSDL document:

```
/*  the creation of the Service and the Call */
Service service = new Service();
Call call = (Call)service.createCall();

/* the definition of a variable "endpoint" containing the address of the service */
String endpoint = "http://localhost:8080/axis/services/FileDownloadService";

/* the method to set the endpoint address in the Call object */
call.setTargetEndpointAddress( new java.net.URL(endpoint) );

'* the definition of the operation name */
call.setOperationName(new QName("http://soapinterop.org/", "downloadFile"));

/* the invocation */
byte[] ret = (byte[])call.invoke( new Object[] { args[0] } );
```

Example: \demos\SourceFiles\Axis\Client

e-Macao-16-5-563

Axis Summary 10

Invoking a Service referencing to the WSDL document:

```
String wsdlLocation = "http://localhost:8080/axis/services/FileDownloadService?wsdl";
String nameSpace = "http://localhost:8080/axis/services/FileDownloadService";
QName serviceName = new QName(nameSpace, "FileDownloadService");
Service service = new Service(wsdlLocation, serviceName);

QName portName = new QName(nameSpace, "FileDownloadService");
String operationName = "downloadFile";
Call call = (Call)service.createCall( portName, operationName);

byte[] ret = (byte[])call.invoke( new Object[] { args[0] } );
```

Example: \demos\SourceFiles\Axis\Dynamic

e-Macao-16-5-564

Axis Summary 11

Invoking a Service using a generated stub:

```
/* Get the stub from the locator object */
FileDownloadServiceLocator locator = new FileDownloadServiceLocator();
FileDownload stub = locator.getFileDownloadService();

/* Call the web service */
byte[] ret = stub.downloadFile(fileName);
```

Example: \demos\SourceFiles\Axis\TestStubs

e-Macao-16-5-565

Axis Summary 12

Developing a Handler:

```
public class MyRequestHandler extends BasicHandler{
  SOAPEnvelope env;
  public MyRequestHandler() {
  }

  public void invoke(MessageContext msgContext) throws AxisFault {
      try{
```

Example: \demos\SourceFiles\Axis\Handlers

A.6. UDDI

UDDI

e-Macao-16-5-567

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

A.6.1. Introduction

e-Macao-16-5-568

UDDI Outline

1) Introduction
2) Concepts
3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag
4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service
5) Summary

e-Macao-16-5-569

UDDI Motivation

Once services have been properly described, these descriptions should be made available to those interested in using them.

Service discovery is a process for locating service providers and retrieving service description documents that have been published in a service registry.

Two types of service discovery:

1) static: occurs at application design time and is done by a human designer
2) dynamic: occurs at runtime and is done by the application

For doing so, it is needed to standardize the web service registry.

Such standardization is done by UDDI project.

e-Macao-16-5-570

UDDI Project

UDDI project is an industry initiative attempting to create a platform-independent, open framework for:

1) describing,
2) discovering
3) integrating

business services.

UDDI specifications define a registry service for:

1) web services
2) other electronic and non-electronic services.

e-Macao-16-5-571

UDDI Registry Service

A UDDI registry service is a web service.

The UDDI registry service manages information about:

1) service providers
2) service implementations
3) and service metadata

e-Macao-16-5-572

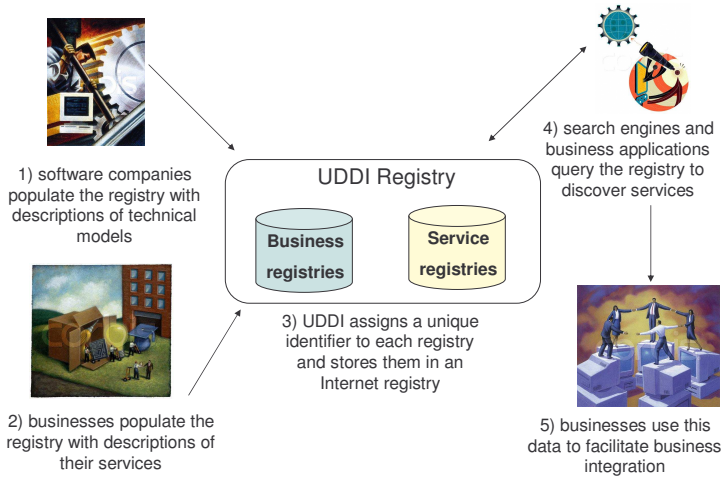
UDDI Usage

UDDI is used by:

1) providers - use UDDI to advertise their services
2) consumers - use UDDI to:
   a) discover services that suit their requirements
   b) obtain the service metadata needed to consume them

e-Macao-16-5-573

UDDI Process



e-Macao-16-5-574

UDDI Goals

Primary goal of UDDI:

- the specification of a framework for describing and discovering web services.

Two main goals for UDDI registry specifications:

1) support developers in finding information about services
2) enable dynamic binding

e-Macao-16-5-575

UDDI Data Structures and APIs

UDDI defines data structures and APIs for:

1) publishing service descriptions in the registry
2) querying the registry to look for published descriptions

e-Macao-16-5-576

UDDI History

The initiative was announced on 6th September 2000.

The first three UDDI versions were developed by uddi.org.

After completion of UDDI version 3.0, uddi.org submitted the specifications to the Organization for the Advancement of Structured Information Standards (OASIS).

In April 2003, the UDDI version 2.0 specifications were approved as a formal OASIS standard.

UDDI version 3.0.2 has been published as a Committee Draft in October 2004.

The specifications are available at:
http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

e-Macao-16-5-577

UDDI Services

UDDI defines a number of lookup services allowing clients to look up and retrieve information to access a web service.

Four services are provided to clients:

1) white pages to look up a web service by business identification
2) yellow pages to look up a web service by topic
3) green pages for searches through web services features

A service for providers:

4) UDDI business registry to publish/request information about web services

e-Macao-16-5-578

UDDI Members

UDDI members are companies that are committed to the:

1) enhancement,
2) evolution
3) world-wide acceptance

of the UDDI registry.

For instance, some UDDI members:

a) Cisco Systems
b) IBM
c) Intel
d) Microsoft
e) NEC Corporation
f) Oracle
g) SAP
h) Sun Microsystems
i) ...

e-Macao-16-5-580

Different Operators – One Registry

A company needs to register its information with only one operator, called the custodian.

UBR is based on: “register once, publish everywhere”.

The information contained in one registry is replicated in the other registries, not instantaneously, but at least every 12 hours.

A company can update its information only through its custodian.

e-Macao-16-5-581

UDDI Specifications

The UDDI specifications define:

1) SOAP APIs that applications use to query and to publish information to a UDDI registry
2) XML Schema of the registry data model and the SOAP message formats
3) WSDL definitions of the SOAP APIs
4) UDDI registry definitions of various identifier/category systems that may be used to identify and categorize UDDI registrations

A.6.2. Concepts

e-Macao-16-5-582

UDDI Outline

1) Introduction
2) Concepts
3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag
4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service
5) Summary

e-Macao-16-5-583

UDDI - UUID

A service registry maintains information about
1) businesses
2) services
3) technical information
4) specification of services

Instances of these data structures are kept separate and are identified by a Universally Unique Identifier (UUID).

UUIDs are assigned when the data structure is first inserted in the registry.

UUIDs are hexadecimal strings whose structure and generation algorithm is defined by the ISO/IEC 11578:1996 standard.

e-Macao-16-5-584

UDDI – Other Identifiers

UDDI allows to define additional international identifiers.

These identifiers are assigned to business and technical information in order to retrieve data according to them.

Two identifier types have been adopted and made core part of the UDDI operator registries:

1) D-U-N-S
2) Thomas Register

e-Macao-16-5-585

D-U-N-S Identifier

D-U-N-S number is a unique nine-digit identification sequence which provides unique identifiers of single business entities.

D-U-N-S is provided by Dun & Bradstreet.

Dun & Bradstreet is a company providing global business information, tools, and insights.

Reference: http://www.dnb.com

e-Macao-16-5-586

Thomas Register Identifier

Thomas Register identifiers are used in Thomas Global Register.

The Thomas Global Register is a directory of manufacturers and distributors from 28 countries.

The registry is classified by products and services categories.

Reference: http://www.thomasregister.com

e-Macao-16-5-587

UDDI Core Identifier Systems

The UDDI - UUIDs for these identifier systems are:

Name	UUID
D-U-N-S	uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823
Thomas Registry	uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039

In order to take advantage of these identification systems, businesses need to provide the relevant codes when publishing information.

This information is provided using the `identifierBag` element.

e-Macao-16-5-588

Categorization and Classification

In order to improve the search procedure UDDI provides a method to perform intelligent searches through categorization and classification.

Categorization: is the process of creating categories.

Classification: is the process of assigning objects to these predefined categories.

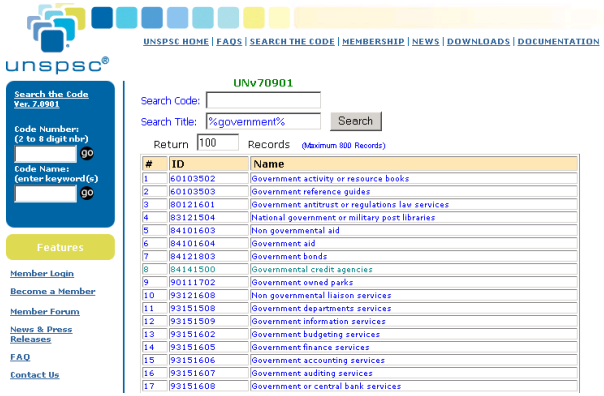
UDDI defines a set of built-in classification schemes (or taxonomies):

North American Industry Classification System (NAICS) http://www.census.gov/epcd/www/naics.html	classifies businesses by industry
United Nations Standard Products and Services Code (UNSPSC) http://www.unspsc.org/	classifies products and services
ISO 3166 http://www.iso.org/iso/en/prods-services/iso3166ma	classifies geographic locations

e-Macao-16-5-589

Classification Codes: UNSPSC

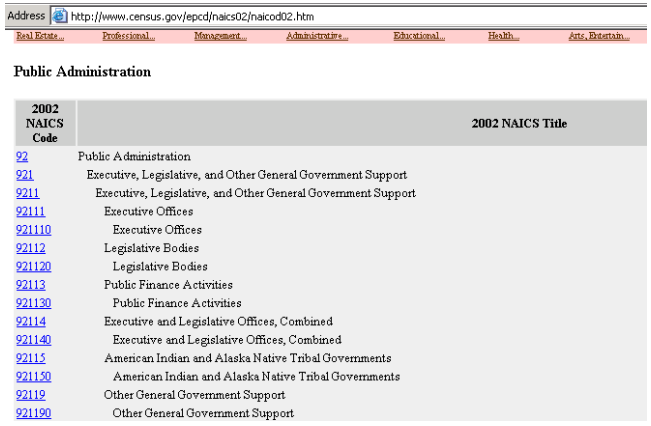
Examples of UNSPSC classification codes:



e-Macao-16-5-590

Classification Codes: NAICS

Examples of NAICS classification codes for public administration:



e-Macao-16-5-591

UDDI Classification Schemes

The UDDI - UUIDs for the classification schemes are:

Name	Type	UUID
NAICS	business	uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
UNSPSC	product and services	uuid:CD153257-086A-4237-B336-6BDCBDCC6634
ISO 3166	geographic	uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88

In order to take advantage of these classification schemes businesses need to provide the relevant classification information as they publish their entries.

This is done using the `categoryBag` element.

e-Macao-16-5-592

UDDI Core tModels

UDDI registries store `tModels`.

tModels represent technical specifications.

UDDI describes the classification schemes NAICS, UNSPSC and ISO 3166 as `tModels`.

e-Macao-16-5-593

UDDI Type Taxonomy

UDDI type taxonomy has been established to assist in general categorization of `tModels`.

UDDI type taxonomy is described as a tModel:

```
tModel name= uddi-org:types
tModel description = UDDI Type Taxonomy
tModel UUID: uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4
```

The categorization information for each tModel is added in the `categoryBag` element to indicate the type of `tModel`.

e-Macao-16-5-594

uddi-org:types: Values 1

Some of the values defined in uddi-org:types taxonomy include:

1) `namespace`: represents a scoping constraint or domain for a set of information. Similar to the namespace functionality used for XML
2) `specification`: is used for `tModels` that define interactions with a web service
3) `xmlSpec`: is used to indicate that the interaction with the service is via XML
4) `soapSpec`: is used to indicate that the interaction with the service is via SOAP

e-Macao-16-5-595

uddi-org:types: Values 2

5) `wsdlSpec`: is used to indicate that the web service is described using WSDL
6) `protocol`: is used for a `tModel` describing a protocol of any sort
7) `transport`: is used for a `tModel` specifying specific types of protocols: HTTP, FTP, and SMTP

A.6.3. Data Types

e-Macao-16-5-596

UDDI Outline

1) Introduction
2) Concepts
3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag
4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service
5) Summary

e-Macao-16-5-597

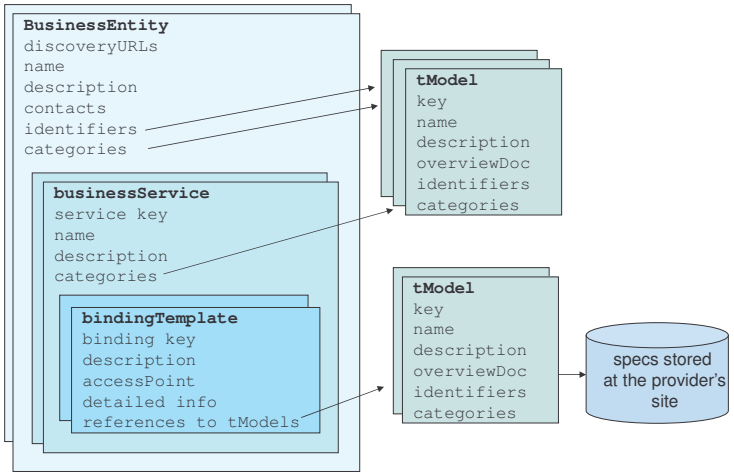
UDDI Data Types

UDDI version 2.0 is modeled using five data types:

1) businessEntity: describes an organization that provides web services
2) businessService: describes a group of related web services offered by a businessEntity
3) bindingTemplate: describes the technical information necessary to use a particular web service
4) tModel: (technical model) is a generic container for any kind of specification
5) publisherAssertion: is used to define a relationship between two or more businessEntity elements

e-Macao-16-5-598

UDDI Data Model



e-Macao-16-5-599

BusinessEntity

The structure is used to represent all known information about a business or entity that publishes descriptive information about the entity as well as the services that it offers.

The structure includes:

1) attributes:
   a) businessKey
   b) operator
   c) authorizedName
2) elements:
   a) discoveryURLs
   b) name
   c) description
   d) contacts
   e) businessServices
   f) identifierBag
   g) categoryBag

e-Macao-16-5-600

BusinessEntity Attributes

1) businessKey: UUID for a given instance of the businessEntity structure

2) operator: is the certified name of the UDDI registry site operator that manages the master copy of the businessEntity data

3) authorizedName: is the recorded name of the individual that published the businessEntity data.

```
<?xml version="1.0" encoding="utf-8" ?>
<businessEntity businessKey="5774466b-089d-4fa8-b8cb-fa2ead5329c5"
   operator="Microsoft Corporation"
   authorizedName="NiceWeather UDDI Publisher"
  . . .
```

e-Macao-16-5-601

BusinessEntity Elements 1

1) discoveryURLs: (optional) is used to hold pointers to alternate, file based service discovery mechanisms.

```
<discoveryURLs>
 <discoveryURL
   useType="businessEntity">http://uddi.microsoft.com/discovery?
   businessKey=5774466b-089d-4fa8-b8cb-fa2ead5329c5
 </discoveryURL>
</discoveryURLs>
```

2) name: (repeating element) are the human readable names recorded for the businessEntity, adorned with a unique xml:lang value

```
<name xml:lang="en">NiceWeather</name>
<name xml:lang="es">BuenTiempo</name>
```

e-Macao-16-5-602

BusinessEntity Elements 2

3) description: (optional repeating element) one or more short business descriptions. One description is allowed per language code supplied.

```
<description xml:lang="en">UDDI businessEntity for NiceWeather.
</description>
<description xml:lang="es">UDDI businessEntity para BuenTiempo.
</description>
```

e-Macao-16-5-603

BusinessEntity Elements 3

4) contacts: (optional) list of contact information including:
   a) useType: attribute describing the type of contact in freeform text
   b) description: (optional) descriptions in more than one language of the reasons for using the contact
   c) personName: is the name of the person or the name of the job role
   d) phone: (optional)
   e) email: (optional)
   f) address: (optional repeating element) this structure represents the printable lines suitable for addressing an envelope

```
<contact useType="Technical Information">
   <description>Contact for technical information</description>
   <personName>Susan Carroll</personName>
   <phone useType="Main Office">853.782.923</phone>
   <email useType="CTO">susancarroll@niceweather.com</email>
   <address useType="Main Office" sortCode="10001">
      <addressLine>2001 Kuong Building</addressLine>
      <addressLine>Macao</addressLine>
   </address>
</contact>
```

e-Macao-16-5-604

BusinessEntity Elements 4

5) businessServices: (optional) list of one or more logical business service descriptions

6) identifierBag: (optional) list of "name-value" pairs that can be used to record identifiers for a businessEntity

7) categoryBag: (optional) list of "name-value" pairs that are used to tag a businessEntity with specific taxonomy information. For instance: industry, product or geographic codes

e-Macao-16-5-605

BusinessEntity Example

```
<businessEntity businessKey="5441234-763E-11D5-B565-000782FD9C23">
  <name xml:lang="en">NiceWeather</name>
  <description xml:lang="en">UDDI businessEntity for NiceWeather.
  </description>
  <contacts>
    <contact useType="Technical Information">
     <description>Contact for technical information</description>
     <personName>Susan Carroll</personName>
     <phone useType="Main Office">853.782.923</phone>
     <email useType="CTO">susancarroll@niceweather.com</email>
     <email useType="General Information">info@niceweather.com</email>
     <address useType="Main Office" sortCode="10001">
        <addressLine>2001 Kuong Building</addressLine>
        <addressLine>Macao</addressLine>
     </address>
    </contact>
    <contact> ... </contact>
  </contacts>
 <businessServices> . . . </businessServices>
<identifierBag> . . . </identifierBag>
<categoryBag> . . . </categoryBag>
</businessEntity>
```

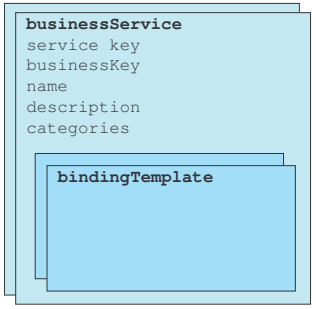
e-Macao-16-5-606

BusinessService

The `businessService` element is the root element for describing a logical business service.

The structure includes:

1) attributes:
   a) `serviceKey`
   b) `businessKey`
2) elements:
   a) `name`
   b) `description`
   c) `bindingTemplates`
   d) `categoryBag`



e-Macao-16-5-607

BusinessService Attributes

1) `serviceKey`: UUID for identifying a given `businessService`.

2) `businessKey`: is a direct reference to the `businessEntity` that is associated with it.

```
<businessService serviceKey="1T264170-2E3E-TE97-M9A8-F11111JE9WBH"
                 businessKey="5441234-763E-11D5-B565-000782FD9C2" >
```

reference to businessEntity

e-Macao-16-5-608

BusinessService Elements

1) `name`: (optional repeating element) are the human readable names recorded for the `businessService`, adorned with a unique `xml:lang` value

2) `description`: (optional) descriptions in more than one language of the logical service family

3) `bindingTemplates`: this structure holds the technical service description information related to a given business service family

4) `categoryBag`: (optional) list of "name-value" pairs that are used to tag a `businessService` with specific taxonomy information. For instance: industry, product or geographic codes.

e-Macao-16-5-609

BusinessService Example

```
<businessServices>
   <businessService serviceKey="1T264170-2E3E-TE97-M9A8-F11111JE9WBH"
                    businessKey="5441234-763E-11D5-B565-000782FD9C2" >
   <name>ask Data Submission</name>
   <description>NiceWeather ask data submission service.</description>

 <bindingTemplates>
   . . .
 </bindingTemplates>

 <categoryBag>
   . . .
 </categoryBag>

</businessServices>
```

e-Macao-16-5-610

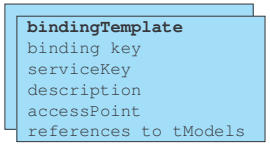
BindingTemplate

The `bindingTemplate` element contains the technical information necessary to invoke a specific web service.

The same logical service may have more than one type of binding (SOAP-HTTP, HTTP browser-based binding, etc.), each of them is described in a separate `bindingTemplate` element.

The structure includes:

1) attributes:
   a) `bindingKey`
   b) `serviceKey`
2) elements:
   a) `description`
   b) `accessPoint`
   c) `hostingRedirector`
   d) `tModelInstanceDetails`



e-Macao-16-5-611

BindingTemplate Attributes

1) `bindingKey`: UUID for identifying a given `bindingTemplate`

2) `serviceKey`: is a direct reference to the `businessSevice` that is associated with it.

```
<bindingTemplate bindingKey="2T5FDS12-04T4-GTT6-08GF-Y67E454357T89"
                 serviceKey= "1T264170-2E3E-TE97-M9A8-F11111JE9WBH">
```

reference to businessService

e-Macao-16-5-612

BindingTemplate Elements 1

1) `description`: (optional) descriptions in more than one language of the technical service entry point

2) `accessPoint`: is an attribute-qualified element that is used to convey the entry point address suitable for calling a particular web service. A single attribute named URLType is provided with the following values:

   a) `mailto`
   b) `http`
   c) `https`
   d) `ftp`
   e) `fax`
   f) `phone`
   g) `other`

3) `hostingRedirector`: used to designate that a `bindingTemplate` entry is a pointer to a different `bindingTemplate` entry. Is a required element if `accessPoint` is not provided. Is adorned with a `bindingKey` attribute, giving the redirected reference to a different bindingTemplate

e-Macao-16-5-613

BindingTemplate – Elements 2

3) `tModelInstanceDetails`: list of zero or more `tModelInstanceInfo` elements.

The `tModelInstanceInfo` is a distinct fingerprint used to identify services including one attribute (`tModelKey`) and two elements:

   a) `tModelKey`: is a unique key reference to a `tModel` describing the implementation details of the service
   b) `description`: (optional) descriptions in more than one language describing what role a `tModel` reference plays in the overall service description
   c) `instanceDetails`: (optional) is a structure providing additional information required to understand the details relative to the `tModelKey` reference, or to provide further parameters and settings support

e-Macao-16-5-614

BindingTemplate Example

```
<bindingTemplate bindingKey="2T5FDS12-04T4-GTT6-08GF-Y67E454357T89"
                 serviceKey= "1T264170-2E3E-TE97-M9A8-F11111JE9WBH">
  <description>SOAP based ask data submission service.</description>
  <accessPoint URLtype="http:">
        http://www.example.com/WeatherService/askData
  </accessPoint>

  <tModelsInstanceDetails>
    <tModelsInstanceInfo
        tModelKey="uuid:67DF6F55-SG56-976F-9U77-570425RFD68J">
        <description>HTTP address</description>
    </tModelsInstanceInfo>
  </tModelsInstanceDetails>

</bindingTemplate>
```

e-Macao-16-5-615

BindingTemplate - instanceDetails

The `instanceDetails` element is added to the `tModelInstanceInfo` to specify additional service implementation details. It has the following structure :

a) `description`: (optional) descriptions in more than one language describing the purpose and/or the use of the particular instanceDetails entry

b) `overviewDoc`: (optional) is a structure referencing to remote descriptive information or instructions related to proper use of the `bindingTemplate` technical sub-element. The structure contains:
- `description`
- `overviewURL`

c) `instanceParms`: (optional) used to contain setting parameters or a URL reference to a file containing setting or parameters required to use a specific facet of a `bindingTemplate` description

e-Macao-16-5-616

instanceDetails Example

```
<tModelsInstanceDetails>
   <tModelsInstanceInfo
     tModelKey="uuid: 9OC7WMI1-0998-0375-NE00-0702IBA09049">
     <description>Reference to tModel with Web Service interface
                  definition
     </description>
     <instanceDetails>
       <overviewDoc>
         <description>
            Reference to additional semantic specification of the
            web service.
         </description>
         <overviewURL>
           http://www.example.com/WeatherService/additionalFeatures.xml
         </overviewURL>
       </overviewDoc>
     </instanceDetails>
</tModelInstanceInfo>
</tModelsInstanceDetails>
. . .
```

e-Macao-16-5-617

tModel Structure

The primary role that a tModel plays is to represent a technical specification.

The structure includes:

1) attributes:
   a) `tModelKey`
   b) `operator`
   c) `authorizedName`

2) elements:
   a) `name`
   b) `description`
   c) `overviewDoc`
   d) `identifierBag`
   e) `categoryBag`

e-Macao-16-5-618

tModel Attributes

1) `tModelKey`: is a unique key for a given `tModel` structure

2) `operator`: is the certified name of the UDDI registry site operator that manages the master copy of the `tModel` data

3) `authorizedName`: is the recorded name of the individual that published the `tModel` data

e-Macao-16-5-619

tModel Elements

1) `name`: the name recorded for the `tModel`

2) `description`: (optional repeating element) one description is allowed per national language code supplied

3) `overviewDoc`: is a structure referencing to remote descriptive information or instructions related to proper use of the `tModel`. The structure contains:
   - `description`
   - `overviewURL`

4) `identifierBag`: (optional) is an optional list of "name-value" pairs that can be used to record identification numbers for a `tModel`

5) `categoryBag`: (optional) is a list of "name-value" pairs that are used to tag a `tModel` with specific taxonomy information

e-Macao-16-5-620

tModel Example 1

```
<tModel tModelKey="uuid:67DF6F55-SG56-976F-9U77-570425RFD68J">
    <name>Ask Weather Data Service</name>
    <description xml:lang="en"> Service interface definition for ask
       weather data submission service.</description>
    <overviewDoc>
       <description xml:lang="en">Reference to the WSDL document that
         contains the service interface definition for the ask data
         submission service.</description>
       <overviewURL>
          http://www.example.com/WeatherService/askData.wsdl
       </overviewURL>
    </overviewDoc>
```

reference to the web service description

e-Macao-16-5-621

tModel Example 2

reference to DUNS tModel

reference to UDDI type taxonomy tModel

```
    <identifierBag>
      <keyedReference keyName="DUNS"
         keyValue="00-111-1111"
         tModelKey="uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823" />
    </identifierBag>

    <categoryBag>
      <keyedReference keyName="uddi-org:types"
          keyValue="soapSpec"
          tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4" />
      <keyedReference keyName="uddi-org:types"
          keyValue="wsdlSpec"
          tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB" />
      <keyedReference keyName="Weather"
          keyValue="84902934"
          tModelKey="uuid:J6LIJ84T-3874-8301-Y9JY-2JS76GFD60J8" />
     </categoryBag>

</tModel>
```

e-Macao-16-5-622

publisherAssertion 1

The `publisherAssertion` is a declaration done by two `businessEntity`s in order to establish a relationship between them.

The relation becomes visible when both `businessEntity`s published the same information.

The structure includes:

a) `fromKey`: UUID referencing the first `businessEntity`
b) `toKey`: UUID referencing the second `businessEntity`
c) `keyedReference`: designates the relationship between the business entities by three attributes:
   - `tModelKey`: reference the relationship type system
   - `keyName`
   - `keyValue`

   } are used to indicate the specific type of relationship

e-Macao-16-5-623

publisherAssertion 2

According to the relationship type system defined in the UDDI specification:

Values for the `keyValue` attribute are:

1) `parent-child`: the business referenced by the `fromKey` is the parent of the business referenced by the `toKey`
2) `peer-peer`: both businesses referenced are partners or affiliates
3) `identity`: both businesses referenced are the same. Is typically used to assert different divisions, units and departments of the same organization

Providing, publisher-assertion capabilities, UDDI allows large corporations to describe aspects of their businesses - such as: divisions, partners, and subsidiaries - to users of the UBR.

e-Macao-16-5-624

publisherAssertion Example

```
<publisherAssertion>
  <fromKey>5441234-763E-11D5-B565-000782FD9C23</fromKey>
  <toKey>U7O0J86-JU77-MN88-G86G-096YY8EV9JK1</toKey>
  <keyedReference
     keyName="subsidiary"
     keyValue="parent-child"
     tModelKey="uuid:807A2C6A-EE22-470D-ADC7-E0424A337C03" />
</publisherAssertion>
```

reference to UDDI relationships tModel

This declaration models the relationship between the company NiceWeather (`fromKey`) and its subsidiary in Hong Kong (`toKey`).

For this example, the `keyValue` attribute defines that NiceWeather is the parent-company of NiceWeather Hong Kong.

e-Macao-16-5-625

identifierBag

UDDI defines the notion of attaching identifiers to data using the `identifierBag`

Two of the core data types support attaching identifiers to data:

1) `businessEntity`
2) `tModel`

An `identifierBag` is an element that holds zero or more instances of something called `keyedReference`.

e-Macao-16-5-626

identifierBag Example

```
<identifierBag>
   <keyedReference keyName="DUNS"
      keyValue="00-111-1111"
      tModelKey="uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823" />
 </identifierBag>
```

reference to DUNS tModel

Is a general-purpose structure for a "name-value" pair with one additional attribute referencing a `tModel` structure

The reference to a `tModel` structure makes the identifier scheme extensible, allowing `tModel`s to be used as a conceptual namespace qualifiers.

e-Macao-16-5-627

categoryBag

The `categoryBag` is the container that describes relevant classification information that businesses provide when publishing their services.

```
<categoryBag>
      <keyedReference keyName="uddi-org:types"
         keyValue="soapSpec"
         tModelKey="uuid:T6TGU76T-9876-3234-4J90-34J997T78TG5" />
</categoryBag>
```

Three of the core data types support attaching identifiers to data:

1) `businessEntity`
2) `businessService`
3) `tModel`

e-Macao-16-5-628

categoryBag Example

reference to UNSPSC tModel

```
<categoryBag>
  <keyedReference keyName="Weather Station"
      keyValue="41114410"
      tModelKey="CD153257-086A-4237-B336-6BDCBDCC6634"/>
  <keyedReference keyName="Macao"
      keyValue="MO"
      tModelKey="4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88"/>
</categoryBag>
```

reference to ISO 3166 tModel

This `categoryBag` corresponds to the `businessEntity` of NiceWeather.

We are defining that the business belongs to the "41114410" category that corresponds to "weather stations" according to UNSPSC codes, and we are locating the company in Macao according to ISO-3166.

A.6.4. UDDI Registry

e-Macao-16-5-629

UDDI Outline

1) Introduction
2) Concepts
3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag
4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service
5) Summary

e-Macao-16-5-630

Using a UDDI Registry

A UDDI is itself an instance of a web service.

Entries in the registry can be published and queried using SOAP-based service interface.

The WSDL service interface definitions for a UDDI registry can be found at:

UDDI Inquiry API v2.0	http://uddi.org/wsdl/inquire_v2.wsdl
UDDI Publication API v2.0	http://uddi.org/wsdl/publish_v2.wsdl
UDDI portTypes API v3.0	http://uddi.org/wsdl/uddi_api_v3_portType.wsdl
UDDI Bindings API v 3.0	http://uddi.org/wsdl/uddi_api_v3_binding.wsdl

e-Macao-16-5-631

UDDI Business Registry 1

Four organization hosts nodes in the UDDI Business Registry.

Most of them also host a test registry.

Three URLs are provided for each:

Host Organization	Access Type	URL
IBM Business Registry	Web	http://uddi.ibm.com
	Inquiry	http://uddi.ibm.com/ubr/inquiryapi
	Publish	http://uddi.ibm.com/ubr/publishapi
IBM Test Registry	Web	http://uddi.ibm.com/testregistry/registry.html
	Inquiry	http://www-3.ibm.com/services/uddi/testregistry/inquiryapi
	Publish	https://uddi.ibm.com/testregistry/publishapi

e-Macao-16-5-632

UDDI Business Registry 2

Host Organization	Access Type	URL
Microsoft Business Registry	Web	http://uddi.microsoft.com
	Inquiry	http://uddi.microsoft.com/inquire
	Publish	https://uddi.microsoft.com/publish
Microsoft Test Registry	Web	http://test.uddi.microsoft.com
	Inquiry	http://test.uddi.microsoft.com/inquire
	Publish	https://test.uddi.microsoft.com/publish
SAP Business Registry	Web	http://uddi.sap.com
	Inquiry	http://uddi.sap.com/uddi/api/inquiry
	Publish	https://uddi.sap.com/uddi/api/publish
SAP Test Registry	Web	http://udditest.sap.com
	Inquiry	http://udditest.sap.com/UDDI/api/inquiry
	Publish	https://udditest.sap.com/UDDI/api/publish

e-Macao-16-5-633

UDDI Business Registry 3

Host Organization	Access Type	URL
NTT Business Registry	Web	http://www.ntt.com/uddi
	Inquiry	http://www.uddi.ne.jp/ubr/inquiryapi
	Publish	https://www.uddi.ne.jp/ubr/publishapi

e-Macao-16-5-634

Task 78: UDDI Registry

1) access the UDDI v2 IBM Business Registry: http://uddi.ibm.com
2) select option “Search UDDI Business Registry”
3) search for “Business” – starting with: “Public Administration”
4) select Administration Division
5) access the file in the DiscoveryURL
6) save this file in your PC

e-Macao-16-5-635

Task 79: Test UDDI Registry

7) open the file:

   a) where is the master copy of the businessEntity data?

   b) who registered the information for this businessEntity?

   c) access the business contact on the web page containing the businessEntity information

   d) what is the information described for the contact? check the address details with the information on the XML file

   e) how many services related does the business have?

   f) what is the information provided for this service and how can you access this service?

e-Macao-16-5-636

Task 80: Test UDDI Registry

8) go back to the page where the results of step 3 where shown

9) select “services” for the Administration Division

10) what is the information shown in the page?

11) what links are provided?

12) select “home page”

13) what information is provided?

e-Macao-16-5-637

Publishing Service Descriptions 1

Most UDDI operators require the user to register before publishing any UDDI entries.

The registration process provides a publisher account consisting on a user-id and a password.

Entries in the registry are owned by the publisher who created them, and only the owner can update or delete a registry entry.

The UDDI publication APIs provide support for creating, updating, and deleting the following entries:

1) `businessEntity`
2) `businessService`
3) `bindingTemplate`
4) `tModel`
5) `publisherAssertions`

e-Macao-16-5-638

Authentication Token

Before using a publication API, an authentication token must be obtained, using the `get_authToken` API call.

Authentication tokens are required for all publication APIs. They represent an active session with the registry.

Authentication tokens are valid for a period of time defined by the registry.

The `discard_authToken` message is used to indicate the registry that the token can be discarded.

e-Macao-16-5-639

APIs for Publishing

APIs for publishing the four primary data types:

Datatype	Save API	Delete API
bindingTemplate	save_binding	delete_binding
businessEntity	save_business	delete_business
serviceBusiness	save_service	delete_service
tModel	save_tModel	delete_tModel

The `get_registeredInfo` API call is used to obtain a complete list of `businessEntity` and `tModel` entries owned by the publisher.

e-Macao-16-5-640

Publishing Publisher-Assertions

Five APIs are used to process publisher assertions:

1) `add_publisherAssertions`
2) `delete_publisherAssertions`
3) `get_publisherAssertions`: gets the full list of publisher assertions associated with a publisher's assertion collection
4) `get_assertionsStatusReport`: determines the status of current assertions
5) `set_publisherAssertions`: adds new assertions or updates existing assertions

e-Macao-16-5-641

WSDL and UDDI

UDDI provides a method for publishing and finding businesses and services information.

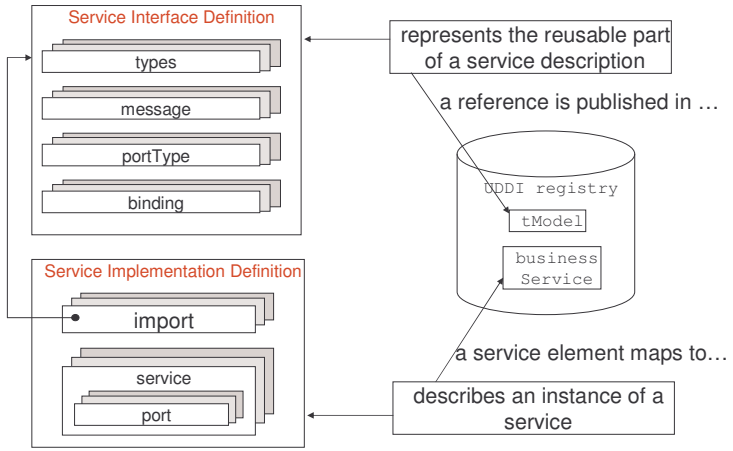
UDDI provides support for many different types of service descriptions:

1) WSDL
2) plain ASCII text
3) RDF
4) others

The service description information defined in WSDL is complementary to the information found in a UDDI registry.

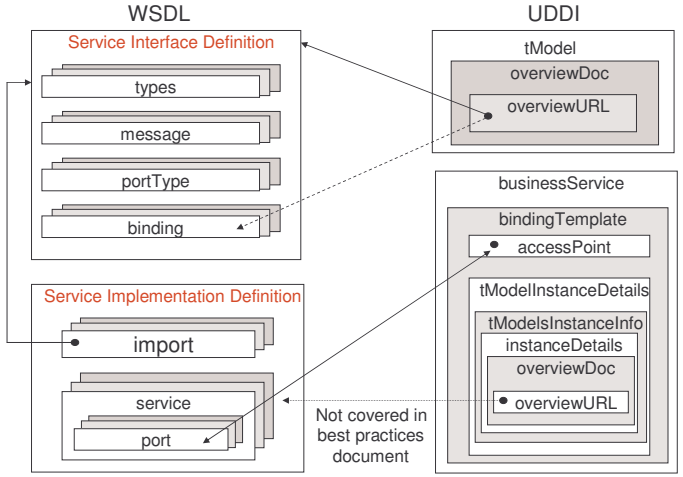
e-Macao-16-5-642

Mapping from WSDL to UDDI 1



e-Macao-16-5-643

Mapping from WSDL to UDDI 2



e-Macao-16-5-644

Mapping Co-Relations

1) Service Interface Definition - UDDI-`tModel`:
   a) the `overviewDoc` field in each new `tModel` will point to the corresponding WSDL document
2) Service Implementation Definition – UDDI-`businessService`:
   a) a `bindingTemplate` is created for each access endpoint. The network address of the access point is the `accessPoint` element
   b) one `tModelInstanceInfo` is created in the `bindingTemplate` for each wsdlSpec `tModel` that defines interfaces and bindings supported by the service

e-Macao-16-5-645

Procedure for Publishing Services

1) the WSDL service interface definition is created
2) the WSDL service interface definition is registered as UDDI `tModels`. Such models are called `wsdlSpec tModels`
3) programmers will build services conforming to the service definitions
4) the new service must be deployed and registered in the UDDI registry. A UDDI `businessService` data structure is created and registered

e-Macao-16-5-646

Publishing a Service Example 1

1) the WSDL service interface definition is created
2) the WSDL service interface definition is registered as UDDI `tModel`.

```
<tModel authorizedName="…" operator="…"
        tModelKey="9OC7WMI1-0998-0375-NE00-0702IBA09049">
  <name>Ask Weather Data Service</name>
  <description xml:lang="en">
       WSDL description of a standard ask data about weather service.
  </description>
  <overviewDoc>
     <description xml:lang="en">WSDL source document.</description>
     <overviewURL> "http://www.example.com/WeatherService/askData.wsdl"
     </overviewURL>
  <overviewDoc>
  <categoryBag>
     <keyedReference
        tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
        keyName="uddi-org:types"
        keyValue="wsdlSpec" />
  </categoryBag>
</tModel>
```

reference WSDL binding element

tModel is categorized as a WSDL specification

e-Macao-16-5-647

Publishing a Service Example 2

3) after the service is deployed, is registered as a `businessService`:

```
<businessService serviceKey="1T264170-2E3E-TE97-M9A8-F11111JE9WBH"
                 businessKey="5441234-763E-11D5-B565-000782FD9C23">
  <name>Ask Data Service</name>
  <description>Ask data submission service.</description>

 <bindingTemplates>
  <bindingTemplate bindingKey="2T5FDS12-04T4-GTT6-08GF-Y67E454357T89"
                   serviceKey= "1T264170-2E3E-TE97-M9A8-F11111JE9WBH">
    <description>SOAP based ask data submission service.</description>
    <accessPoint URLType="http:">
        http://www.example.com/WeatherService/askData
    </accessPoint>

  <tModelsInstanceDetails>
    <tModelsInstanceInfo
       tModelKey="uuid: 9OC7WMI1-0998-0375-NE00-0702IBA09049">
       <description>Reference to tModel with Web Service interface
            definition
       </description>
```

URL where the WS can be invoked

reference to tModel (wsdlSpec)

e-Macao-16-5-648

Publishing a Service Example 3

```
  <instanceDetails>
      <overviewDoc>
         <description>
             Reference to WSDL service implementation document.
         </description>
         <overviewURL>
        http://www.example.com/WeatherService/askDataImplementation.wsdl
     </overviewURL>
       </overviewDoc>
  </instanceDetails>
</tModelInstanceInfo>
</tModelsInstanceDetails>
</bindingTemplate>
</bindingTemplates>
. . .
</busienessService>
```

reference to service implementation definition (not defined in best practices)

e-Macao-16-5-649

Service with Multiple Bindings 1

The WSDL service interface definition contains multiple bindings:

```
<portType name="CancelUserPortType">
. . .
</portType>

<portType name="AskDataPortType">
. . .
</portType>

<binding name="CancelUserSoapBinding">
. . .
</binding>

<binding name="AskDataSoapBinding">
. . .
</binding>
```

e-Macao-16-5-650

Service with Multiple Bindings 2

Create a `tModel` for each binding definition using an `XPointer` in the `overviewURL` element

```
. . .
  <overviewDoc>
     <description>
            Reference to WSDL service implementation document.
     </description>
     <overviewURL>
        http://www.example.com/WeatherService/MultipleBindings.wsdl
        xmlns(wsdl=http://schemas.xmlsoap.org/wsdl/)
        xpointer (//wsdl:binding[@name="AskDataSoapBinding"])
     </overviewURL>
  </overviewDoc>
. . .
```

e-Macao-16-5-651

Finding Service Description 1

When finding service descriptions, two types of inquiry APIs are available:

1) find APIs:
   a) `find_binding`: returns the contents of a `bindingTemplate`
   b) all others (business, service, tModel): retrieve a list of references (UDDI keys) to UDDI data entries matching the specified search criteria

2) get APIs: return the actual contents of a data entry

e-Macao-16-5-652

Finding Service Description 2

APIs for inquiring the four primary data types:

Datatype	Find API	Get API
bindingTemplate	find_binding	get_bindingDetail
businessEntity	find_business	get_businessDetail
serviceBusiness	find_service	get_serviceDetail
tModel	find_tModel	get_tModelDetail

e-Macao-16-5-653

Task 81: Find Business

Objective: find businesses in the IBM test UUDI registry. The name must match “Business”, case sensitive match, 100 instances.

```
1) cd demos\UDDI\FindBusiness

2) dir
FindBusinessExample.class
FindBusinessExample.java

1) notepad FindBusinessExample.java

2) java -cp demos\UDDI\FindBusiness FindBusinessExample
```

e-Macao-16-5-654

Task 82: Find Service

Objective: find a service in the IBM test UUDI registry. The tModel key is: “AFFC30D0-D83E-11D5-8055-0004AC49CC1E”.

```
1) cd demos\UDDI\FindService

2) dir
FindServiceExample.class
FindServiceExample.java

1) notepad FindServiceExample.java

2) java -classpath demos\UDDI\FindService
       FindServiceExample
```

A.6.5. Summary

e-Macao-16-5-655

UDDI Outline

1) Introduction
2) Concepts
3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag
4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service
5) Summary

e-Macao-16-5-656

UDDI Summary 1

UDDI is a platform independent, open framework for describing, discovering and integrating business services

Two types of service discovery:

1) static
2) dynamic

e-Macao-16-5-657

UDDI Summary 2

How it works:

a) service providers publish information about businesses and services

b) UDDI assigns unique identifiers to the information provided

c) service requestors query the registry

d) data returned is used to invoke web services

e-Macao-16-5-658

UDDI Summary 3

Four services provided:

a) white pages to look up a web service by the business

b) yellow pages to look up a web service by topic

c) green pages to look up a service through web services features

d) publish information

e-Macao-16-5-659

UDDI Summary 4

Global registry hosted by UDDI operators: IBM, Microsoft, SAP, NTT

UDDI principle: “register once – publish everywhere”

UDDI provides two core systems for identifying businesses and services:

1) D-U-N-S
2) Thomas Register

UDDI provides three classification schemes:

1) NAICS
2) UNSPSC
3) ISO-3199

e-Macao-16-5-660

UDDI Summary 5

UDDI is modeled using five data types:

1) `businessEntity`
2) `businessService`
3) `bindingTemplate`
4) `tModel`
5) `publisherAssertion`

e-Macao-16-5-661

UDDI Summary 6

1) businessEntity: represents all the information about a business

2) businessService: describes a logical business service

3) bindingTemplates: contains the technical information needed to invoke a web service

4) tModel: represents a technical model specification

5) publisherAssertion: establishes a relation between two businessEntitys

e-Macao-16-5-662

UDDI Summary 7

1) identifierBag and categoryBag can be defined for businessEntity and tModels.

2) categoryBag can also be defined for businessService

3) UDDI provides the UDDI type taxonomy for assisting in general categorization of the tModels themselves

e-Macao-16-5-663

UDDI Summary 8

1) APIs are provided for publishing the four core data types:
   a) save_business
   b) save_service
   c) save_binding
   d) save-tModel

2) four APIs are also provided to delete the information related to these core data types.

3) five APIs are used to process publisher assertions:
   add / delete / get_publisherAssertions, get_assertionsStatusReport and set_publisherAssertions

4) for retreving information, two types of APIS are available:
   1) find_business - find_(service/binding/tModel)
   2) get_businessDetail - get_(service/binding/tModel)Detail

e-Macao-16-5-664

UDDI Summary 9

The WSDL service interface definition of a web service is created and published as a tModel.

The overviewURL of the tModel points to the WSDL document.

After the service implementation is built and deployed, it is published as a businessService.

A bindingTemplate is created for each access endpoint.

The accessPoint contains the network address of the service Implementation.

If the WSDL service interface definition contains multiple bindings, a tModel is created for each binding definition using an XPointer in the overviewURL element.

A.7. Security

Security

e-Macao-16-5-666

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

A.7.1. Security Basics

e-Macao-16-5-667

Security Outline

1) Security Basics
2) Web Service Security
3) Digital Signatures

e-Macao-16-5-668

Security Context

e-Business as well as e-Government relies on the exchange of information between partners over insecure networks.

Sending messages over insecure networks implies risks.

Messages could be:

a) stolen
b) lost
c) modified

e-Macao-16-5-669

Security Requirements

Four security requirements must be addressed to ensure the safety of information exchanged among partners:

a) confidentiality
b) integrity
c) authentication
d) non-repudiation

} protect messages

One security requirement to assure resources:

1) authorization } protects resources

e-Macao-16-5-670

Confidentiality

Guarantees that exchanged information is protected against eavesdroppers.

For example:

a) license's information should not be exposed to outsiders

b) credit card information should not be wiretapped by third parties

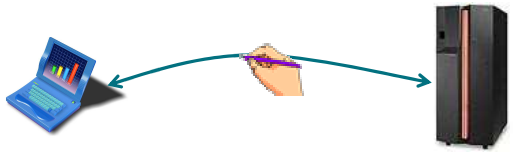
e-Macao-16-5-671

Integrity

Assures that a message is not accidentally or deliberately modified in transit.

For example:

a) social security benefits's information should not be modified as it moves between citizens and government



e-Macao-16-5-672

Authentication

Guarantees that access to e-applications and data is restricted to those who can provide appropriate proof of identity.

For example:

a) in order to track the status of a license application, subscribers are required to provide an ID and password as proof of their identity

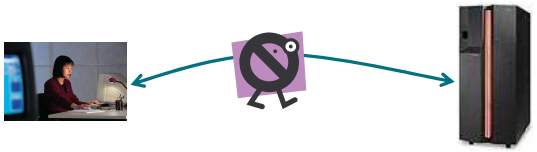
e-Macao-16-5-673

Non-Repudiation

Guarantees that the message's sender cannot deny having send it.

For example:

a) with non-repudiation, once an application license is submitted, the business cannot repudiate it



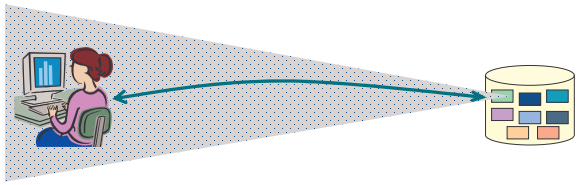
e-Macao-16-5-674

Authorization

Decides whether an entity with a given identity can access a particular resource

For example:

a) a particular citizen can only view the information related to him

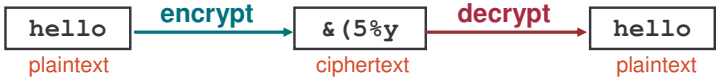


e-Macao-16-5-675

Cryptography

Cryptography technologies provide a basis for protecting messages exchanged between partners.

A process based on algorithms transforming a clear text message – plaintext, in encrypted data - ciphertext.



e-Macao-16-5-676

Cryptography Algorithms

Most of the algorithms use a key to encrypt and decrypt.

According to the keys used, encrypting algorithms can be classified in:

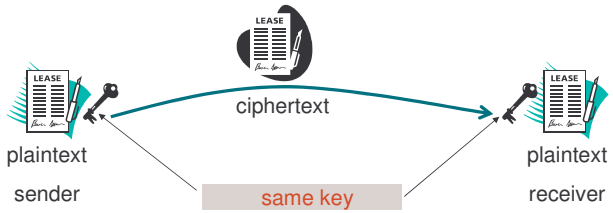
1) symmetric
2) asymmetric

Different technologies	Symmetric key	Asymmetric key
Encryption	3DES – AES – RC4	RSA15
Digital signature	HMAC-SHA1 HMAC-MD5	RSA-SHA1

e-Macao-16-5-677

Symmetric Encryption

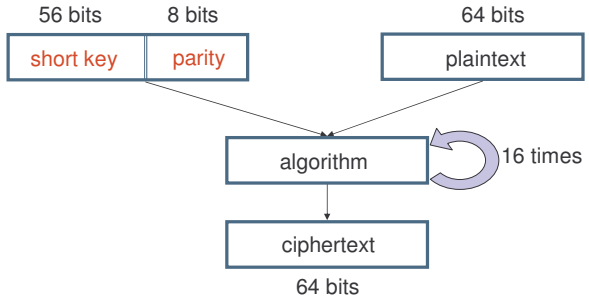
Requires the use of the same key for encryption and decryption.



e-Macao-16-5-678

DES Algorithm

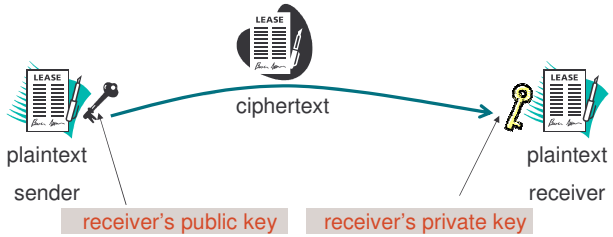
DES – Data Encryption Standard: developed by IBM and approved by the National Bureau of Standards (NBS).



e-Macao-16-5-679

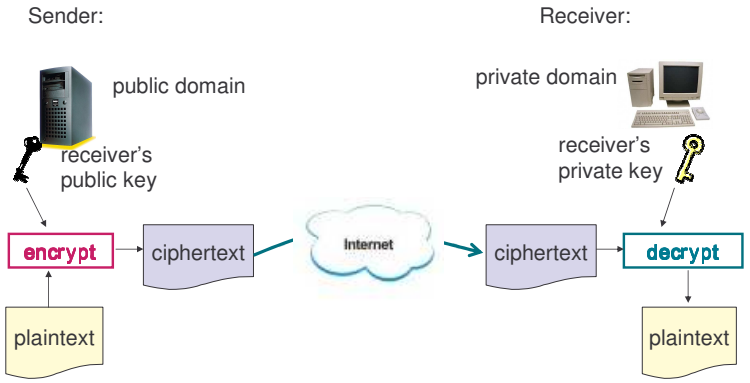
Asymmetric Encryption

Uses two keys: public and private key



e-Macao-16-5-680

Asymmetric Encryption Process



e-Macao-16-5-681

Symmetric Digital Signature

Symmetric digital signature technology is called Message Authentication Code (MAC) technology.

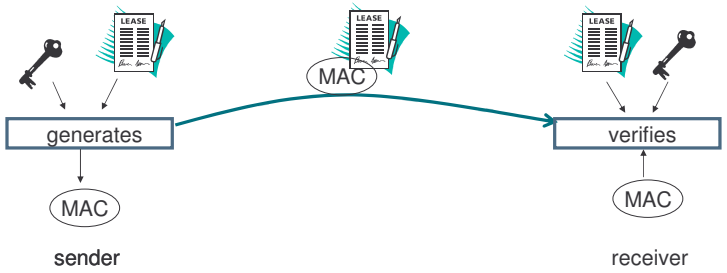
MAC relies on mathematical algorithms known as hashing functions to ensure data integrity.

A hashing function takes data as input and produces smaller data called digest.

In MAC, the digest is created with a key in addition to the input data.

e-Macao-16-5-682

Symmetric Digital Signature



Keyed-Hashing for Message Authentication Code (HMAC) is an example of MAC.

HMAC is combined with hashing functions such as MD5 and SHA-1. Therefore, the algorithm names: HMAC-SHA1 and HMAC-MD5.

e-Macao-16-5-683

Digital Signature

The asymmetric digital signature technology is called digital signature.

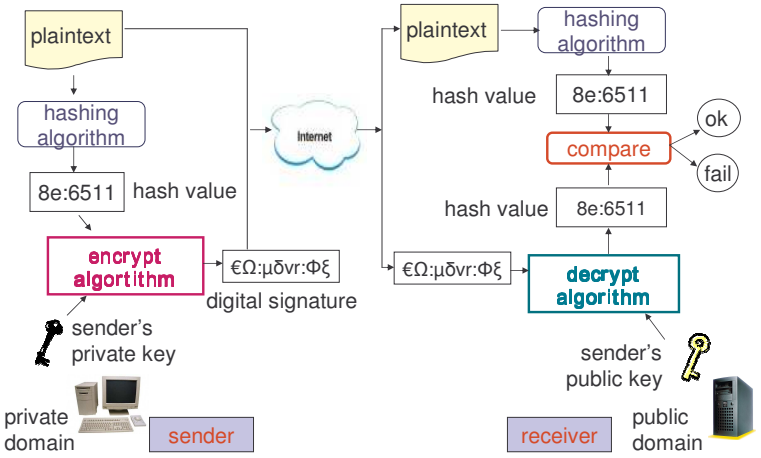
The sender signs the plaintext with his private key.

Signing means creating a signature value that is sent with the original plaintext.

Like MAC algorithms, digital signature algorithms are also combined with hashing functions such as SHA-1 (SHA: Secure Hash Algorithm).

e-Macao-16-5-684

Digital Signature Process



e-Macao-16-5-685

Use of Digital Signature

The digital signature technology ensures:

a) non-repudiation: the receiver can assure the message is signed by the sender because he is using the sender's public key to decrypt the message

b) integrity: the receiver can assure that the message was not changed during the transmission

e-Macao-16-5-686

Public Key Infrastructure

How can the receiver know that "Person A" is the holder of the public key?

Public Key Infrastructure (PKI) provides a solution.

In PKI an "authority" issues digital certificates.

Digital certificates are used to bind a party to a public key.

e-Macao-16-5-687

Different Solutions

Different solutions may be applied to assure security:

1) password authentication
2) HTTP Basic Authentication
3) digital signature authentication
4) secure protocols – SSL

e-Macao-16-5-688

Password Authentication

Password authentication is the most commonly used authentication method on the Internet:

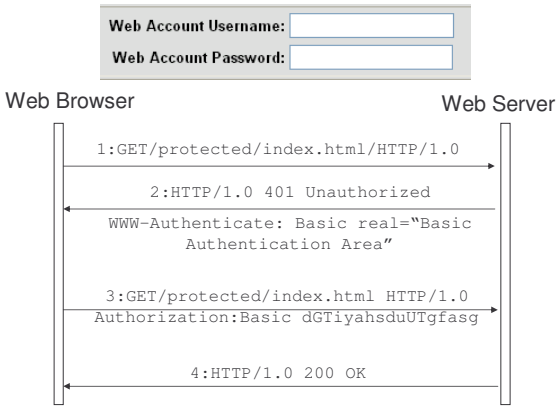
1) a client shows its ID or username and password
2) the server checks the ID and password in a user registry

Password authentication to access Web servers over HTTP is called HTTP Basic Authentication (BASIC-AUTH) and its defined in RFC 2617

e-Macao-16-5-689

HTTP Basic Authentication

Is an interaction protocol between a Web browser and a Web server.



e-Macao-16-5-690

Digital Signature Authentication

Digital signatures can also be used for authentication.

Is more convenient than password authentication – since a certificate authority manages certificates.

Use of certificates:

1) client certificates are not widely used - not easy for users to install certificates.

2) server certificates are commonly used - when web browsers need to authenticate servers.

e-Macao-16-5-691

Security Protocols - SSL

Security protocols allows to share symmetric keys between partners in a secure manner.

SSL – Secure Socket Layer defined by Netscape for Web browsers is the most widely used protocol on the Internet:

1) enables to share symmetric keys

2) performs authentication

e-Macao-16-5-692

SSL Protocol

Client:

1) accesses the server

3) prepares a random number (a seed for generating a symmetric key)

4) encrypts the seed number with a public key contained in the server certificate

5) sends the encrypted data to the server

7) generates a symmetric key based on the seed number

Server:

2) returns its certificate

6) decrypts the received data to extract the seed number

7) generates a symmetric key based on the seed number

e-Macao-16-5-693

SSL Authentication

SSL authentication is based on encryption.

After the negotiation has been completed, the client and the server can authenticate themselves:

1) the client authenticating the server:
   a) client sends application data encrypted with the symmetric key
   b) server sends response encrypted with the symmetric key
   c) client can authenticate the server

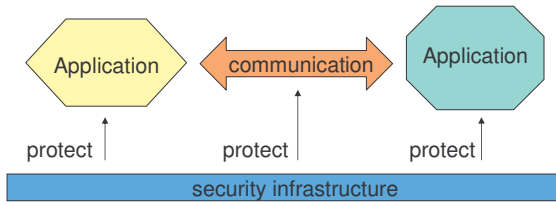
2) the server authenticating the client – two ways:
   1) HTTP Basic Authentication
   2) the client is required to decrypt a random number encrypted with its public key

e-Macao-16-5-694

Security Infrastructure

To solve the difficulties of combining security technologies properly, security infrastructures have been developed and are use in real systems.

A security infrastructure is a basis on which applications can interact with each other securely.



e-Macao-16-5-695

Different Security Infrastructures

Each security infrastructure has different design requirements and vary in terms of their design and architecture.

Three security infrastructures will be presented:

a) user registries

b) Public Key Infrastructure

c) Kerberos

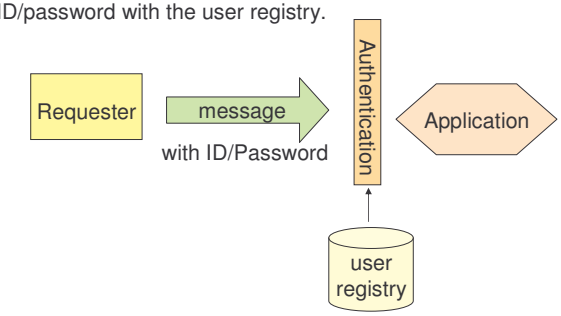
e-Macao-16-5-696

User Registries

The most basic security infrastructure used for authentication.

User registries manages users identification and password.

An authentication module is placed “in front” of applications checking each ID/password with the user registry.



e-Macao-16-5-697

User Registries Usage

The advantage is its simplicity.

The disadvantage is that they may be cracked. Once a password is stolen, the attacker can easily access a system.

Operating systems, Data Base Management Systems (DBMS) and HTTP servers incorporate user registries.

The cost for development and management of user registries is cheaper than other mechanisms.

e-Macao-16-5-698

Public Key Infrastructure

Public Key Infrastructure provides a basis to certify holders of public keys.

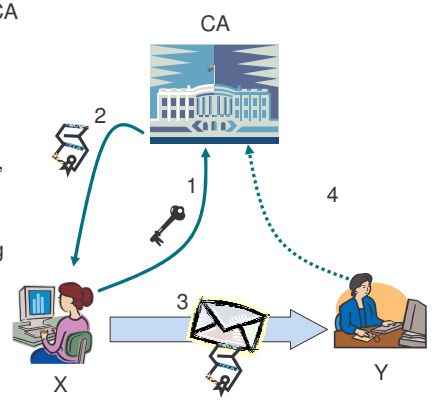
The key constructs of PKI are:

1) certificate: a proof of identity
2) certificate authority: an entity that issues certificates.

e-Macao-16-5-699

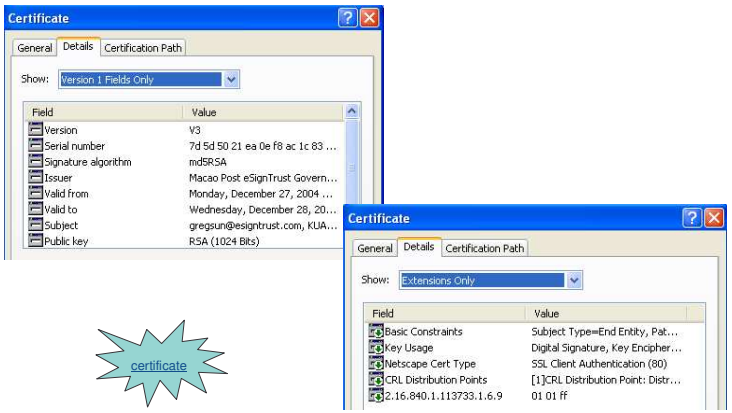
Use of Certificates

1) X registers its public key in CA
2) CA issues a certificate to X
3) X signs a message with the private key and sends it to B, attaching the certificate
4) Y verifies the signature using a public key included in the certificate
5) Y verifies if the certificate is signed by CA



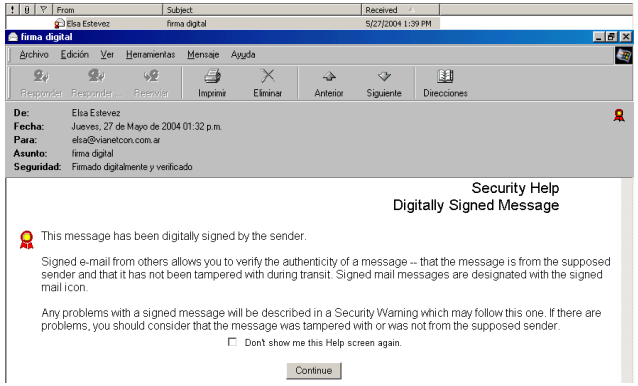
e-Macao-16-5-700

Certificate Example



e-Macao-16-5-701

Valid Signed Message Example

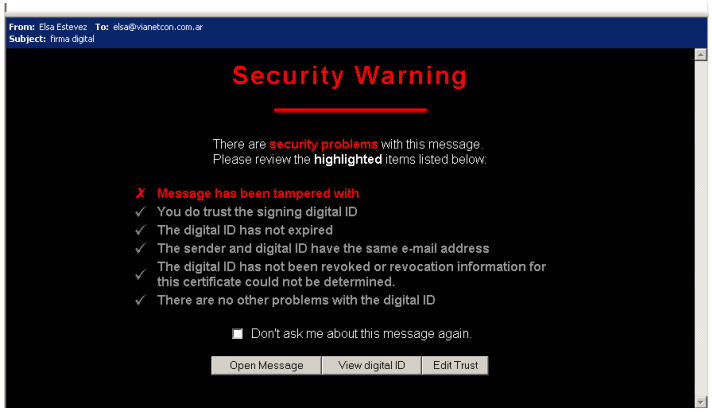


e-Macao-16-5-702

Invalid Signed Message Example 1

e-Macao-16-5-703

Invalid Signed Message Example 2



e-Macao-16-5-704

Kerberos

Kerberos was initially developed for workstation users who wanted to access a network.

Key requirements:

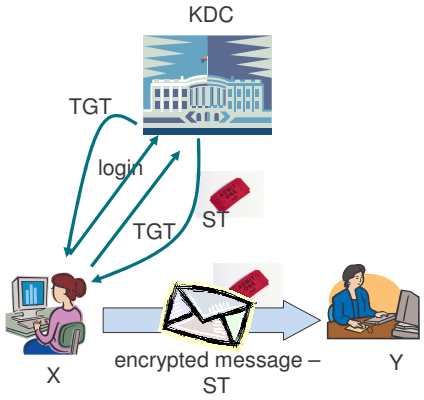
1) Single Sign ON (SSO): a user provides an ID and a password only once to access various applications within a certain interval

2) no use of public key cryptography

e-Macao-16-5-705

Use of Kerberos

1) X logins in KDC (Key Distribution Center) using password authentication and requests a Ticket-Granting Ticket
2) X receives TGT from KDC
3) X requests and receives a service ticket (ST) for Y, showing the TGT to the KDC
4) X can now access Y by including the ST in a request message



e-Macao-16-5-706

Ticket-Granting Ticket

The TGT contains:

1) user's ID
2) session key
3) TGT expiration time

As long as the TGT is valid, X can get various STs without giving its ID and password. Therefore, single sign on is achieved.

When issuing the ST, KDC encrypts X's information with Y's information. Thus, only Y can decrypt X's ID in the ST.

ST contains a session key between X and Y, so they can securely exchange messages with encryption and digital signatures.

e-Macao-16-5-707

Security Domains

Each security infrastructure has a scope.

The scope determines the participants and resources managed by the security infrastructure.

User registries and Kerberos have explicit databases defining their scope.

CA implicitly prescribes a set of participants in PKI.

The scope of the security infrastructure is called security domain.

e-Macao-16-5-708

Multiple Security Domains

Multiple security domains exist in the real world.

Is out of question considering a single security infrastructure to integrate them.

Web services security addresses how to integrate security domains based on different security infrastructure.

A.7.2. Web Services Security

e-Macao-16-5-709

Security Outline

1) Security Basics
2) Web Service Security
3) Digital Signatures

e-Macao-16-5-710

Web Services Security

Each business has its own security infrastructure.

Web services need to interoperate over different security domains.

The security model for web services defines:

1) concepts
2) architecture

e-Macao-16-5-711

WS Security Model Concepts

Some concepts used in the WS security model include:

1) security token
2) subject
3) claim
4) web service endpoint policy
5) security token service

e-Macao-16-5-712

Security Token

A security token is a piece of information related to security.

For instance a security token can be:

1) a X.509 certificate,
2) Kerberos ticket,
3) username,
4) mobile device security token from a SIM card,
5) etc.

e-Macao-16-5-713

Subject

A subject is an entity about which the claims expressed in the security token apply.

For instance:

1) a person
2) an application
3) business

e-Macao-16-5-714

Claim

A claim is a statement about a subject.

A claim can be done by:

1) the subject itself
2) a third party that associates a subject with a claim

For instance, claims:

1) may be about keys that may be used to sign or encrypt messages
2) may be statements of the security token itself
3) may be used to assert the user's identity or an authorized role

e-Macao-16-5-715

Web Service Endpoint Policy

A Web service endpoint policy are the claims and related information that web services require in order to process messages.

Endpoint policies may be expressed in XML.

Endpoint policies can be used to indicate requirements related to:

1) authentication – proof of user
2) authorization – proof of execution capabilities
3) other requirements

e-Macao-16-5-716

Security Token Service (STS)

A security token service is a third party issuing security tokens.

For instance:

1) the certificate authority in PKI
2) Key Distribution Center in Kerberos

A security token service is a web service.

e-Macao-16-5-717

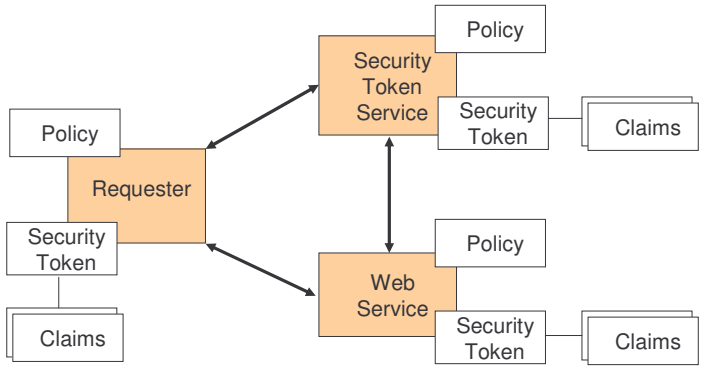
WS Security Architecture

The web services security architecture defines an abstract model for managing security based on three parties:

a) requestor
b) web service
c) security token service

e-Macao-16-5-718

Security Model for WS



Each party has its own claims, security token and policy.

e-Macao-16-5-719

Scenario for the Security Model

1) One end - the requestor:
    a) wants to invoke a web service
    b) has claims, such as identity and privileges
2) other end - the web service:
    a) has a policy – requires encryption of messages and authentication of requestors

e-Macao-16-5-720

Sending Security Claims

How security claims can be represented in messages?

WS security model suggest that all security claims should be included in the security token that is attached to the request message.

For instance:

1) identification via password
2) X.509 certificate

are security claims, therefore they are represented as security tokens attached to the message.

e-Macao-16-5-721

WS Security Specifications

On April 2004 OASIS officially announced Web Services Security v1.0, composed of:

1) Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
2) Web Services Security Username Token Profile 1.0
3) Web Services Security X.509 Certificate Token Profile
4) two XML schema documents:
    a) secext.xsd
    b) utility.xsd

Reference: http://www.**oasis**-open.org/committees/wss/

e-Macao-16-5-722

WSS - SOAP Message Security 1

Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) specification proposes a standard set of SOAP (1.1 and 1.2) extensions.

It can be used when building secure web services to implement message content integrity and confidentiality.

Provides support for:

1) multiple security token formats
2) multiple trust domains
3) multiple signature formats
4) multiple encryption technologies

e-Macao-16-5-723

WSS - SOAP Message Security 2

The specification provides three main mechanisms:

1) ability to send security tokens as part of a message
2) message integrity
3) message confidentiality

They do not provide a complete security solution for WS.

This specification is a building block that can be used in conjunction with other WS extensions.

e-Macao-16-5-724

SOAP Security Namespaces

The XML namespaces URI that must be used by implementations are:

wsse: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

wsu: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

The XML namespaces URI for digital signature and encryption are:

ds: http://www.w3.org/2000/09/xmldsig#

xenc: http://www.w3.org/2001/04/xmlenc#

e-Macao-16-5-725

SOAP Message Security Model

The specification uses security tokens combined with digital signatures to protect and authenticate messages.

Security tokens:

1) state claims
2) can be used to declare the binding between authentication keys and security identities

Signatures are used to verify the message origin and integrity:

1) bind the identity of the sender with the message
2) confirm the claims in a security token

e-Macao-16-5-726

Message Protection

The specification provides a means to protect a message by encrypting and/or digitally signing a body, a header, or any combination of them.

Message integrity is provided by XML Signature [XMLSIG] in combination with security tokens to ensure that modifications to messages are detected.

Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens to keep portions of a SOAP message confidential.

The specification defines syntax and semantics of signatures within `<wsse:Security>` element.

e-Macao-16-5-727

Rules for Invalid Messages

A message recipient should reject messages:

a) containing invalid signatures
b) messages missing necessary claims
c) messages whose claims have unacceptable values

These are unauthorized or malformed messages.

e-Macao-16-5-728

SOAP Security Example 1

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
              xmlns:ds="...">
  <S11:Header>
   <wsse:Security  xmlns:wsse="...">
        <xxx:CustomToken wsu:Id="MyID"
             xmlns:xxx="http://www.example.com/token">FHUIORv...
        </xxx:CustomToken>

        <ds:Signature>

          <ds:SignedInfo>
            <ds:CanonicalizationMethod
               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
            <ds:Reference URI="#MsgBody">
              <ds:DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
```

1) what is being signed

2) type of canonicalization being used

e-Macao-16-5-729

SOAP Security Example 2

```
          <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
          <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI="#MyID"/>
               </wsse:SecurityTokenReference>
          </ds:KeyInfo>
        </ds:Signature>

   </wsse:Security>
  </S11:Header>

  <S11:Body wsu:Id="MsgBody">
     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">QQQ
     </tru:StockSymbol>
  </S11:Body>

</S11:Envelope>
```

e-Macao-16-5-730

Id Attribute

There are many situations where elements within SOAP messages need to be referenced.

The specification introduces `wsu:ID` attribute to reference elements.

Example:

```
...
<ds:Reference URI="#MsgBody">
...
<S11:Body wsu:Id="MsgBody">
```

e-Macao-16-5-731

Security Header

The <wsse:Security> header block provides a mechanism for attaching security-related information targeted at a specific recipient.

```
<S11:Envelope>
   <S11:Header>
        . . .
        <wsse:Security soap:role="…" soap:mustaunderstand=".." >
            . . .
        </wsse:Security>
              . . .
   </S11:Header>
    . . .
</S11:Envelope>
```

This header is extensible by design -it supports many types of security information.

e-Macao-16-5-732

Security Headers Rules 1

A message may have multiple <wsse:Security> headers if they are targeted for separate recipients.

Only one <wsse:Security> header may omit the role attribute (actor in SOAP 1.1).

Two <wsse:Security> headers must not have the same value for the role attributes (actor in SOAP 1.1).

e-Macao-16-5-733

Security Headers: Rules 2

Message security information targeted for different recipients must appear in different <wsse:Security> header blocks.

The <wsse:Security> header block without a specified role (or actor) may be processed by anyone, but must not be removed prior to the final destination or endpoint.

e-Macao-16-5-734

Security Tokens

The extensibility of the <wsse:Security> header allows to insert security tokens based on XML into the header.

The security token may be:

a) user name token

b) binary security token

c) XML token

e-Macao-16-5-735

User Name Token

The <wsse:UsernameToken> element is introduced as a way of providing a username.

This element is optionally included in the <wsse:Security> header.

It contains a mandatory UserName and an optional Password sub-elements.

It is used for password authentication, such as HTTP Basic Authorization.

Disadvantage - the plaintext representation is extremely insecure.

e-Macao-16-5-736

User Name Token Example

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." >
  <S11:Header>
     <wsse:Security>
          <wsse:UsernameToken wsu:ID="…" >
              <wsse:Username>Mary</wsse:Username>
          </wsse:UsernameToken>
     </wsse:Security>
      . . .
  </S11:Header>
  . . .
</S11:Envelope>
```

e-Macao-16-5-737

Binary Security Token

Binary security tokens, such as: X.509 certificates or Kerberos tickets, or other non-XML formats require a special encoding format for inclusion.

The <wsse:BinarySecurityToken> element defines two attributes:

1) <ValueType>: indicates what is the security token:

X509v3	X.509 v3 digital certificate
Kerberos5TGT	Kerberos ticket-granting ticket
Kerberos5ST	Kerberos service ticket

2) <EncodingType>: specifies how the security token is encoded, using a URI

e-Macao-16-5-738

Binary Security Token Example

```
<wsse:BinarySecurityToken wsu:ID="Kerberosv5ST"
                ValueType="Kerberosv5ST"
                EncodingType="wsse:Base64Binary" />
```

e-Macao-16-5-739

XML Token

The specification also defines multiple mechanisms for identifying and referencing security tokens using:

1) `wsu:ID` attribute

2) `wsse:SecurityTokenReference` element

e-Macao-16-5-740

Security Token Reference

A security token conveys a set of claims.

Sometimes these claims reside somewhere else and need to be retrieved by the receiving application.

The `<wsse:SecurityTokenReference>` element provides an extensible mechanism for referencing security tokens.

```
<wsse:SecurityTokenReference wsu:ID="...">
    . . .
</wsse:SecurityTokenReference>
```

e-Macao-16-5-741

Use of Security Token Reference

The `<wsse:SecurityTokenReference>` element can be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to retrieve the key information from a security token placed somewhere else.

It is recommended to use it when applying XML Signature and XML Encoding to reference the security token used for the signature or encryption.

```
<wsse:SecurityTokenReference>
    <wsse:Reference URI="#MyID"/>
</wsse:SecurityTokenReference>
```

A.7.3. Digital Signatures

e-Macao-16-5-742

Security Outline

1) Security Basics
2) Web Service Security
3) Digital Signatures

e-Macao-16-5-743

Signatures

Signatures are used to:

1) enable message recipients to determine whether the message was altered in transit
2) verify that the claims in a particular security token apply to the producer of the message

The specification allows multiple signatures and signature formats to be attached to a message.

Each signature may refer to different or overlapping parts of a message.

e-Macao-16-5-744

Signature Algorithms

The specification builds on XML Digital Signature Specification.

XML Digital Signature specification (XML Signature) defines how to sign part of an XML document in a flexible manner, using two canonicalization algorithms:

1) XML Canonicalization (Inclusive Canonicalization)
2) Exclusive XML Canonicalization

Neither one solves all possible problems that can arise.

e-Macao-16-5-745

Signature Algorithms: Problems

Two problems:

1) XML allows different documents to be considered equivalent. For instance: duplicate namespace declaration can be removed or created

2) if the signature covers something like “nms:abc”, its meaning may change if nms is redefined

Related to the second problem:

a) it could be solved by expanding all the values

b) mechanisms like XPATH considers nms1=http://example.com to be different from nms2=http://example.com

e-Macao-16-5-746

Canonicalization Example

Document 1:

```
<?xml version ="1.0" encoding="us-ascii" ?>
<example
     a="a"
     b="b"
></example>
```

Document 2:

```
<?xml version ="1.0" encoding="us-ascii" ?>
<example a="a" b="b" />
```

These two documents appear quite different, although with canonicalization are translated both to:

```
<?xml version ="1.0" encoding="us-ascii" ?>
<example a="a" b="b"></example>
```

e-Macao-16-5-747

Inclusive Canonicalization

The fundamental difference between Inclusive and Exclusive Canonicalization is the namespace declarations.

Inclusive Canonicalization copies all the declarations that are currently in force, even if they are defined outside the scope of the signature.

Problem: if the file is moved into another XML document which has other declarations, the signature will be invalid.

XML-C14N specifies inclusive canonicalization.

e-Macao-16-5-748

Exclusive Canonicalization

Exclusive Canonicalization copies only the namespaces that are "visibly used", those that are part of the XML syntax.

It does not look into attributes values or element content, so the namespaces declarations required to process these are not copied.

It allows you to create a list of the namespaces that must be declared.

Exclusive canonicalization is useful when you have a signed XML document that you wish to insert into other XML documents.

EXC-C14N specifies exclusive canonicalization.

The specification strongly recommends the use of exclusive canonicalization.

e-Macao-16-5-749

Signing Messages

The `<wsse:Security>` header block may be used to carry a signature compliant with XML Signature specification within a SOAP envelope.

Multiple signature entries may be added within one `<wsse:Security>` header block.

To add a signature, a `<ds:Signature>` element must be inserted at the top of the existing content of the `<wsse:Security>` header block.

e-Macao-16-5-750

XML Signature

In XML Signature, an element `Signature` is defined with its descendants under the namespace http://www.w3.org/2000/09/xmldsig#

The WS-Security defines how to embed the `Signature` element in SOAP messages as a header entry.

e-Macao-16-5-751

Signature Example

```
<S11:Header>
 <wsse:Security  xmlns:wsse="...">
  <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
      <ds:Reference URI="#MsgBody">
         <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
   <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
  </ds:Signature>
 </wsse:Security>
</S11:Header>
```

e-Macao-16-5-752

Signing Messages 1

The part signed is specified by `Reference` and is transformed by the method specified in `CanonicalizationMethod`.

The digest value is calculated with an algorithm specified by the `DigestMethod` element.

The value is inserted in the `DigestValue` element represented in Base64.

```
<ds:SignedInfo>
   <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <ds:SignatureMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
   <ds:Reference URI="#MsgBody">
       <ds:DigestMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
   </ds:Reference>
</ds:SignedInfo>
```

e-Macao-16-5-753

Signing Messages 2

The value of the part is not signed directly. The `SignedInfo` element is signed.

The `SignedInfo` element is canonicalized and signed with the algorithm specified in the `SignatureMethod` element.

The calculated value is inserted in `SignatureValue` element with Base64 format.

```
<ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
```

e-Macao-16-5-754

Security Summary 1

Security requirements include:

1) confidentiality: protect messages against eavesdroppers
2) integrity: protect messages against deliberate or accidental modifications
3) authentication: guarantees access to those who provide proof of identity
4) non-repudiation: guarantees that the sender not deny the message
5) authorization: decides whether an entity can access a particular resource

e-Macao-16-5-755

Security Summary 2

Cryptography technologies provide a basis for protecting messaged exchanged between partners.

Cryptography algorithms are classified in: symmetric and asymmetric.

Symmetric algorithms require the use of the same key for encryption and decryption.

Asymmetric algorithms uses a public and a private key.

Digital signatures assures integrity and non-repudiation of messages.

e-Macao-16-5-756

Security Summary 3

Different technologies can be applied:

1) Password authentication
2) HTTP Basic Authentication
3) Digital Signature Authentication
4) Security protocols – SSL

To solve the difficulties of combining these technologies security infrastructure have been developed.

e-Macao-16-5-757

Security Summary 4

Some security infrastructure:

1) User registries – managing user identifications and passwords
2) Public Key Infrastructure – certificate authority providing certificates
3) Kerberos – a key distribution center provides tickets based on a single sign on

e-Macao-16-5-758

Security Summary 5

To manage different security infrastructures, web services define a security model based on three parties:

1) requester
2) web service
3) security token service

Each of them posses:

1) policy
2) security token
3) claims

e-Macao-16-5-759

Security Summary 6

On April 2004 OASIS announced Web Services Security v1.0 composed by a set of specifications.

WS Security SOAP Message Security 1.0 propose a set of SOAP extensions to assure integrity and confidentiality of the message contents.

Security headers may include:

1) security tokens:
   a) user name token
   b) binary security token
   c) XML token
2) digital signatures

e-Macao-16-5-760

Acknowledgements

I would like to thank to:

1) Tomasz Janowski and Adegboyega Ojo - for their valuable comments

2) Gabriel Oteniya - for his help in developing the examples

3) Audience - for your presence and valuable comments

B. Assessment

B.1. Set 1

1. (8%)	Which line of the following xml document is wrong: 1 <foo:aaa xmlns:foo="ns1" xmlns="ns2"> 2 <foo:aaa> 3 <foo:bbb xmlns:foo=""> 4 <foo:bbb>abcd</foo:bbb> 5 </foo:bbb> 6 <ccc xmlns =""> 7 abcd 8 </ccc> 9 </foo:aaa>	
	A.	Line 1
	B.	Line 3
	C.	Line 6
	D.	Line 7
Answer		B

2. (8%)	What will be the class name of a Java class which maps to the following XML schema, : <schema> <complexType name="Address"> <sequence> <element name="number" type="xsd:int"/> <element name="street" type="xsd:string"/> <element name="city" type="xsd:string"/> </sequence> </complexType> </schema>	
	A.	Address
	B.	number
	C.	sequence
	D.	schema
Answer		A

3. (8%)	Which one of the following is not a part of a SOAP message?	
	A.	Envelope
	B.	Port
	C.	Header
	D.	Body
Answer		B

4. (9%)	Which one of the following statement is wrong?	
	A.	SOAP is a protocol for exchanging information in a decentralized and distributed environment.
	B.	SOAP deals with objects with remote object references.
	C.	SOAP clients do not hold any stateful references to remote objects.
	D.	SOAP uses XML to send and receive messages.
Answer		B

5. (9%)	Which one of the following statement about WSDL document is wrong?	
	A.	The `binding` element has attributes: `name` and `portType`.
	B.	must contain a root element: `definitions`.
	C.	like Java interfaces, is a contract between the server and client developers.
	D.	The `service` element describes the location of the web service.
Answer		A

6. (8%)	Which one of the following statement is correct?	
	A.	UDDI specifications define registry service for web services only.
	B.	UDDI uses the publsherAssertion structure to define the relationship between two businessEntities.
	C.	A web service provider can update its information through any UDDI Business Registry.
	D.	Answer A, B and C
Answer		B

7. (8%)	Which one of the following statement is wrong?	
	A.	X.509 is a kind of security token.
	B.	Digital signature can be used to determine whether the message was altered in transit.
	C.	Certificate authority (CA) is an entity which issues certificates.
	D.	Exclusive XML Canonicalization is also known as XML Canonicalization.
Answer		D

8. (9%)	A JAX-RPC client cannot invoke the service using:	
	A.	Client side stubs automatically generated by a tool
	B.	A client must invoke the service by looking up the Java interface in UDDI
	C.	Dynamic Invocation Interface (DII)
	D.	Dynamic proxies
Answer		B

9. (9%)	Canonical XML specification is used to	
	A.	Determine if two XML documents are logically equivalent
	B.	Digitally encrypt an XML document
	C.	Digitally sign an XML documents
	D.	Register a web service
Answer		A

10. (8%)	SOAP with Attachments API for Java (SAAJ) is	
	A.	An API to for Java messaging.
	B.	An API for remote procedure calls.
	C.	An API to produce, consume and manipulate the XML structure for the SOAP message programmatically.
	D.	An API for describing, discovering and integrating business services.
Answer		C

11. (8%)	In a WSDD file, which one is not a valid value for the `provider` attribute of the `service` element?	
	A.	Java:RPC
	B.	Java:MSG
	C.	Java:Handler
	D.	Java:EJB
Answer		C

12. (8%)	In a WSDL file, `operation` element within the `portType` element may define:	
	A.	Input message
	B.	Output message
	C.	Fault message
	D.	Answer A, B and C
		D

B.2. Set 2

1. (9%)		Canonical XML specification is used to
	A.	Register a web service
	B.	Digitally encrypt an XML document
	C.	Digitally sign an XML documents
	D.	Determine if two XML documents are logically equivalent
Answer		D

2. (8%)		In a WSDD file, which one is not a valid value for the `provider` attribute of the `service` element?
	A.	Java:RPC
	B.	Java:MSG
	C.	Java:Handler
	D.	Java:EJB
Answer		C

3. (8%)		In a WSDL file, `operation` element within the `portType` element may define:
	A.	Input message
	B.	Output message
	C.	Fault message
	D.	Answer A, B and C
Answer		D

4. (9%)		Which one of the following statement is wrong?
	A.	SOAP is a protocol for exchanging information in a decentralized and distributed environment.
	B.	SOAP deals with objects with remote object references.
	C.	SOAP clients do not hold any stateful references to remote objects.
	D.	SOAP uses XML to send and receive messages.
Answer		B

5. (9%)	A JAX-RPC client cannot invoke the service using:	
	A.	Client side stubs automatically generated by a tool
	B.	A client must invoke the service by looking up the Java interface in UDDI
	C.	Dynamic Invocation Interface (DII)
	D.	Dynamic proxies
Answer		B

6. (8%)	Which line of the following xml document is wrong: 1 <foo:aaa xmlns:foo="ns1" xmlns="ns2"> 2 <foo:aaa> 3 <foo:bbb xmlns:foo=""> 4 <foo:bbb>abcd</foo:bbb> 5 </foo:bbb> 6 <ccc xmlns =""> 7 abcd 8 </ccc> 9 </foo:aaa>	
	A.	Line 1
	B.	Line 3
	C.	Line 6
	D.	Line 7
Answer		B

7. (8%)	What will be the class name of a Java class which maps to the following XML schema, : <schema> <complexType name="Address"> <sequence> <element name="number" type="xsd:int"/> <element name="street" type="xsd:string"/> <element name="city" type="xsd:string"/> </sequence> </complexType> </schema>	
	A.	Address
	B.	number
	C.	sequence
	D.	schema
Answer		A

8. (8%)	Which one of the following is not a part of a SOAP message?	
	A.	Envelope
	B.	Port
	C.	Header
	D.	Body
Answer		B

9. (9%)	Which one of the following statement about WSDL document is wrong?	
	A.	The `binding` element has attributes: `name` and `portType`.
	B.	must contain a root element: `definitions`.
	C.	like Java interfaces, is a contract between the server and client developers.
	D.	The `service` element describes the location of the web service.
Answer		A

10. (8%)	Which one of the following statement is correct?	
	A.	UDDI specifications define registry service for web services only.
	B.	UDDI uses the publsherAssertion structure to define the relationship between two businessEntities.
	C.	A web service provider can update its information through any UDDI Business Registry.
	D.	Answer A, B and C
Answer		B

11. (8%)	Which one of the following statement is wrong?	
	A.	X.509 is a kind of security token.
	B.	Digital signature can be used to determine whether the message was altered in transit.
	C.	Certificate authority (CA) is an entity which issues certificates.
	D.	Exclusive XML Canonicalization is also known as XML Canonicalization.
Answer		D

12. (8%)	SOAP with Attachments API for Java (SAAJ) is	
	A.	An API to for Java messaging.
	B.	An API for remote procedure calls.
	C.	An API to produce, consume and manipulate the XML structure for the SOAP message programmatically.
	D.	An API for describing, discovering and integrating business services.
Answer		C

Web Services and Java

Elsa Estevez, Tomasz Janowski

and Gabriel Oteniya

UNU-IIST, Macau

The Course

1) objectives - what do we intend to achieve?

2) outline - what content will be taught?

3) resources - what teaching resources will be available?

4) organization - duration, major activities, daily schedule

Course Objectives

1) explain the concept of web services

2) present three XML technologies comprising web services:

   a) Simple Object Access Protocol (SOAP)
   b) Web Services Description Language (WSDL)
   c) Universal Description Discovery and Integration (UDDI)

3) present the best practices in developing web services with Java

4) motivate the use of web services for e-government

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

Introduction Outline

An overview of web services (WS).

Main points:

1) WS definition, components, process, properties
2) Service-Oriented Architectures (SOA)
3) WS and SOA
4) WS architecture stack and interoperability
5) WS implementation

SOAP Outline

SOAP = Simple Object Access Protocol

A protocol for exchanging XML messages in a distributed environment.

Main points:

1) introduction to SOAP
2) messaging framework: envelope and its components, processing rules
3) data structures and rules for encoding data and service requests
4) protocol binding framework
5) using SOAP to send binary data

WSDL Outline

WSDL = Web Services Description Language.

A language for describing web services with XML.

Main points:

1) introduction to WSDL
2) WSDL language structure
3) transmission primitives
4) WSDL extension mechanisms
5) WSDL and Java

AXIS Outline

Presentation of a particular SOAP engine - Apache AXIS.

Open source project.

Main points:

1) AXIS concepts and architecture

2) web service invocation

3) AXIS tools and configuration

4) web service deployment

5) service lifecycle

UDDI Outline

UDDI = Universal Description, Discovery and Integration

An open, platform-independent framework for describing, discovering and integrating business services.

Main points:

1) introduction

2) concepts

3) data types

4) registries

Security Outline

WS-Security describes enhancements to SOAP messaging in order to provide message integrity and confidentiality.

Main points:

1) basic security concepts

2) web service security

3) digital signatures

Course Resources - Books

1) Building Web Services with Java, Making sense of XML, SOAP, WSDL, and UDDI (2nd ed.) – Steve Graham, et. al. – Sams Publishing, 2004

2) Java Web Services Unleashed – Robert J. Brunner et al – Sams, 2002

3) Web Services Concepts, Architectures and Applications – Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju – Springer, 2004

4) WebSphere Application Developer Web Services Handbook – http://publib-b.boulder.ibm.com/abstracts/sg246891.html

Course Resources - Organizations

1) W3C – World Wide Web Consortium, http://www.w3.org

2) OASIS – Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org

3) Apache – Apache Software Foundation, http://www.apache.org

Course Resources - Specifications

1) SOAP - http://www.w3.org/TR/soap/
2) AXIS - http://ws.apache.org/axis/
3) WSDL - http://www.w3.org/TR/wsdl
4) UDDI - http://www.uddi.org/
5) WS-Security - http://www.oasis-open.org/

Course Logistics

1) duration - 42 hours

2) activities - lectures and development

3) timing

a)	Monday	9:00-13:00	14:30-17-45
b)	Tuesday	9:00-13:00	14:30-17-45
c)	Wednesday	9:00-13:00	
d)	Thursday	9:00-13:00	14:30-17-45
e)	Friday	9:00-13:00	

4) sessions - 7 morning, 5 afternoon

5) style - interactive and tutorial

Course Prerequisite

1) basic Java
2) distributed Java
3) XML, XML namespaces, XML Schema

Introduction

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

Service Definition

Definition by W3C:

A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requestors entities.

To be used, a service must be realized by a concrete provider agent.

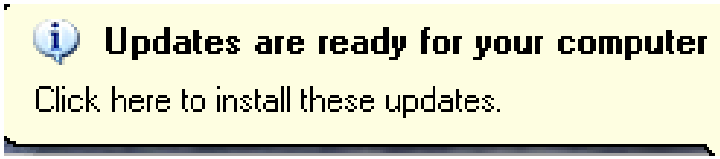
[Web Services Glossary
http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/]

Service Concepts

A service:

1) is a resource and has an owner
2) is provided by a person or an organization
3) must be realized by a (software) provider agent
4) performs one or more tasks
5) is used by a requestor agent

Example: a service for updating software



Web Service (WS)

A web service is a software application that applies XML to exchange data with other applications on other computers.

Features of web services:

1) Web services operate over any network (the Internet or a private Intranet) to achieve specific tasks.

2) The tasks performed by a web service are methods or functions that other applications can invoke and use.

3) Web service requests/responses can be sent/received between different applications on different computers belonging to different businesses.

Web Service Definition

Definition by W3C:

A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network:

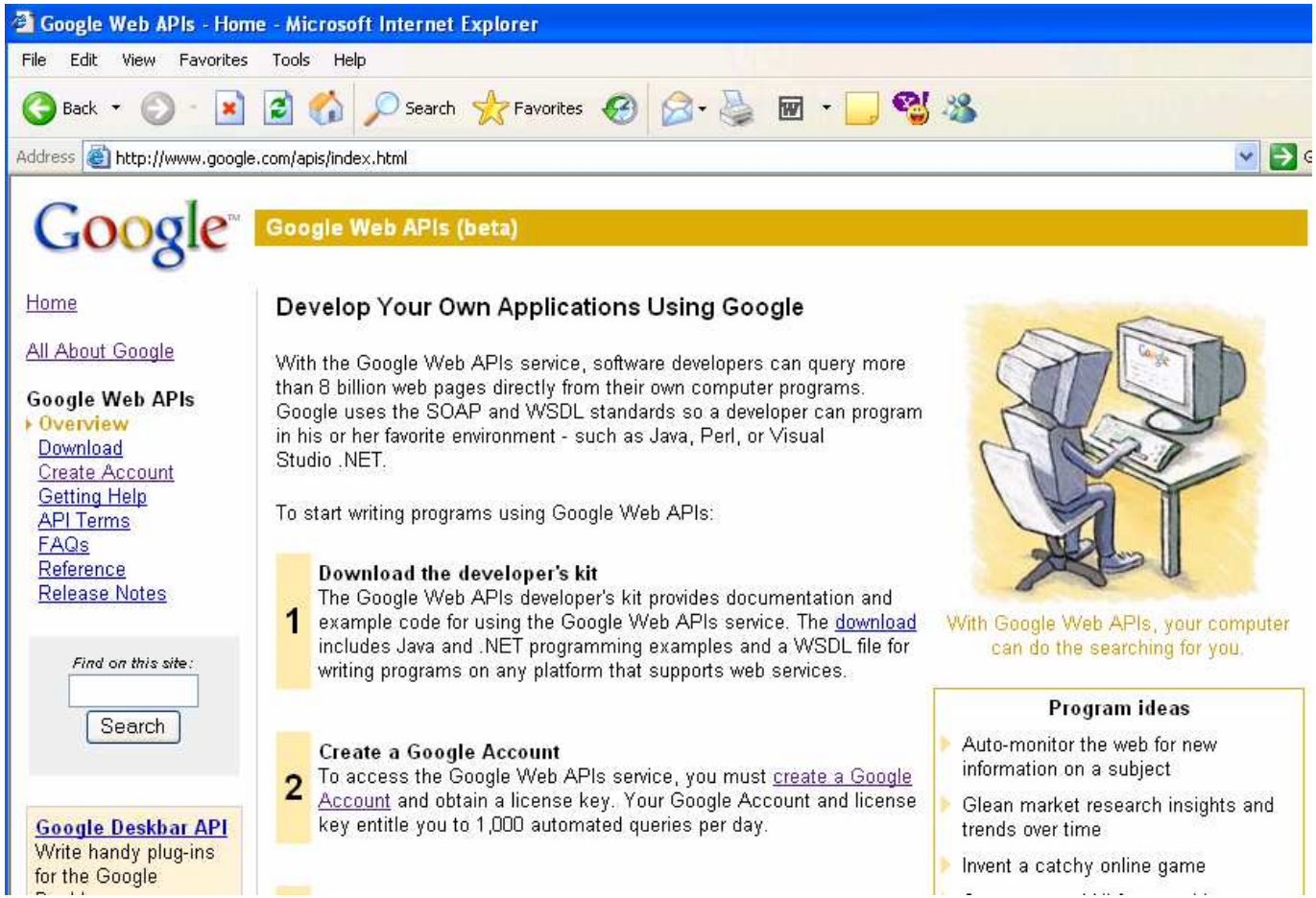
1) *It has an interface described in a machine-processable format (specifically WSDL).*

2) *Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web standards.*

[Web Services Glossary
http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/]

Web Service Example

A Google web service for Internet search - http://www.google.com/apis:



Task 1: Google Search

Copy from the server in `\WebServices` the folder `demos` to your local PC, in drive E: under a folder with your own name.

Objective: automatic search in Google using a WS

```
1) cd demos\Ws\FirstExample

2) dir
googleapi.jar
GoogleApiDemo.class
GoogleSearch.wsdl

1) copy googleapi.jar \j2sdk1.4.2_04\jre\lib\ext

2) java -cp \demos\WS\FirstExample GoogleApiDemo Macao
```

Service Description

A service description is data describing the capabilities of a web service:

1) all information needed in order to invoke the web service

2) the key concept for Service-Oriented Architectures (SOA)

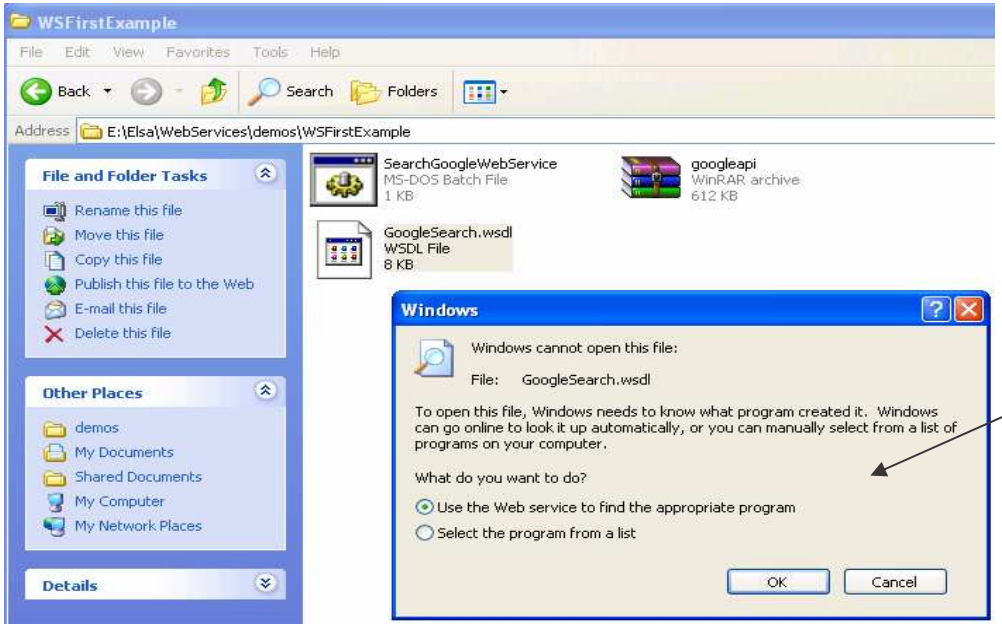
The standard for writing service descriptions is WSDL.

Task 2 : Google Description

1) `cd demos\Ws\FirstExample`

2) double-click `GoogleSearch.wsdl`

3) the following window appears:



applying a web service

4) open the file with a browser - WSDL document describing the service

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

SOA Definition

SOA = Service-Oriented Architecture

SOA is a software architecture where all software-implemented tasks and processes are designed as services to be consumed over a network.

Keywords:

1) architecture
2) service

SOA Approach

SOA approach:

The focus of design is the service interface.

A service:

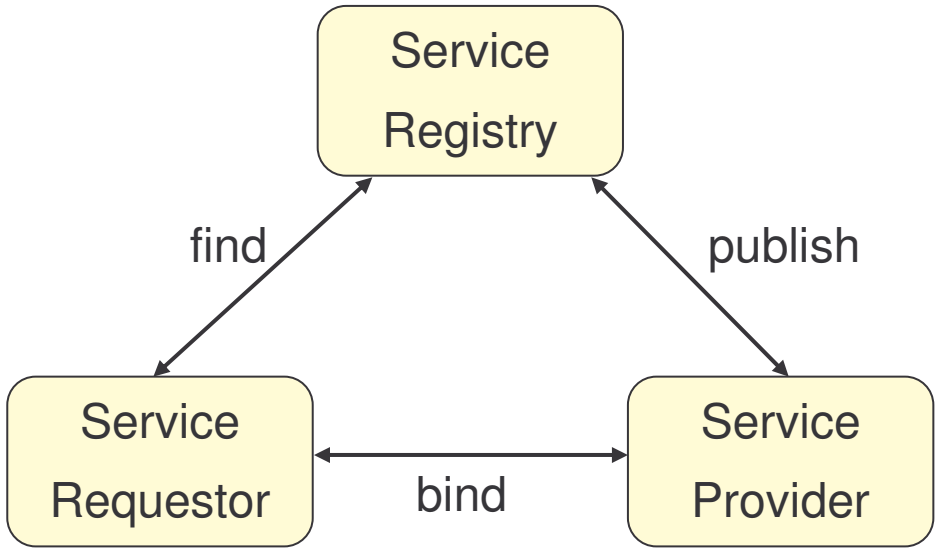
1) has a well-defined interface
2) can be potentially invoked over a network
3) can be reused in multiple business contexts

An application:

1) is integrated at the interface and not implementation level
2) is built to work with any implementation of a contract, resulting in a loosely coupled and more flexible system

SOA Components

1) roles
   a) service provider
   b) service requestor
   c) service registry
2) operations
   a) publish
   b) bind
   c) find



SOA Roles: Service Provider

What a service provider does?

1) creates a service description
2) deploys the service in a runtime environment to make it accessible to other entities over the network
3) publishes the service description to one or more services registries
4) receives messages invoking the service from service requestors

Any entity that hosts a network-available web service is a service provider.

SOA Roles: Service Requestor

What a service requestor does?

1) finds a service description published in a service registry
2) applies the service description to bind and invoke the web service hosted by a service provider

A service requestor can be any consumer of a web service.

SOA Roles: Service Registry

What a service registry does?

1) accepts request from service providers to publish and advertise web service descriptions
2) allows service requestors to search the collection of service descriptions contained within the service registry

The role of service registry is to enable match-making between service providers and service requestors.

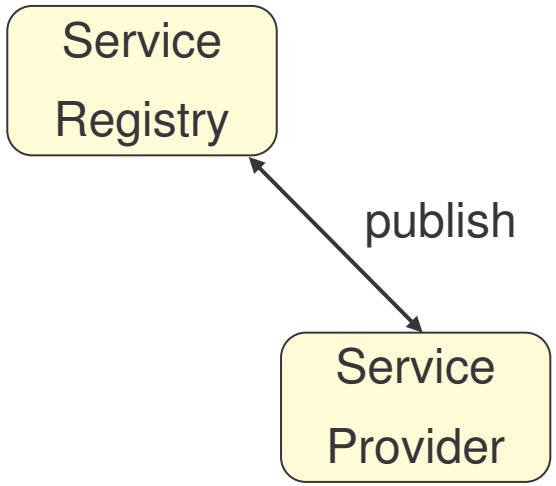
Once the match has been found, the interactions are carried out directly between the service requestor and the service provider.

SOA Operations: Publish

The publish operation is an act of service registration or service advertisement.

When a service provider publishes its web service in a service registry, it is advertising the service to the whole community of potential service requestors.

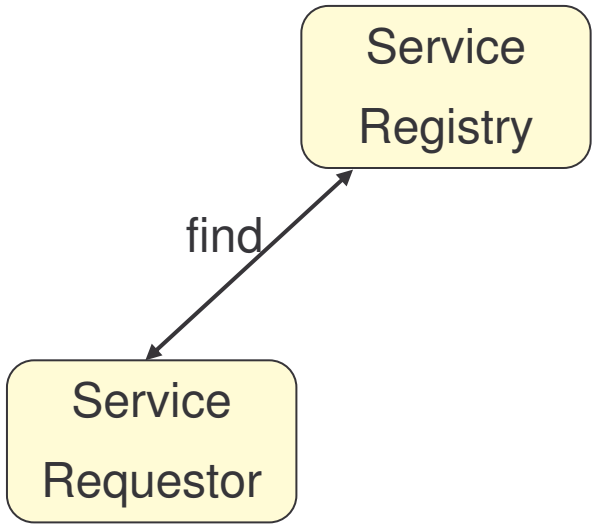
The details of the publish operation depends on how the service registry is implemented.



SOA Operations: Find

The find operation is an act of looking for a service satisfying certain conditions:

1) service requestor states a search criteria, such as: the type of the service, its quality, etc.
2) service registry matches the search criteria against the published web service descriptions



The result is a list of service descriptions that match the search criteria.

Details of the operation depend on the implementation of the service registry.

SOA Operations: Bind

The bind operation creates the client-server relationship between service requestor and service provider.

The operation can be:

1) dynamic - creating a client-side proxy on-the-fly based on the service description to invoke the web service
2) static - the developer hard-codes the way the client invokes the web service

SOA Properties 1

SOA is a form of distributed systems architecture.

It is characterized by:

1) logical view - a service is an abstraction is what actual programs, databases, businesses processes etc. are able to do.

2) message exchange – a service is defined in terms of the messages exchanged between provider and requestor agents and not in terms of the properties of the agents themselves

SOA Properties 2

3) abstraction – SOA hides the implementation details of the underlying languages, process and database structures, etc.

4) meta-data – a service is described by machine-processable meta-data

5) small number of operations – a service tends to rely on a small number of operations with relatively large and complex messages

6) network orientation - services are oriented to their use over a network

7) platform-neutral - messages are sent in a standardized format delivered through the interfaces. XML is typically used.

SOA Benefits 1

SOA enables the agents participating in the message exchange to be loosely coupled, which in turn allows for more flexibility:

1) a client is only coupled to a service, not to a server - the integration of the server takes place outside the scope of the client application

2) functional components and their interfaces are separated - new interfaces can be easily added

3) old and new functionality can be encapsulated as software components that provide and receive services

SOA Benefits 2

4) the control of business processes can be isolated:
   a) business-rule engine can control the workflow of a business process
   b) depending on the state, the engine invokes different services
5) services can be incorporated dynamically during runtime
6) service bindings are specified using configuration files and can be easily adapted to satisfy new needs

Service Description in SOA

The key to SOA is service description:

1) it is published by the service provider in the service registry

2) it is returned to the service requestor as a result of the search operation

3) it specifies to the service requestor:
   a) how to bind and invoke the web service
   b) what information is returned as a result of the invocation

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

WS Components

A web service includes three basic components:

1) a mechanism to find and register interest in a service

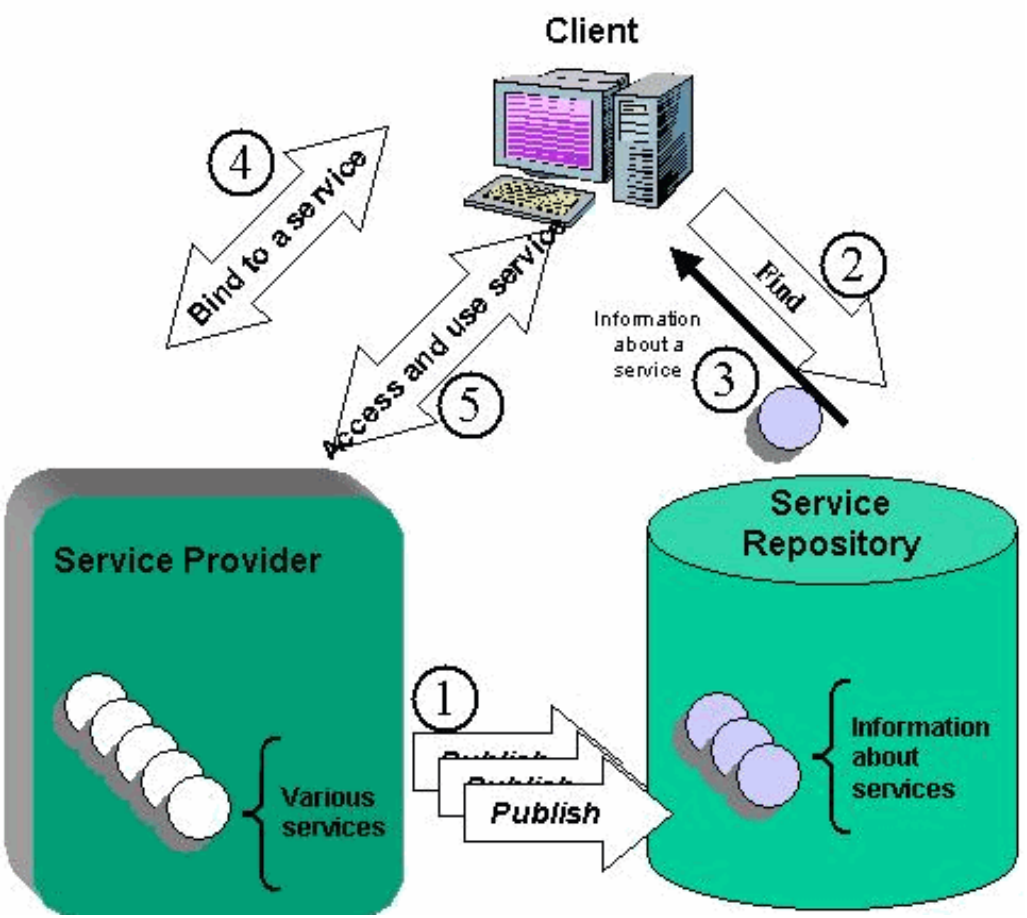
2) a definition of the service's input and output parameters

3) a transport mechanism to access a service

Web services also include other technologies that can be used to provide additional features such as security, transaction processing and others.

WS Process

1) a service provider publishes a service to an external repository
2) a client looks up for a service in the repository
3) the repository returns information about the service:
   - call format
   - provider address
4) the client binds to the underlying service
5) the client calls and accesses the service



[courtesy Al Saganich]

WS and Others

Web services do not introduce new functionality.

Similar functionality is provided by:

1) Sun/RPC
2) DCOM
3) Enterprise Java Beans
4) etc.

The difference is how this functionality is provided.

CORBA Application

Recall the application developed in the Distributed Programming course.

A client requests a file from the server. The server sends the file to the client. When received, the client saves the file on the local machine.

The steps involved:

1) define a service interface in IDL
2) map the IDL interface to Java (done automatically)
3) implement the interface (`FileInterface.idl`)
4) develop the server (`FileServer.java`)
5) develop a client (`FileClient.java`)
6) run the naming service, the server, and the client

CORBA Example 1

Running the application:

1) the server is running on the auxiliary PC:

   - run the CORBA naming service:

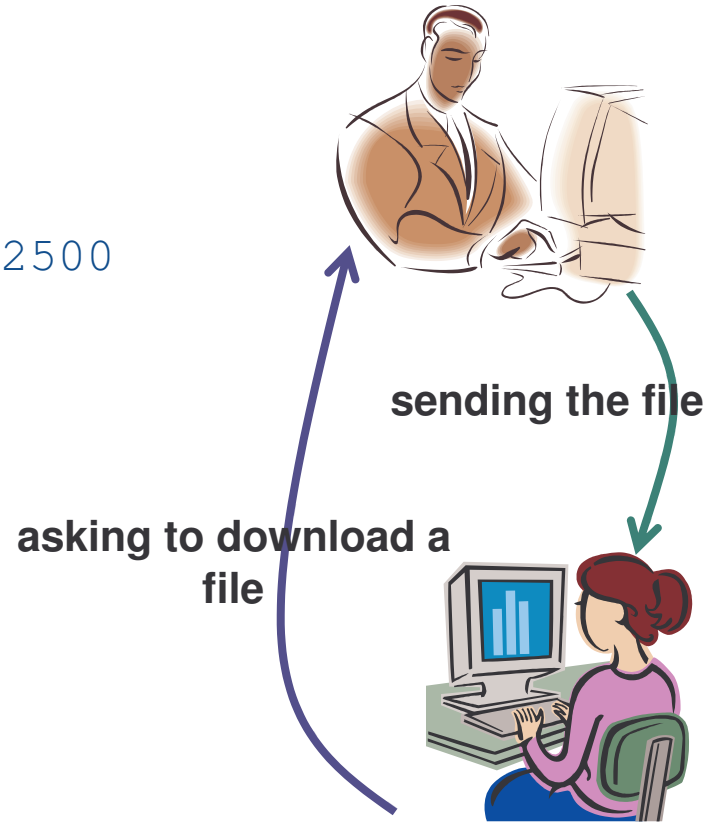
     ```
     tnameserv -ORBInitialPort 2500
     ```

   - start the server:

     ```
     java FileServer -ORBInitialPort 2500
     ```

2) the client is running on the current PC:
   run

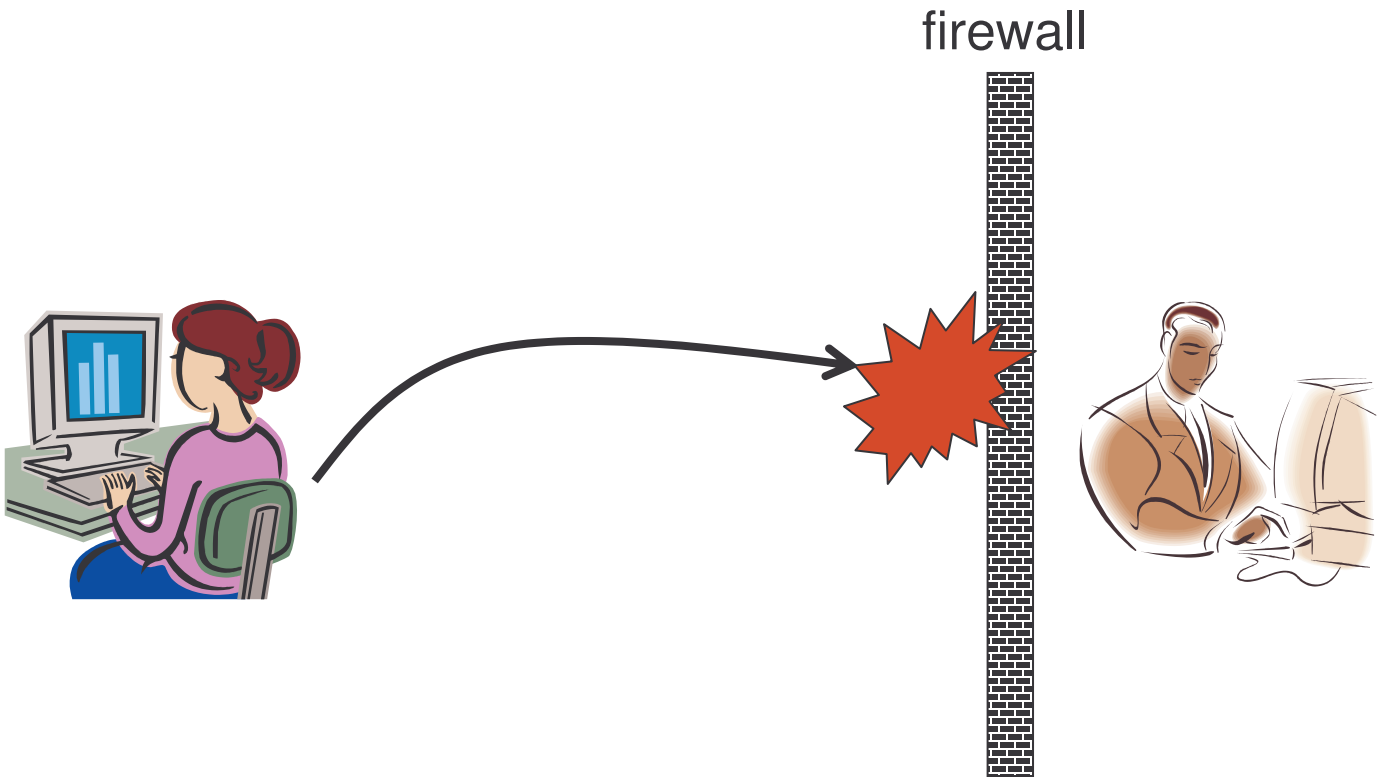
   ```
   run_CORBA_Client.bat
   ```



CORBA Example 2

What happens if we enable a firewall on the server side?

Let's try again to run the client application:



Web Service Application

Consider the same application, but built as a web service.

A client requests a file from the server. The server sends the file to the client. When received, the client saves the file on the local machine.

The steps involved:

1) setup the SOAP server
2) develop the server
3) develop the client
4) start the web server
5) deploy the server as a web service
6) run the client application

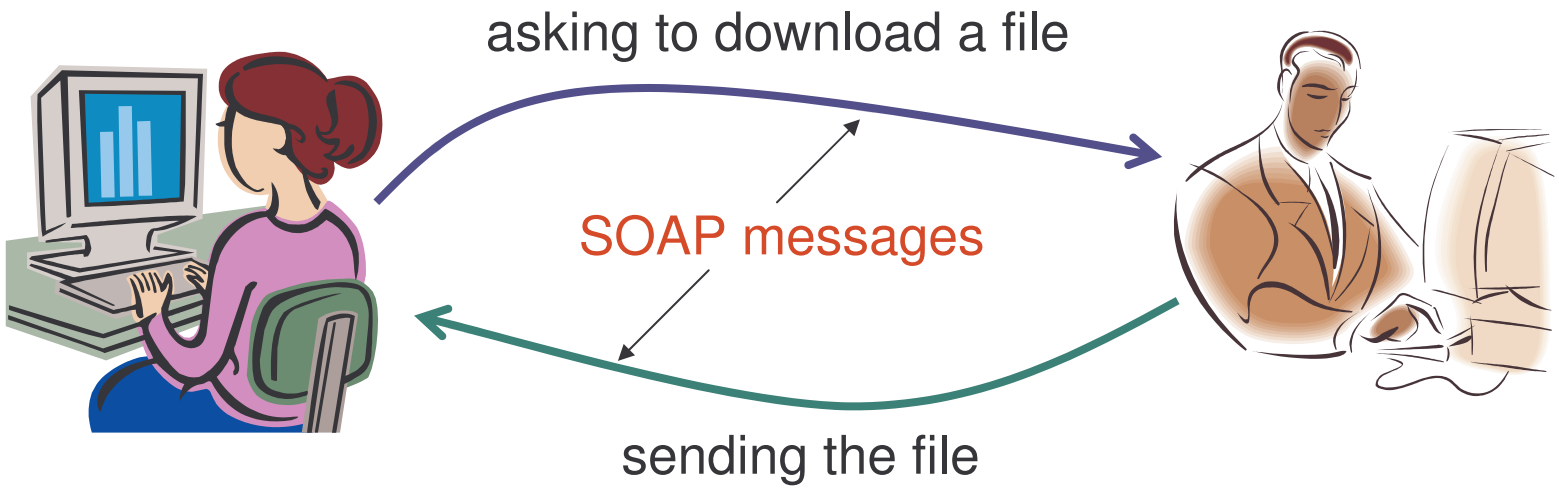
Web Service Example 1

Running the application:

1) the server is running on the auxiliary PC - start the web server

2) the client is running on the current PC

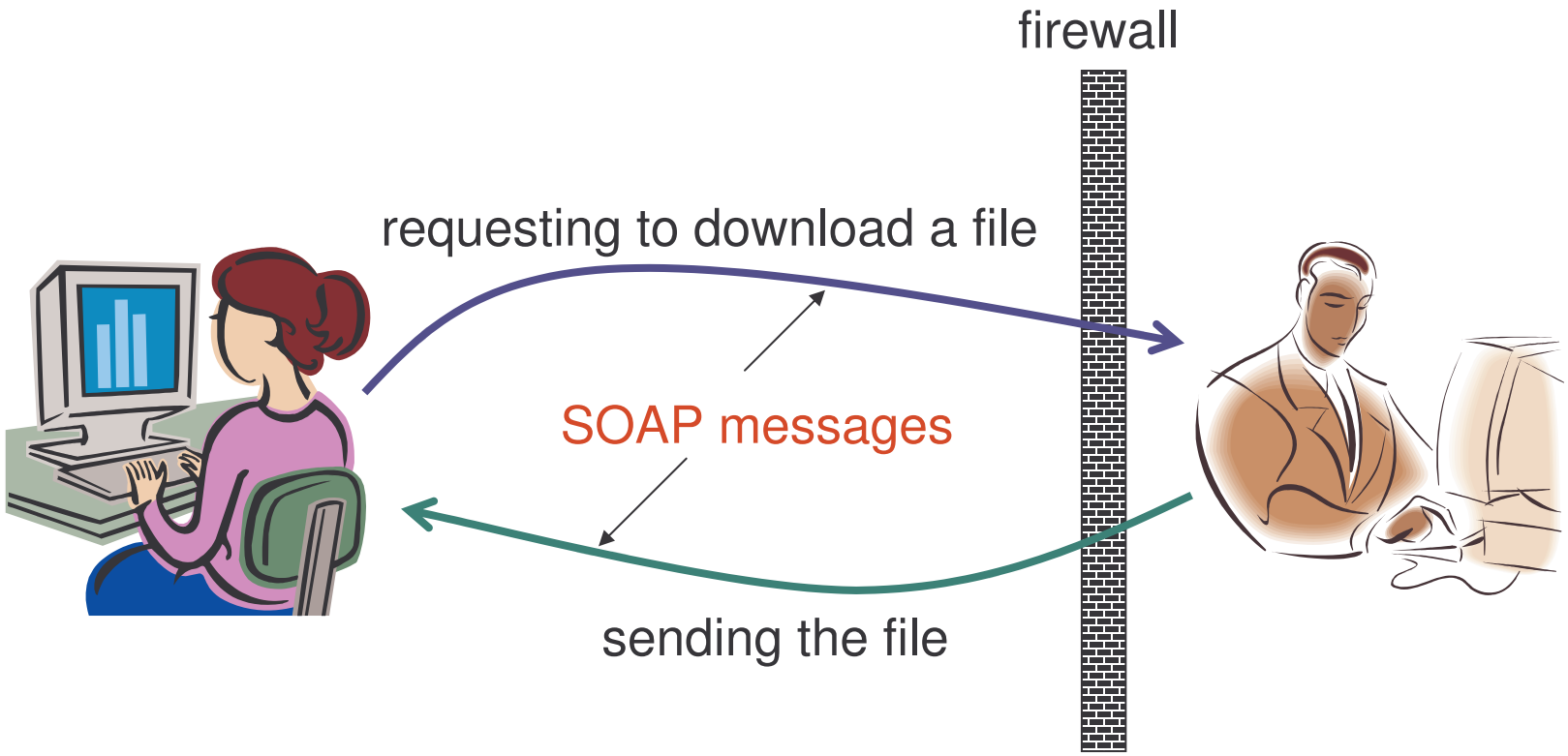
```
run_client.bat
```



Web Service Example 2

What happens if we enable a firewall on the server side?

Let's try again to run the client application:



Comparison: Communication

What is the observable difference between CORBA and WS applications? With the firewall enabled, the CORBA application was unable to run.

One advantage of SOAP is its explicit definition of HTTP binding through the process of hiding another protocol inside HTTP messages.

This allows SOAP messages to pass through a firewall unimpeded.

Firewalls will usually allow HTTP protocol through port 80, while they will restrict the use of other protocols or ports.

Comparison: Functionality

The same functionality in CORBA and WS.

The difference is how WS provides this functionality:

1) data is formatted for transfer using XML
2) data is passed using standard communication protocols
3) the exposed service is well defined in an XML vocabulary
4) services are found in standard ways with XML vocabularies

WS provides more a flexible design than CORBA.

Comparison: Standards

The main difference with past Distributed Computing Environments is adopted standards and implementations:

1) a standard lookup service – UDDI
2) a standard definition mechanism – WSDL
3) a standard way for two parties to communicate – SOAP

The foundation technology for all three (and more) is XML.

Web Service: Request Message

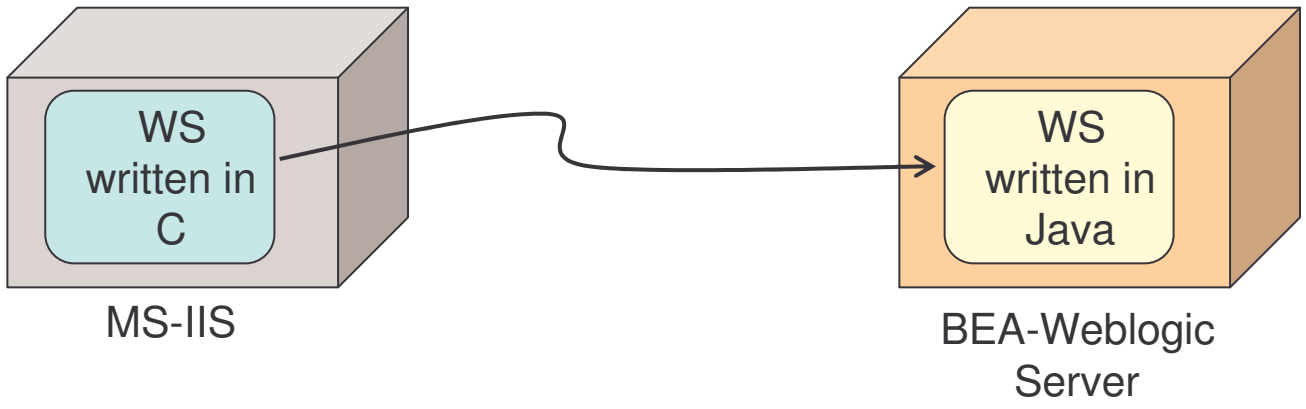
A request message looks as follows:

```
<soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:downloadFile
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://soapinterop.org/">
      <ns1:arg0
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xsi:type="soapenc:string"> name_of_file
      </ns1:arg0>
    </ns1:downloadFile>
  </soapenv:Body>
</soapenv:Envelope>
```

Web Service Implementation

The standards used by web services are defined with little concern for the underlying implementation mechanism.

Therefore: a web service written in C and running on Microsoft IIS can access a web service written in Java, running on BEA WebLogic Server.



Web Service Environments

Several environments exist to build, deploy and access web services.

Best known:

1) Microsoft's .Net Platform
2) Sun's Java 2 Platform

We rely on Java Web Services.

Traditional Communication

Traditional system communication:

1) systems must be tightly bound

2) data must be transferred in such a way that two systems agree beforehand on the format

3) various “network normal forms” were created to decide how bytes, integers, etc. were to be encoded for transfer

XML-Based Communication

Before - no common data-definition mechanism.

With XML:

1) common, well-defined data and representation
2) well-defined set of validity and well-formedness rules

Web service communication relies on XML syntax to write messages.

WS - Business Perspective

Web services and business processes/goals:

1) a web service is an implementation of a business process or a step within such a process
2) a web service is made available over a network to internal and/or external business partners to achieve specific business goals

Web services promote integration of applications within an organization and between different business partners.

Key feature: to allow for rapid construction of business applications by combining web services built internally with those of business partners.

Web Service Usage

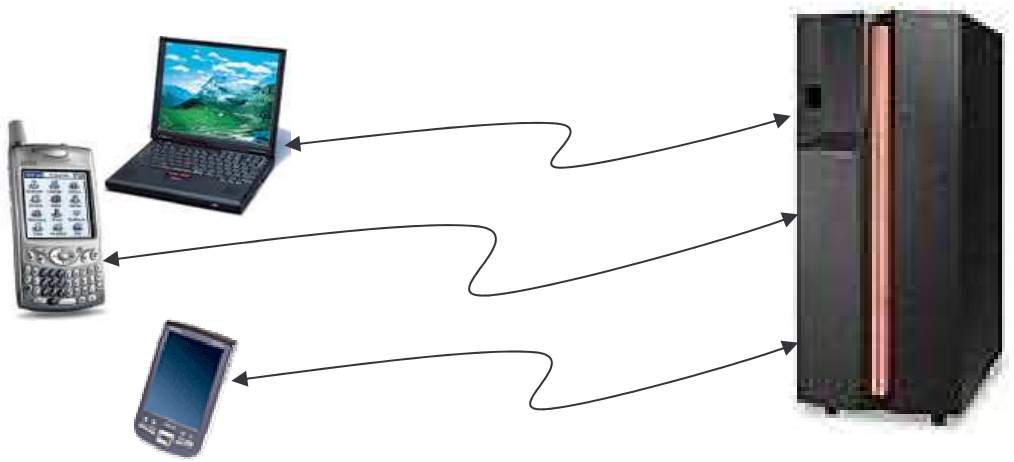
Two main scenarios of web service usage:

1) application integration
2) B2B partner integration over the Internet

WS Usage: Application Integration

Legacy systems can be wrapped as web services and made available for integration with other systems.

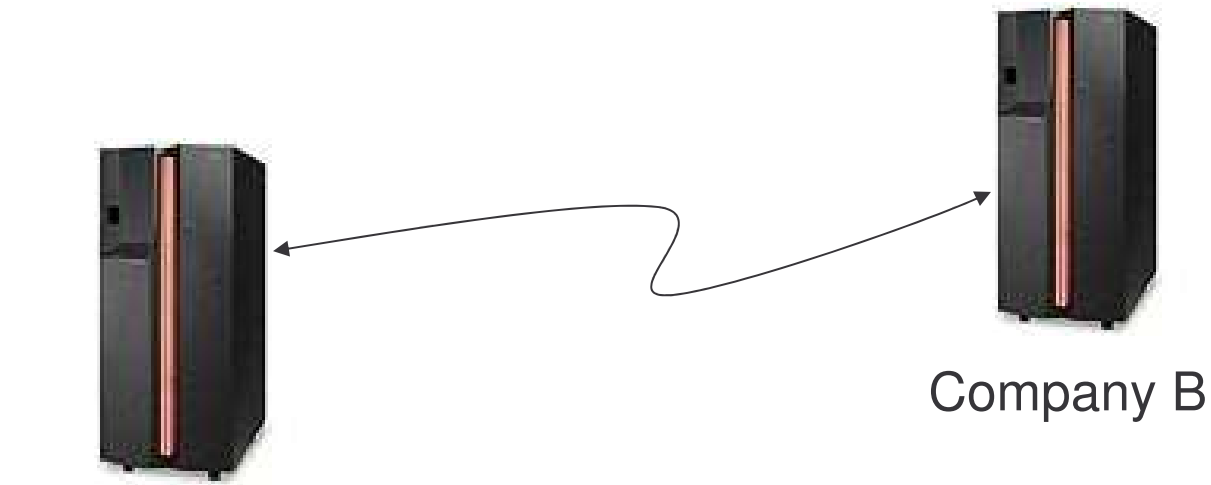
Applications exposed as web services are accessible by other applications running on different hardware platforms and written in different languages.



WS Usage: B2B Integration

Business-to-business (B2B) partner integration over the Internet.

B2B integrates business systems of two or more companies to support cross-enterprise business processes, e.g. supply chain management.



WS Properties 1

1) self-contained - no additional software is required for WS:
   a) client-side: a programming language with XML/HTML client support
   b) server-side: a web server and a SOAP server are needed

2) loosely coupled - client and server only knows about messages - a simple coordination level that allows for more flexible re-configuration

3) web-enabled – WS are published, located and invoked across the web, using established lightweight Internet standards

4) language-independent and interoperable - client and server may be implemented in different environments and different languages

WS Properties 2

5) composable - WS can be aggregated using workflow techniques to perform higher-level business functions

6) dynamically bound - with UDDI and WSDL, the discovery and binding of web services can be automated

7) programmatic access - the web services approach does not provide a graphical user interface but operates at the command level

8) wrap existing applications - stand-alone applications can easily be integrated by implementing a web service as an interface

Web Service Benefits

1) platform integration - the platform-neutrality of WS allows combining business systems using different devices (PDAs, cell phones, desktops) with service providers of all sizes and shapes

2) software integration - systems supporting new or modified business processes can be rapidly delivered by wrapping the existing functionality

3) standard technology - open standards enable developers to choose among different products, avoiding vendor-dependence

4) small businesses integration - the low cost of WS allows small businesses to deploy and participate in WS applications

5) easy integration - interface-based development using web service descriptions reduces the time to integrate applications

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

SOA and WS

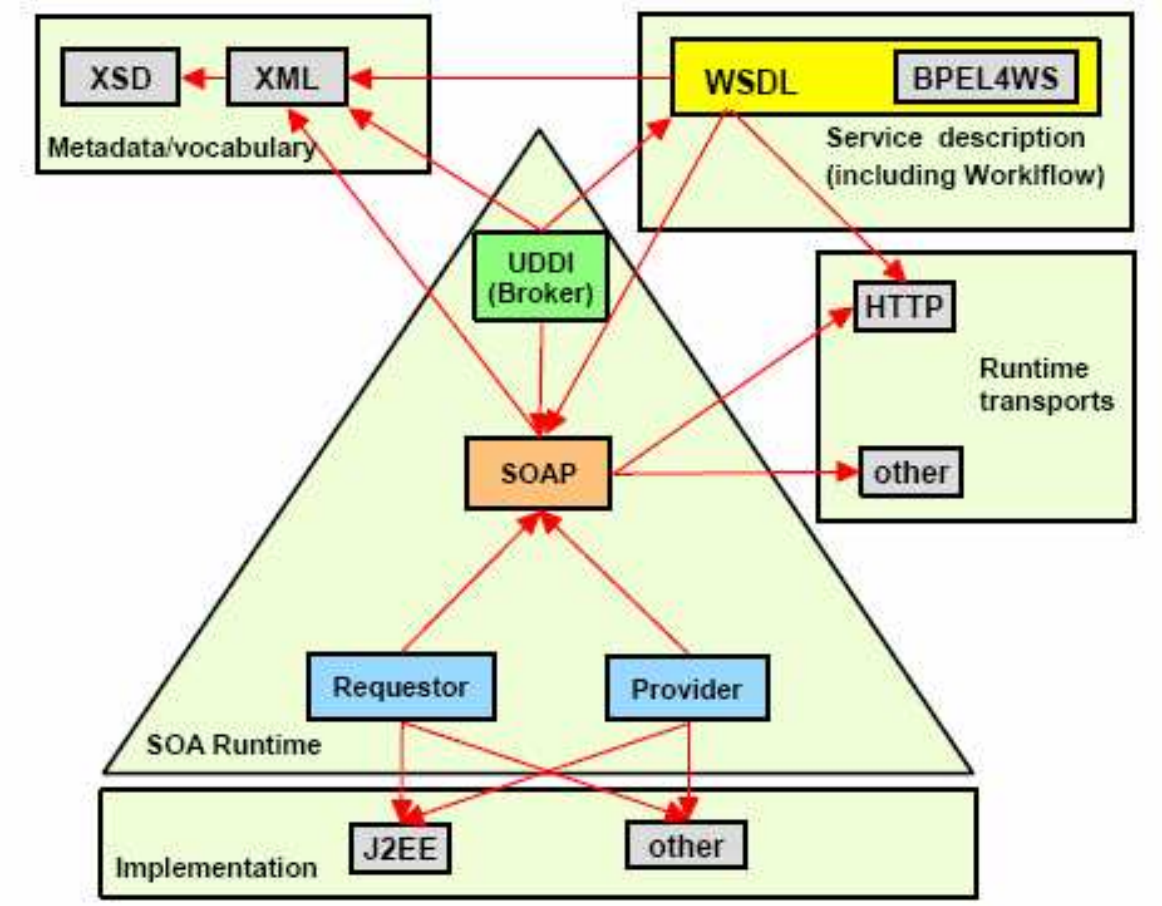
Service-Oriented Architecture:

1) provides an approach for building systems focused on a loosely coupled set of components (services) that can be dynamically composed
2) promotes seamless software integration as a business benefit

Web Services:

1) one approach to building SOA
2) provide a standard for a particular set of XML-based technologies that can be used to build SOA systems

WS-Based Approach to SOA



[courtesy IBM]

Using SOA and WS

SOA and WS are most appropriate for the applications that:

1) can operate over the Internet, accepting that reliability and performance of communication cannot be guaranteed in this case

2) do not require that all service requestors and providers are upgraded at the same time

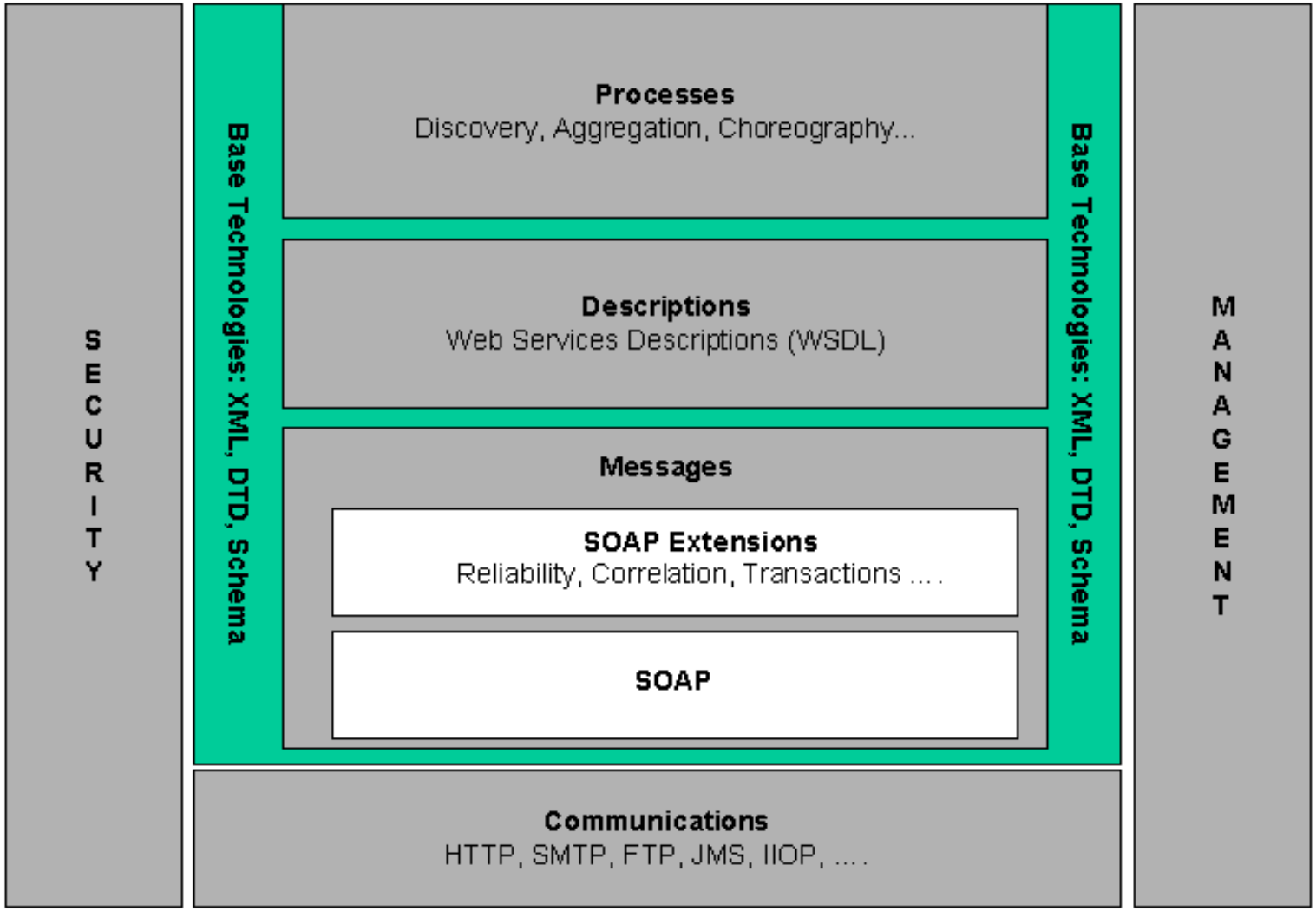
3) consist of the components running remotely on different execution platforms and vendor products

4) were designed using legacy technology but need to be exposed for use over a network, using a web service wrapping

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

Web Services Architecture Stack



[courtesy W3C]

Communications Layer

Web Services are essentially transport-neutral.

A web service message can be transported using HTTP or HTTPS, as well as more specialized transport mechanisms, such as e.g. JMS.

Web services insulate the designer from most of the details and implications of the message transport layer.

Messaging Layer

SOAP = Simple Object Access Protocol

A protocol to exchange structured information in a distributed environment.

SOAP extensions:

1) WS-ReliableMessaging - a standard for web services messaging to guarantee the receipt of messages for WS requestors and providers

2) WS-Transactions - a series of standards related to WS invocations in transactions (atomicity, consistency, isolation and durability semantics)

Descriptions Layer

WSDL = Web Services Description Language

A language that allows a service provider to specify the functional characteristic of its web services.

WSDL extensions:

1) WS-Policy - augment WSDL with non-functional constraints on WS

2) WS-ResourceProperties – describes how to define and access properties of resources through WS

Processes Layer: Discovery

Discovery - locating a machine-processable description of a web service that may have been previously unknown and that meets certain criteria.

UDDI = Universal Description, Discovery and Integration

UDDI defines a way to store and retrieve information about web services.

Processes Layer: Choreography

Choreography - defines how multiple cooperating independent agents exchange messages in order to perform a task to achieve a given goal.

WS-CDL = WS Choreography Description Language

WS-CDL describes peer-to-peer collaborations where ordered message exchanges result in accomplishing a common business goal.

WS Interoperability

Web Services tackle the set of problems related to loosely coupled, dynamically configured heterogeneous distributed computing.

WS Specifications:

1) A series of smaller, purpose-focused specifications dealing with narrow problems (security, transactions, etc.) in isolation.
2) Each WS specification is designed to be composed with the others.
3) WS designers determines which specifications their system needs and implement them accordingly.

WS-I Organization

Web Services Interoperability organization (WS-I):

1) WS-I is to standardize combinations of WS specifications that can be used to increase the level of interoperability between web services.

2) WS-I promotes the Basic Profile - implementation guidelines for how non-proprietary WS specifications, such as SOAP, WSDL, UDDI, should be used together for best interoperability.

WS-I website - http://www.ws-i.org/

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

Apache Axis

Apache and Axis:

1) Apache is an open-source HTTP server - http://ws.apache.org/

2) Axis is an Open Source SOAP engine - http://ws.apache.org/axis/

Axis converts Java objects to SOAP data for sending/receiving messages.

Apache Axis - Modules

Axis implements the standard Java API for Web Services - JAX-RPC.

Axis:

1) is compiled in the JAR file `axis.jar`
2) implements the JAX-RPC API declared in:
   a) `jaxrpc.jar`
   b) `saaj.jar`

All these files can be packaged into a web application called `axis.war` that can be deployed in a servlet container.

Servlet - Java class that can respond to HTTP requests.

Apache Axis - Requirements

What is needed?

1) Java 1.4
2) Tomcat 4.x

New versions of Java and Tomcat are not supported yet.

Tomcat

What is Tomcat?

1) Servlet container used in the official Reference Implementation of the Java Servlet and JavaServer Pages technologies.

2) A free, open-source implementation.

3) Was developed under the Jakarta project at the Apache Software Foundation.

4) Tomcat reference - http://jakarta.apache.org/tomcat

Installing Apache Axis

Steps for installing Apache Axis:

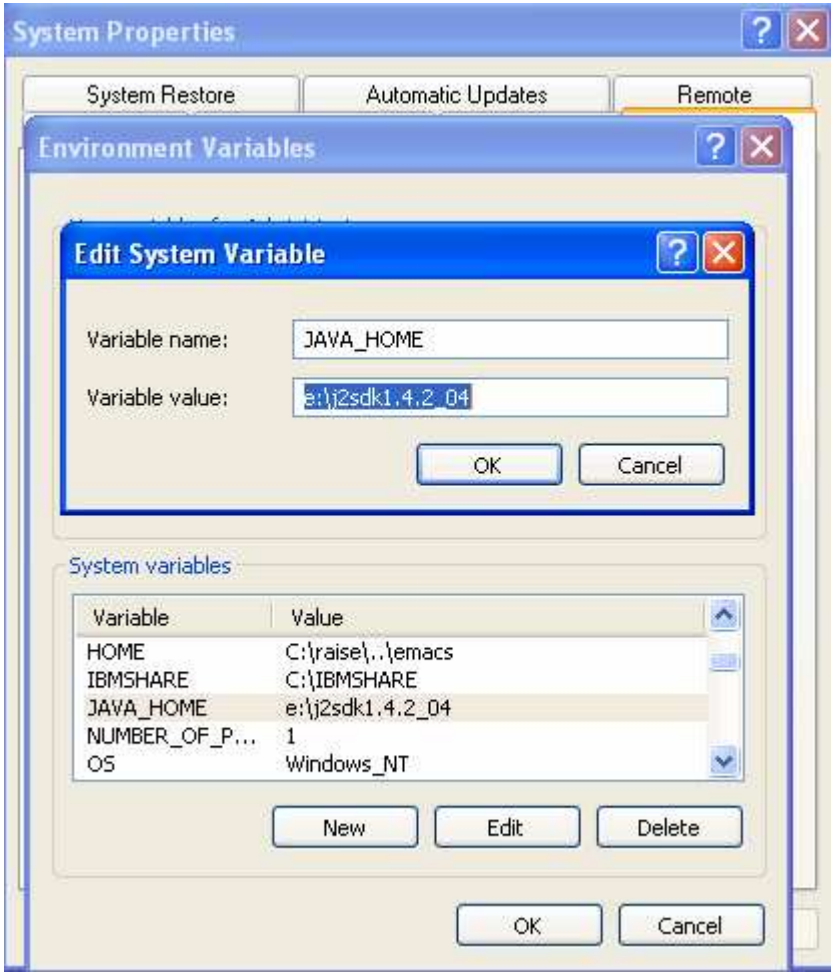
1) update the JAVA_HOME variable
2) install Tomcat
3) install Apache Axis
4) deploy Axis
5) validate the installation

Task 3: Update JAVA_HOME

1) pointing at My Computer press the right bottom and select Properties

2) select Advanced and Environment Variables

3) select System Variables and modify JAVA_HOME to contain the path to the j2sdk1.4 installation directory



Task 4: Installing Tomcat

1) visit http://jakarta.apache.org/tomcat

2) select Download - Binaries

3) select Download - Tomcat

4) select Tomcat 4

5) select Binary - 4.1.31.exe

6) save the file:
jakarta-tomcat-4.1.31.exe
on your own directory

Task 5: Installing Tomcat

7) execute: jakarta-tomcat-4.1.31.exe

8) Answer the following:

a) Using Java Development Kit found in j2sdk1.4.2_04 → OK

b) To the window about Apache License → I Agree

c) Setup Installation Options:

1) Tomcat

2) JSP Development Shell Extensions

3) Tomcat Start Menu Group

4) Documentation and Examples → Next

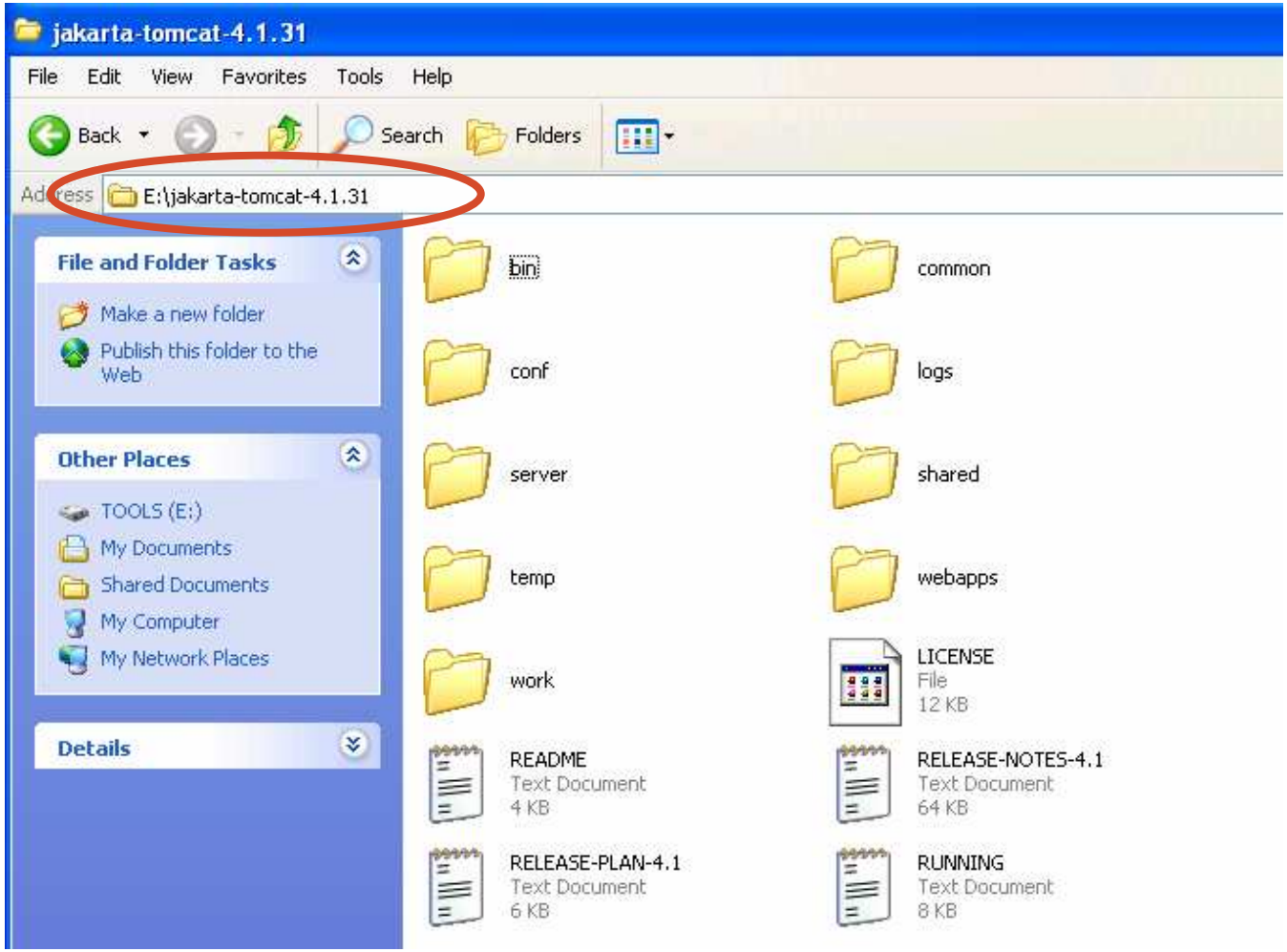
d) Destination Folder: → D:\Tomcat 4.1 or E:\Tomcat 4.1

Install → Next

e) HTTP/1.1 Connector Port: 8080

User name: admin → Finish

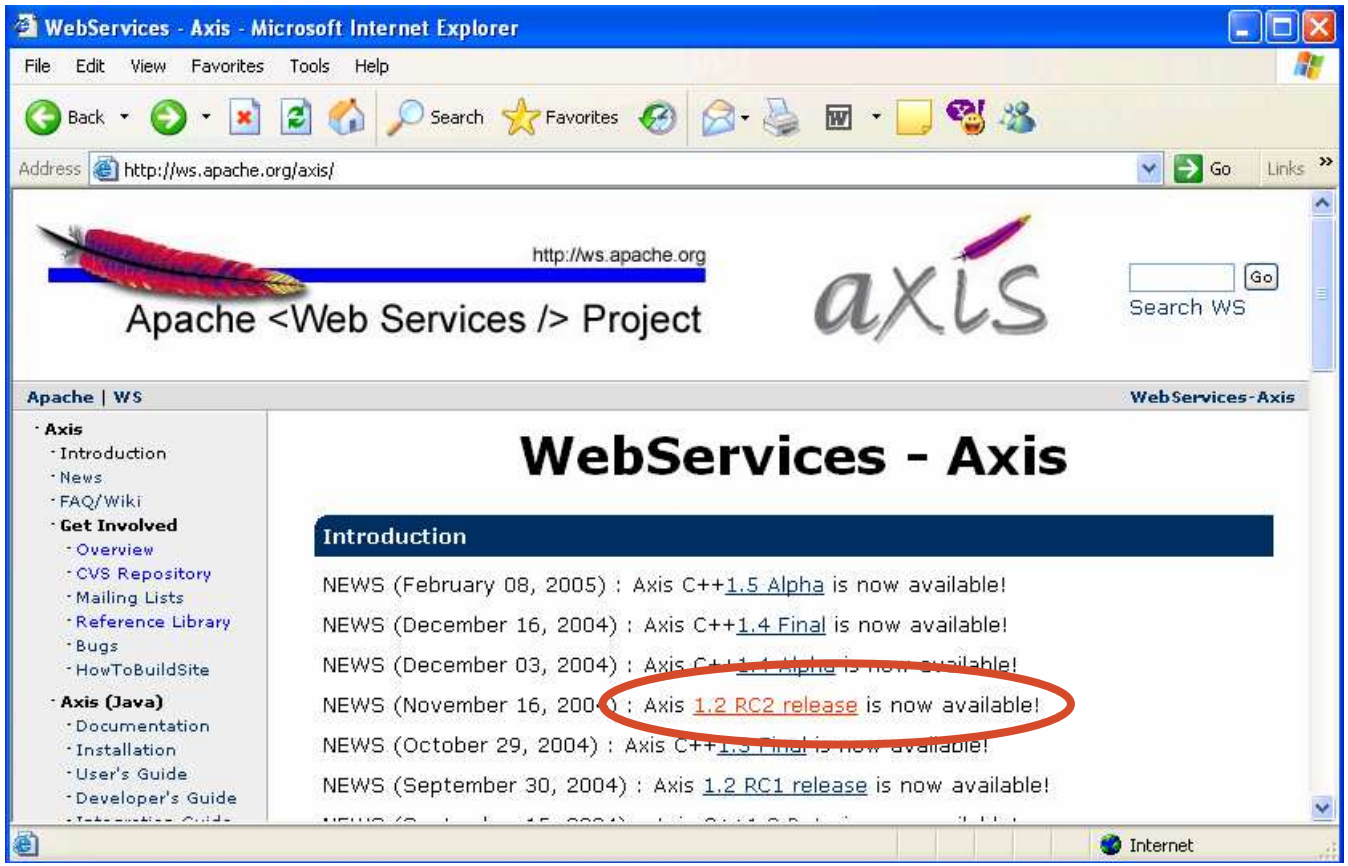
Task 6: Verifying Installation



Task 7: Installing Axis 1

1) visit http://ws.apache.org/axis/

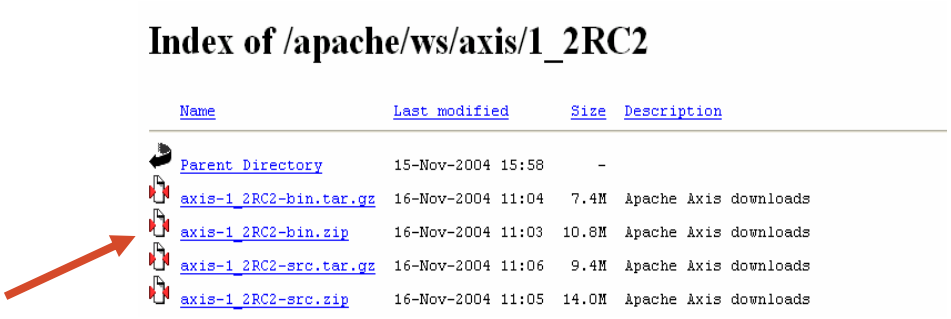
2) select Axis 1.2 RC2 release



Task 8: Installing Axis 2

3) select the Apache download mirrors

4) download the file: axis-1_2RC2-bin.zip to your own directory

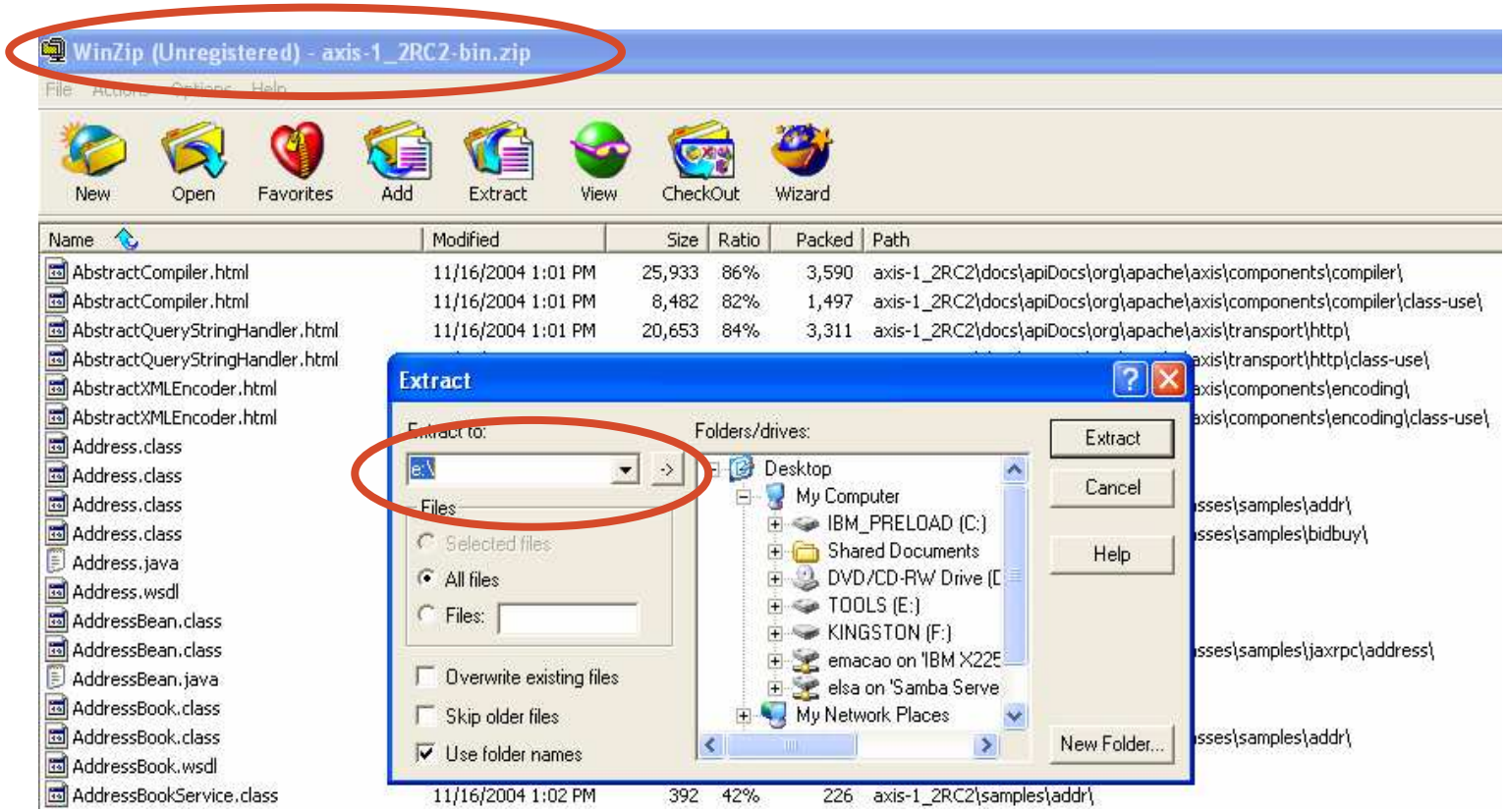


Index of /apache/ws/axis/1_2RC2

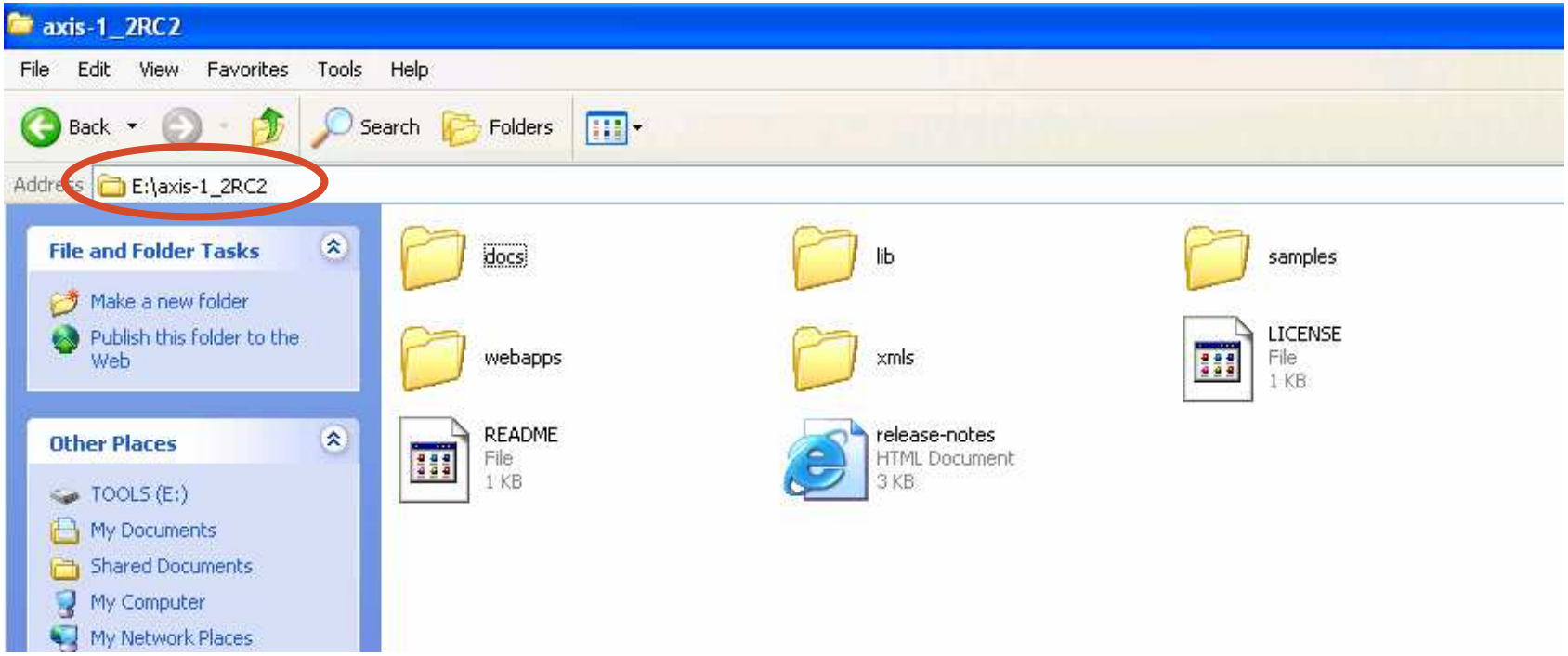
Name	Last modified	Size	Description
Parent Directory	15-Nov-2004 15:58	-	
axis-1_2RC2-bin.tar.gz	16-Nov-2004 11:04	7.4M	Apache Axis downloads
axis-1_2RC2-bin.zip	16-Nov-2004 11:03	10.8M	Apache Axis downloads
axis-1_2RC2-src.tar.gz	16-Nov-2004 11:06	9.4M	Apache Axis downloads
axis-1_2RC2-src.zip	16-Nov-2004 11:05	14.0M	Apache Axis downloads

Task 9: Installing Axis 3

5) uncompress: axis-1_2RC2-bin



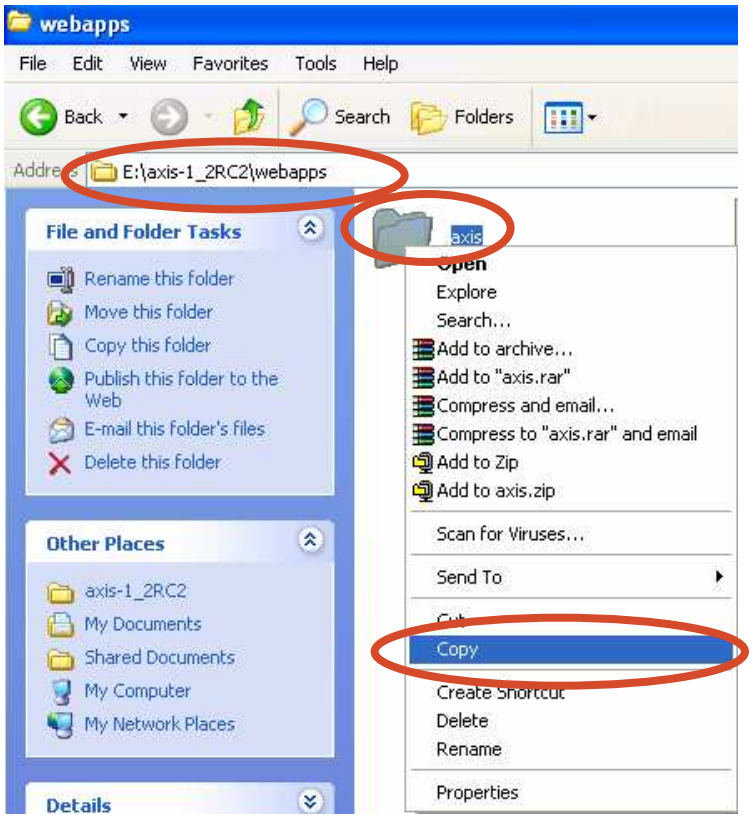
Task 10: Verify Installation

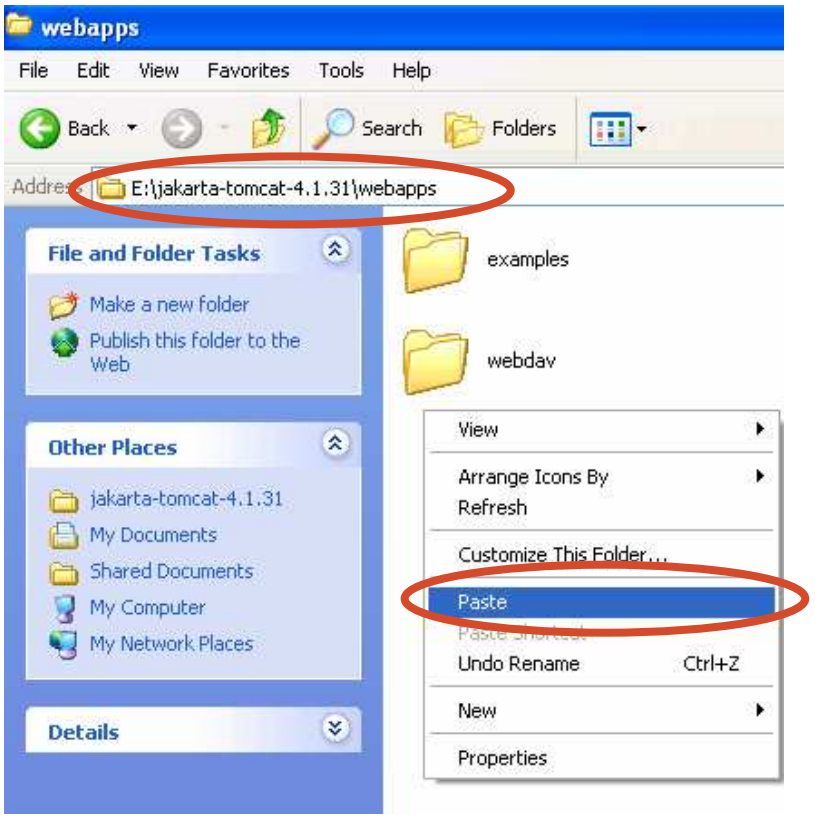


Task 11: Deploy Axis

In order to deploy Apache Axis in Tomcat:

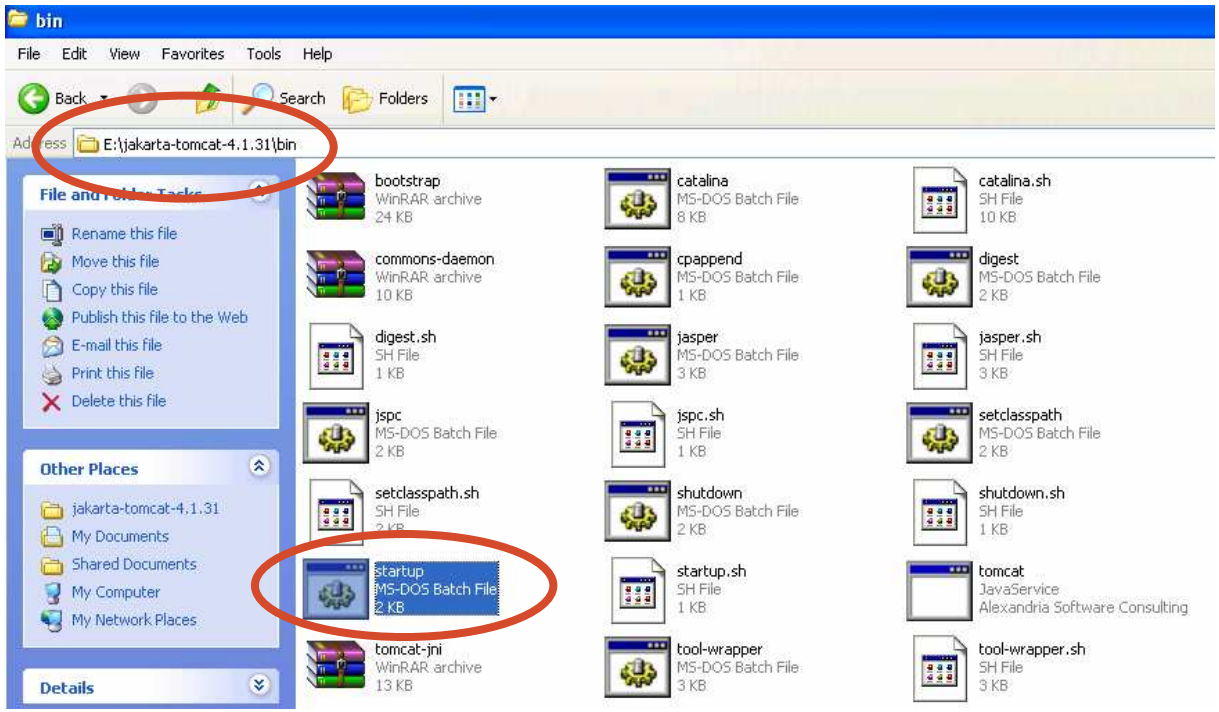
1) in E:\axis-1_2RC2\webapps, copy the folder: axis
2) in E:\jakarta-tomcat-4.1.31\webapps, paste the folder: axis





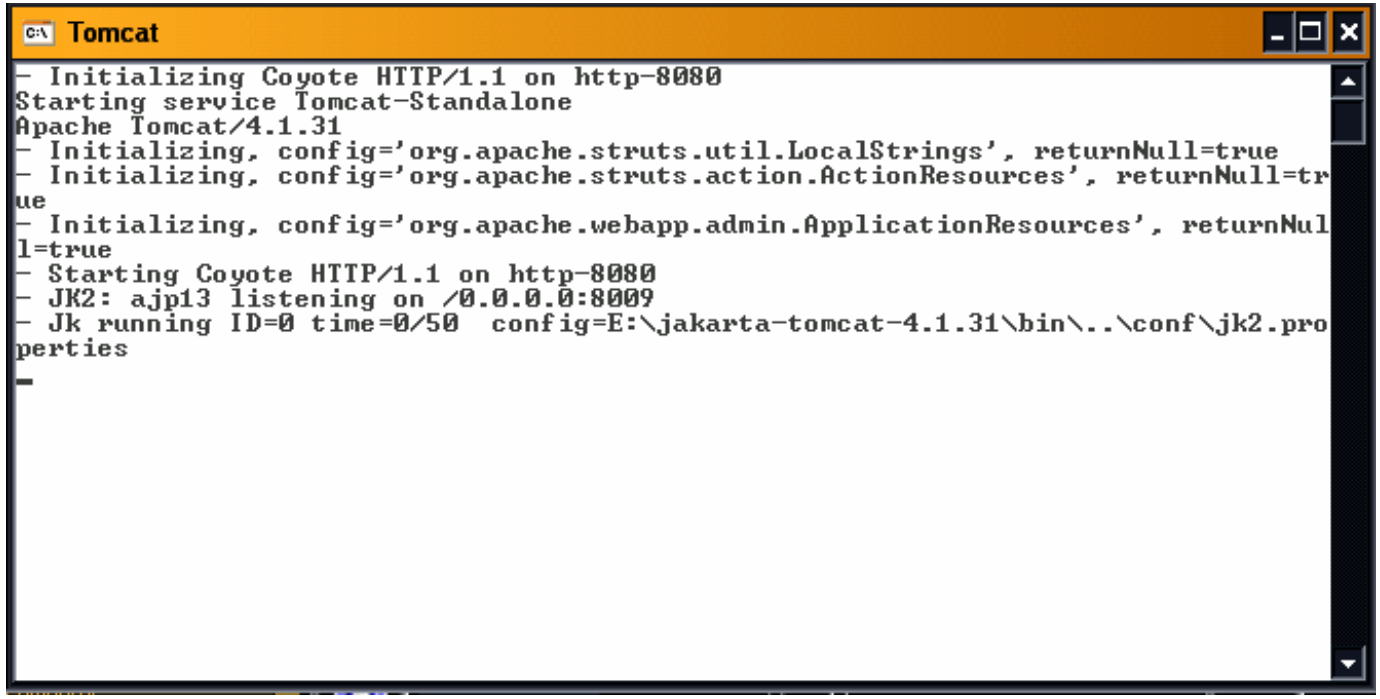
Task 12: Validate Installation 1

Start Tomcat - double-click E:\jakarta-tomcat-4.1.31\bin\startup.bat



Task 13: Validate Installation 2

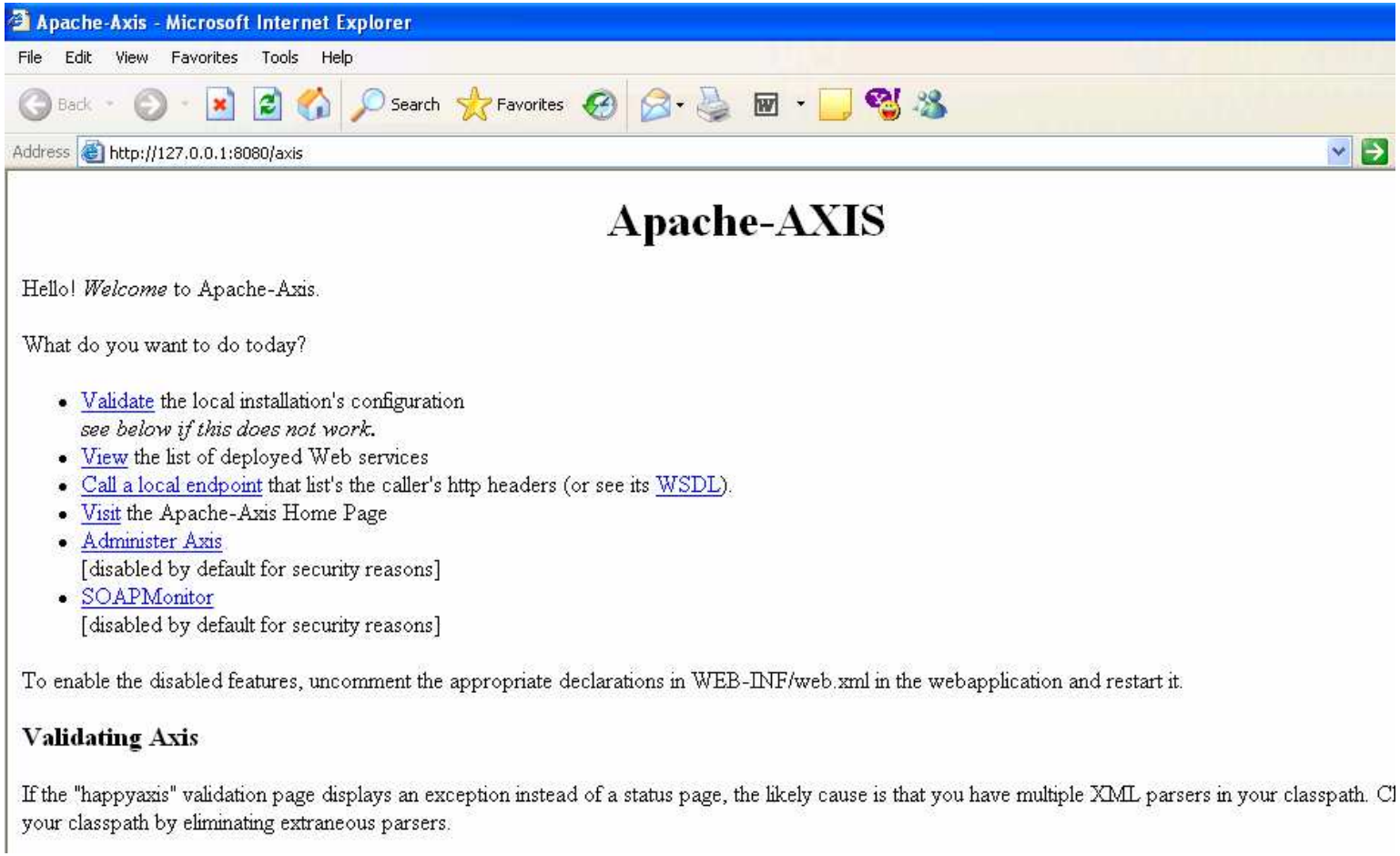
Starting Tomcat will cause the following window to appear:



(colors are inverted)

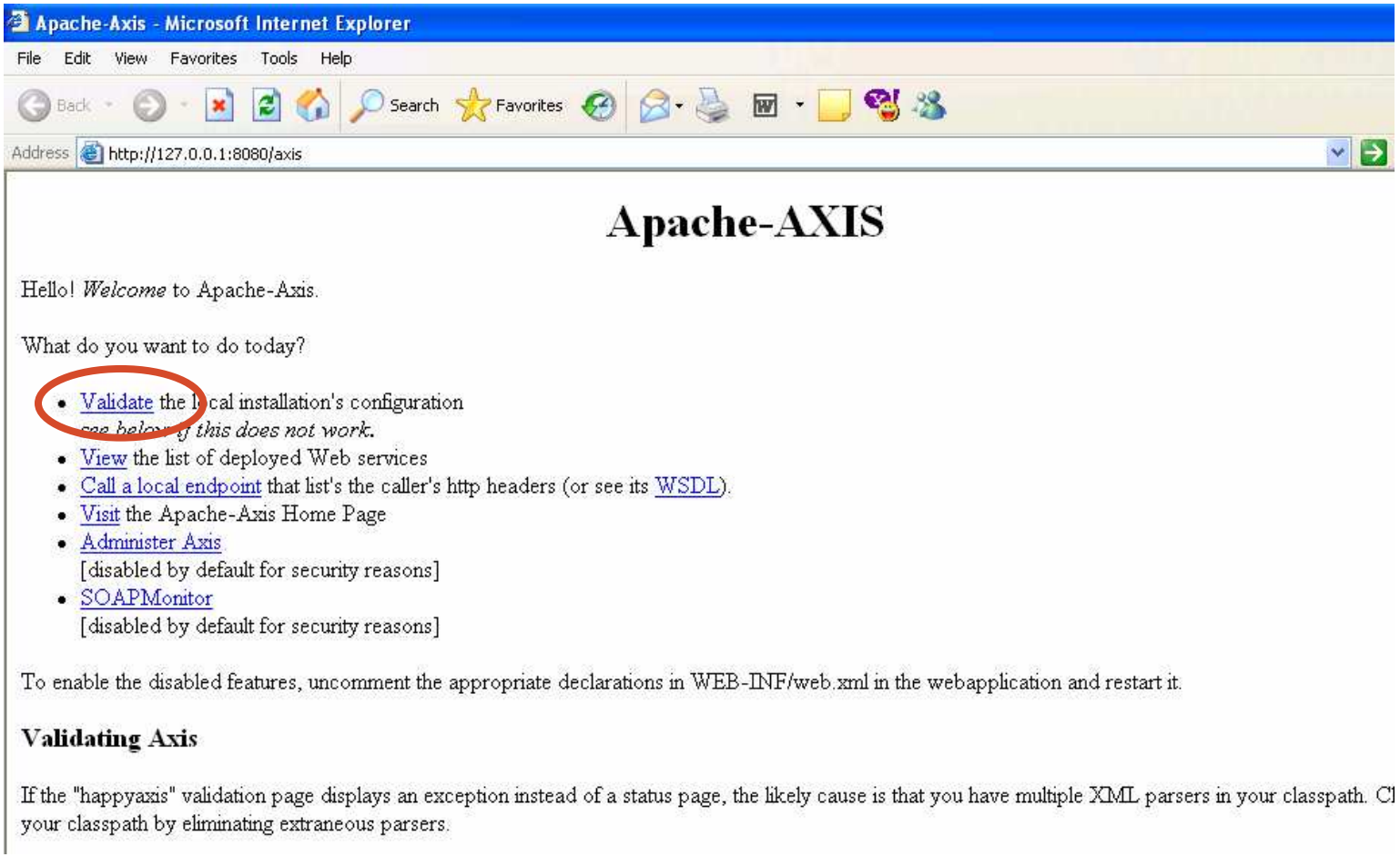
Task 14: Validate Installation 3

Navigate to the start page of the webapp http://localhost:8080/axis



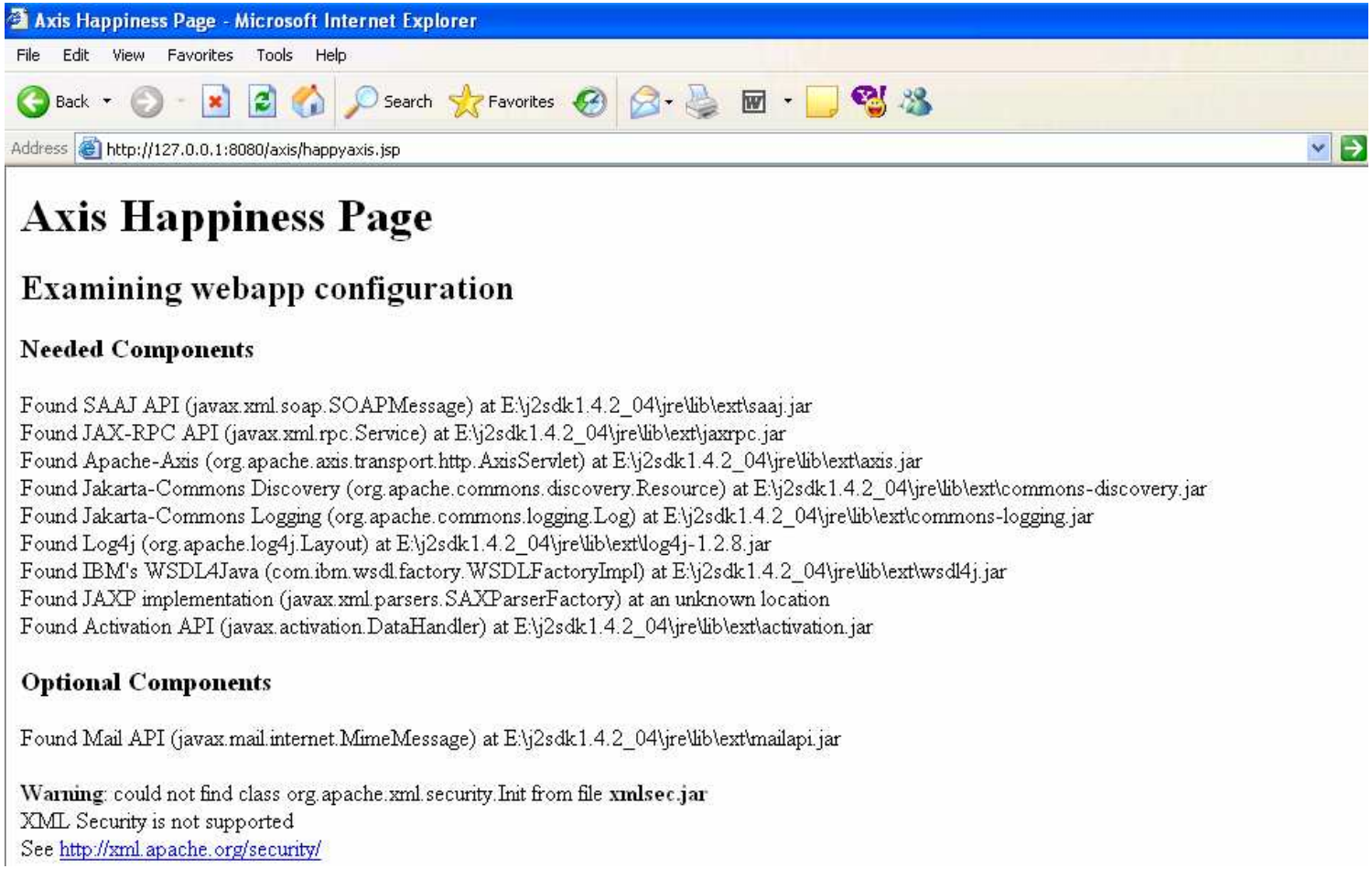
Task 15: Validate Installation 4

Validate Axis installation - follow the link Validate



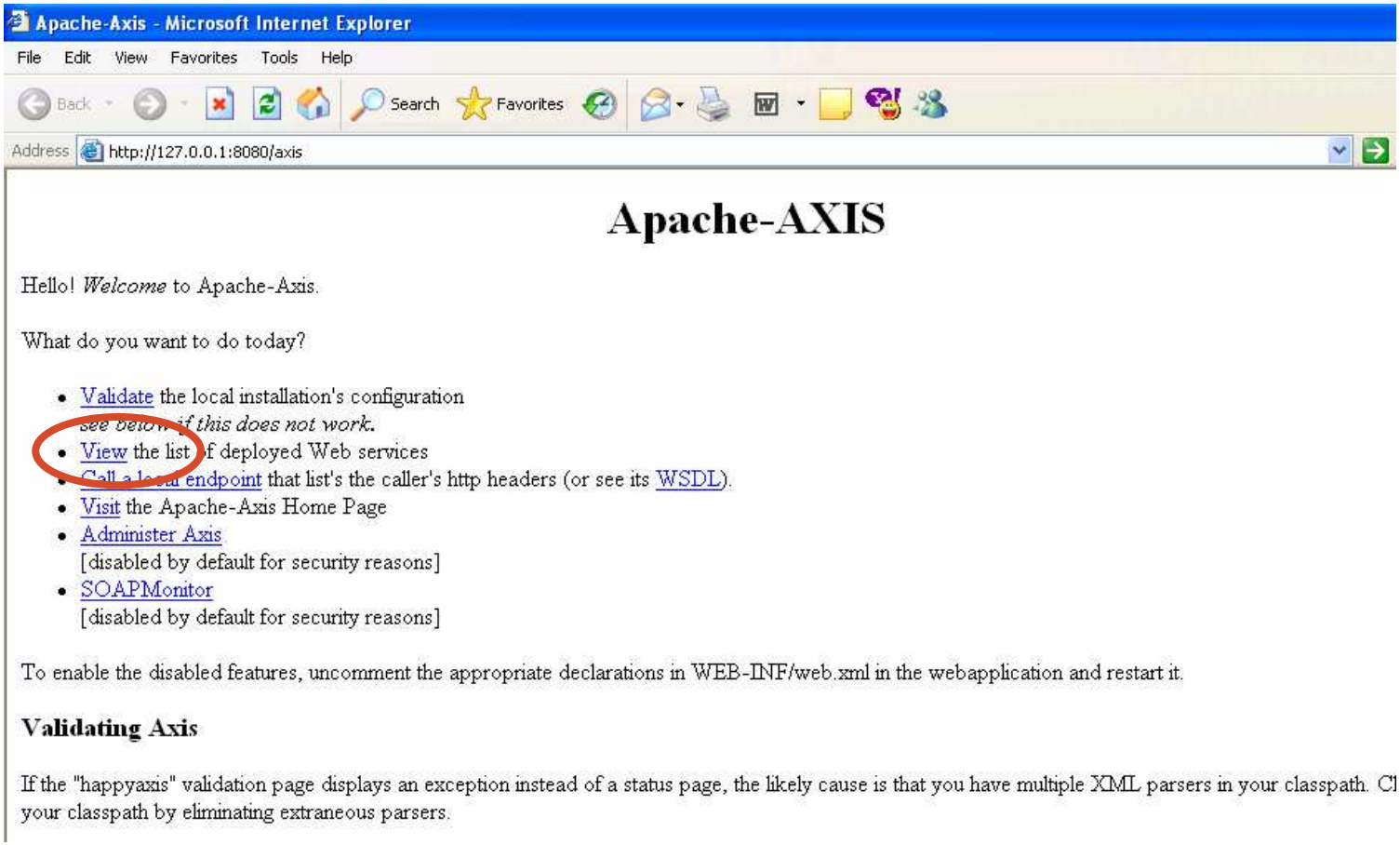
Task 16: Validate Installation 5

If the installation was successful then the following page is displayed:



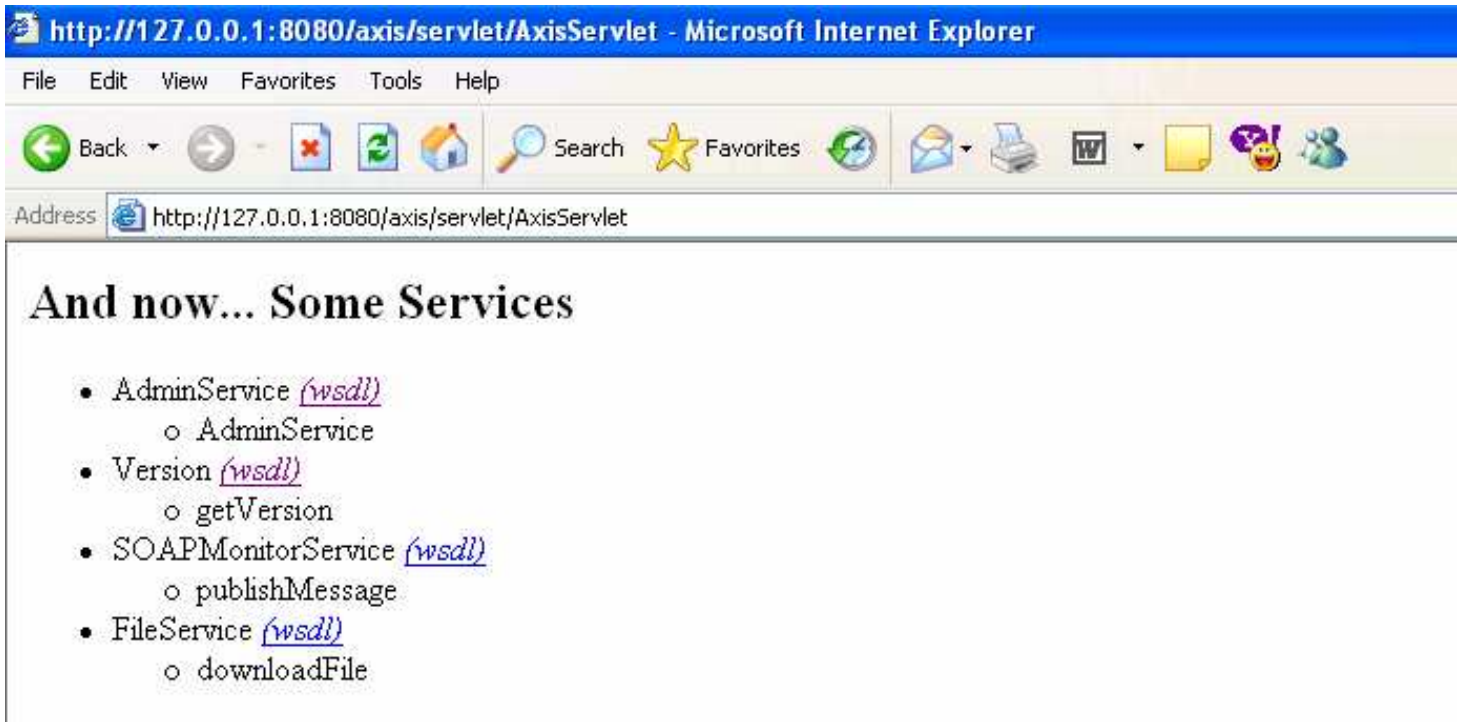
Task 17: Executing WS 1

Navigate to the start page of http://localhost:8080/axis and click on View to list the deployed web services:



Task 18: Executing WS 2

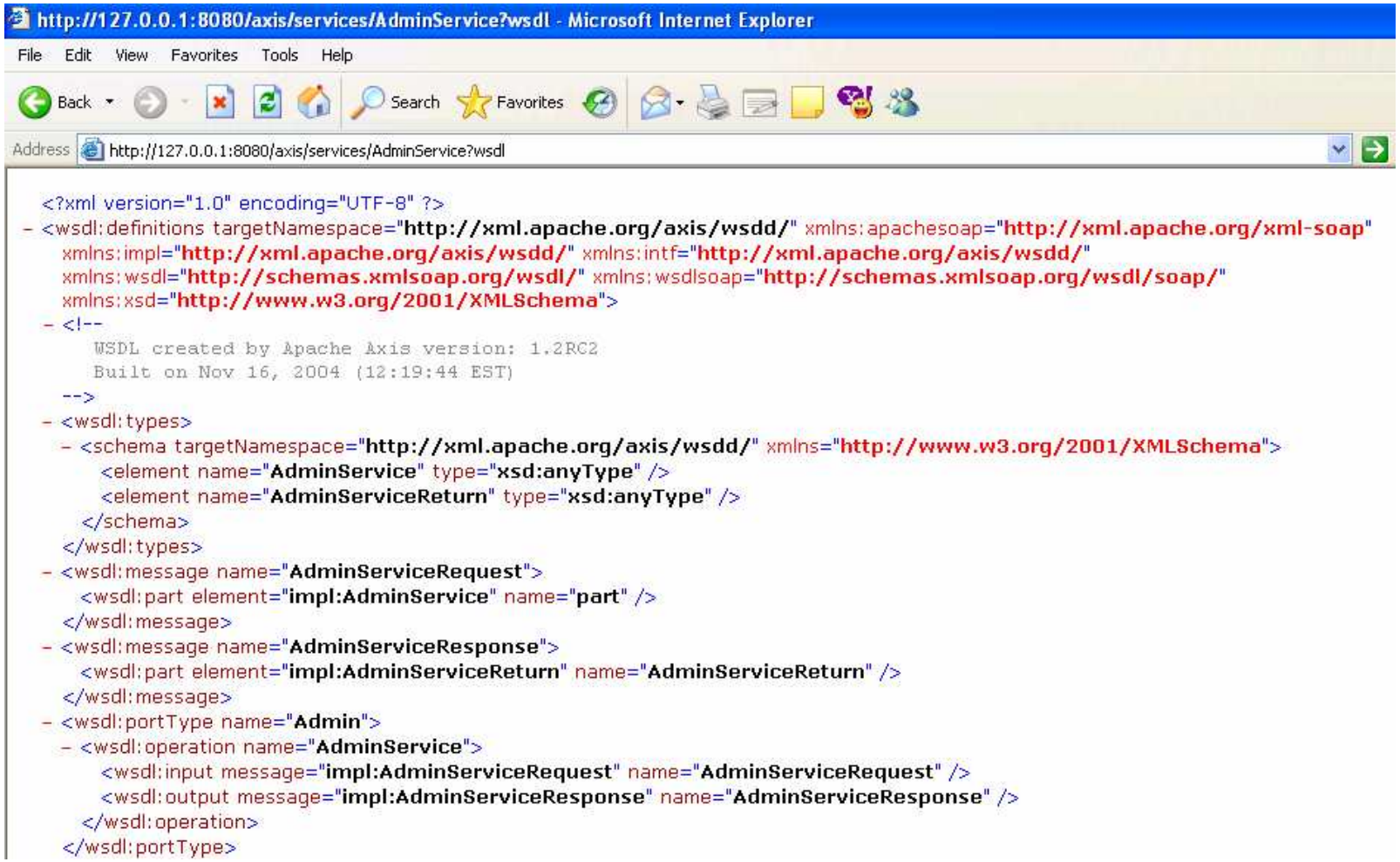
This page is displayed:



Click on AdminService (*wsdl*).

Task 19: Executing WS 3

Here is a web service description in WSDL:



Task 20: Testing WS 1

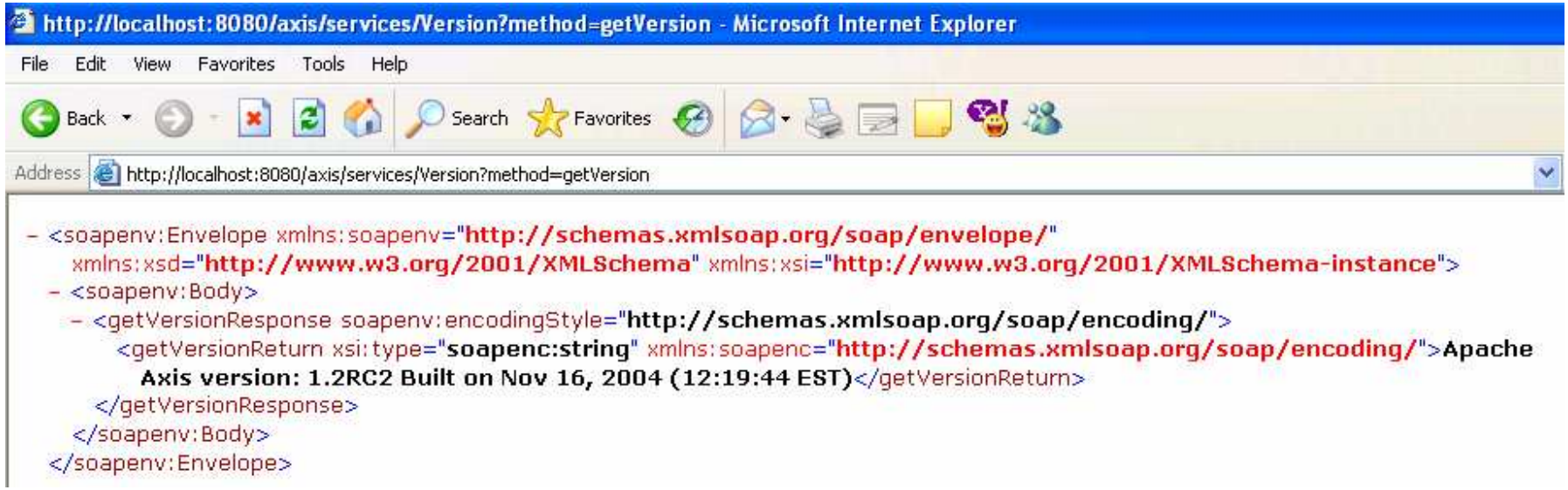
We are going to invoke the getVersion service which returns a message with the version number of the Axis installation.

Open the browser at

http://localhost:8080/axis/services/Version?method=getVersion

Task 21: Testing WS 2

In response, the following message is obtained:



A SOAP envelope!

Deploying a Web Service

Axis uses a deployment descriptor to deploy a web service.

A deployment descriptor is an Axis-specific XML file that tells Axis how to deploy (or undeploy) a web service, and how to configure Axis itself.

Deploying a web service:

1) copy the class that is being deployed as a web service to

```
\Tomcat 4.1\webapps\axis\WEB-INF\classes
```

2) write the deployment descriptor

3) run `AdminClient`

WS Deployment: Descriptor 1

To deploy a web service, the root of the XML deployment descriptor document must be the tag `<deployment>`.

The mandatory child of the `<deployment>` element is:

```
<service name="name" provider="provider">
        …
</service>
```

It is used to deploy/undeploy an Axis Service, where:

1) `name` - name of the web service

2) `provider` - specifies the particular provider of the web service such as: Java-RPC, Java-EJB, etc.

WS Deployment: Descriptor 2

The different options of the service may be specified as follows :

```
<parameter name="name" value="value"/>
```

and common ones include:

1) `className` - the backend implementation class

2) `allowedMethods` - each provider can determine which methods are allowed to be exposed as web services

WS Deployment: Descriptor 3

```
<deployment
   xmlns="http://xml.apache.org/axis/wsdd/”
   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="FileDownloadService" provider="java:RPC">
    <parameter name="className“ value="FileDownload"/>
    <parameter name="allowedMethods" value="*"/>
 </service>

</deployment>
```

WS Deployment: AdminClient

With Tomcat running, execute AdminClient:

```
java org.apache.axis.client.AdminClient deploy.wsdd
```

where:

1) `deploy.wsdd` is the name of the deployment descriptor.

2) AdminClient is a tool that comes with Axis that allows to deploy/undeploy web services and to configure the Axis engine.

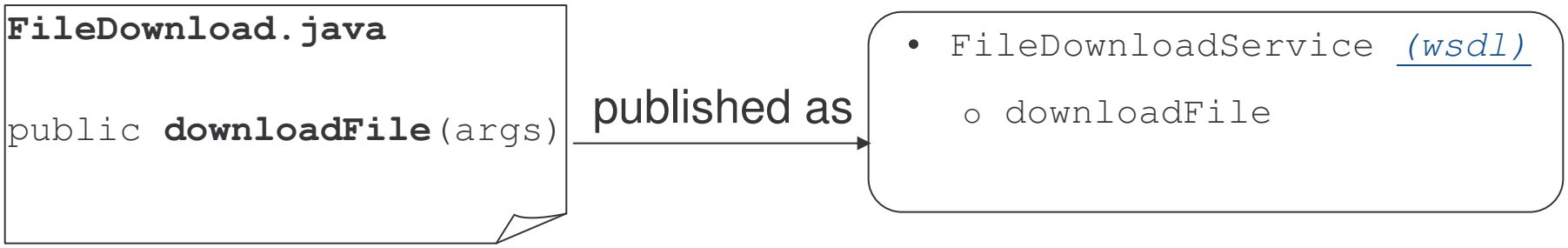
WS Deployment Example

We developed a web service that allows to download a file:

1) Java class is FileDownload.

2) The class has one method downloadFile that transmits a file.

3) The name of the web service is FileDownloadService.



Task 22: WS Deployment 1

1) change and check directory

```
> cd demos\WS\deployWebService
> dir
      deploymentDescriptor.wsdd
deployService.bat
FileDownload.class
```

2) Copy the Java class

`demos\WS\deployWebService\FileDownload.class`

to `\Tomcat 4.1\webapps\axis\WEB-INF\classes`

Task 23: WS Deployment 2

3) Edit the file `deploymentDescriptor.wsdd`:

   - changing `name_of_the_service` to `FileDownloadService`
   - changing `name_of_the_class` to `FileDownload`
   - and save it as `FileDownloadDescriptor.wsdd`

4) Edit the file `deployService.bat`

   - changing `deploymentdescriptor.wsdd` into `FileDownloadDescriptor.wsdd`
   - and save it as `deployFileDownload.bat`

5) Execute `deployFileDownload.bat`.

Task 24: WS Testing

1) view the list of the deployed web services at http://localhost:8080/axis
2) select View
3) click on the wsdl corresponding to `FileDownloadService`
4) this is the service description prepared by Axis for your service
5) in the address of the browser, remove `?wsdl` at the end of the line
6) this is the execution of your web service

Introduction Outline

1) Definitions
2) Service-Oriented Architecture
3) Web Services (WS)
4) Relating SOA and WS
5) WS Architecture Stack
6) Implementation Details
7) Summary

Introduction Summary 1

SOA is a Service-Oriented Architecture where software business processes are defined as services to be consumed over a network.

SOA defines three roles for agents: service provider, service requestor and service registry, and three operations: publish, find and bind.

Web Services (WS) is one approach to implementing SOA.

WS-based SOA produce a loosely coupled and flexible applications.

Introduction Summary 2

Web Services are software applications that use XML to exchange data with other applications.

Web Services provide a seamless integration framework of software, execution platforms and businesses.

Web Services use open standards technologies such as HTTP and SOAP for communication, WSDL for definition and UDDI for identification.

Introduction Summary 3

WS are build on three technologies:

1) SOAP - a protocol for exchanging structured information in a distributed system based on XML
2) WSDL - a language for describing web services
3) UDDI - a specification that defines a way to store and retrieve information about web services

Other purpose-focused specifications are available, such as WS-Security, WS-Reliable Messaging and others.

A system designer has to determine which specifications are needed for the system and implement or deploy them accordingly.

Introduction Summary 4

Axis is a SOAP engine, a free and open-source product from Apache.

Axis runs as an application of the Tomcat web server.

Axis provides a deployment descriptor that allows to deploy web services.

SOAP

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

SOAP History 1

1997	Microsoft considers supporting XML-based distributed computing consisting of applications communicating via RPC using standard data types on top of XML/HTTP.
1998	DevelopMentor (a Microsoft ally) and Userland (a private company) joined the discussion inventing the SOAP name. Within Microsoft, the process was stalled: some people promoted the DCOM wire protocol via HTTP tunneling, instead of pursuing XML.
1998	Userland publishes a version of the SOAP specification as XML-RPC.
1999	SOAP 1.0 appeares, entirely based on HTTP.

SOAP History 2

2000	SOAP 1.1 is submitted as a note to W3C with IBM as a coauthor – a more generic version including other protocols: 1) IBM immediately releases a Java SOAP implementation that was donated to the Apache XML Project for open source development. 2) Sun voices support to SOAP and started working on integrating Web Services into J2EE platform. 3) Many vendors also begin working on WS implementations.
2001	First draft of SOAP 1.2 is presented.
2003	SOAP 1.2 becomes a W3C recommendation.

SOAP Definition

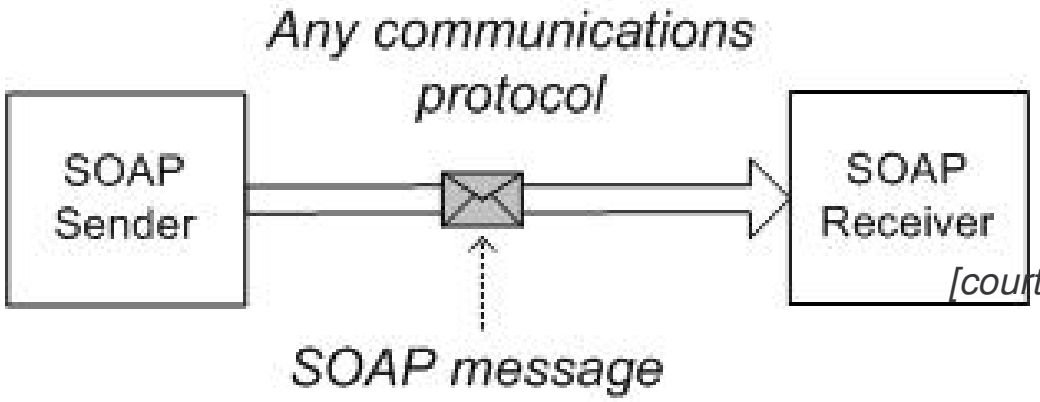
W3C (World Wide Web Consortium) definition:

- SOAP is a lightweight protocol intend for exchanging structured information in a decentralized, distributed environment.
- SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols.
- The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

[SOAP Version 1.2 Part1:Messaging Framework
http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/]

SOAP Features

SOAP defines a way to move XML messages from point A to point B:



[courtesy Aaron Skonnard]

It does this by providing an XML-based messaging framework that is:

1) extensible
2) usable over a variety of underlying networking protocols
3) independent of programming models

Features: Extensible

1) SOAP is simple by design

2) SOAP lacks various distributed system features:

   - security
   - routing
   - transactions
   - etc.

3) SOAP defines a communication framework that allows additional features to be added as layered extensions.

Features: Protocol-Independent

4) SOAP can be used over any protocol:

   - TCP
   - HTTP
   - SMTP
   - etc.

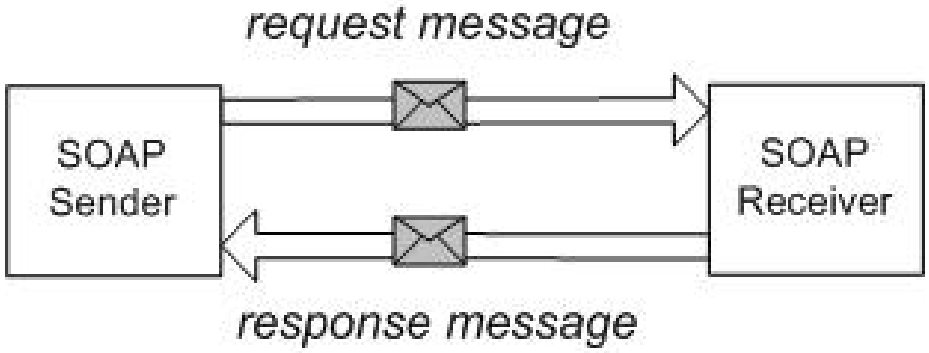
5) SOAP provides a flexible framework for defining bindings to arbitrary protocols to maintain interoperability.

6) SOAP provides an explicit binding for HTTP.

Features: Model-Independent

SOAP is independent of any distributed programming model:

7) allows for any programming model not tied to RPC
8) defines a model for processing individual, one-way messages, or combine multiple messages into an overall message exchange



[courtesy Aaron Skonnard]

9) allows for any number of message exchange patterns: request/response, solicit/response, notifications, peer-to-peer

SOAP and ebXML

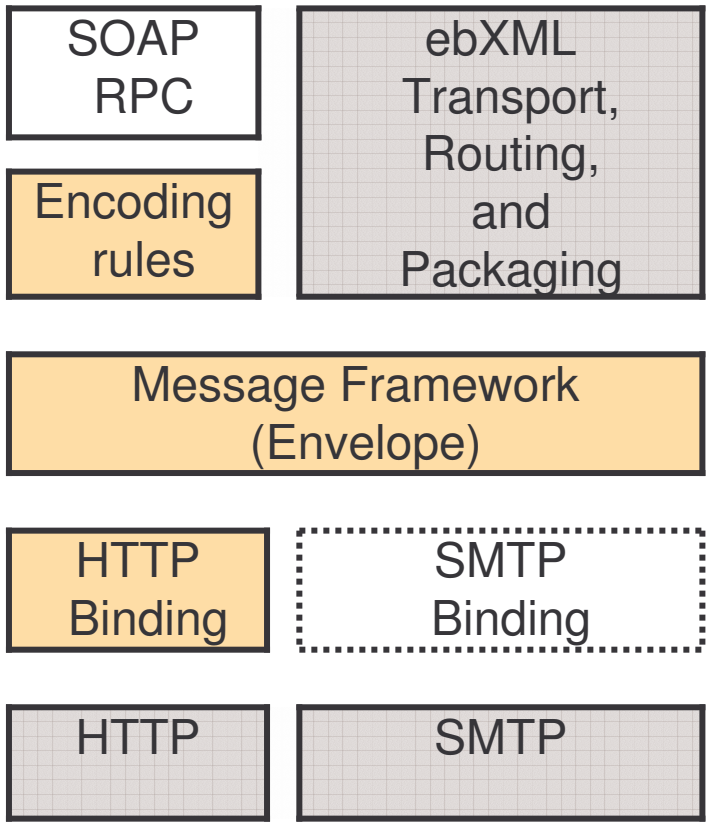
SOAP:

1) messaging framework
2) encoding rules
3) binding to HTTP protocol

ebXML:

1) messaging framework
2) SOAP bindings
3) own encoding rules

e-Commerce solutions with XML.

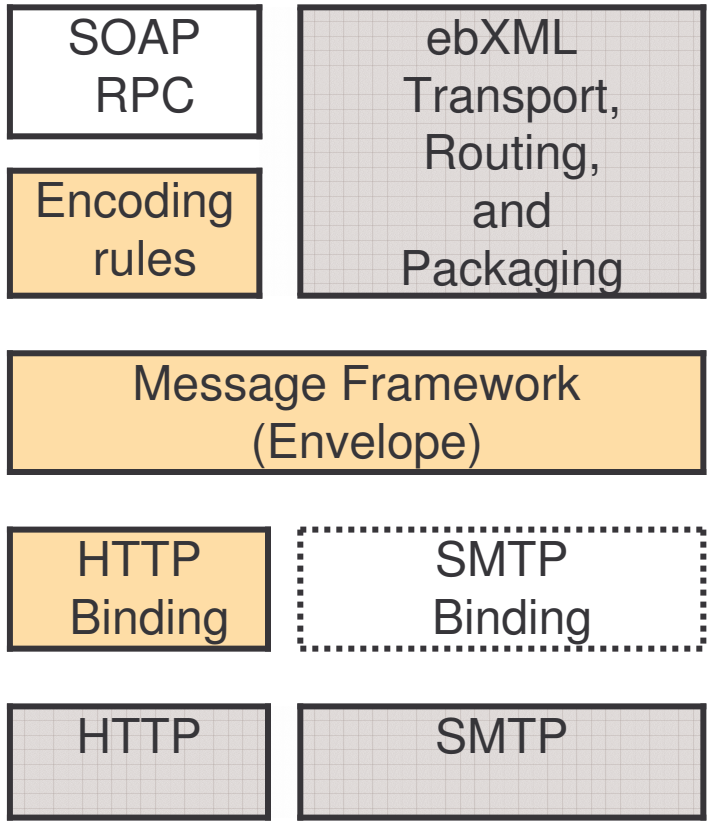


SOAP Toolbox

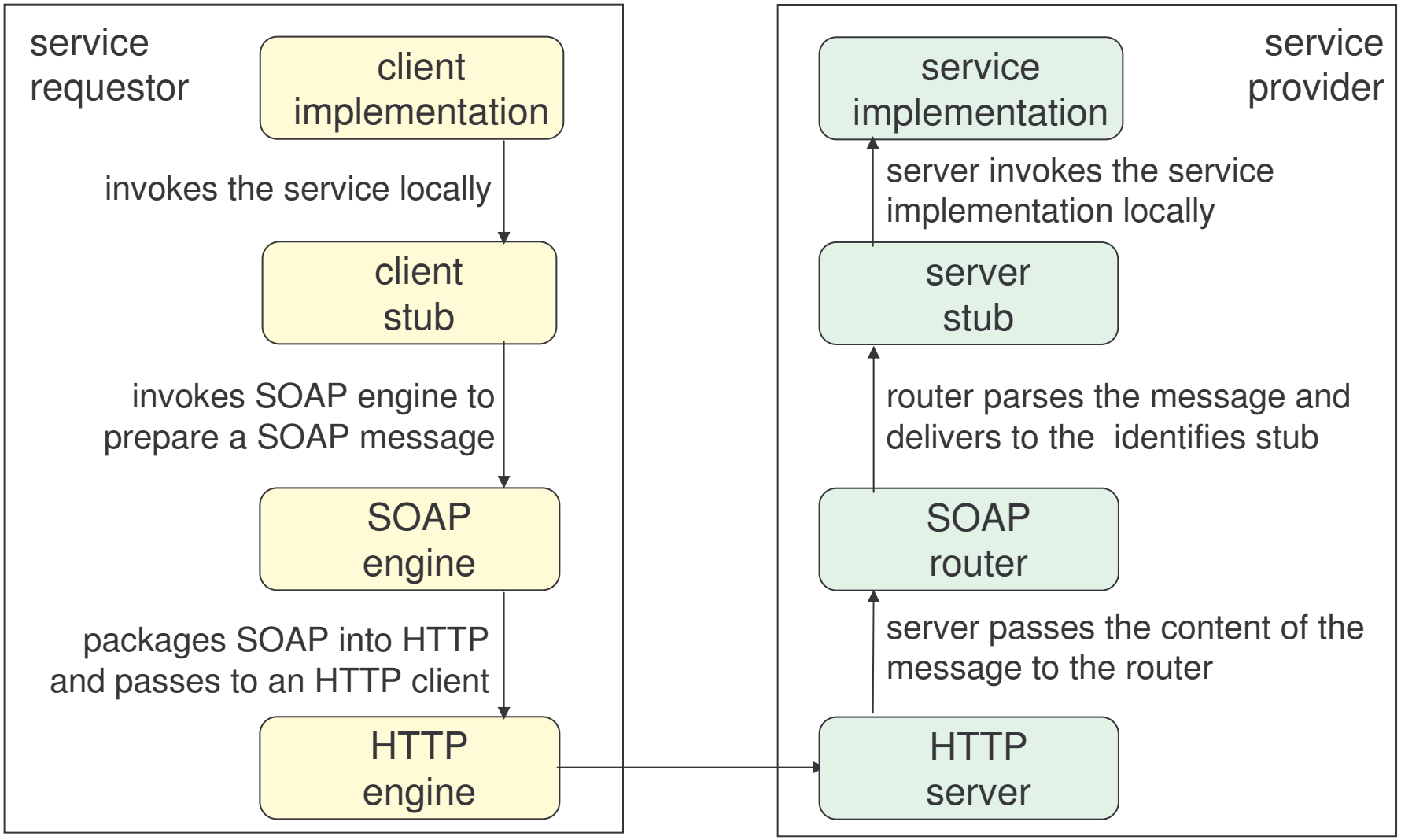
SOAP is like a toolbox.

SOAP requests could be made:

- with the SOAP envelope, HTTP binding and some encoding or
- with the SOAP envelope, SOAP encoding and SMTP or
- any other combinations



SOAP Implementation Model



SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

SOAP Message

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <soapenv:Body>
   <getVersionResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <getVersionReturn xsi:type="soapenc:string"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
         Apache Axis version: 1.2RC2 Built on Nov 16, 2004 …
      </getVersionReturn>
   </getVersionResponse>
 </soapenv:Body>

</soapenv:Envelope>
```

An XML document!

SOAP Messaging

The SOAP messaging framework defines a suite of XML elements for "packaging" arbitrary XML messages for transport between systems:

1) envelope
2) header
3) body
4) fault
5) etc.

SOAP Namespaces

All XML elements belong to the following namespaces:

1) SOAP 1.1 - http://schemas.xmlsoap.org/soap/envelope
2) SOAP 1.2 - http://www.w3.org/2003/05/soap-envelope

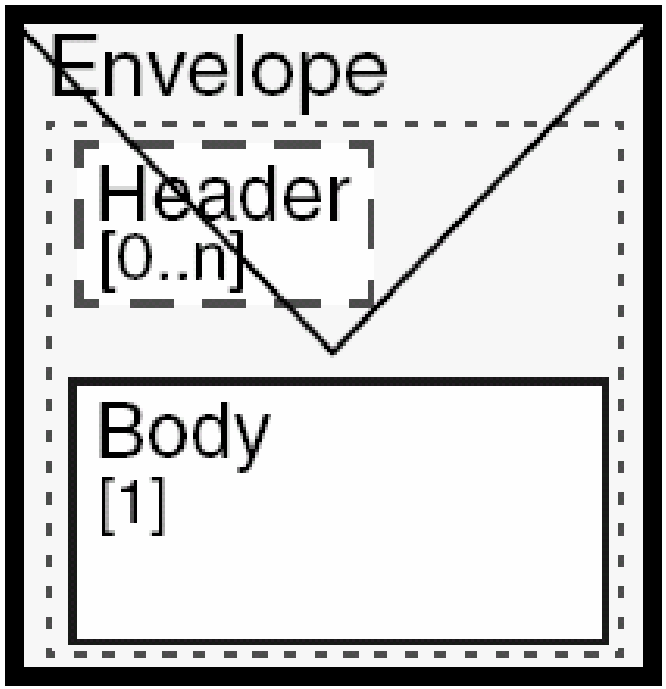
SOAP Envelope 1

A SOAP message is an envelope with zero or more headers and one body:

1) envelope is a container for control information, recipient address and the message itself

2) headers contain control information

3) body contains the message information



[courtesy IBM]

SOAP Envelope 2

`Envelop` is always the root element of a SOAP message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope /">
  <soap:Header>...</soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

The namespace is specified in the envelope for:

1) defining the envelope elements
2) controlling the SOAP version

Additional namespaces may be defined as well.

SOAP Header

`Header` is a generic place-holder for application independent information.

A header:

1) provides a mechanism for extending SOAP messages in a decentralized and modular way

2) allows to pass control information to the receiving SOAP server

SOAP Header: Example

This header introduces a namespace and two elements:

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

  <soap:Header>
       <t:transaction xmlns:t="http://example.org/transac">
              <t:loginTime>10:20:00</t:loginTime>
         <t:logoutTime>10:21:00</t:logoutTime>
     </t:transaction>
  </soap:Header>

 ...

</soap:Envelope>
```

SOAP Header Attributes

SOAP 1.2 provides mechanisms to specify who should deal with headers and what to do with them.

For this purpose it includes attributes:

1) `role`
2) `mustUnderstand`
3) `relay`

Also it is possible to define:

4) `encodingStyle`

SOAP 1.1 has `actor` attribute instead of `role`, with the same semantic.

Mandatory/Optional Headers

Headers may be mandatory or optional.

If a header is mandatory:

1) the receiver must process the header

2) if the receiver is unable to process the header, it must fail

`mustUnderstand` attribute indicates if a header is mandatory or optional.

SOAP Body

The SOAP `Body` element represents a mechanism for exchanging information intended for the ultimate recipient of the message.

`Body` represents the message payload – a generic container that includes any number of elements from any namespace.

In the simplest case the body of a SOAP message includes:

- message name
- reference to a service instance
- parameters with values and optional type references

SOAP Body: Request Example

Request message to transfer funds between bank accounts:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <x:TransferFunds xmlns:x="urn:examples-org:banking">
      <x:from>983-23456</x:from>
      <x:to>672-24806</x:to>
      <x:amount>1000.00</x:amount>
    </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

SOAP Body: Response Example

Response message send back to the sender:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
     <x:TransferFundsResponse xmlns:x="urn:examples-org:banking">
      <x:balances>
        <x:account>
          <x:id>983-23456</x:id>
          <x:balance>34.98</x:balance>
        </x:account>
        <x:account>
          <x:id>672-24806</x:id>
          <x:balance>1267.14</x:balance>
        </x:account>
      </x:balances>
     </x:TransferFundsResponse>
  </soap:Body>
</soap:Envelope>
```

Task 25: Sending Request 1

Objective:

Send a SOAP message to `FileDownloadService` deployed in the Introduction, asking to download a file.

The request message contains:

1) a header - contains user name and password
2) body - contains the method invocation

The client application has two command-line parameters:

1) the path to the downloaded file
2) the name to save this file on the client machine

Task 26: Sending Request 2

```
> cd demos\SOAP\Request

> dir
FileTransferRequest.class
MacaoNews.txt

> mkdir E:\WebServices

> copy MacaoNews.txt to \WebServices

> java -cp \demos\SOAP\Request FileTransferRequest
     E:\WebServices\MacaoNews.txt
     news.txt
```

Task 27: Sending Request 3

Based on the request message:

1) what is the SOAP version?
2) what is the structure of the header?
3) what is the user name?
4) what is the password?
5) what is the structure of the body?

```
> dir
```

Task 28: Receiving Response

Objective: receive a response to the message sent.

```
> cd demos\SOAP\Response

> dir
FileTransferResponse.class

> java -cp \demos\SOAP\Response FileTransferResponse
      \WebServices\MacaoNews.txt
      news.txt
```

SOAP Fault

The `Fault` element is used to represent errors:

1) processing errors
2) errors understanding a mandatory header
3) all abnormal situations

Faults are specified within the body of a SOAP message.

SOAP Fault: Example

Response message with the `Insufficient funds` error:

```
<soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
                <soap:Fault>
                <soap:Code>
                        <soap:Value>soap:Sender</soap:Value>
                </soap:Code>
                <soap:Reason>Insufficient funds</soap:Reason>
                <soap:Detail>
                <x:TransferError xmlns:x="urn:examples-
 org:banking">
                        <x:sourceAccount>22-342439</x:sourceAccount>
                        <x:transferAmount>100.00</x:transferAmount>
                        <x:currentBalance>89.23</x:currentBalance>
                </x:TransferError>
                </soap:Detail>
        </soap:Fault>
        </soap:Body>
</soap:Envelope>
```

Task 29: Generating a Fault 1

Objective:

Generate a fault message when looking for the service “FileService” instead of “FileDownloadService”.

```
> cd demos\SOAP\Fault


> dir
FileTransferResponse.class


> java -cp \demos\Soap\Fault FileTransferResponse
           \WebServices\Macao.txt
           news.txt
```

Task 30: Generating a Fault 2

Based on the response message:

1) Where is the fault element located?
2) What is the structure of the fault element?

Extending SOAP: Wrong Way

Suppose we want to add authentication information to the message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <x:TransferFunds xmlns:x="urn:examples-org:banking">
      <x:from>983-23456</x:from>
      <x:to>672-24806</x:to>
      <x:amount>1000.00</x:amount>
      <credentials>
                <username>dave</username>
                <password>evad</password>
            </credentials>
    </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

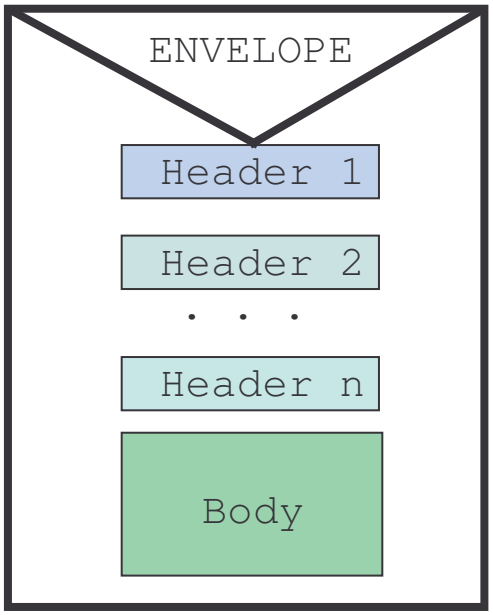
Not the right way: other applications in need of security must develop their own solutions to the problem. Ultimately, interoperability suffers.

Extending SOAP: Right Way

An envelope wraps whatever XML content is sent in a message.

The header is used to insert message extensions without modifying its body.

Each individual header represents one piece of extensibility information that travels with the message.



Use of Headers

Headers can contain any kind of data.

They are generally used to:

1) extend the messaging infrastructure:
   a) infrastructure headers are processed by middleware
   b) the application does not see the headers, only their effects
   c) examples: security credentials, reliable messaging, etc.

2) define additional data:
   a) these headers are defined by the application
   b) called: vertical extensibility
   c) for instance extra-data to accompany non-extensible schemas

Headers for Extensions

For common needs such as security, it makes more sense to define standard SOAP headers that everyone agrees on.

Then, vendors can build support for the extended functionality into their generic SOAP infrastructure and everyone wins.

```
<soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
        <s:credentials xmlns:s="urn:examples-org:security">
                <s:username>dave</s:username>
                <s:password>evad</s:password>
        </s:credentials>
  </soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

mustUnderstand Attribute

A global SOAP attribute `mustUnderstand` indicates whether or not a receiver is required to understand the header block before processing.

For example:

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
     <s:credentials
                xmlns:s="urn:examples-org:security"
          soap:mustUnderstand="1">
                ...
     </s:credentials>
       </soap:Header>
 ...
</soap:Envelope>
```

mustUnderstand Values

Two values:

1) `mustUnderstand="1"` – if a receiver cannot support the header, a fault should be returned with `soap:mustUnderstand` status code.

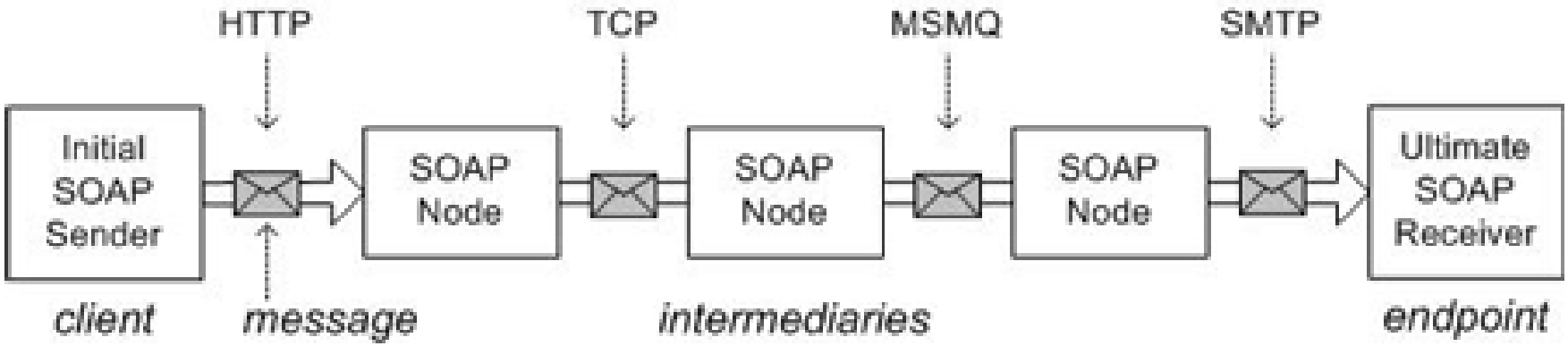
2) `mustUnderstand="0"` or `mustUnderstand` attribute is absent, the receiver can ignore those headers and continue processing.

It may also have values `false` (`0`) and `true` (`1`).

SOAP Processing Model

SOAP defines a processing model that outlines rules for processing a SOAP message as it travels from a SOAP sender to a SOAP receiver.

The model allows for architectures with multiple intermediary nodes:



[courtesy Aaron Skonnard]

SOAP Nodes

A SOAP node can be:

1) initial SOAP sender
2) ultimate SOAP receiver
3) SOAP intermediary

SOAP Intermediaries

SOAP intermediaries:

1) they are applications that can process parts of a SOAP message as it travels from its origination point to its final destination point

2) can accept and forward SOAP messages, and usually they do carry out some form of message processing

The route taken by a SOAP message, including all intermediaries it passes through, is called the SOAP message path.

Reasons for Intermediaries 1

There are three major reasons for using intermediaries:

1) crossing-trust domains

   nodes that allow some requests to cross the trust domain boundary and deny access to others

2) ensuring scalability

   a) nodes that provide flexible buffering and routing of messages based on message parameters
   b) nodes that provide information about network traffic and the availability and load of network nodes

Reasons for Intermediaries 2

3) providing special services:

a) encrypting and digitally signing a message, or decrypting and checking the digital signature

b) making a persistent copy of the request message, providing a token that can be used to reference the transaction in the future (notarization or non-repudiation)

c) enabling to find out the path that the message has followed, with arrival and departure times to and from intermediaries

Intermediaries: Sender-View

Message senders may or may not be aware of intermediaries:

1) transparent intermediary - the client knows nothing about it, it believes the message is sent to the service end-point.

   The security intermediary would likely be transparent.

2) explicit intermediary – it involves specific knowledge on the part of the client. The client knows the message will pass through the intermediary.

   The notarization intermediary would likely be explicit.

Intermediaries: Process-View

Intermediaries also differ with respect to processing:

1) forwarding intermediaries - nodes doing specific processing based on the contents of the incoming message.

   For instance a notarization node making a copy of the message based on what is defined on a particular header.

2) active intermediaries - nodes doing processing and eventually modifying the message in the ways not defined by the message contents.

   For instance a node at a boundary of a company to the outside world adding digital signatures to all outbound messages.

Nodes and Roles

While processing a message a SOAP node assumes one or more roles that influence how the headers are processed.

A SOAP node has its role declared.

When it receives a message for processing:

1) it must process all mandatory headers targeted at one of its roles
2) it may process any optional headers targeted at one of its roles

Role Attribute

The `role` attribute is defined optionally in the header element:

1) The value of `role` is a URI that identifies the name the intermediary who should handle the header entry.

2) The URI might mean:

   a) a particular node - the server `XY` or
   b) a class of nodes - any cache manager along the message path

3) A node can play multiple roles, e.g. `XY` server as a cache manager.

Predefined Roles

SOAP defines three special values for `role`:

1) `http://www/w3.org/2002/06/soap-envelope/role/next`

   Each SOAP intermediary and the ultimate SOAP receiver must act in this role and MAY additionally assume zero or more other roles.

2) `http://www/w3.org/2002/06/soap-envelope/role/ultimateReceiver`

   The final recipient of the SOAP message - processes the body. The end-receiver must act in this role. Intermediaries must not act in this role.

3) `http://www/w3.org/2002/06/soap-envelope/role/none`

   SOAP nodes must not act on this role. Headers addressed to this role should never be processed. They are used to carry data.

Roles: Example 1

A SOAP message with roles:

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

Mandatory header targeted at a SOAP node that plays the `http://example.org/security` role:

```
   <a:Security
      xmlns:a="http://example.com"
      soap:role="http://example.com/security"
      soap:mustUnderstand="true" >
      …
   </a:Security>
```

Roles: Example 2

Optional header targeted at the next node in the message path:

```
<b:NextExample xmlns:b="http://example.com"
    soap:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    soap:mustUnderstand="false" >
    …
</b:NextExample>
```

Header targeted at a SOAP node with the `ultimateReceiver` role:

```
<c:NoRoleDef xmlns:c="http://example.com">
    …
</c:NoRoleDef>

</soap:Header>

…
</soap:Envelope>
```

Processing Rules

The contract implied by a header is between the sender and the first node satisfying the role at which it is targeted.

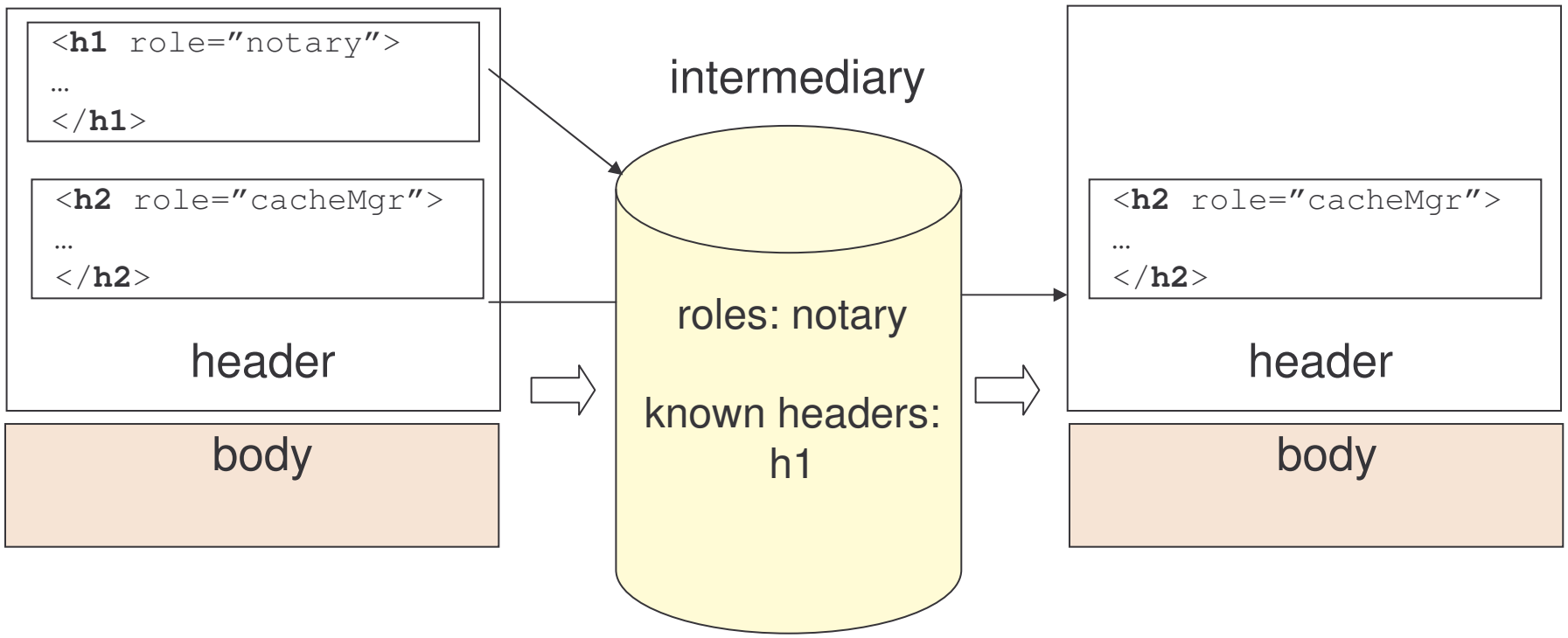
Two rules:

1) if a SOAP node successfully processes a header, it is required to remove the header from the message

2) if the SOAP node happens to be the ultimate receiver, it must also process the SOAP body.

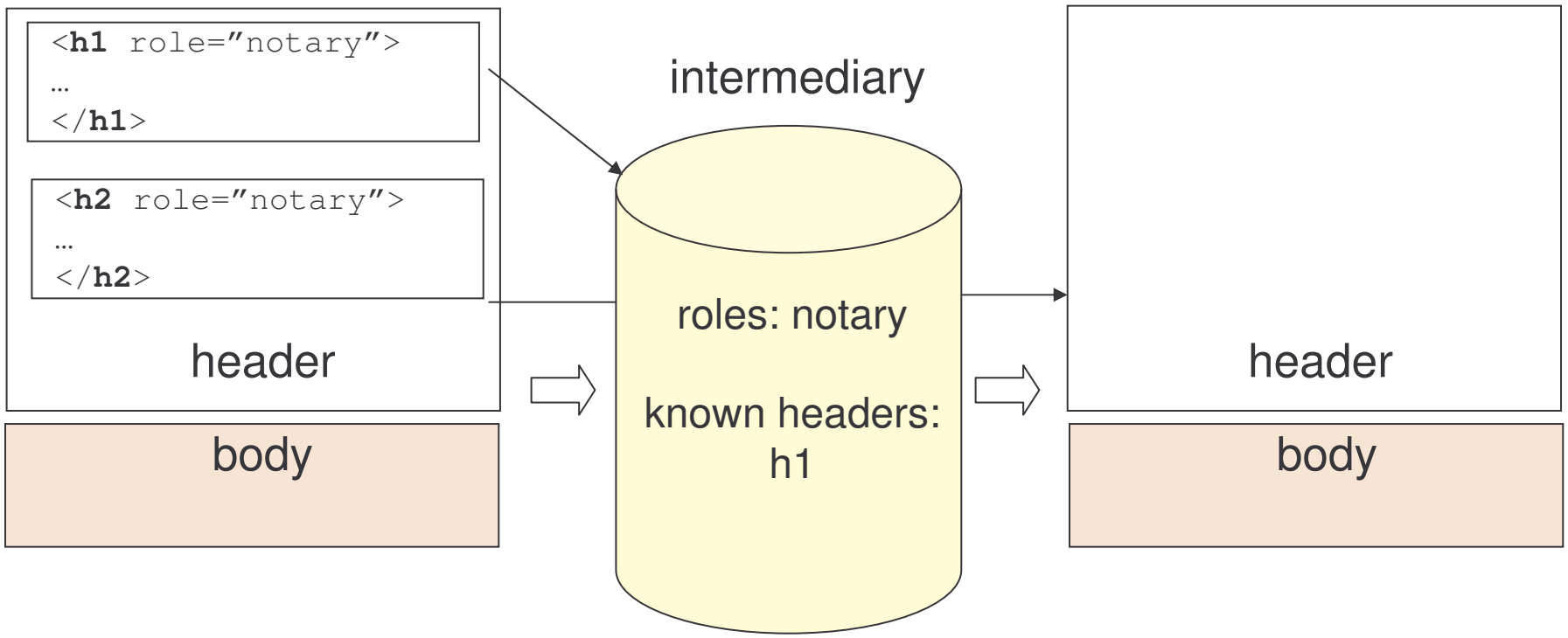
SOAP nodes are allowed to reinsert headers, but doing so changes the contract parties – it's now between the current and the next node.

Processing Rules: Example 1



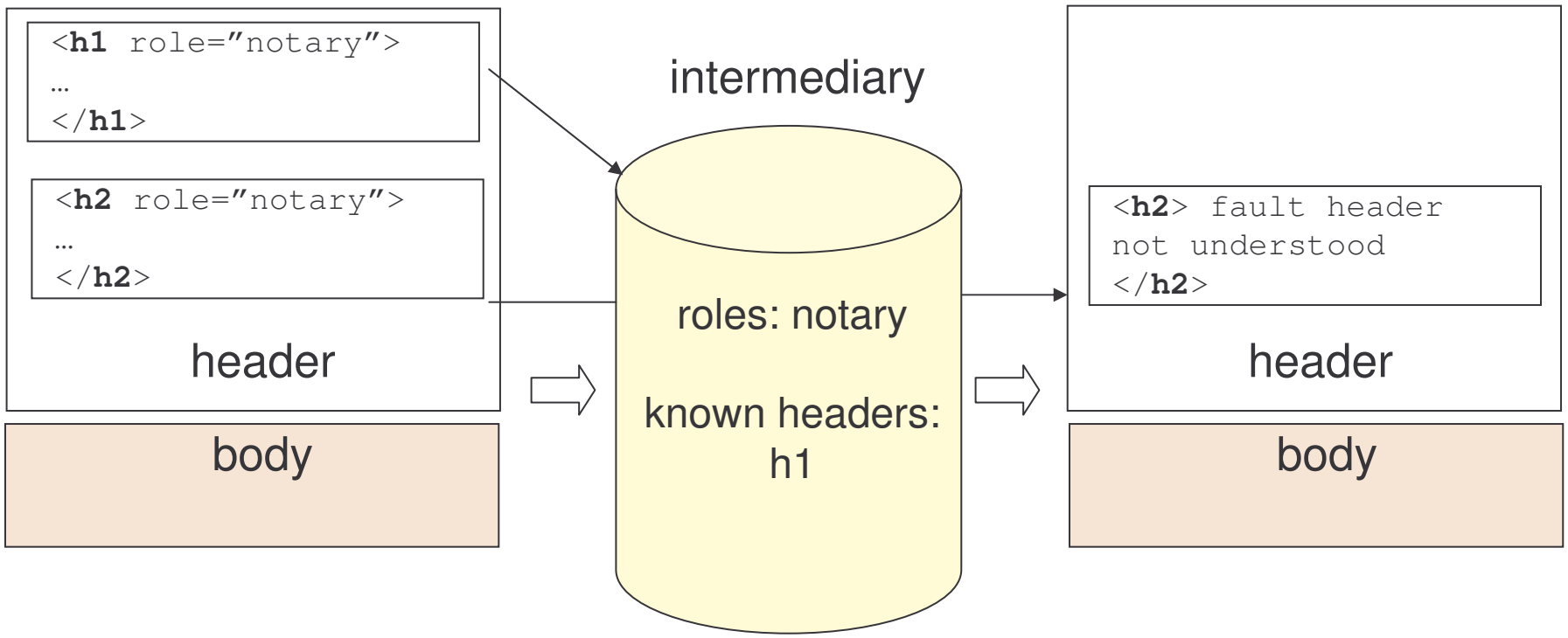
- h1 and h2 are optional headers
- h1 is processed and removed
- h2 is forwarded untouched

Processing Rules: Example 2



- h1 and h2 are optional headers
- h1 is processed and removed
- h2 is not understood and removed

Processing Rules: Example 3



- h1 and h2 are mandatory headers
- h1 is processed and removed
- h2 is not understood and a fault is generated

Relay Attribute

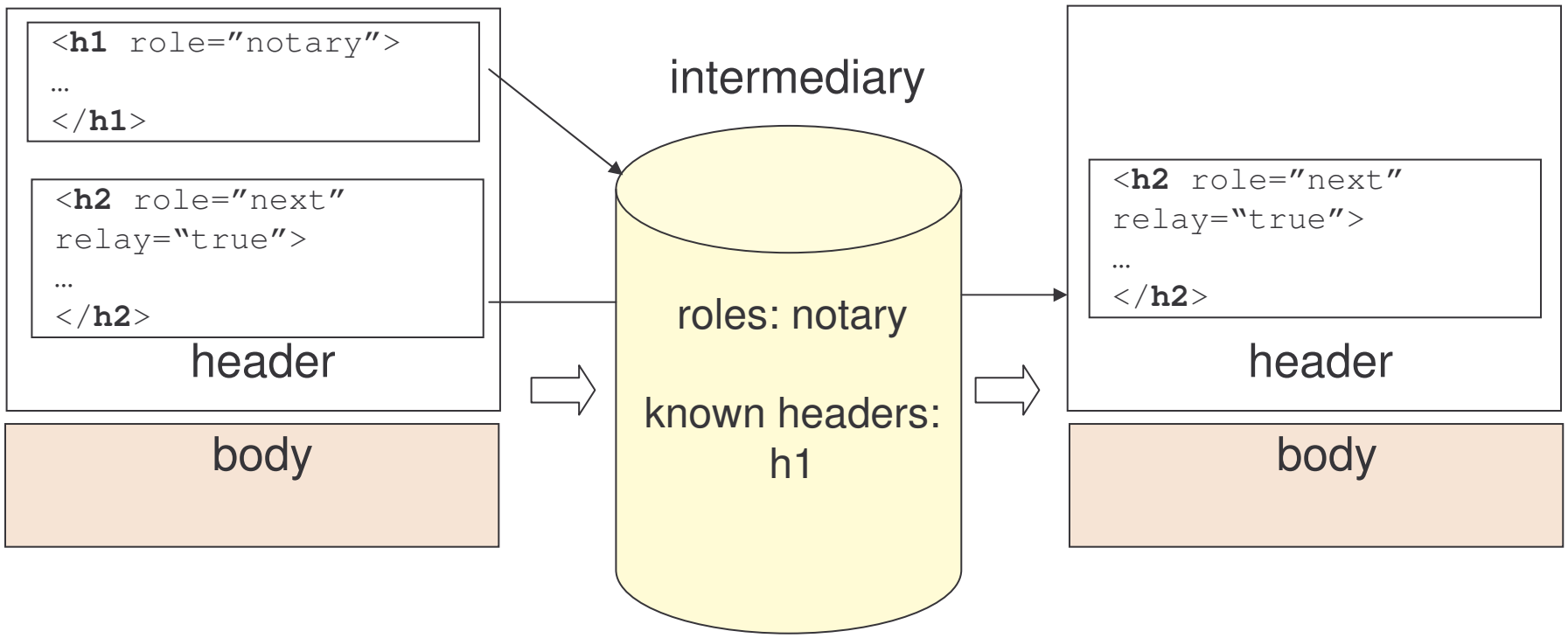
The `relay` attribute is used to indicate to the intermediaries that:

- if a header they do not understand is targeted at them
- then this header should still be passed through

The attribute may equal `true` or `false`.

When a header targeted at a given intermediary has `relay="true"`, it forwards the header regardless of whether it understands it.

Relay Attribute: Example



- h1 is processed and removed
- h2 is forwarded due to the relay attribute

Versioning

SOAP applies XML namespaces to define the protocol version.

The SOAP version is the URI of the SOAP envelope namespace:

1) `http://schemas.xmlsoap.org/soap/envelope` for SOAP 1.1
2) `http://www.w3.org/2003/05/soap-envelope` for SOAP 1.2

An engine supporting a later SOAP version should know previous versions.

The SOAP specification defines processing rules related to SOAP versions.

Versioning: Processing Rules

Rules to follow by the SOAP engine:

- If the message version is the same as a version the engine knows, it should process the message.
- If the message version is older than the one the engine knows, the engine should generate a `VersionMismatch` fault and attempt to negotiate the protocol version with the client.
- If the message version is newer than the one the engine knows, the engine must generate a `VersionMismatch` fault.

Error Handling

A SOAP fault message is a normal SOAP message with a single `Fault` element inside the body.

Components of the `Fault` element include:

1) `Code` - mandatory
2) `Subcodes` - optional
3) `Reason` - mandatory
4) `Node` - optional
5) `Role` - optional
6) `Details` - optional

Fault Elements: Code

The `Code` element includes two sub-elements: a mandatory `Value` element and an optional `Subcode` element.

1) `Value` specifies the type of a fault

2) `Subcode` specifies additional information

Fault Elements: Value 1

Here are the possible values of the Value sub-element:

1) `VersionMismatch`

   The namespace of the received SOAP envelope is not compatible with the SOAP version of the receiver.

2) `mustUnderstand`

   The node does not recognize the block that includes the `mustUnderstand` attribute.

3) `Sender`

   The node cannot process the message because of incorrect or missing data from the sender, e. g. the message is not properly formatted.

Fault Elements: Value 2

4) Receiver

The error is not due to the message itself but rather to the state in which the server was when processing the message.

5) DataEncodingUnknown

The node does not understand the encoding style.

For example:

```
<soap:Fault>
    <soap:Code>
        <soap:Value>soap:Sender</soap:Value>
    <soap:Code>
    . . .
</soap:Fault>
```

Fault Elements: Subcode 1

SOAP allows developers to specify an arbitrary hierarchy of fault subcodes for providing further details about the fault cause.

The `Subcode` element contains:

1) a mandatory `Value` element and

2) an optional `Subcode` sub-element

Each subcode may contain another subcode, to whatever level of nesting.

Fault Elements: Subcode 2

For example:

```
<soap:Body>
  <soap:Fault>
   <soap:Code>
       <soap:Value>soap:Sender</soap:Value>
       <soap:Subcode>
         <soap:Value>bk:InvalidAccount</soap:Value>
       </soap:Subcode>
   </soap:Code>
  </soap:Fault>
</soap:Body>
```

Fault Elements: Reason

The `Reason` element contains human-readable descriptions of the fault.

It contains the `Text` sub-element which includes the fault description.

`Text` may appear several times inside `Reason`.

```
<soap:Fault>

  <soap:Code>…</soap:Code>

  <soap:Reason>
    <soap:Text xml:lang="en">Processing Error</soap:Text>
    <soap:Text xml:lang="cn">处理错误</soap:Text>
    <soap:Text xml:lang="es">Error de Procesamiento</soap:Text>
  </soap:Reason>

</soap:Fault>
```

Fault Elements: Node and Role

Two subelements of `Fault`:

1) The `Node` element specifies which SOAP node was processing the message when the fault has occurred.

   It contains a URI.

2) The `Role` element specifies which role the node was playing when the fault has occurred.

   `Role` behaves in the same way as the headers' `role` attribute.

Fault Elements: Detail

The `Detail` element includes machine-readable data related to the fault.

```
<soap:Envelope
 xmlns:soap="http:www.w3.org/2003/05/soap-envelope"
 xmlns:ac="http://www.example.com">
 <soap:Body>
       <soap:Fault>
             <soap:Code> . . . </soap:Code>
             <soap:Reason>
                   <soap:Text xml:lang="en">
                         Invalid account!
                   </soap:Text>
             </soap:Reason>
             <soap:Detail>
             <ac:LineNumber>10</ac:LineNumber>
             <ac:ColumnNumber>57</ac:ColumnNumber>
             </soap:Detail>
       </soap:Fault>
 </soap:Body>
</soap:Envelope>
```

Task 31: Generate Sender Fault

Objective:

Send a SOAP message that will generate a fault caused by the wrong operation name specified - “downloadf” instead of “downloadFile”.

```
> cd demos\Soap\SenderFault

> dir
FileTransferSenderFault.class

> java -cp \demos\Soap\SenderFault
            FileTransferSenderFault
            \WebServices\MacaoNews.txt news.txt
```

Task 32: Generate Receiver Fault

Objective:

Send a SOAP message that will generate a fault because the server could not find the service “FileService”.

```
> cd demos\SOAP\ReceiverFault

> dir
 FileTransferReceiverFault.class

> java -cp \demos\Soap\ReceiverFault
            FileTransferReceiverFault
            \WebServices\MacaoNews.txt news.txt
```

Faults in Headers

Since a fault is a SOAP message, it can also carry headers.

Problem:

1) a message may contain several mandatory headers
2) one node fails to understand one header
3) how can the header causing the fault be identified?

Solution: SOAP introduces the `NotUnderstood` header:

1) `NotUnderstood` is included for each header in the original message that was not understood.
2) The `qname` attribute of `NotUnderstood` specifies the name of the header that was not understood.

Faults in Headers: Example 1

Suppose a SOAP node receives the following message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Header>
      <a:Header1
         xmlns:a="http://example.com/header1"
         soap:mustUnderstand="true"/>
      <b:Header2
         xmlns:b="http://example.com/header2"
         soap:mustUnderstand="true"/>
   </soap:Header>
   <soap:Body>…</soap:Body>
</soap:Envelope>
```

If the SOAP node does not understand `Header2`,
it would return a message as follows…

Faults in Headers: Example 2

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xml="http://www/w3.org/XML/1998/namespace">
   <soap:Header>
       <soap:NotUnderstood
           qname= "b:Header2"
           xmlns:b="http://example.com/header2"/>
   </soap:Header>
   <soap:Body>
       <soap:Fault>
           <soap:Code>
               <soap:Value>soap:mustUnderstand</soap:Value>
           </soap:Code>
           <soap:Reason>
               <soap:Text xml:lang="en">
                   One or more mandatory headers not understood!
               </soap:Text>
           </soap:Reason>
       </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Upgrade Header

SOAP provides a standard mechanism to indicate which versions of SOAP are supported by a node when generating a `VersionMismatch` fault.

1) An `Upgrade` header is used when a version mismatch fault occurs, to specify which SOAP versions are supported by the node.

2) The different supported version are specified in the `SupportedEnvelope` sub-element of `Upgrade`.

3) The `SupportedEnvelope` elements are ordered by preference, from the most preferred to the least.

Upgrade Header: Example

```
<soap:Envelope
       xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
       xmlns:xml="http://www/w3.org/XML/1998/namespace">
       <soap:Header>
              <soap:Upgrade>
                     <soap:SupportedEnvelope qname="ns1:Envelope"

       xmlns:ns1="http://www.w3.org/2003/05/soap-envelope" />
                     <soap:SupportedEnvelope qname="ns2:Envelope"

       xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/" />
              </soap:Upgrade>
       </soap:Header>
       <soap:Body>
              <soap:Fault>
                     <soap:Code>
                            <soap:Value>VersionMismatch</soap:Value>
         </soap:Code>
         <soap:Reason>
            <soap:Text xml:lang="en">Version Mismatch </soap:Text>
         </soap:Reason>
       </soap:Fault>
```

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

Data Model and Encoding

In order to be able to send Java and others programming language objects inside SOAP envelopes, SOAP defines:

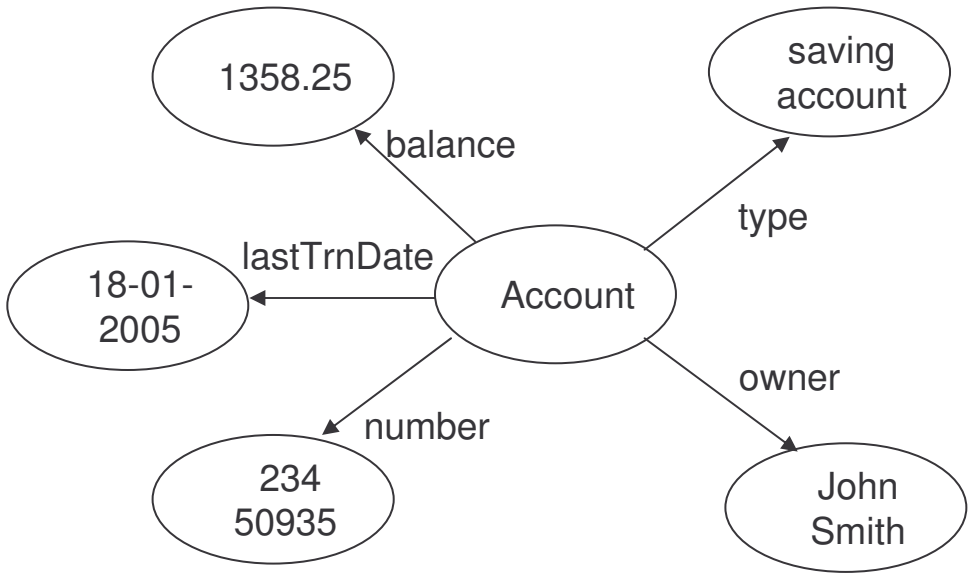
1) SOAP Data Model - an abstract representation of the data structures such as the ones handled by Java or C#

2) SOAP Encoding - a set or rules to map the data model into XML for sending the data inside SOAP envelopes

Data Model

The SOAP data model represents data structures as connected graphs, where nodes represent values and edges represent labels.

SOAP data model



Java object

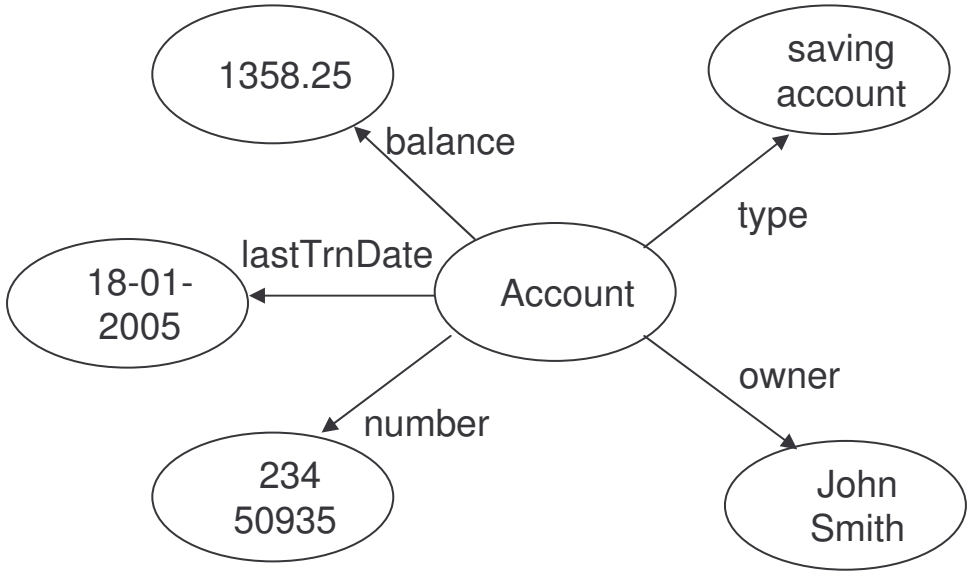
```
class Account {
   int number;
   String owner;
   String type;
   double balance;
   int lastTrnDate;
}
```

Simple Values

Simple values are nodes with only incoming edges.

They correspond to basic data types found in most programming languages, such as `int`, `string`, etc.

For instance `type`, `balance`, `lastTrnDate`, `number`, or `owner` below are all simple values:



Compound Values

Compound values are nodes with outgoing edges.

There are two types of compound values:

1) structures

2) arrays

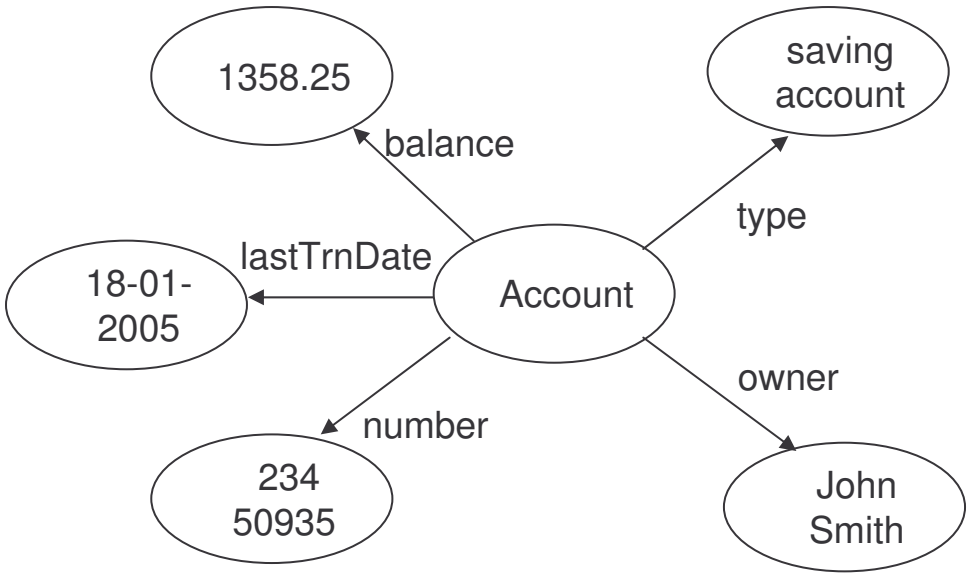
Compound Values: Structures

Structures are compound values where the outgoing edges have names.

They correspond to the named aggregated types.

Each element has a unique name called accessor, which is an XML tag.

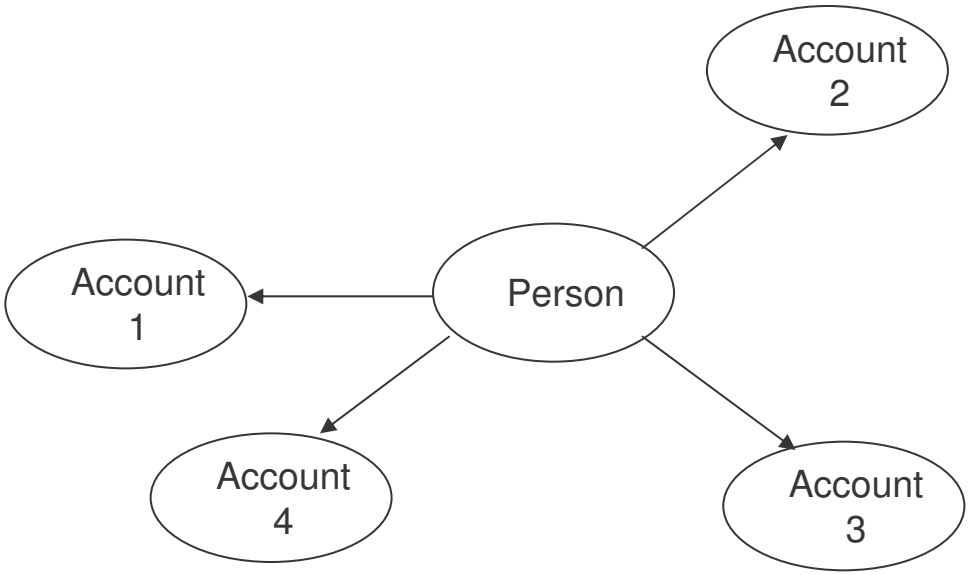
`Account` below is a structure:



Compound Values: Arrays

Arrays are compound values where the outgoing edges are only distinguished by their position (first edge, second edge, etc.).

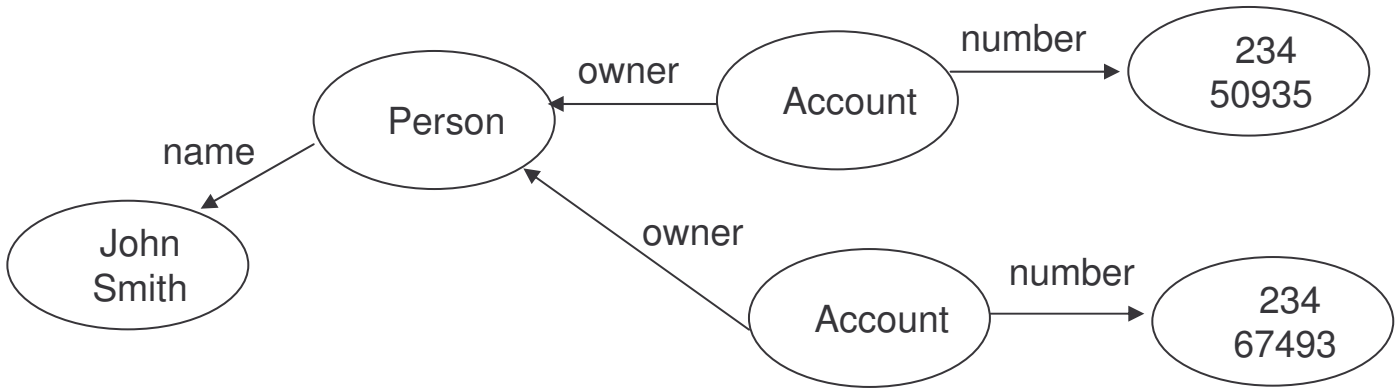
For instance:



Multirefs

Multirefs is a value which is referred from more than one value.

For instance: John Smith is the owner of two different accounts below



Encoding

SOAP encoding describes how the SOAP data model is written with XML.

SOAP encoding is identified by the URI
`http://www.w3.org/2003/05/soap-encoding`.

When serializing XML using encoding rules, processors should use the `encodingStyle` attribute to indicate the SOAP encoding in use.

The `encodingStyle` attribute can appear in:

1) message headers
2) message bodies
3) `Detail` sub-element of `Fault`

or any of their children.

Encoding Example

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>

      <ns1:downloadFileResponse
         soapenv:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:ns1="http://soapinterop.org/">
         <downloadFileReturn
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
            xsi:type="soapenc:base64“>
            TW9..QogDQo=
         </downloadFileReturn>
      </ns1:downloadFileResponse>

   </soapenv:Body>
</soapenv:Envelope>
```

Task 32: Encoding

1) browse: http://www.w3.org/2003/05/soap-encoding

2) Based on the encoding rule definitions:

   a) what are the different values of a node type?
   b) what are the possible attributes for an array?
   c) what is base64?

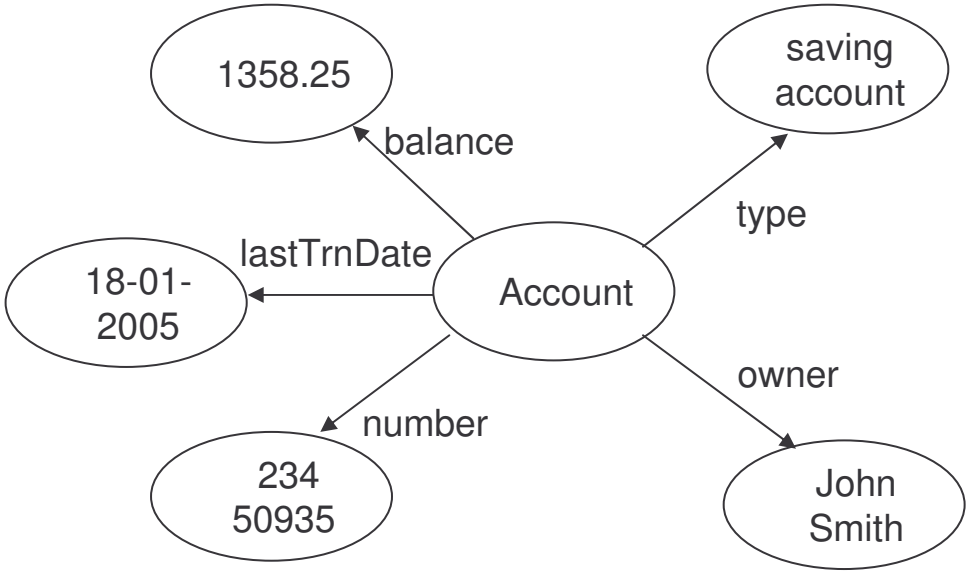
Encoding Rule

Each outgoing edge becomes an XML element which contains:

1) a text value, if the edge points to a terminal node
2) further sub-elements, if the edge points to a node which itself has outgoing edges.

Encoding Rule Example

```
<account
  soapenv:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
  <number>23450935</number>
  <owner>John Smith</owner>
  <type>saving account</type>
  <balance>1358.25</balance>
  <lastTrnDate>20050118</lastTrnDate>
</account>
```



Encoding a Simple Value

For example, in a SOAP message containing:

```
<arg0 xsi:type="xsd:string">
   c:\WebServices\MacaoNews.txt
</arg0>
```

1) `xsi:type` means that `<arg0>` will take string values

2) `xsd:string` is the XML schema string type

In general, all encoded elements provide the `xsi:type` attribute to help recipients decode a message.

Simple Value Example

Java:

```
float balance=1358.25;
String owner="John Smith";
```

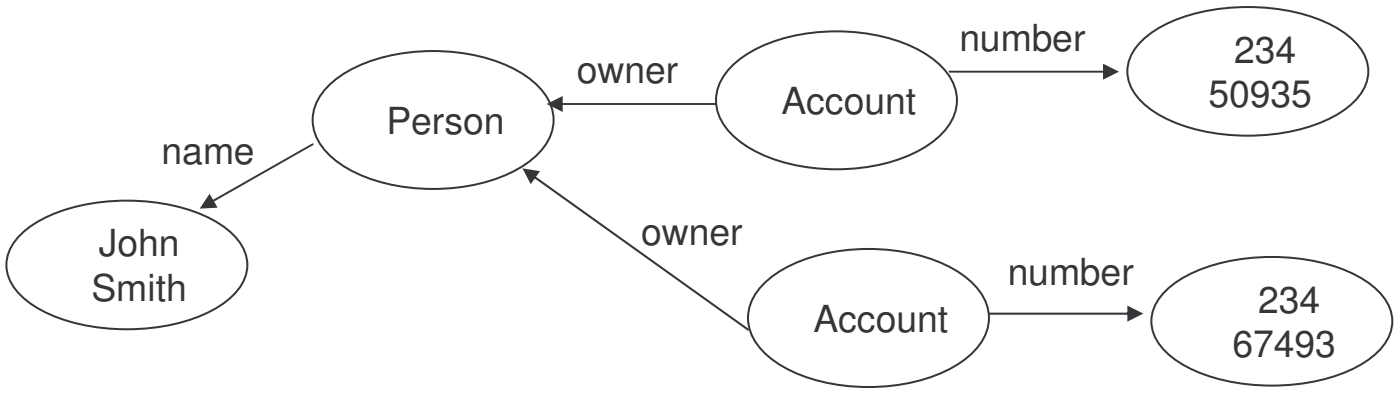
SOAP Encoding:

```
<balance xsi:type="xsd:float">
   1358.25
</balance>
<owner xsi:type="xsd:string">
   John Smith
</owner>
```

Encoding Multirefs

An ID attribute is used to identify objects that are referred to elsewhere.

```
<person id="1"
   soapenv:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
   <name>John Smith</name>
   <account id="2">
       <number>23450935</number>
       <owner>ref="1"</owner>
   </account>
   <account id="3">
       <number>23467493</number>
       <owner>ref="1"</owner>
   </account>
</person>
```



Encoding Arrays

An array in the SOAP object model is encoded in XML using a compound element with two attributes:

1) `itemType` - specifies the data type of the array elements
2) `arraySize` - specifies how many elements are in the array

For example:

```
<myAccounts
   soapenc:itemType="xsd:integer"
   sopaenc:arraySize="3">
   <item>23450935</item>
   <item>23467493</item>
   <item>23426741</item>
</myAccounts>
```

Encoding Multidimensional Arrays

Multidimensional arrays are supported by listing each dimension in the `arraySize` attribute separated by spaces.

The values are serialized as a single list of items in row-major order:

```
<myArray
    soapenc:itemType="xsd:string"
    sopaenc:arraySize="2 3">
    <item>blue</item>
    <item>yellow</item>
    <item>white</item>
    <item>red</item>
    <item>pink</item>
    <item>green</item>
</myArray>
```

blue	yellow	white
red	pink	green

Task 33: Encoding 1

Objective:

Send a request to receive the creation date, contents, size and name of a given file. Accept an object of the `FileAttribute` class in response.

Serialize and encode the `FileAttribute` object before sending.

```
> cd SOAP\Encoding

> dir
deployFileDownloadEncodedService.wsdd
deployService.bat
FileAttribute.class
FileDownloadEncoded.class
FileTransferRequestEncoding.class
FileTransferResponseEncoding.class
responseFormatted.txt
```

Task 34: Encoding 2

Deploy the web service:

```
> copy FileAttribute.class
            Tomcat 4.1\webapps\axis\WEB-INF\classes

> copy FileDownloadEncoded.class
            Tomcat 4.1\webapps\axis\WEB-INF\classes
```

Double-click `deployService.bat`

Test the web service: browse http://localhost:8080/axis and View.

Task 35: Encoding 3

Run the request and send the output to a log file (request.txt):

```
> java -cp \demos\SOAP\Encoding
             FileTransferRequestEncoding
             \WebServices\MacaoNews.txt > request.txt
> notepad request.txt
```

Run the response and send the output to a log file (response.txt):

```
> java -cp \demos\SOAP\Encoding
             FileTransferResponseEncoding
             \WebServices\MacaoNews.txt > response.txt
> notepad response.txt
```

Analize the response:

```
> notepad responseFormatted.txt
```

Encoding a Request

A SOAP request message is modeled as a structure with an accessor element for each input and output parameter:

```
<getBalance
   encodingStyle="http://www/w3.org/2003/05/soap-encoding">
   <accountNumber xsi:type="xsd:int">
      23450935
   </accountNumber>
</getBalance>
```

1) the only accessor is `accountNumber`

2) accessor names correspond to the names of parameters, their `type` attributes correspond to the programming language data types

3) parameters must appear in the same order as in the method signature

4) the name of the structure element is the procedure or method name

Encoding Specific Faults

SOAP defines some fault codes specifically for encoding problems.

These are recommended values to be sent in the `Subcode` value when the `Sender` code is used.

They all relate to problems with the sender's data serialization.

1) `MissingID` – generated when a `ref` attribute in the received message does not correspond to any of the `id` attributes in the message

2) `DuplicateId` – generated when more than one element in the message has the same `id` attribute value

3) `UntypedValue` – generated optionally to indicate that a type in the received message could not be determined by the receiver

Encoding RPC Request

Using a SOAP structure to represent a method call:

```
public float getBalance(int arg)
```

A request message representing a call to this method in SOAP is:

```
<soap:Envelope>
        <soap:Body>
                <orgNS:getBalance
                        xmlns:orgNS="http://myOrganization.com/"

        soap:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
                        <arg0 xsi:type="xsd:int">23450935</arg>
                </orgNS:getBalance>
        </soap:Body>
</soap:Envelope>
```

The structure contains one accessor for each argument.

The content of the `arg` element is the value for the argument.

Encoding RPC Response

The response is also modeled as a structure which name is the method name with `Response` element appended.

Here is a possible response to the previous message:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <orgNS:getBalanceResponse
         xmlns:orgNS="http://myOrganization.com/"
         xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
         soap:encodingStyle=
            "http://www/w3.org/2003/05/soap-encoding">
         <rpc:result>ret</rpc:result>
         <ret xsi:type="xsd:decimal">1358.25</ret>
      </orgNS:getBalanceResponse>
   </soap:Body>
</soap:Envelope>
```

Encoding RPC Return Values

SOAP specifies that an RPC response structure containing a return value must contain an accessor element called `result`.

The value of this element specifies the accessor's name containing the return value for the invocation.

```
<soap:Body>
        <orgNS:getBalanceResponse
                xmlns:orgNS="http://myOrganization.com/"
                xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
                soap:encodingStyle="http://www/w3.org/2003/05/soap-
 encoding">
                <rpc:result>ret</rpc:result>
                <ret xsi:type="xsd:decimal">1358.25</ret>
        </orgNS:getBalanceResponse>
</soap:Body>
```

Encoding RPC Out Parameters

Suppose we have this Java method:

```
public float getBalance(int arg, out String status)
```

The method now returns the account balance and the status.

The request message is as the one shown previously.

Encoding RPC Out Parameters

The response looks as follows:

```
<soap:Envelope
      xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <orgNS:getBalanceResponse
          xmlns:orgNS="http://myOrganization.com/"
          xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
          soap:encodingStyle=
              "http://www/w3.org/2003/05/soap-encoding">
          <rpc:result>ret</rpc:result>
          <ret xsi:type="xsd:decimal">1358.25</ret>
          <status xsi:type="xsd:string">active</status>
      </orgNS:getBalanceResponse>
   </soap:Body>
</soap:Envelope>
```

RPC In-Out Parameters

In web services, all parameters are passed by value.

Therefore the notion of `in-out` and `out` parameters does not involve passing objects by reference, but exchanging copies of the data.

The client code should create the perception that the state of the object that has been passed to the method has been modified.

Communication Styles

SOAP enables two communication styles:

1) document-style

   The message has no fixed structure, so the interacting applications must agree beforehand on this structure.

2) RPC-style

   Synchronous method invocation - pre-defined message structure.

Document Style

Also known as a message-oriented style:

1) a request is an XML document
2) an optional response is also an XML document

Two interacting applications agree beforehands upon the structure of the documents exchanged, then use SOAP messages to transport them.

Very flexible communication style that provides the best interoperability, using synchronous or asynchronous communication.

Document Style Example

The response message in document-style:

```
<soap:Envelope
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <soap:Body>
  <orgNS:returnBalance
      xmlns:orgNS="http://myOrganization.com/"
      soap:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
    <orgNS:balance orgNS:type="xsd:float">1235.95
    </orgNS:balance>
  </orgNS:returnBalance>
 </soap:Body>
</soap:Envelope>
```

Task 36: Document Style 1

Objective:

Generate a SOAP message in a document style, with a purchase order for two XML books with ISB-12345 and the total price 125.12.

```
1) cd demos\SOAP\DocumentStyle

2) dir
deployDocumentStyleService.wsdd
deployService.bat
DocumentStyleClient.class
DocumentStyleService.class
```

Task 37: Document Style 2

3) deploy DocumentStyleService:

```
a) copy documentStyleService.class
     to   Tomcat/webapps/axis/WEB-INF/classes
b)        execute deployService
```

4) test the service: http://localhost:8080/axis ⟶ View

5) what is the difference with the previous service deployed?

   a) methods?

   b) WSDL?

Task 38: Document Style 3

6) execute the client sending the output to a log file:

```
java -cp \demos\SOAP\DocumentStyle DocumentStyleClient
   > log.txt
```

7) `notepad log.txt`

RPC Style

RPC-style is a synchronous invocation of an operation returning a result:

1) One SOAP message encapsulates the request.

   The body of the request message contains the actual call including the name of the procedure being invoked and the input parameters.

2) Another SOAP message encapsulates the response.

   The body of the response contains the result and output parameters.

The two interacting applications agree upon the RPC method signature.

RPC Style Example

The response message in RPC-style:

```
<soap:Envelope
     xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
    <orgNS:getBalanceResponse
         xmlns:orgNS="http://myOrganization.com/"
         xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
         soap:encodingStyle="http://www/w3.org/2003/05/soap-encoding">
       <rpc:result>ret</rpc:result>
       <ret xsi:type="xsd:decimal">1358.25</ret>
       <status xsi:type="xsd:string">active</status>
    </orgNS:getBalanceResponse>
 </soap:Body>
</soap:Envelope>
```

Task 39: RPC Request

Objective: generate a SOAP message in the RPC-style.

```
1)  cd demos\SOAP\RPCStyle

2)  dir
FileTransferRequest

3)  Java -cp demos\SOAP\RPCStyle FileTransferRequest
    e:\webservices\macaonews.txt news.txt > log.txt

4)  notepad log.txt
```

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

Protocol Binding Framework

SOAP enables exchange of messages using a variety of protocols.

The set of rules for carrying a SOAP message within or on top of another protocol for the purpose of exchange is called binding.

SOAP protocol binding framework:

1) provides general rules for the specification of protocol bindings

2) describes the relationship between bindings and SOAP nodes that implement those bindings

Binding and Transfer

For instance, SOAP HTTP binding describes how to take a SOAP infoset at one node and serialize it across an HTTP connection to another node.

The job of the binding is to move the infoset from one node to another. The way the infoset is represented in the “wire” is up to the binding author.

Bindings have the freedom to specify custom serializations in order to improve efficiency, security, etc.

Binding URI

Communicating parties must agree on what binding to use.

Thus, bindings are named with URIs.

SOAP HTTP binding URI:

http://www.w3.org/2003/05/soap/bindings/HTTP

SOAP Features

A feature extends SOAP with some specific functionality.

SOAP poses no constraints on the potential scope of features.

A feature description is identified by a URI, so that all applications referencing it are assured the same semantics.

SOAP Feature Examples

Examples of features:

1) reliability
2) security
3) routing
4) Message Exchange Patterns (MEPs):
    a) request/response
    b) one-way
    c) peer-to-peer conversations

Expressing Features

The SOAP extensibility model provides two mechanisms through which features can be expressed:

1) SOAP Processing Model

2) SOAP Protocol Binding Framework

Features with SOAP Processing

Describes the behavior of a single SOAP node with respect to the processing of an individual message

Characteristics:

1) features are expressed by modules

2) a module is a way to perform functions using the SOAP processing model via headers

Features with Protocol Binding

Mediates the act of sending and receiving SOAP messages by a SOAP node via an underlying protocol.

Characteristics:

1) features are expressed by bindings

2) a binding is a way to perform functions below the SOAP processing model

Features Method Comparison

Processing Model:

1) enables SOAP nodes, that are able to implement features to express them within the SOAP envelope as SOAP headers

2) header can be intended for any SOAP node along the SOAP message path

Protocol Binding Framework:

1) a protocol binding operates between two adjacent SOAP nodes along the message path

2) different protocols can be used along the path

3) some protocols are equipped, either directly or through an extension, with mechanisms for providing certain features

SOAP Modules

SOAP modules define the syntax and semantics of the extensions provided by headers, including constraints, rules, preconditions, and data formats .

A SOAP module realizes zero or more SOAP features.

SOAP modules are named with URIs so they can be referenced, versioned, and reasoned about.

Feature Specification

The specification of a feature must include:

1) a URI used to name the feature

2) the information required at each node to implement the feature

3) processing required at each node to implement the feature including handling of communication failures that might occur

4) the information to be transmitted from node to node

Feature Example

Suppose an application requires a “secure channel” feature.

The URI for this feature is http://www.myOrganization.com/secureChannel

The abstract feature describes that messages must travel from node to node in an unsnoopable fashion (reasonable level of security).

Alternatives:

1) Since HTTPS meets the security requirement specified by the feature, the feature would be satisfied by this protocol binding.
2) We can use a SOAP module (e.g. WS-Security) that provides encryption and signing of SOAP messages across any binding.

We can decide in some situations to engage the SOAP module, and not to do so in others (e.g. when using the HTTPS binding).

Message Exchange Patterns

MEP is a common type of feature.

A MEP specifies:

1) how many messages are involved in interaction
2) where the messages originate
3) where they end up

Each binding must support one or more MEP.

SOAP specifies two standard MEPs:

1) request-response
2) SOAP-response

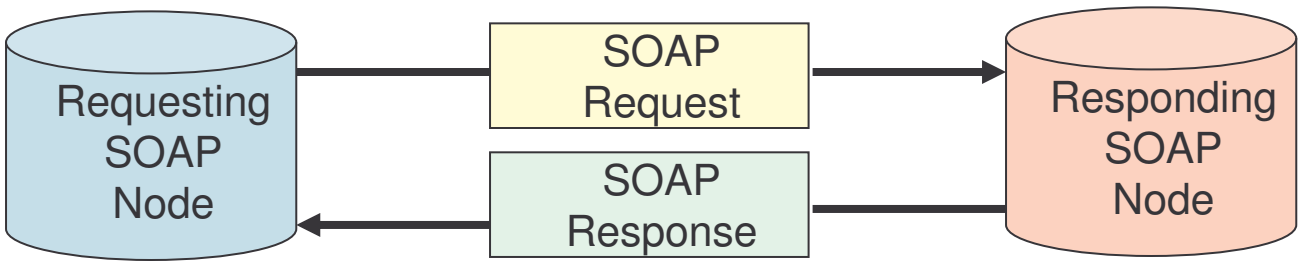
Request-Response MEP

Request-Response MEP involves two nodes:

1) requesting node sends a SOAP message to the responding node
2) responding node replies with a SOAP message that returns to the requesting node

Important features:

1) the response message is correlated to the request message
2) if a fault is generated at the responding node, the fault is delivered as part of the response message



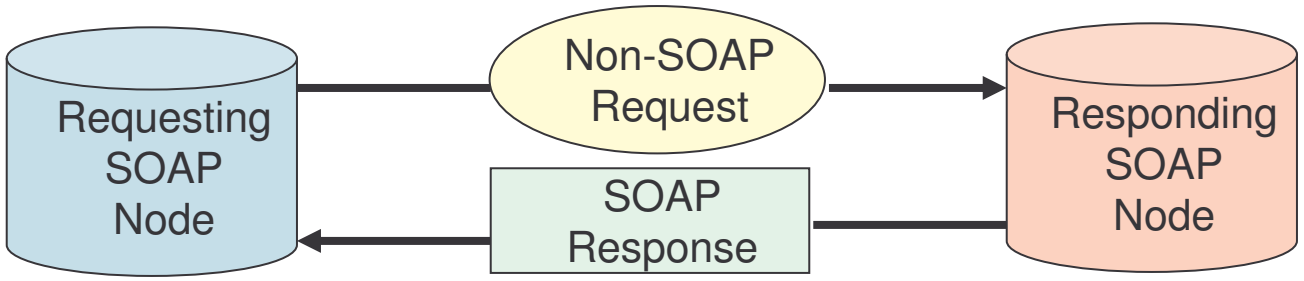
SOAP Response MEP

SOAP Response MEP involves two nodes:

1) the requesting message is not a SOAP message
2) the responding node replies to the request with a SOAP message

Important features:

1) the request does not trigger the execution of the SOAP processing model on the receiving node
2) it allows a request to be something as simple as an HTTP GET



Request-Response MEP Example

One alternative is to implement the request-response MEP by an HTTP binding. Another one is using SOAP headers:

```
<soap:Envelope
 xmlns:saop="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
       <r:replyTo
              soap:mustUnderstand="true"
              xmlns:r="http://myOrganization/requestResponse">

       <destination>udp://anotherHost.com:6777</destination>
       </r:replyTo>
       <r:correlationID
              soap:mustUnderstand="true"
              xmlns:r="http://myOrganization/requestResponse">
              1202
       </reqresp:correlationID>
 </soap:Header>

 <soap:Body>…</soap:Body>
</soap:Envelope>
```

MEPs with HTTP Binding

The SOAP HTTP binding supports both MEPs.

1) request/response MEP with HTTP binding:
   - ✓ SOAP request message → HTTP request
   - ✓ SOAP response message → HTTP response

2) SOAP response MEP with HTTP binding:
   - ✓ non-SOAP request → HTTP GET request
   - ✓ SOAP response message → HTTP response

HTTP binding also specifies how to map faults to particular HTTP status codes and status codes to the web services invocations.

SOAP HTTP Protocol Binding

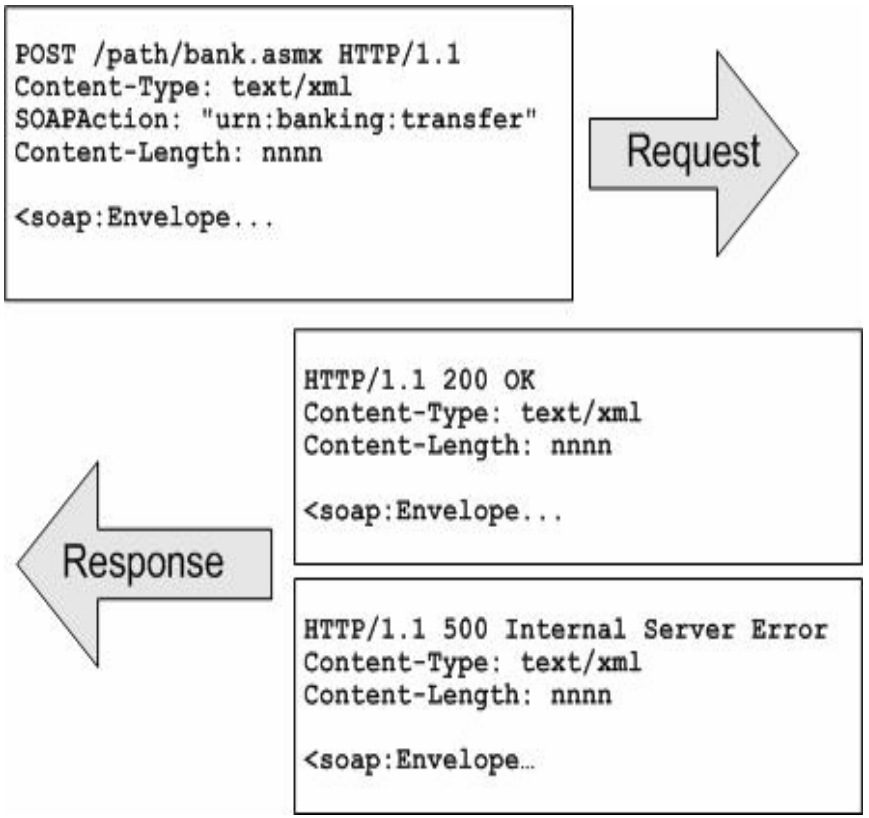
Example rules:

1) SOAP request/response maps naturally to the HTTP model

2) content-type header for HTTP request/response messages must be set to application/soap+xml

3) request messages must use POST and the URI identifying the SOAP processor

4) HTTP response should use 200 status if no errors occurred or 500 if the body contains a fault



[courtesy Aaron Skonnard]

SOAP HTTP Binding Details

HTTP GET request does not have a payload area and therefore cannot be used to carry SOAP messages.

The SOAP HTTP binding also implements two features:

1) SOAP action feature
2) web method feature

SOAP 1.1 Action Feature

In SOAP 1.1 HTTP binding, a custom header `SOAPAction` is required to:

1) let know the receiver that the content of the message is SOAP
2) convey the intent of the message via a URI

The decision to use a value for the `SOAPAction` header field is up to the web service designer.

Many implementations use this URI dispatching a particular piece of code on the backend, especially for document-style communication.

For instance, the same body content may be sent to two different methods, and the `SOAPAction` URI may be used to differentiate between them.

SOAP 1.2 Action Feature

SOAP 1.2 uses the `application/soap+xml` media type. The `SOAPAction` header is no longer needed to identify SOAP messages.

This media type specifies an optional `action` parameter used in SOAP 1.2, instead of an HTTP-specific header to carry the `SOAPAction` URI.

The binding implementing this feature must place the value of the URI in the action parameter. The message looks like:

```
POST /axis/TheService.jws
Content-Type: application/soap+xml; charset=utf-8
Action="http:// myOrganization.com/specificAction"
...
```

Web Method Feature

The web method feature was defined to integrate the semantics of SOAP with the semantic of HTTP.

Bindings to HTTP should use this feature to give control to applications over the web methods (GET, POST, PUT, …) used sending SOAP message.

When sending a message, the HTTP binding will use the verb specified in this property instead of the default POST.

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

SOAP and Binary Data

Suppose that in the bank application we would like to send an image of the account statement.

Since XML cannot encode binary data, a solution might be to use the XML Schema type `base64binary` and encode images as `base64` text:

```
<soap:Envelope xmlns:soap="..." xmlns:xsi="...">
 <soap:Body>
  <accountData>
   <number>23450935</number>
   <owner>John Smith</owner>
   <statement imageType="jpg" xsi:type="base64binary">
      4f3t68j …
   </statement>
  </accountData>
 </soap/Body>
</soap:Envelope>
```

Not efficient! E-mail is using the MIME standard.

SOAP with Attachments 1

In 2000, HP and Microsoft released a specification SOAP with Attachments.

SwA describes a simple way to use the multiref encoding in SOAP 1.1 to reference MIME-encoded attachment parts.

Here is the previous example in SwA:

```
MIME-Version:1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
type=application/soap+xml;start="<account@myOrganization.com>"

--MIME_boundary
Content-Type: application/soap+xml: charset=UTF-8
Content-Transfer-Encoding: 8 bit
Content-ID:<account@myOrganization.com>
```

SOAP with Attachments 2

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 <soap:Body>
        <accountData>
                <number>23450935</number>
                <owner>John Smith</owner>
                <statement href="cid:statemet@myOrganization.com"/
        </accountData>
 </soap/Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID:<statement@myOrganization.com>
. . . Binary JPG image . . .
--MIME_boundary
```

The problem is that this approach introduces a data structure that is explicitly outside the realm of the XML data model.

Task 40: SOAP with Attachments

Objective: send a request for uploading an attached file.

```
1)  cd \demos\SOAP\WithAttachments

2) dir
deployFileUploadService.wsdd
deployService.bat
FileUploadRequest.class
FileUploadService.class
```

Task 41: SOAP with Attachments

Deploy the web service:

3) copy FileUploadService.class
   to Tomcat 4.1\webapps\axis\WEB-INF\classes

4) double-click: deployService.bat

Test the web service:

5) http://localhost:8080/axis --> View

Task 42: SOAP with Attachments

Run the request and send the output to a log file (`request.txt`):

```
6) java -cp \demos\SOAP\WithAttachments  FileUploadRequest
       \WebServices\MacaoNews.txt macao.txt > request.txt

7) notepad request.txt
```

SOAP Outline

1) Introduction

2) Messaging
   a) envelope
   b) headers
   c) processing model
   d) error handling

3) Data Structures
   a) data model
   b) data encoding
   c) request encoding

4) Protocol Binding
   a) binding
   b) features and modules
   c) communication patterns

5) SOAP and Binary Data

6) Summary

SOAP Summary 1

SOAP is a lightweight protocol that allows to move XML messages in a distributed environment.

SOAP provides:

1) a messaging framework
2) data model and encoding rules
3) bindings to various communication protocols

SOAP is extensible.

SOAP is independent of any protocol or programming language.

SOAP Summary 2

1) XML messages are packed in envelopes for transmission.

2) A SOAP envelope contains zero or more headers and a body.

3) A header is a container for control information or application data sent to a SOAP server. It provides a mechanism to extend SOAP messages.

4) A body is a container to exchange information.

5) A fault is a structure to inform a sender that something went wrong while processing the message by the receiver.

SOAP Summary 3

1) A SOAP message can visit several intermediaries before it reaches the ultimate receiver. All these nodes constitute the message path.

2) SOAP defines a processing model that outlines rules for processing a message through the message path.

3) Headers may be addressed to specific intermediaries.

4) Headers may include four attributes:

   a) `role`: the name of the intermediary who should handle the header
   b) `mustUnderstand`: is processing the header mandatory?
   c) `relay`: should the header be passed to the next node?
   d) `encodingStyle`: a URI for the SOAP encoding

SOAP Summary 4

1) SOAP data model defines an abstract representation of the common data structures handled by programming languages.

2) SOAP encoding provides a set of rules to map the data model to XML.

3) SOAP also provides the rules for encoding requests, faults and RPC.

4) SOAP supports two communication styles for invoking a service:

   a) document style: interacting applications must agree upon the structure of the documents exchanged
   b) RPC style: synchronous service invocation, defined message structure

SOAP Summary 5

1) SOAP protocol binding framework provides the general rules for specification of protocol bindings.

2) A SOAP protocol binding describes how to take a SOAP infoset at one node and serialize it across the protocol connection to another node.

3) A feature is a unit of SOAP extension, implemented via SOAP modules or bindings. Message Exchange Pattern (MEP) is a common binding.

4) Two standard MEPs: request-response and SOAP response.

5) SOAP HTTP binding satisfies both standard MEPs.

6) Binary data may be sent by SOAP using a text encoding or using SOAP with attachments.

WSDL

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

Service Description

A client needs to use a web service to exchange SOAP messages, but:

1) what to include on the body of the message?

2) is any security SOAP header required?

3) what is the format of the response message?

4) what protocol is required?

5) where to send the message?

Service Description Need

A service description is needed.

Service descriptions are needed for the three SOA operations:

1) publish

2) find

3) bind

Service Description Components

A service description has two major components:

1) functional description - defines details of how the service is invoked, where is invoked, etc.

2) non-functional description - provides other details that are secondary to the message but instructs the requestor's runtime environment to add SOAP headers, such as: security policy

Functional Description

The functional description describes the operations available in the web service and the syntax of the messages required to invoke them.

The functional description is composed of:

1) service interface definition - describes:
   a) what messages must be sent
   b) how to use the various messaging protocols
   c) which encoding schemes must be used in order to format messages acceptable by the service provider

2) service implementation definition - describes where the service is located

Both definitions use the Web Service Description Language (WSDL).

Non-Functional Description

The non-functional description adds more information about the service:

1) why a service requestor should invoke the service - what business function the web service addresses and how it fits into a broader business process

2) who is the service provider, if the service provider carries out auditing, ensures privacy, etc.

3) specific aspects of the service not dependent on the domain, such as security

Currently, the most widely adopted approach to describing non-functional requirements is the combination of: WS-Policy and WS-PolicyAttachment.

Service Description Layers

A web service is described using a combination of techniques.

These are the questions that a service description should answer and which layer is providing this information:

who ?	non-functional description
what ?	service interface
where ?	service implementation
why ?	non-functional description
how ?	service interface and non-functional description

WSDL History

Two prior IDL languages for web services:

1) IBM's Network Accessible Service Specification Language (NASSL)

2) Microsoft's SOAP Contract Language (SCL)

WSDL is the result of merging NASSL and SCL.

WSDL Recommendation

IBM, Microsoft and other companies submitted WSDL 1.1 to the W3C for standardization in March 2001.

The specification is available at: http://www.w3.org/TR/wsdl

On 2004, the Web Services Description Working Group has released the First Public Working Draft of WSDL 2.0.

Still, not a recommendation.

WSDL Service Description

A WSDL service description is an XML document conformant to the WSDL schema definition.

This document, without any extensions, is not a complete service description, since it only covers the functional part.

WSDL is “the IDL for Web Services” describing:

1) what a service does - the operations (methods) the service provides, and the data (arguments and returns) needed to invoke them

2) how a service is accessed - details about data formats and protocols necessary to access the service operations

3) where a service is located - details of the protocol-specific network address, such as a URL

WSDL and IDLs

As WSDL describes service interfaces, it has a role and purpose similar to that of an IDL in conventional middleware platforms, but:

1) IDL only specifies a service interface: name and signature

2) the location of the requested object is transparent and unknown to the client

3) describes a single entry point (single RPC interaction)

4) objects are accessed through a concrete middleware platform

1) WSDL also defines the mechanisms to access the service

2) the WS middleware at the client site should be able to identify the location of the service

3) involves the exchange of several asynchronous messages

4) services are accessed using different protocols

Repositories of WSDL Documents

Public repositories of WSDL documents:

1) http://www.salcentral.com
2) http://www.xmethods.com
3) http://www.grandcentral.com/

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

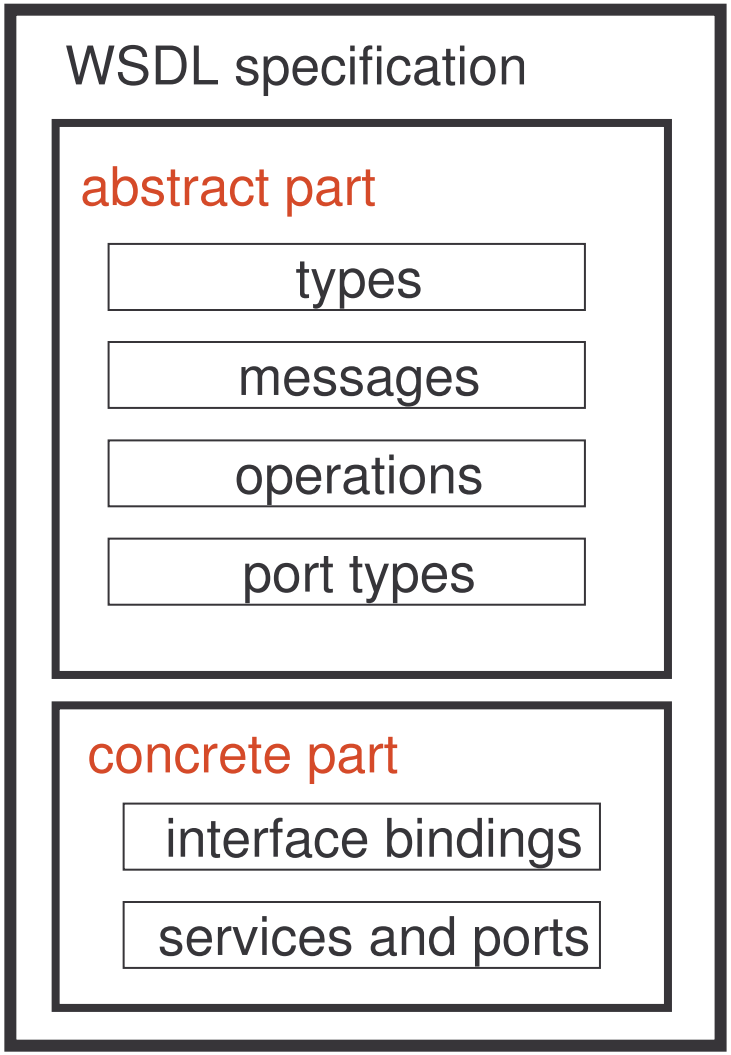
6) Summary

WSDL Structure

WSDL specifications include:

1) abstract part - conceptually analogous to conventional IDL

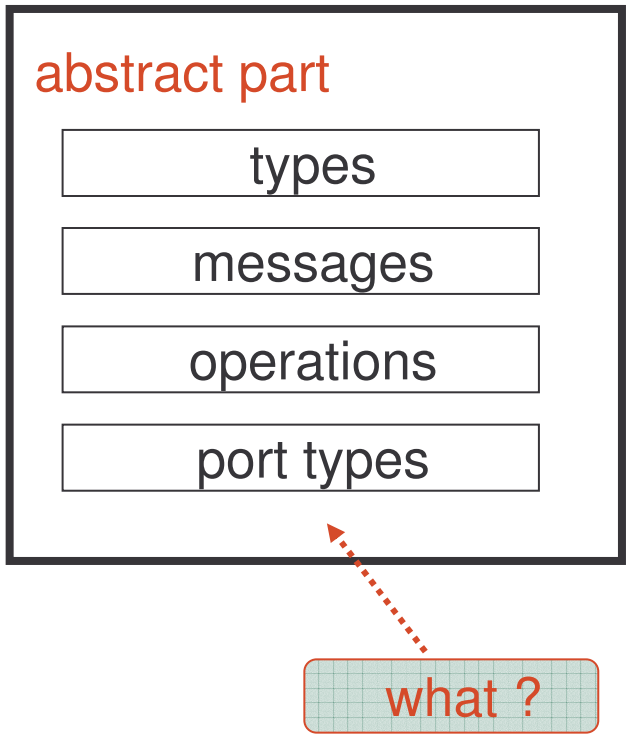
2) concrete part - defines protocol binding and other information



WSDL Structure - Abstract

The abstract part includes:

1) port type - logical collection of related operations
2) operation - abstract description of an action supported by the service
3) message - data exchanged in a single logical transmission
4) types - data structures that will be exchanged as parts of messages



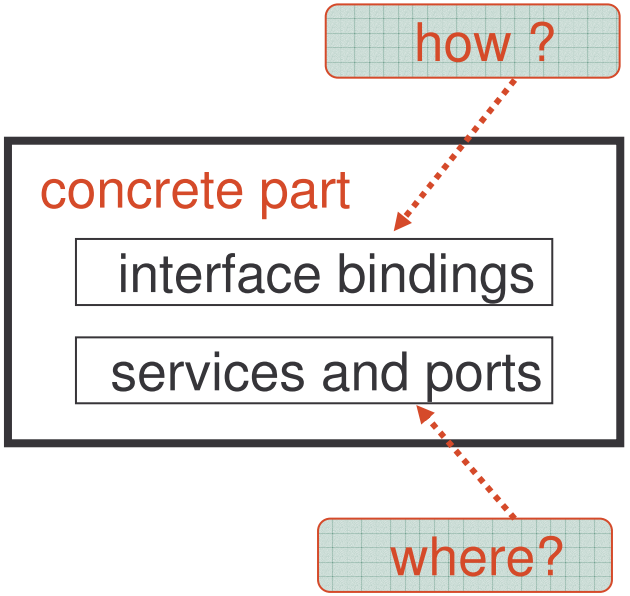
There is no:

1) concrete binding
2) encoding specified
3) service implementing the set of ports

WSDL Structure - Concrete

The concrete part includes:

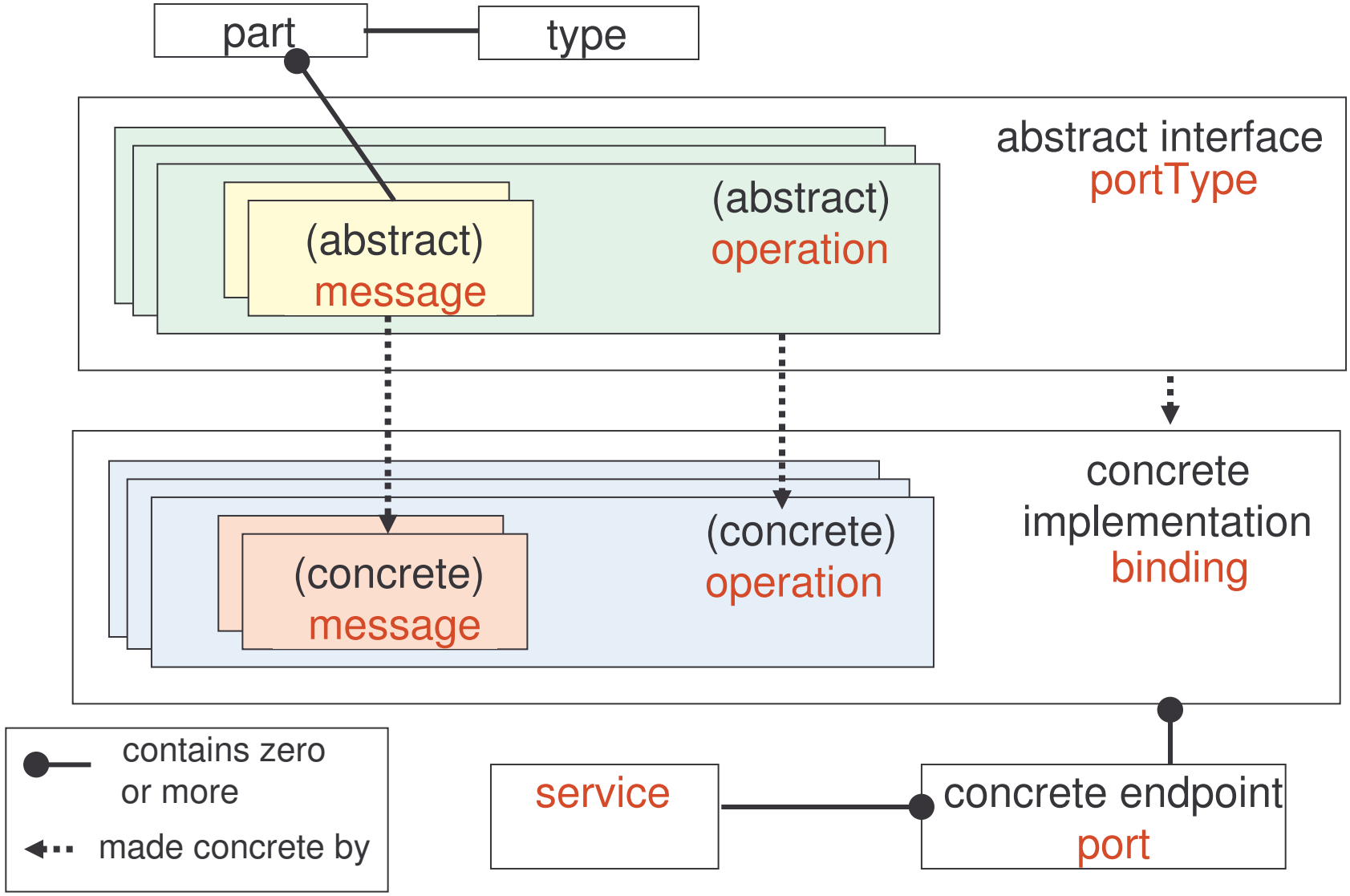
1) interface bindings - message encoding and protocol binding for all operations and messages defined in a given port-type
2) ports - combine the interface binding information with a network address specified by a URI
3) services - are logical groupings of ports



These allow:

1) a specific web service at different web addresses (different servers)
2) different ports (interface bindings) for the same port type, allowing the same functionality to be accessible via multiple transport protocols and interaction styles

WSDL Information Model



WSDL Document Example 1

```
<?xml version="1.0"?>
<definitions
   name= "Orders"
   targetNamespace="http://www.example.com/orders"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:tns=http://www.example.com/orders">
```

abstract part

```
<message name="OrderMsg">
   <part name="productId" type="xsd:string"/>
   <part name="quantity"  type="xsd:integer"/>
</message>
```

messages

```
<portType name="OrderPortType">
  <operation name="orderProductRequest">
    <input message = "OrderMsg"/>
  </operation>
</portType>
```

operation and port type

WSDL Document Example 2

WSDL Document Structure

The root element of a WSDL document is a `definitions` element.

This element can contain:

1) an optional `types` element
2) zero or more `message` elements
3) zero or more `portType` elements (usually one)
4) zero or more `binding` elements (usually one)
5) zero or more `service` elements (usually one)
6) zero or more `documentation` elements
7) zero or more `import` elements

A WSDL document must conform to the XML Schema defined at:
http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd

Definitions Element

The `definitions` element contains:

1) `name` attribute - corresponds to the name of the web service. It is only for documentation and is optional

2) `targetNamespace` attribute - a URI for the entire WSDL file

3) default namespace - all elements without a namespace prefix, such as `message` or `portType`, are assumed to be part of the default WSDL namespace: `http://schemas.xmlsoap.org/wsdl/`

4) other XML namespace declarations

Definitions Example

A Web Service that converts temperature in Fahrenheit to Celsius.

The service supports a single operation FahrenheitToCelsius, deployed using the SOAP protocol over HTTP.

```
<definitions
 name= "TemperatureConverterService"
 targetNamespace="http://www.converter.com/TemperatureConverter"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns=http://www.converter.com/TemperatureConverter”
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">

 ...
</definitions>
```

Task 43: Definitions Element

Objective: Write a WSDL file describing a web service that will send an email with a custom message and error description to a developer or help desk. Two bindings will be provided: SOAP and HTTP-GET

1) `cd \demos\WSDL\Example1`

2) create a document “Myexample.wsdl” and add:

   a) the XML declaration

```
<?xml version="1.0" encoding="utf-8" ?>
```

   b) the root element for the WSDL document

```
<wsdl:definitions
```

Task 44: Definitions Element

3) add the definitions of namespaces for:

```
targetNamespace="http://example.com/ErrorMailer"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

and:

```
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://example.org/ErrorMailer"
```

4) save the file

Types Element

The `types` element encloses the definitions of user-defined XML types and elements for later use in the contents of the message.

A WSDL document can have at most one `types` element, and when present, it typically contains a single schema definition.

XML Schema is the predominant type system used, although `types` allows to describe other type systems.

In order to build interoperable web services, WSDL should only use the datatypes defined with XML Schema.

Types Example

```
<types>
   <xsd:schema
     targetNamespace="http://www.converter.com/TemperatureConverter">
        <xsd:element name="tempCelsius" type="xsd:float" />
        <xsd:element name="tempFahrenheit" type="xsd:float" />
   </xsd:schema>
</types>
```

Using the `schema` element to define XML datatypes and elements requires to include a `targetNamespace` attribute.

Many designers use the same value as the one used in WSDL `definitions` element.

Task 45: Types

Objective: Add the types definition to “MyExample.wsdl”

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” and add:

   a) the type and the schema elements:

```
<wsdl:types>
  <s:schema  targetNamespace="http://example.com/ErrorMailer">
```

Task 46: Types

b) element `SendError` defined as a complex type with two elements `LicenseKey` and `ErrorMessage`, both of type string:

```
<s:element name="SendError">
     <s:complexType>
        <s:sequence>
             <s:element name="LicenseKey" type="s:string" />
             <s:element name="ErrorMessage" type="s:string" />
        </s:sequence>
     </s:complexType>
</s:element>
```

Task 47: Types

c) element `SendErrorResponse` defined as a complex type, with an element `SendErrorResult` of type string:

```
<s:element name="SendErrorResponse">
   <s:complexType>
     <s:sequence>
        <s:element name="SendErrorResult
                  type="s:string" />
     </s:sequence>
   </s:complexType>
</s:element>
```

3) no more type definitions are needed. Add:

```
    </s:schema>
</wsdl:types>
```

4) save the file

Message Element

A `message` is the construct that describes the abstract form of an input, output or a fault message.

A message describes the data being communicated.

Each message has a unique name within the WSDL document and contains a collection of `parts`.

```
<message name="FahrenheitToCelsiusRequest">
      <part name="tempFahrenheit" type="xsd:float" />
</message>

<message name="FahrenheitToCelsiusResponse">
      <part name="tempCelsius" type="xsd:float" />
</message>
```

A `message` may have several `parts`.

A `part` may belong to several `messages`.

Part Element

Parts provide a flexible mechanism for describing the logical content of messages.

A `part` element has two properties:

1) name - represented by the `name` attribute, which must be unique among all the `part` elements of the `message` element

2) kind - defined as either a `type` or an `element` attribute:
   a) `element` - the payload of the message on the wire is precisely the XML element
   b) `type` - any element conforming to the type

Message Part Example

These are messages for the operation asking for information about weather:

```
<message name="WeatherRequest">
   <part name="userID" type="xsd:string" />
   <part name="city"   type="xsd:string' />
</message>


<message name="WeatherResponse">
  <part name="weather"  element="tns:weatherData" />
</message>
```

Task 48: Message Parts

Objective: Add the message definitions to “MyExample.wsdl”. Four messages are needed to formulate a request and response for both bindings.

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” and add:

   a) two messages `SendErrorSoapIn` and `SendErrorSoapOut`. Each message has a part `parameters`. Both parts are elements of type “SendError”and “SendErrorResponse” respectively.

   b) two messages `SendErrorHttpGetIn` and `SendErrorHttpGetOut`. The first has two parts: `LicenseKey` and `ErrorMessage`, both of type string. The other message has only one part `Body` of type string.

3) save the file

PortType Element

`portType` is a collection of one or more related operations describing the interface of a web service.

`portType` definition is a collection of `operation` elements.

Generally, WSDL documents contain only one `portType` element, because different web service interface definitions are written with different documents.

`portType` has a single `name` attribute.

The name of `portType` together with the namespace of the WSDL document define a unique name for the `portType`.

Operation Element

`operation` defines a method of a web service, including the name of the method, input parameters, and the output or return type of the method.

All operations in a `portType` must have different names.

Each `operation` may define:

1) input message
2) output message
3) fault message

An `operation` in WSDL is the equivalent of a method signature in Java.

PortType Operation Example

The example defines one port type with one operation:

```
<portType name="TemperatureConverter_Service">

    <operation name="FahrenheitToCelsius">
        <input message="FahrenheittoCelsiusRequest"/>
        <output message="FahrenheittoCelsiusResponse "/>
    </operation>

</portType>
```

Notes:

1) operations and messages are modeled separately in order to support flexibility and simplify reuse of existing information

2) two operations with the same parameters can share one abstract message definition

Task 49: PortTypes and Operations

Objective: Add two portType definitions to “MyExample.wsdl”, one for each binding. Both contain the operation `sendError` with the corresponding messages.

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” and add:

   a) one portType `ErrorMailerSoap`

   b) other portType `ErrorMailerHttpGet`

3) save the file

Abstract – Concrete Definitions

We already defined:

1) types
2) messages
3) portTypes
4) operations

} abstract, reusable portions of a WSDL definition

We didn't define yet how to relate these definitions with SOAP headers, SOAP bodies or SOAP encodings:

1) is this service invoked using a SOAP message or a simple HTTP POST of an XML payload?
2) is the service invoked with an RPC or a document style?

These aspects relate to the concrete implementation and are defined using the `binding` element.

Binding Element 1

The `binding` element specifies how to format messages in a protocol-specific manner:

1) message encoding
2) protocol binding

for all operations and messages defined in a given port type.

Each `portType` can have several `binding` elements associated with it.

Each binding specifies how to invoke operations using particular transport protocols. For instance: SOAP over HTTP, SOAP over SMTP, etc.

Binding Element 2

The `binding` element has two attributes:

1) `name` - must be unique among all binding elements defined in the WSDL document

2) `type` - identifies which portType the binding describes

Binding Example

```
<binding
     name="TemperatureConverter_ServiceSOAPBinding"
     type="TemperatureConverter_Service"
    . . .
</binding>
```

relates the binding with the portType

Conventions:

1) the name of the binding is combined with:
   a) the `portType` name (e.g.TemperatureConverter_Service),
   b) the name of the protocol to which the binding maps (e.g.SOAP)
   c) the word "Binding"
2) most WSDL documents contain only a single binding

Binding Protocol

To which protocol is the portType mapped by the binding?
We need to inspect the definitions inside the `binding` element.

The definitions inside the `binding` element are standard WSDL extensions that depends on the binding. WSDL specification describes extensions for:

1) SOAP/HTTP
2) HTTP GET/POST
3) SOAP with MIME attachments

Once the transport protocol is selected, find the WSDL convention that corresponds to the pair and fill in the details.

Most web services define at least a SOAP binding.

SOAP Binding Protocol

The `soap:binding` element has two attributes:

1) `style` - specifies the communication style. The values include:
    a) `document` – operation is document-oriented; messages carry documents that are agreed upon by the two applications
    b) `rpc` – operation is RPC-oriented; messages carry the input parameters and return values of the procedure call

2) `transport`- specifies the communication protocol that is used to transport the messages. The values include:
    1) http://schemas.xmlsoap.org/soap/http
    2) http://schemas.xmlsoap.org/soap/SMTP
    3) http://schemas.xmlsoap.org/soap/ftp
    4) other URI

SOAP Binding Protocol Example

This binding defines that the included operations will use a document-oriented style and HTTP as the communication protocol.

```
<binding
     name="TemperatureConverter_ServiceSOAPBinding"
     type= "TemperatureConverter_Service" >
     <soap:binding style="document"
               transport="http://schemas.xmlsoap.org/soap/http" />
   . . .
</binding>
```

this declaration applies to the whole binding, specifying that all operations defined will be SOAP messages

Different Styles and Restrictions

1) if a `portType` references messages whose `parts` use the `element` attribute, it should only use `style="document"`

2) if a `portType` references messages whose `parts` use the `type` attribute, only a `style="RPC"` should be used.

Binding Protocol Operations

An `operation` element within a binding specifies the binding information for that operation:

```
<operation name="FahrenheitToCelsius">
  <soap:operation soapAction:"urn:temperatureconverter-service"/>
 . . .
</operation>
```

The `soap:operation` element provides information for the operation:

1) `soapAction` attribute specifies the value of the soapAction in the HTTP header for this operation.

Binding Protocol Encoding Rules

The binding also specifies the encoding rules used in serializing parts of a message into XML:

1) literal encoding - takes the WSDL types defined in XML Schema and “literally” uses those definitions to represent the XML content of messages. Abstract WSDL types becomes concrete types

2) SOAP encoding - considers the XML Schema definitions as abstract entities and translates them into XML using SOAP encoding rules

Literal encoding is used for document style interactions.

SOAP encoding is used for RPC style interactions.

One part of the message can be encoded literally in the header and other part can use the SOAP encoding in the body.

Operation Encoding Example

```
<binding
  name="TemperatureConverter_ServiceSOAPBinding"
  type= "TemperatureConverter_Service">
  <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="FarenheitToCelsius">
    <soap:operation soapAction="urn:temperatureconverter-service" />
    <input>
       <soap:body use="literal"/>
    </input>
    <output>
       <soap:body use="literal" />
    </output>
   </operation>
</binding>
```

The content of the input and output messages are sent in the body of the message. The content of the body is literally an XML element.

Only one part was defined for the input and output messages.

Binding Protocol Body Example

The input message that takes a data value 98.0, in the body of the message is described below:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi:"http:// www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <tempFarenheit xmlns="http://www/converter.com/TermperatureConverter">
   98.0</tempFarenheit>
 </soapenv:Body>
</soapenv:Envelope>
```

Task 50: SOAP Binding

Objective: Add the SOAP binding to “MyExample.wsdl”. The communication style is “document” and messages are encoded literally.

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” and add the binding:

```
<wsdl:binding name="ErrorMailerSoap" type="tns:ErrorMailerSoap" >
   <soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
      style = "document" />
   <wsdl:operation name="SendError">
      <soap:operation
          soapAction="http://example.com.ErrorMailer/SendError"
          style = "document" />
      <wsdl:input>  <soap:body use="literal" /> </wsdl:input>
      <wsdl:output> <soap:body use="literal" /> </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

3) save the file

Port Element

The only purpose of the `port` element is to specify the network address of the end-point hosting the web service.

`port` is a single end-point defined as a combination of a binding and a network address.

There can be many ports for a binding, just like many implementations for the same interface.

The `soap:address` element is used to give a port an address.

Port Example

```
<port
  name="TemperatureConverter_ServicePort"
  binding="TemperatureConverter_ServiceSOAPBinding"

    <soap:address
          location://localhost:8080/soap/servlet/TempConverter" />
</port>
```

reference to the binding

network address of the web service

The name identifies the port.

Service Element

A `service` is a collection of `ports`.

Although a WSDL document can contain a collection of `service` elements, by convention a WSDL document contains a single service.

Usage: group the ports that are related to the same service interface (`portType`) but expressed by different protocols (`binding`).

Service Example

```
<service name="TemperatureConverter_Service">
  <port
    binding="TemperatureConverter_ServiceSOAPBinding"
    name="TemperatureConverter_ServicePort">
    <soap:address
          location://localhost:8080/soap/servlet/TempConverter" />
  </port>
  <port>… </port>
</service>
```

Task 51: Service and Ports

Objective: Add the service and port definitions to “MyExample.wsdl”. The address of the service is: http://www.example.com/ErrorMailer/Errormail.

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” adding the service and port definitions. The location is
   `=“http://www.example.com/ErrorMailer/Errormail”`

3) save the file

Documentation Element

The `documentation` element is used to provide useful, human-readable information about the web service description.

Any WSDL element can contain a `documentation` element, usually as its first child.

One conventional use is declaring that the WSDL file is an interoperable description: it is compliant with the WS-I basic profile. This use of the `documentation` element appears in the `service` element.

Documentation Example

```
<service name="TemperatureConverter_Service">
 <port binding="TemperatureConverter_ServiceSOAPBinding"
       name="TemperatureConverter_ServicePort">
   <documentation>
      <wsi:Claim
         conformsTo="http://ws-i.org/profiles/basic/1.0" />
   </documentation>
 <soap:address location://localhost:8080/soap/servlet/TempConverter" />
 </port>
</service>
```

Task 52: Documentation

Objective: Add documentation comments to “MyExample.wsdl”.

```
1) cd \demos\WSDL\Example1
```

2) edit “MyExample.wsdl” adding a documentation element inside the service definition.

3) save the file

Import Element

The `import` element is used to include other WSDL documents or XML Schemas into a WSDL document.

The use of the `import` element allows to:

1) separate the different elements of a service definition into independent documents
2) import these documents as needed

It helps writing clearer WSDL descriptions by separating the definitions according to their level of abstraction and maximizing reusability.

For example, data structures modeled as XML Schemas can be imported by several WSDL documents defining different services.

Import Example

```
<definitions
    name= "TemperatureConverterService"
    targetNamespace="http://www.converter.com/TemperatureConverter"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns=http://www.converter.com/TemperatureConverter”
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <types>
     <xsd:schema>
               <import namespace=“http://www.converter.com/schemas”
                 location=“http://converter.com/temperature.xsd"/>
    </xsd:schema>
    </types>

    ...
</definitions>
```

Import Conventional Use

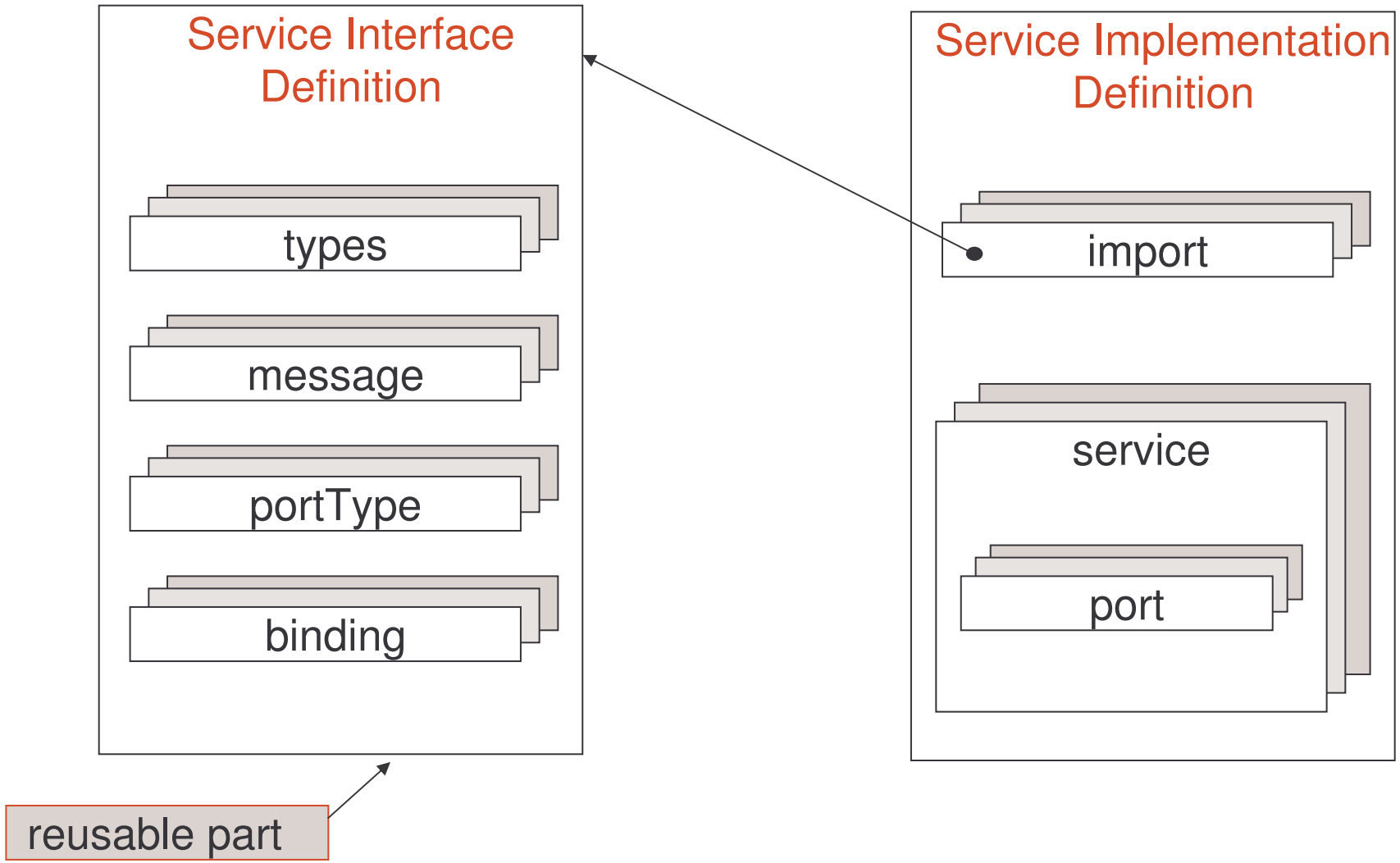
Many designers split their WSDL design into two parts:

1) service interface definition
2) service implementation definition

Interface definition contains: `types`, `message`, `portType` and `binding` elements and encapsulates the reusable components of a service description.

Each organization wanting to implement a web service conformant to this interface definition would describe an implementation definition containing the `port` and `service` elements.

Interface versus Implementation



WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

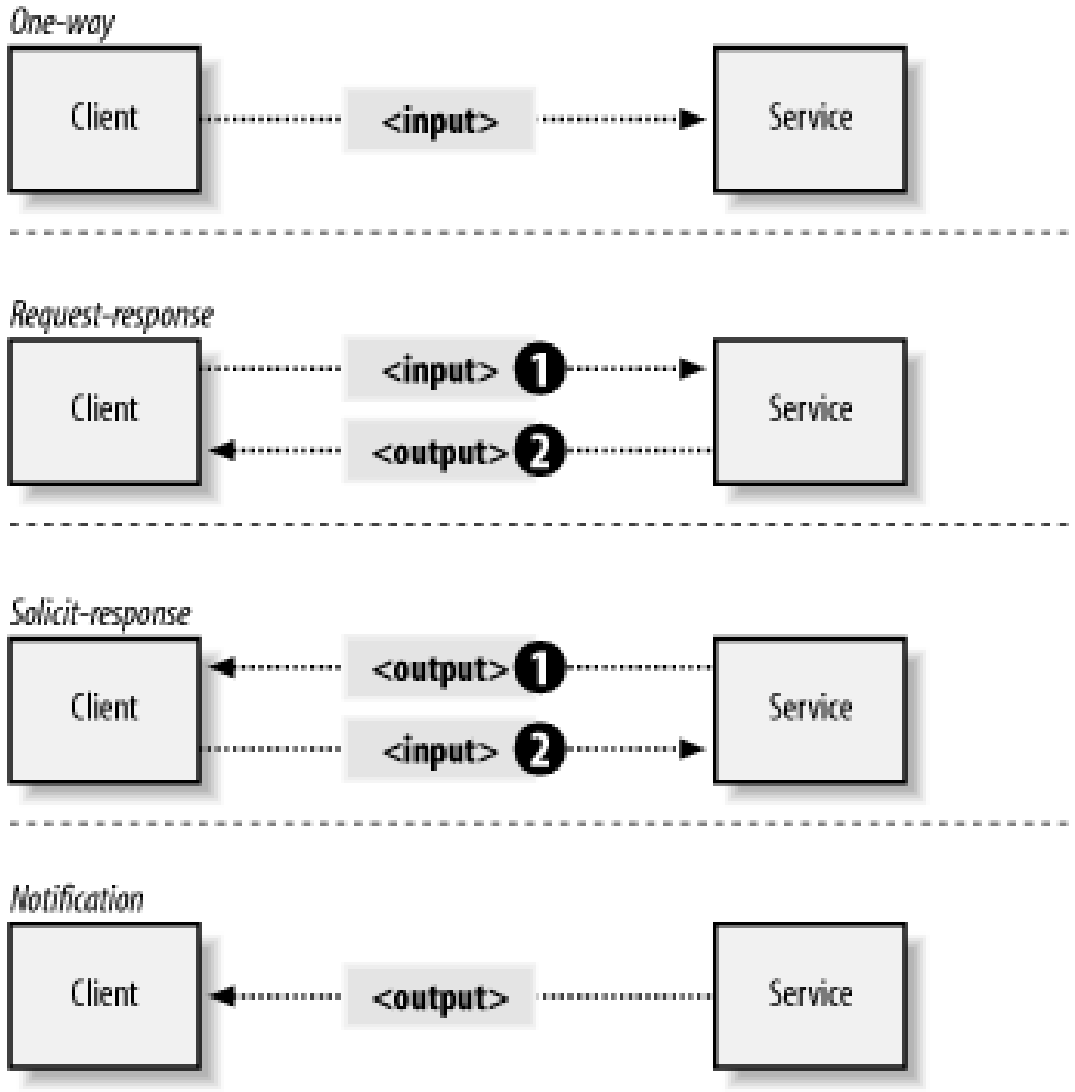
6) Summary

Transmission Primitives 1

WSDL supports four basic operation patterns called transmission primitives:

One way	The service receives a message. The operation has a single input element.
Request-Response	The service receives a message and sends a response. The operation has one input and one output element. To encapsulate errors fault elements can be specified.
Solicit-Response	The service sends a message and receives a response. The operation has one output element and one input element. To encapsulate errors fault element can also be specified.
Notification	The service sends a message. The operation has a single output element.

Transmission Primitives 2



[Courtesy Ethan Cerami]

Web Service Examples

Four different examples will be explained, one for each transmission primitive.

All examples are related to a virtual organization providing informational services about weather to its subscribed users.

WS Example One Way

1) `CancelUser` - a message is received asking to cancel the subscription service for a user.

    a) input message contains user identification and password

WS Example Request-Response

2) `AskData` - a user request for information related to the weather in a particular city and for a date, and receives a response.

   a) input message contains user identification, city and date

   b) output message contains temperature and humidity

WS Example Solicit-Response

3) `ServiceInterruption` - the service provider notifies a user that the service will be interrupted and waits for a response

   a) output message contains notification

   b) input message contains acknowledgement

WS Example Notification

4) `NotifyBadWeather` - the service provider sends a notification to a user when bad weather is forecast in a city where the user lives

    a) output message contains temperature, humidity and notification

WS Example Definitions

WSDL document structure:

```
<?xml version="1.0"?>
<definitions name="WeatherServices"
    targetNamespace="http://www.example.com/WeatherService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.com/WeatherService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  >

  <types>
   ...
  </types>
   ...
   ...
</definitions>
```

It will be reused for all four examples.

WS Example Types 1

Types for all four weather examples:

```
<types>
  <xsd:schema
    targetNamespace="http://www.example.com/WeatherService"
     <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"
      schemaLocation= ="http://schemas.xmlsoap.org/soap/encoding/" />

     <xsd:element name="notification"    type="xsd:string" />
     <xsd:element name="acknowledge"     type="xsd:string" />
     <xsd:element name="errorString"     type="xsd:string" />

     <xsd:complexType name="userData">
       <xsd:sequence>
         <xsd:element name="userId"      type="xsd:string" />
         <xsd:element name="password"    type="xsd:string" />
       </xsd:sequence>
     </xsd:complexType>
```

WS Example Types 2

```
    <xsd:complexType name="dataRequest">
      <xsd:sequence>
        <xsd:element name="city"        type="xsd:string" />
        <xsd:element name="date"        type="xsd:dateTime" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="weatherData">
      <xsd:sequence>
        <xsd:element name="temperature" type="xsd:float" />
        <xsd:element name="humidity"    type="xsd:float" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
...
</definitions>
```

One-Way Operations

A one-way operation has only an input message. It acts like a data sink.

No response message (output or fault) going back to the requestor.

Basic functionality - change the state of the service provider

Many one-way messages at this level end up being request-response messages at the network transport level (response is the HTTP-level acknowledgement).

For a one-way operation, the HTTP response must not contain a SOAP envelope. Most clients will ignore it if it does appear.

One Way Example 1

Service Example: `CancelUser` - a message is received asking to cancel the subscription service for a user.

```
<message name="cancelUser">
  <part name="userCancel" type="tns:userData" />
</message>

<portType name="cancelUserPortType">
   <operation name="cancelUser">
     <input message="tns:cancelUser" />
   </operation>
</portType>


<binding name="CancelUserSOAPBinding"
    type="tns:cancelUserPortType">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />
```

One Way Example 2

```
<operation name="cancellation">
   <soap:operation
      soapAction= "http://www.example.com/WeatherService/cancel" />
   <input>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
</operation>
</binding>


<service name="CancelUser">
   <port name="CancelUser"
         binding="CancelUserSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

Request-Response Operations

The most common form of operation because many web services are deployed using SOAP over HTTP.

Defines:

1) input message (request)
2) output message (response)
3) optional collection of fault messages

Basic functionality:

1) retrieve information about a web service object
2) change the state of the service provider
3) include information about the new state in the response

Like input and output elements, the fault element refers to a message which describes the data contents of the fault.

Request Response Example 1

Service Example: `AskData` - a user request for information related to the weather in a particular city and for a date, and receives a response.

```
<message name="askDataRequest">
  <part name="userIdent"         element="tns:userID" />
  <part name="userRequestData"   type="tns:dataRequest" />
</message>

<message name="askDataResponse">
   <part name="cityDateWeatherData" type="tns:weatherData" />
</message>

<message name="askDataLoginError">
   <part name="errorString" element="xsd:string" />
</message>

<message name="askDataDataError">
   <part name="errorString" element="xsd:string" />
</message>
```

fault messages must have a single part

Request Response Example 2

```
<portType name="askDataPortType">
   <operation name="askData">
     <input message="tns:askDataRequest" />
     <output message="tns:askDataResponse" />
     <fault message="tns:askDataLoginError"
            name="HeaderErrorMessage" />
     <fault message="tns:askDataDataError"
            name="BodyErrorMessage" />
   </operation>
</portType>
```

Request Response Example 3

```
<binding name="AskDataSOAPBinding"
    type="tns:askDataPortType">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="askData">
   <soap:operation
      soapAction= "http://www.example.com/WeatherService/askData" />
   <input>
     <soap:header message="tns:askDataRequest" part="userIdent"
                  use="literal" >
     <soap:headerfault message="tns:HeaderErrorMessage"
           part="errorString" use="literal" />
     </soap:header>
     <soap:body parts="userRequestData" use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
```

a part of the input message on the header

best practice

Request Response Example 4

```
    <output>
      <soap:body use="encoded"
         namespace="http://www.example.com/WeatherService"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
    <fault name="BodyErrorMessage">
      <soap:fault name="BodyErrorMessage"
         namespace="http://www.example.com/WeatherService"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </fault>
</operation>
</binding>

<service name="AskData">
    <port name="AskData"
          binding="AskDataSOAPBinding">
       <soap:address
          location="http://www.example.com/WeatherService" />
    </port>
</service>
</definitions>
```

Solicit-Response Operations

A solicit-response operation models a push operation similar to a notification. It expects an input (response) from the service requestor.

Defines:

1) output message (solicit)
2) optional fault messages (solicit)
3) input message (response)

Basic functionality:

1) notify the service requestor about the result of some event by the service provider
2) waits for an answer from the service requestor

Solicit Response Example 1

Service Example: `ServiceInterruption` - the service provider notifies a user that the service will be interrupted and waits for a response

```
<message name="serviceInterruptionSolicit">
  <part name="interruptionComment" element="tns:notification" />
</message>

<message name="serviceInterruptionResponse">
   <part name="userAcknowledge"    element="tns:acknowledge" />
</message>

<portType name="serviceInterruptionPortType">
   <operation name="serviceInterruption">
     <output message="tns:serviceInterruptionSolicit" />
     <input message="tns:serviceInterruptionResponse" />
</operation>
</portType>
```

Solicit Response Example 2

```
<binding name="ServiceInterruptionSOAPBinding"
    type="tns:serviceInterruptionPortType">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="serviceInterruption">
   <soap:operation
    soapAction="http://www.example.com/WeatherService/Interruption"/>
   <input>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
   <output>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </ouput>
   </operation>
</binding>
```

Solicit Response Example 3

```
<service name="ServiceInterruption">
   <port name="ServiceInterruption"
         binding="ServiceInterruptionSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

Notification Operations

A notification operation is like a one-way operation, but the message is pushed by the service provider.

Output messages are pushed to the service requestor as the result of an event occurring on the service provider side, such as: time-out or operation completion.

Basic functionality - notify the service requestor about an event.

The notification style of interaction is commonly used in systems built around asynchronous messaging.

It is not possible to describe the semantic of these operations in WSDL (where to push the messages) without extensions, except by text comments.

To ensure interoperability of web services, the use of notifications is not recommended.

WS-Notification Example 1

Service Example: `NotifyBadWeather` - the service provider sends a notification to a user when bad weather is forecast in a city where the user lives

```
<message name="notifyBadWeather">
  <part name="notifyWeatherData"  type="tns:weatherData" />
  <part name="notifyCommnent"     element="tns:notification" />
</message>


<portType name="notifyBadWeatherPortType">
   <operation name="notifyBadWeather">
     <output message="tns:notifyBadWeather" />
   </operation>
</portType>


<binding name="NotifyBadWeatherSOAPBinding"
         type="tns:notifyBadWeatherPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />
```

WS-Notification Example 2

```
<operation name="notifyBadWeather">
   <output>
     <soap:body use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </output>
</operation>
</binding>

<service name="NotifyBadWeather">
   <port name="NotifyBadWeather"
         binding="NotifyBadWeatherSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

Operation Default Names

Default names for input and output elements for an operation OP:

	Input	Output
One-Way	OP	not applicable
Request-Response	OPRequest	OPResponse
Notification	not applicable	OP
Solicit-Response	OPResponse	OPSolicit

Fault elements require a name, because several fault elements can be associated with any operation and the fault name is used to distinguish among them.

Operation Parameter Order

Operations using an RPC-binding can specify a list of parameter names via the `parameterOrder` attribute.

The value of this attribute is a space-separated list of message part names with the following rules:

1) the order reflects the order of parameters in the RPC signature
2) the return value is not present in the list
3) if a part name appears in both input and output messages, it is an input/output parameter
4) if a part name appears in only the input message, it is an input parameter
5) if a part name appears in only the output message, it is an output parameter

Parameter Order Example

```
<message name="input">
     <part name="A" element="xsd:int"/>
     <part name="B" element="xsd:long"/>
</message>

<message name="output">
     <part name="A" type="xsd:int"/>
</message>

<portType name="servicePortType">
   <operation name="example" parameterOrder="B A">
      <input message="tns:input"/>
      <output message="tns:output"/>
   </operation>
</portType>
```

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

Functional Extensions

The WSDL language allows most of the WSDL elements to be extended with elements from other namespaces.

The language specification defines standard extensions for:

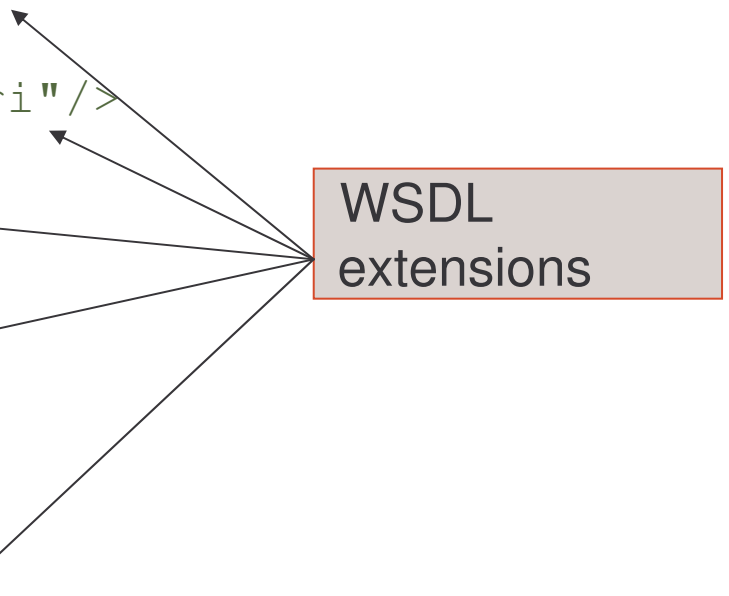
1) SOAP
2) HTTP GET/POST operations
3) MIME attachments

SOAP was already explained.

WSDL HTTP Extension

The HTTP binding extends WSDL with the following elements:

```
<binding .... >
    <http:binding verb="nmtoken"/>
    <operation .... >
      <http:operation location="uri"/>
      <input .... >
          <-- mime elements -->
      </input>
      <output .... >
          <-- mime elements -->
      </output>
     </operation>
    </binding>

    <port .... >
       <http:address location="uri"/>
    </port>
```



HTTP Binding

The `http:binding` element indicates that this binding uses HTTP.

```
<definitions .... >
...
    <binding .... >
        <http:binding verb="nmtoken"/>
    </binding>
...
</definitions>
```

The value of the required `verb` attribute may be GET or POST, or others HTTP requests.

HTTP Operation

An `operation` element within a binding specifies the binding information for that operation.

The `location` attribute specifies a relative URI for the operation.

This URI is combined with the URI specified in the `http:address` element (port definition) to form the full URI for the HTTP request.

The URI value must be a relative URI.

```
<binding .... >
   <operation .... >
       <http:operation location="uri"/>
   </operation>
</binding>
```

HTTP urlEncoded

The `urlEncoded` element indicates that all message parts are encoded into the HTTP request URI using the standard URI-encoding rules.

The names of parameters correspond to the names of the message parts.

Each value contributed by the part is encoded using a `name="value"` pair.

This may be used with GET to specify URL encoding.

For GET, "?" character is automatically appended as necessary.

```
<http:urlEncoded/>
```

```
http://www.example.com/WeatherService/askData?userId=2289193
```

HTTP urlReplacement

The `urlReplacement` element indicates that all message parts are encoded into the HTTP request URI using a replacement algorithm:

1) The relative URI value of `http:operation` is searched for a set of search patterns.
2) The search occurs before the value of the `http:operation` is combined with the value of the location attribute from `http:address`.
3) There is one search pattern for each message part. The search pattern string is the name of the message part surrounded with parenthesis.
4) For each match, the value of the corresponding message part is substituted for the match at the location of the match.
5) Matches are performed before any values are replaced; replaced values do not trigger additional matches.

Message parts MUST NOT have repeating values.

```
<http:urlReplacement/>
```

HTTP mime:content

The `mime:content` element's attribute `type="valid_type"` indicates that the message will appear in the HTTP code as the `valid_type`.

Examples:

1) the message is XML text in the HTTP response:

```
<output>
   <mime:content type="text/xml" />
</output>
```

2) the message is send as a gif file:

```
<output>
      <mime:content type="image/gif"/>
</output>
```

HTTP Address

The `location` attribute of the `http:address` element specifies the base URI for the port.

The value of the attribute is combined with the values of the `location` attribute of the `http:operation` binding element.

```
<port name="...">
  <http:address location="URI" />
</port>
```

HTTP Binding Example

```
<binding name="AskDataHTTPBinding"
         type="tns:askDataPortType">
    <http:binding verb="GET"/>
    <operation name="askData">
     <http:operation location= "askData" />
     <input>
        <http:urlEncoded/>
     </input>
     <output>
        <mime:content type="text/xml"/>
     </output>
   </operation>
</binding>


<service name="AskData">
   <port name="AskData"
         binding="AskDataHTTPBinding">
      <http:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
```

Task 53: HTTP/GET Binding

Objective: Add the HTTP/GET binding to “MyExample.wsdl”. The location is “`/SendError`”, the verb is “`GET`” and the message parts in the HTTP request are encoded.

1) `cd \demos\WSDL\Example1`

2) edit “MyExample.wsdl” and add the `ErrorMailerHttpGet` binding

3) save the file

Task 54: Port for HTTP Binding

Objective: Add the port definition of the service for the HTTP/GET binding, to "MyExample.wsdl". The location is: "`http://www.example.com/ErrorMailer/Errormail`".

1) `cd \demos\WSDL\Example1`

2) edit "MyExample.wsdl" and add the port definition and the final tag of the `definitions` element

3) save the file

Functional Extensions

The WSDL language allows most of the WSDL elements to be extended with elements from other namespaces.

The language specification defines standard extensions for:

1) SOAP
2) HTTP GET/POST operations
3) MIME attachments

SOAP and HTTP were already explained.

WSDL MIME Extension

WSDL also supports a standard extension to describe message parts as MIME.

This extension could be used to include a GIF image as part of a message.

Example: we would like to add a map in a graphical file when sending data about the weather.

The response message must include the new part:

```
<message name="askDataResponse">
   <part name="cityDateWeatherData" type="tns:weatherData" />
   <part name="weatherpicture"      type="xsd:binary" />
</message>
```

WSDL with MIME 1

MIME extensions are only for the binding, indicating that the output is modeled as multipart MIME.

no changes

```
<binding name="AskDataSOAPBinding"
         type="tns:askDataPortType">
    <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="askData">
   <soap:operation
      soapAction= "http://www.example.com/WeatherService/askData" />
   <input>
     <soap:header message="tns:askDataRequest" part="userIdent"
                  use="literal" >
       <soap:headerfault message="tns:HeaderErrorMessage"
          part="errorString" use="literal" />
     </soap:header>
     <soap:body parts="userRequestData" use="encoded"
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
   </input>
```

WSDL with MIME 2

```
<output>
    <mime:multipartRelated>
     <mime:part>
     <soap:body use="encoded" />
        namespace="http://www.example.com/WeatherService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </mime:part>
      <mime:part>
        <mime:content part="weatherpicture" type="image/gif" />
        <mime:content part="weatherpicture" type="image/jpeg" />
      </mime:part>
    </mime:multipartRelated>
  </output>
  <fault name="BodyErrorMessage">
    <soap:fault name="BodyErrorMessage"
       namespace="http://www.example.com/WeatherService"
       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </fault>
</output>
</binding>
```

changes

Task 55: WSDL Example 1

Objective: Access to a complete WSDL document published on the web.

1) browse: http://www.errormail.net/EM/ErrorMailer.asmx?wsdl

2) analyze the document

Task 56: WSDL Example 2

Objective: Access the WSDL document of `FileDownloadService`.

1) start Tomcat

2) browse: http://localhost:8080/axis

3) access: view → FileDownloadService (wsdl)

4) analyze the document

Non-Functional Descriptions

How to describe:

a) security requirements

b) transactional capabilities

c) logging features for the invocation of the service

d) auditing realized by the service provider

Non-functional characteristics of a web service can be described using WS-Policy and related specifications.

WS-Policy

WS-Policy version 1.0 was originally published in December 2002, by BEA, IBM, Microsoft, and SAP.

In May 2003, version 1.1 was published.

The WS-Policy family of specifications has three major components:

1) framework

2) assertions

3) attachment

The basic component of the policy framework is a policy assertion.

Policy Assertion

Policy assertion is a concrete statement about requirements, preferences, capabilities and other characteristic of a web service or its operating environment.

Policy assertions describe certain qualities of service such as reliability of messaging or security aspects.

A policy assertion may be a simply statement of fact:

```
<wsrm:DeliveryAssurance Value="wsrm:ExactlyOnce"/>
```

Also, a policy assertion may be a complicated statement, indicating possible sets of requestor-specifiable parameters, etc.

Policy assertions are grouped together to form a policy.

Policy Assertion Example

Two policy assertions are specified:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
```

1) the subject requires a Kerberos V5 service ticket token

```
   <wsse:SecurityToken wsp:Usage="wsp:Required">
       <wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
   </wsse:SecurityToken>
```

2) an XML digital signature is required

```
   <wsse:Integrity wsp:Usage="wsp:Required">
      <wsse:Algorithm Type="wsse:AlgSignature"
            URI="http://www.w3.org/2000/09/xmlenc#aes" />
   </wsse:Integrity>
</wsp:Policy>
```

Policy and Policy Subject

A policy forms a named collection of policy assertions that can be referenced using standard XML mechanisms, by other XML and Web service components such as a WSDL definition.

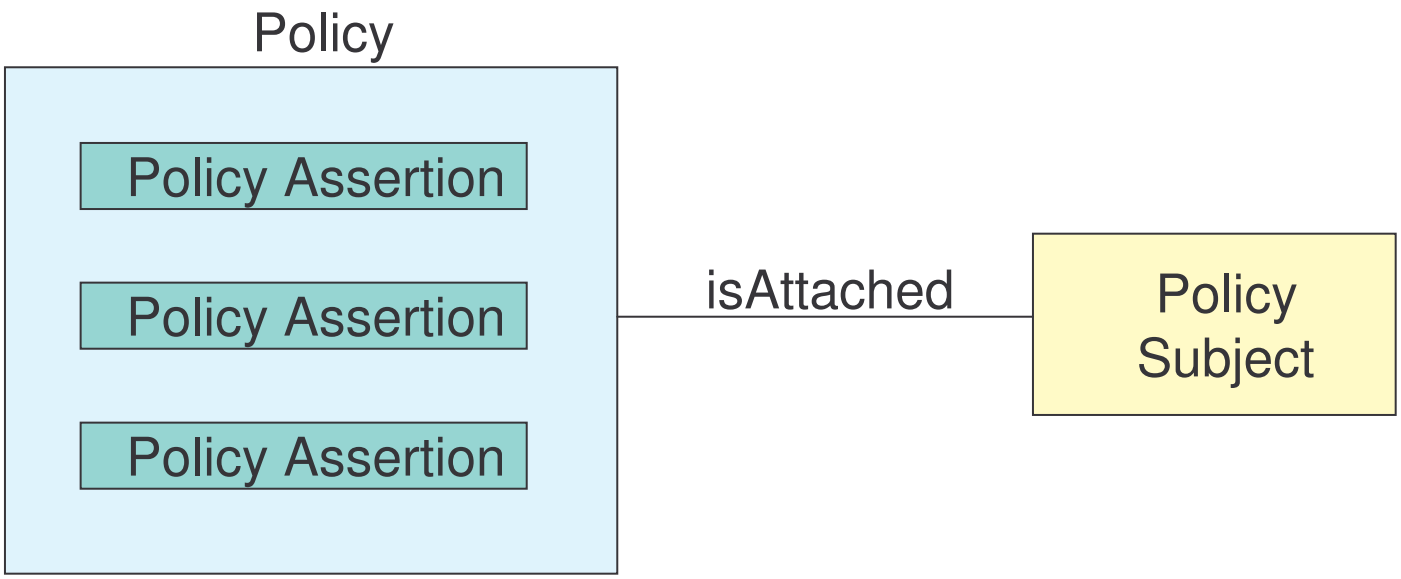
One mechanism to reference a policy is to associate a policy with a policy subject.

A policy subject can be:

1) web service
2) component of a web service description
3) part of a web service's operating environment
4) other entities related to a web service

WS-PolicyAttachments specification describes how to associate policies with policy subjects.

Policy and Policy Subjects



WS-Policy Framework

WS-Policy defines how to group policy assertions into a named collection that can be referenced by other components.

WS-Policy framework consists of:

1) an XML element to act as a container for one or more policy assertions

2) a set of XML elements that describe how the policy assertions grouped by the container are to be combined

3) a set of standard XML attributes that may be associated with policy assertions

Policy Container

General Form:

```
<wsp:Policy  ((name="..." TargetNamespace= "...") | Id="...") >
   <policy-specific assertion>
                ...
   <policy-specific assertion>
   <policy-specific security>
</wsp:Policy>
```

Defines:

1) the policy name
2) policy assertions
3) security policy assertions specific to this policy element

It is assumed that policy assertions are completely independent

Policy operators are needed to provide semantics to policy assertions

Policy Name

Two standard mechanisms to name a policy:

1) by XML QName

```
<wsp:Policy  name="..." TargetNamespace= "..."  >
```

2) by URI – combined with the XML base of the document

```
<wsp:Policy xml:base="http://example.com" Id="Pol1">
```

The URI will be: `http://example.com#Pol1`

One mechanism should be chosen and followed through all the policy work.

Policy Operators

Four operators exists to describe different combinations of policy assertions:

1) All
2) ExactlyOne
3) OneOrMore
4) the basic policy element

Policy Operators Example 1

The following example:

1) defines a policy named Example1 in http://example.com namespace

2) states that all assertions A,B, and C are in effect

```
<wsp:Policy
     name="Example1"
     TargetNamespace="http://example.com" >
<wsp:All>
   <Assertion A/>
   <Assertion B/>
      <Assertion C/>
</wsp:All>
</wsp:Policy>
```

For a policy assertion to be in effect is entirely dependent on the domain of each policy assertion and the policy subject to which the policy is attached

Policy Operators Example 2

The following example:

1) defines a policy named Example2 in http://example.com namespace

2) states that exactly one of the assertions A,B, and C is in effect

```
<wsp:Policy
     name="Example2"
     TargetNamespace="http://example.com" >
<wsp:ExactlyOne>
   <Assertion A/>
   <Assertion B/>
       <Assertion C/>
</wsp:ExactlyOne>
</wsp:Policy>
```

Operators can be nested – any of the assertion elements can be replaced by an operator.

Policy Attributes

WS-Policy framework provides a pair of global XML attributes : `Usage` and `Preference`

These attributes can be added to the various policy assertions.

Usage Attribute

`Usage` describes how the policy assertion is to be interpreted in the context of the policy. Possible values:

1) `Required`: the assertion must apply or an error occurs

2) `Rejected`: the assertion must not apply or an error occurs

3) `Optional`: the assertion may apply or may not apply

4) `Observed`: let the requestor know that a particular assertion will be applied

5) `Ignored`: tell the requestor that if something happens to cause the policy assertion to be in effect then no error message will be emitted

Usage Attribute Example

This policy specifies that assertion A must apply, assertion B must not apply, and assertion C may or may not apply.

```
<wsp:Policy
     name="Example3"
     TargetNamespace="http://example.com" >
  <wsp:ExactlyOne>
      <Assertion A wsp:Usage="Required"/>
      <Assertion B wsp:Usage="Rejected" />
       <Assertion C wsp:Usage="Optional" />
  </wsp:ExactlyOne>
</wsp:Policy>
```

Preference Attribute

This attribute is used in conjunction with the `ExactlyOne` operator.

If there is a choice between a set of policy assertions, this value acts as a hint to the requestor.

The value is integer. The higher the number, the stronger the preference.

Preference Attribute Example

This policy specifies that the requestor has a choice of assertions A, B, and C, and that the service provider would much prefer the requestor to choice assertion A.

```
<wsp:Policy
     name="Example3"
     TargetNamespace="http://example.com" >
  <wsp:ExactlyOne>
      <Assertion A wsp:Preference="100"/>
      <Assertion B wsp:Preference="50" />
       <Assertion C wsp:Preference="10" />
  </wsp:ExactlyOne>
</wsp:Policy>
```

Policy Example

This policy indicates that the subject requires exactly one security token, either a UsernameToken, x509 security token or Kerberos.

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
 <wsp:ExactlyOne wsp:Usage="Required">
    <wsse:SecurityToken>
       <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Preference="10">
       <wsse:TokenType>wsse:x509v3</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken wsp:Preference="1">
       <wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
    </wsse:SecurityToken>
 </wsp:ExactlyOne>
</wsp:Policy>
```

The preference values indicate that the preferred token type is x509, followed by Kerberos, followed by UsernameToken.

Referencing Policies

WS-Policy defines `PolicyReference` elements that allows to include the contents of one policy in another.

The `PolicyReference` element can appear anywhere a policy assertion can appear, and it refers to another policy.

The meaning is that the contents of the included policy element are wrapped with an `All` operator element and inserted in the place of the reference.

Referencing Policies Example 1

Consider this policy specification:

```
<wsp:Policy
     name="Example4"
     TargetNamespace="http://example.com"
     xmlns:tns="http://www.example.com/policies" >
  <wsp:ExactlyOne>
      <wsp:PolicyReference Ref="tns:Example2" />
      <Assertion X />
      <Assertion Y />
   </wsp:ExactlyOne>
</wsp:Policy>
```

Referencing Policies Example 2

It is equivalent to:

```
<wsp:Policy
      name="Example4"
      TargetNamespace="http://example.com"
      xmlns:tns="http://www.example.com/policies" >
  <wsp:ExactlyOne>
      <wsp:All>
         <wsp:ExactlyOne>
              <Assertion A/>
              <Assertion B/>
               <Assertion C/>
         </wsp:ExactlyOne>
      </wsp:All>
       <Assertion X />
       <Assertion Y />
   </wsp:ExactlyOne>
</wsp:Policy>
```

```
<wsp:Policy name="Example2" ...>
  <wsp:ExactlyOne>
     <Assertion A/>
     <Assertion B/>
     <Assertion C/>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Only one of assertions A, B, C, X or Y must be in effect.

Different Policy Assertions

Discipline-specific - policy assertions selected, configured and combined into a policy document, such as:

1) security policy assertions,
2) reliability policy assertions,
3) etc.

Generic assertions - four standard policy assertions defined by WS-Policy in a separate specification called WS-PolicyAssertions

Generic Policy Assertions

Four generic policy assertions:

1) text encoding - declares which character set is used for text that appears in web service messages

2) language assertion - declares the human language expected in messages

3) spec assertion - declares which version of a particular technical specification a web service is compliant with

4) message predicate - declares the exact content of a message going into or coming out of a web service. The contents are defined using the XPath language.

Message Predicate Example

This policy requires:

1) exactly one **wsse:Security** header element, and
2) exactly one child element within the **soap:Body** element

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
  <wsp:MessagePredicate wsp:Usage="wsp:Required">
    count(wsp:GetHeader(.)/wsse:Security) = 1
  </wsp:MessagePredicate>
  <wsp:MessagePredicate wsp:Usage="wsp:Required">
    count(wsp:GetBody(.)/*) = 1
  </wsp:MessagePredicate>
  ...
</wsp:Policy>
```

Task 57: Policy

Objective: Write a policy “MyPolicy” specifying that messages must be sent using UTF-8 encoding, SOAP 1.2, and optionally a digital signature.

1) `cd \demos\WSDL\Example1`

2) create MyPolicy.xml for the specified requirements, using the following namespaces:

```
xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy”
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext”
```

Policy Attachments

A policy can be attached to a policy subject, such as WSDL `portType`, WSDL `message`, UDDI elements or others.

A policy can be attached to a policy subject in two ways:

1) as part of the subject's definition

2) external to the subject's definition

Policy Attachment Example 1

Suppose we want to declare that the language expected for a web service can be English, Chinese or Spanish:

```
<wsp:Policy name = "WeatherLanguages"
            TargetNamespace = "http://www.example.com/policies" >
  <wsp:OneOrMore>
     <wsp:Language Language="en" />
     <wsp:Language Language="cn" />
     <wsp:Language Language="es" />
  </wsp:OneOrMore>
</wsp:Policy>
```

Policy Attachment Example 2

The policy can be referenced from within the `AskData` service declaration, using the `PolicyRefs` attribute:

```
<service name="AskData"
   wsp:PolicyRefs="pol:WeatherLanguages"
   xmlns:wsp="http://schemas.xmloap.org/ws/2002/12/policy"
   xmlns:pol="http://www.examples.com/policies">
   <port name="AskData" binding="AskDataSOAPBinding">
      <soap:address
         location="http://www.example.com/WeatherService" />
   </port>
</service>
</definitions>
```

The `policyRefs` attribute takes a list of QNames allowing to associate a collection of policies to any policy subject.

The `policyURIs` attribute provides the same functionality allowing to associate a collection of policies identified by URI with any policy subject.

Task 58: Policy Attachment

Objective: Attach “MyPolicy” to the Error Mail Service.

1) `cd \demos\WSDL\Example1`

2) edit MyExample.wsdl and below the service definition, add the reference to the policy

3) save the file

Effective Policy

A policy attached to a WSDL element can be inherited by its child elements.

For instance, a policy attached to a `portType` would be inherited by its `input`, `output` and `fault` child elements.

Effective policy is the policy associated with a WSDL element. It can be an inherited policy or a policy directly attached to the element.

Policy Inheritance in WSDL 1

WSDL Element	Effective Policy
`message`	policy associated with `message`
`message/part`	policy associated with `part`, merged with the effective policy of the `part`'s `message` parent
`portType`	policy associated with `portType`
`portType/operation`	policy associated with `operation`, merged with the effective policy of the `operation`'s `portType` parent
`portType/operation/input`	policy associated with `input`, merged with the effective policy of the `input`'s `operation` parent and with the effective policy of the `message` associated with the `input` element
`portType/operation/output`	similar to input
`portType/operation/fault`	similar to input

Policy Inheritance in WSDL 2

WSDL Element	Effective Policy
`binding`	policy associated with the `binding` merged with the effective policy of the associated `portType`
`binding/operation`	policy associated with the `operation` merged with the effective policy of the `operation`'s `binding` parent and merged with the effective policy of the `portType operation`
`binding/operation/input`	policy associated with the `input` merged with the effective policy of the `input`'s `operation` parent and merged with the effective policy of the corresponding `portType/operation/input`
`binding/operation/output`	similar to input
`portType/operation/fault`	similar to input
`service`	policy associated with the `service`
`service/port`	policy associated with the `port` merged with the effective policy of the `port`'s `service` parent

External Policy Attachment 1

Allows policies to be associated with a policy subject independent of that subject's definition and/or representation through the use of a `PolicyAttachment` element.

The `PolicyAttachment` element has three components:

1) the policy scope of the attachment

2) the policy expressions being bound

3) security information (optional)

External Policy Attachment Example

```
<wsp:PolicyAttachment ... >
  <wsp:AppliesTo>
    <x:DomainExpression/> +
  </wsp:AppliesTo>
 ( <wsp:Policy>...</wsp:Policy>
    <wsp:PolicyReference>...</wsp:PolicyReference> ) +
    <wsse:Security>...</wsse:Security> ?
        ...
</wsp:PolicyAttachment>
```

policy scope

policy expressions

security information

```
<wsp:PolicyAttachment>
 <wsp:AppliesTo>
    <wsa:EndpointReference
           xmlns:ad="http://www.example.com/WeatherService" >
      <wsa:Address>http://www.example.com/WeatherService/askData
      </wsa:Address>
      <wsa:PortType>ad:askDataPortType</wsa:PortType>
      <wsa:ServiceName>ad:AskData</wsa:ServiceName>
    </wsa:EndpointReference>
 </wsp:AppliesTo>
 <wsp:PolicyReference URI="http://www.example.com/policies#ASKDATAPOL"/>
</wsp:PolicyAttachment>
```

WSDL: Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

WSDL Mapping to Java

Many tools maps WSDL to Java, both sides:

1) service requestor
2) service provider.

For instance: the Axis WSDL2Java tool.

Some conventions are defined for mapping:

1) portTypes
2) operations
3) messages
4) bindings

portType to Java

1) the `portType` naturally maps into a Java Interface

2) the name of the interface typically takes the name of the `portType`

3) the file is declared in a package named from the targetNamespace URI of the WSDL `definitions` element containing the `portType`

portType to Java Example

The WeatherService.wsdl where the `portType AskData` was defined, include:

```
<?xml version="1.0"?>
<definitions name="WeatherServices"
 targetNamespace=http://www.example.com/WeatherService
```

Thus a piece of the Java Interface for the `AskData PortType` is:

```
Package com.example.www.WeatherService.AskData;

Public interface AskDataPortType extends java.rmi.Remote {
```

Operation to Java

1) for each of the `portType`'s operations, a public method is declared as part of the interface

2) the signature of the method is built from:

   a) the name of the operation

   b) the input and output values defined in the operation

   c) any fault element associated with the operation are included as exceptions thrown by the method.

Operation to Java Example

The method signature generated from the `askData` operation defined in `askDataPortType` is:

```
public com.example.www.WeatherService.cityDateWeatherData
  askData (com.example.www.WeatherService.userID uid,
           com.example.www.WeatherService.dataRequest uRequestData)
  throws java.rmi.RemoteException;
```

Message to Java

1) for those messages that are referenced by `input` and `output` elements, a class is generated for `complexTypes` referenced by the parts of those messages

2) a class is generated for those messages referenced in `fault` elements.

3) the name of the class is taken from the name of the type or element

4) the package from the class is taken from the targetNamespace URI of the XML schema that defines the type or element.

These type-based classes are used as part of the mechanism to serialize and deserialize XML to and from Java.

Message to Java Example

Input element:

```
<input>
  <soap:header message="tns:askDataRequest" part="userIdent"
               use="literal" >
  <soap:headerfault message="tns:HeaderErrorMessage"
        part="errorString" use="literal" />
  </soap:header>
  <soap:body parts="userRequestData" use="encoded"
     namespace="http://www.example.com/WeatherService"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
```

Classes in Java:

1) userId
2) dataRequest
3) fault

defined in package: com.example.www.WeatherService

Binding to Java

1) each binding is generated as a stub class

2) the name of the class is the name of the binding

3) the `targetNamespace` of the `definitions` element is used to define the name of the package

4) this class implements the interface defined by `portType`

This class is a proxy to the service – encapsulates the implementation details associated with how a given `portType` is made concrete by the binding.

Binding to Java Example

```
<binding name="AskDataSOAPBinding"
    type="tns:askDataPortType">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />

   <operation name="askData">
    . . .
```

A class called `AskDataSOAPBindingImpl` is defined.

Implements the interface defined by the `portType askDataPortType`.

Service to Java

The WSDL `service` also generates an interface and class.

They encapsulate details of invoking the service from the client application.

WSDL Outline

1) Introduction

2) The Language
   a) structure
   b) definitions
   c) types
   d) message
   e) part
   f) port type
   g) operation
   h) binding
   i) port
   j) service
   k) documentation
   l) import

3) Transmission Primitives
   a) one way
   b) request-response
   c) notification
   d) solicit-response

4) WSDL Extensions
   a) functional extensions
   b) non-functional extensions

5) WSDL and Java

6) Summary

WSDL Summary 1

WSDL is an XML-based language for describing and accessing web services.

WSDL describes four pieces of critical data:

1) data type declarations for all requests and response messages

2) interfaces describing available functions

3) binding information about the transport protocol

4) address information for locating the service

WSDL Summary 2

The service description is an XML document where the root element called `definitions` may include the following elements:

1) `types` - user-defined types and elements used in messages
2) `message` - composed of `parts`, describes data exchanged
3) `portType` - collection of related operations describing a WS interface
4) `binding` - specification of how to format messages in a protocol-specific manner
5) `service` - a collection of ports specifying network addresses of the end-points hosting the web service
6) `documentation` - human-readable information about the WS
7) `import` - including other WSDL documents or XML Schemas documents

WSDL Summary 3

Conventional use of WSDL includes:

1) Service interface definition:

    a) types
    b) message
    c) portType
    d) binding

2) Service implementation definition:

    a) service and port

WSDL Summary 4

WSDL supports four patterns of operation called transmission primitives:

1) one way - only an input message

2) request-Response - input-output messages and optional faults

3) notification - only an output message

4) solicit-Response - output-input messages and optional faults

WSDL Summary 5

WSDL provides functional extensions for:

1) SOAP

2) HTTP GET/POST operations

3) MIME attachments

WSDL Summary 6

Non-functional aspects of a web service can be specified using WS-Policy and related specifications:

1) A policy assertion describes certain aspects of a service quality: reliability, security, etc.

2) Policy assertions are grouped to form a policy.

3) WS-Policy also specifies how policies can be referenced by other components.

4) A policy can be attached to a subject, such as: WSDL `portType` and `message`, UDDI elements, or others in two ways:
    a) as part of the subject definition
    b) external to the subject definition using WS-PolicyAttachments

AXIS

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

Axis Overview

Axis is a SOAP engine.

Axis consists of several subsystems working together for processing messages.

The Axis engine invokes in sequence a series of handlers to process messages.

The order in which handlers are invoked is determined by:

1) deployment configuration
2) whether the engine is a client or a server

Axis History

IBM contributed with an early implementation of the SOAP protocol to Apache in 1999, known as Apache SOAP.

This implementation was based on SOAP4J and was written in a monolithic style.

Apache Community decided to make re-engineering of this code, and Apache SOAP 2.1 emerged.

The development team started a major refactoring and redesign of the codebase and it was supposed to be called Apache SOAP 3.0.

Axis (Apache eXtensible Interaction System) was chosen instead of Apache SOAP 3.0.

Axis Architecture

A chain of message-processing components that can be developed separately and assembled at deployment time.

These components are called handlers.

Axis replaced the Apache SOAP's DOM-based XML processing used in predecessors, to faster SAX system.

Axis Handlers

Axis handlers tell the engine how to deal with messages that need to be processed.

Handlers can be:

1) built-in the engine
2) included in a module defined by the user

Handlers may:

1) send a request and receive a response
2) process a request and produce a response - called pivot point of the sequence of messages

Handlers Implementations

Handlers are Java classes based around a simple abstract class:

```
apache.axis.handlers.BasicHandler
```

A handler implementation, overrides the following method:

```
void invoke (MessageContext context) throws AxisFault;
```

When a handler is invoked, it may execute different functions:

1) reading and writing pieces of SOAP message,
2) logging information to a database,
3) checking user's identification credentials,
4) . . .

Different Types of Handlers

Handlers are:

1) transport-specific
2) service-specific
3) global

Handlers and Chains

Handlers can be combined into chains.

A chain is a pre-defined ordered collection of handlers.

The overall sequence of handlers comprises three chains:

1) transport
2) global
3) service

Chains

The Axis engine processes chains in the same way as handlers.

A chain class groups handlers together.

The chain class also implements the `Handler` interface.

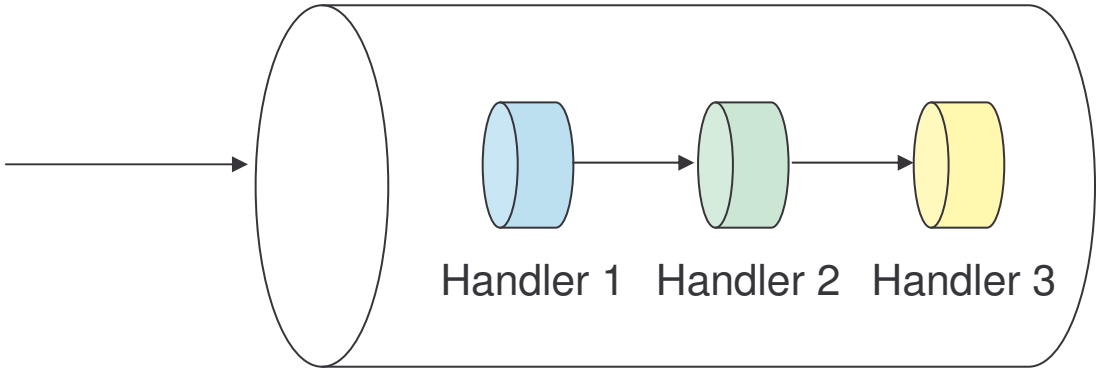
When a chain's `invoke` method is called, it calls the `invoke` method on each of its constituent handlers, which themselves might be chains.

Axis uses two types of chains:

1) simple chains
2) targeted chains

Simple Chains

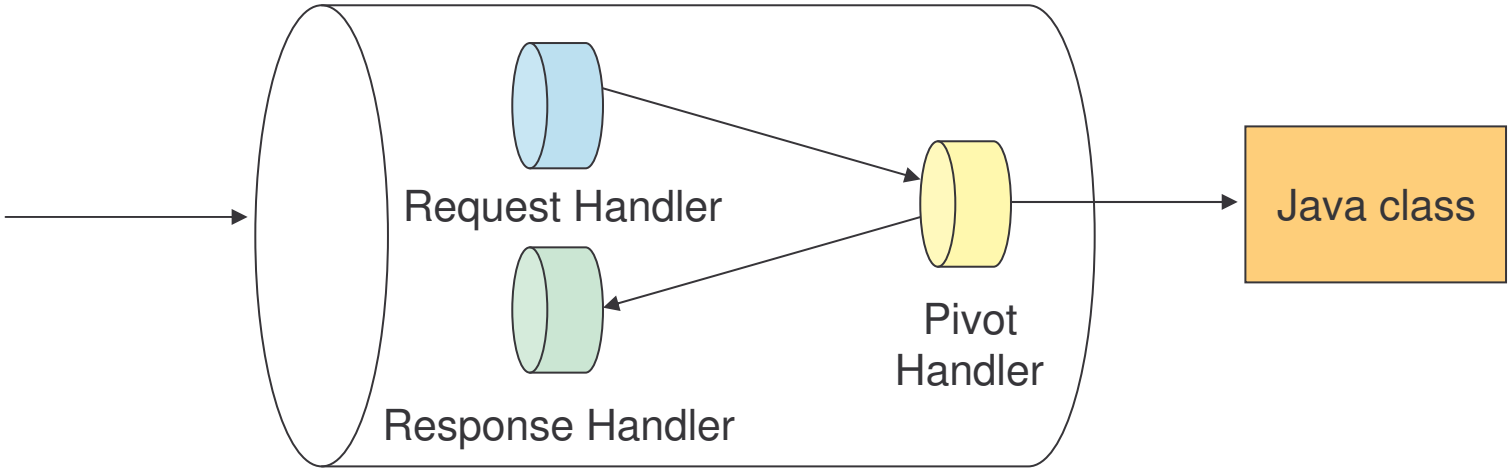
A simple chain is a list of handlers that should be invoked in order.



Targeted Chains

A targeted chain has exactly three handlers:

1) request hander: does the pre-processing work
2) pivot handler: the place where real work is done
3) response handler: does the post-processing work



Deployed services in Axis are invoked by targeted chains. The pivot handler calls the Java class that is exposed as a web service.

Message Context

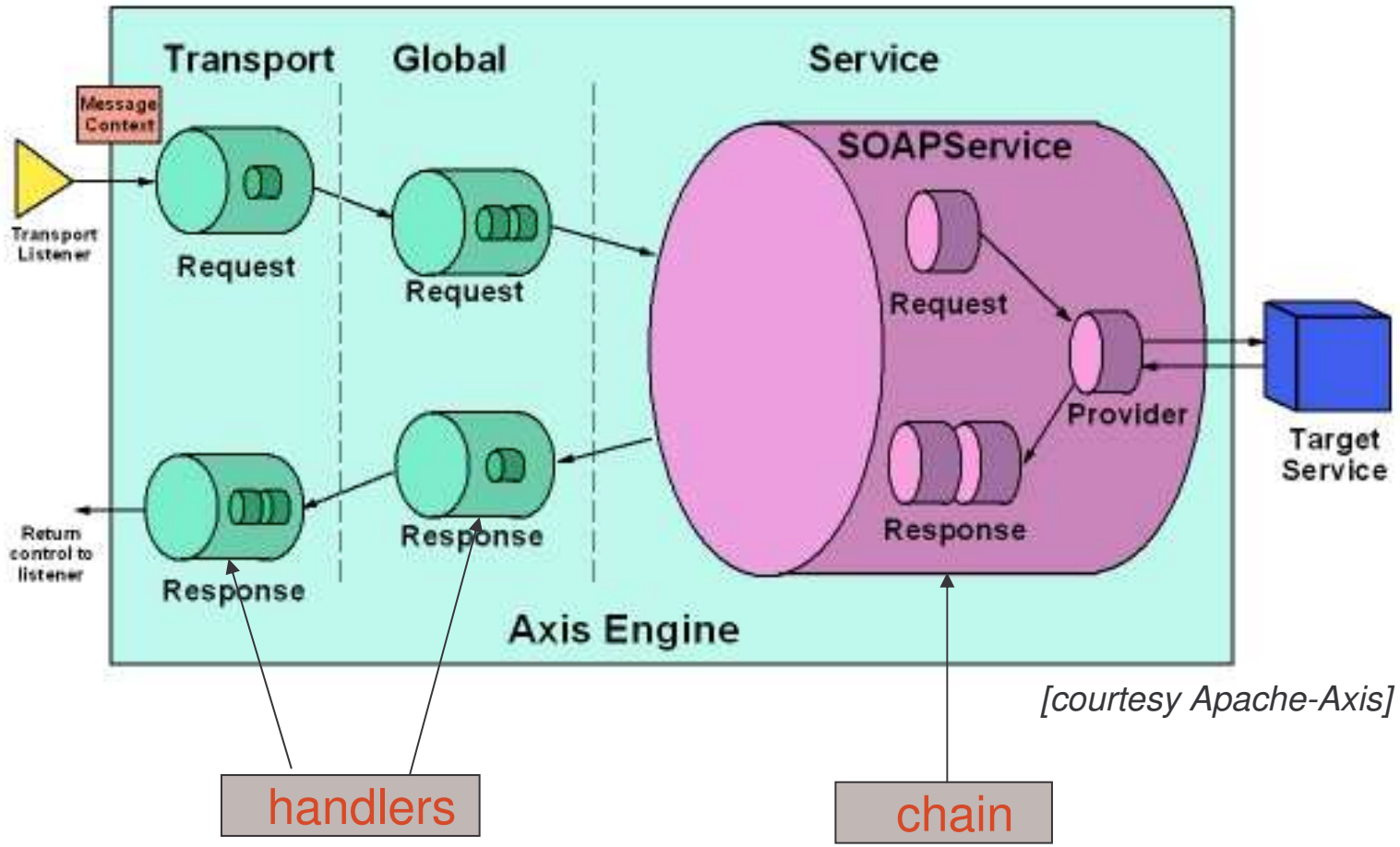
The object passed to each handler is called `MessageContext`.

`MessageContext` is a structure which contains:

1) a request message
2) a response message
3) several properties
4) other fields

Axis processing framework passes the `MessageContext` through the set of handlers that are configured in the engine.

Message Path on the Server



[courtesy Apache-Axis]

Server Side – Listener Request

A message arrives at a transport listener - any software that can take input messages and handle them to Axis.

A transport listener:

– implements a particular SOAP binding

1) packages protocol-specific data into a `Message` object and puts it into a `MessageContext`

2) sets some properties in the `MessageContext`:
   a) `http.SOAPAction` - with the value defined in the HTTP header
   b) `transportName` - “http”
   c) other general properties such as reception-time, etc.

3) hands the `MessageContext` to the AxisEngine

A built-in HTTP listener:

`org.apache.axis.transport.http.AxisServlet`

Server Side – Transport Level

The AxisEngine:

a) looks for a transport chain whose name matches the `transportName` in the `MessageContext`

b) if a transport chain exists, the engine invokes the request handler of this chain passing it the `MessageContext`

This allows the server to implement transport-specific processing.

Transport-specific processing consists of any work that closely relates to the transport over which the message was received.

Example: any process dealing with HTTP headers for an HTTP transport, for instance HTTP authentication.

Server Side – Global Level

After the transport-specific request processing completes without error, the server passes the `MessageContext` to the global request chain.

This chain contains handlers that process all incoming messages, regardless the transport.

The global chain may be used to implement common features to all messages, such as: security or logging.

Server Side – Service Level

After the global chain is finished, the server calls the service handler.

The service handler is a special kind of wrapper called a SOAPService.

The SOAPService class is a targeted chain, including:

1) request chain – allows to insert pre-processing handlers specific for the service invoked

2) provider handler – is the pivot handler that calls the service class

3) response chain – allows to insert post-processing handlers specific for the service that is invoked

Server Side – Listener Response

After the Axis engine processes the message, the control is passed again to the listener.

The listener:

1) takes the message out from the `MessageContext`

2) sends it back to the client as an HTTP response

Deploying Global Handlers Example

Deployment descriptor to deploy a global handler :

```
<deployment xmlns="..."
  <globalConfiguration>
       <requestFlow>
           <handler type="requestHandler"/>
       </requestFlow>
       <responseFlow>
           <handler type="responseHandler"/>
       </responseFlow>
  </globalConfiguration>

 <handler name="requestHandler" type="java:MyRequestHandler">
    ...
 </handler>

 <handler name="responseHandler" type="java:MyResponseHandler">
   ...
 </handler>
</deployment>
```

Task 59: Handlers' Development

Objective: deploy two global handlers on the server side - request and response handlers. Both handlers write the envelope of the messages they process to a file.

```
1)  cd \demos\Axis\Handlers

2)  dir
deployService.bat
FileTransferRequest.class
MyHandlers.wsdd
MyRequestHandler.class
MyResponseHandler.class
```

Analyze the deployment file:

```
3) edit MyHandlers.wsdd
```

Task 60: Deploy Global Handlers

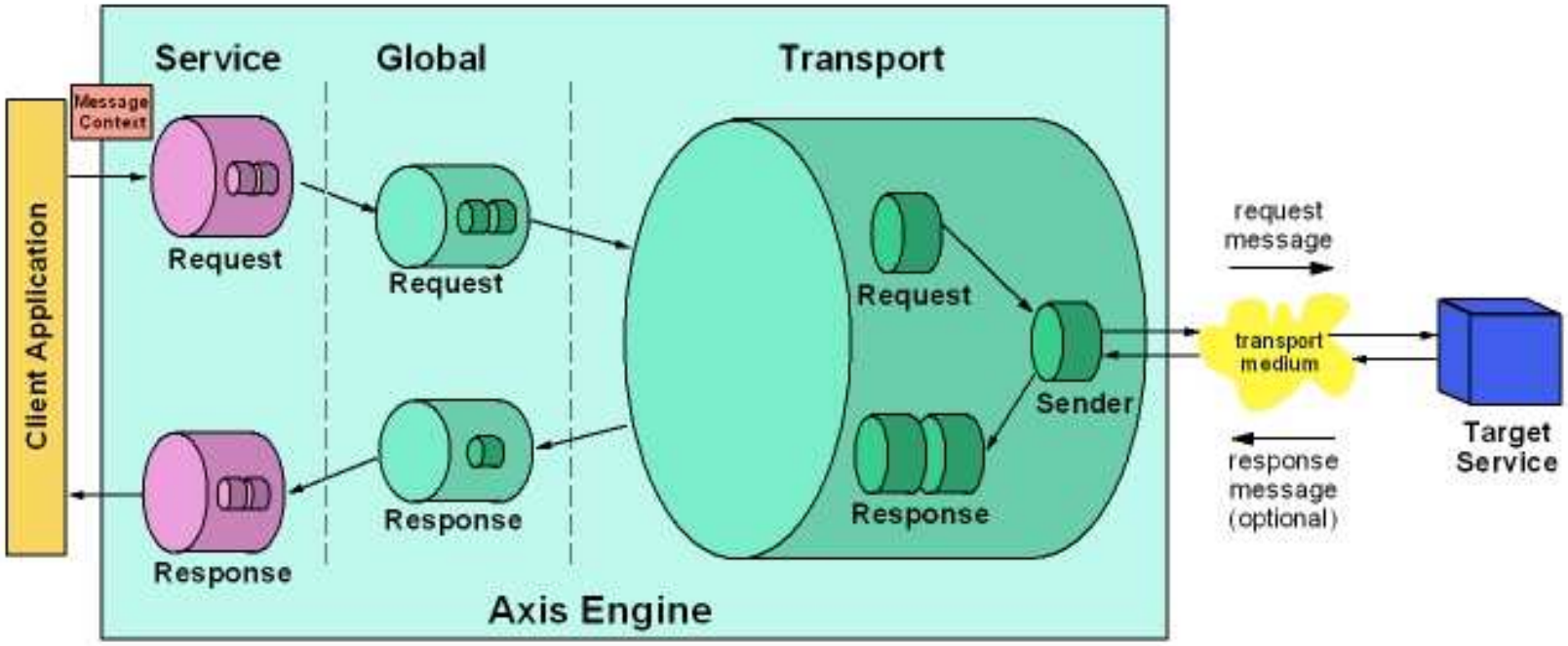
Deploy the handlers:

```
4) copy MyRequestHandler.class  MyResponseHandler.class
   to  Tomcat 4.1\webapps\axis\WEB-INF\classes
5) deployService.bat
6) browse: http://localhost:8080/axis --> View
7) Why they do not appear?
```

Test the handlers:

```
8)  java -cp \demos\Axis\Handlers
      FileTransferRequest \WebServices\MacaoNews.txt Macao.txt
8)  cd Tomcat 4.1\bin
9)  dir
10) edit SOAPRequest.log SOAPResponse.log
```

Message Path on the Client



[courtesy Apache-Axis]

Client Side Message Processing

The `AxisClient` is the class handling the message flow through the various components on the client side.

The `Call` object (`org.apache.axis.client.Call`) is the main client-side entry point to Axis.

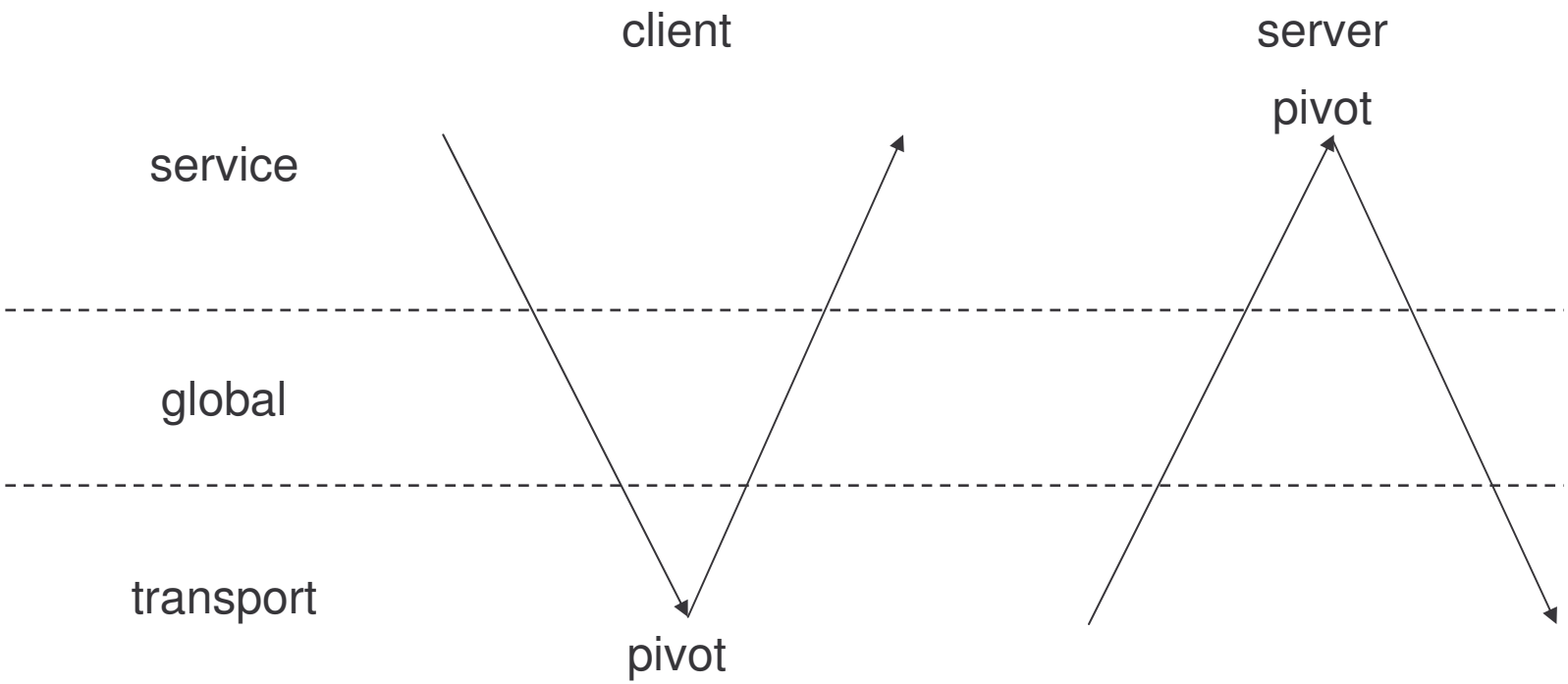
Inside the `AxisClient`, the message flows in a reverse order than in the server.

The last chain, is the transport-specific chain.

The transport chain has a pivot handler called the `sender`.

The sender is responsible for taking the request message out of the `MessageContext` and sending it across the wire in a protocol-specific way.

Sequence of Handlers



JAX-RPC Overview

JAX-RPC: Java API for XML-based RPC.

The fundamental purpose of JAX-RPC is to make communications between Java and non-Java platforms easier:

1) using Web services technologies like XML, SOAP and WSDL
2) providing a simple object-oriented API that Java developers can use to communicate with other technologies

It is possible to use JAX-RPC to:

1) access web services that run in non-Java environments
2) host Java web services, so that non-java applications can access them

JAX-RPC

1) JAX-RPC is designed as a Java API for web services.

2) incorporates XML-based RPC functionality according to the SOAP 1.1 specification

3) requires support for:

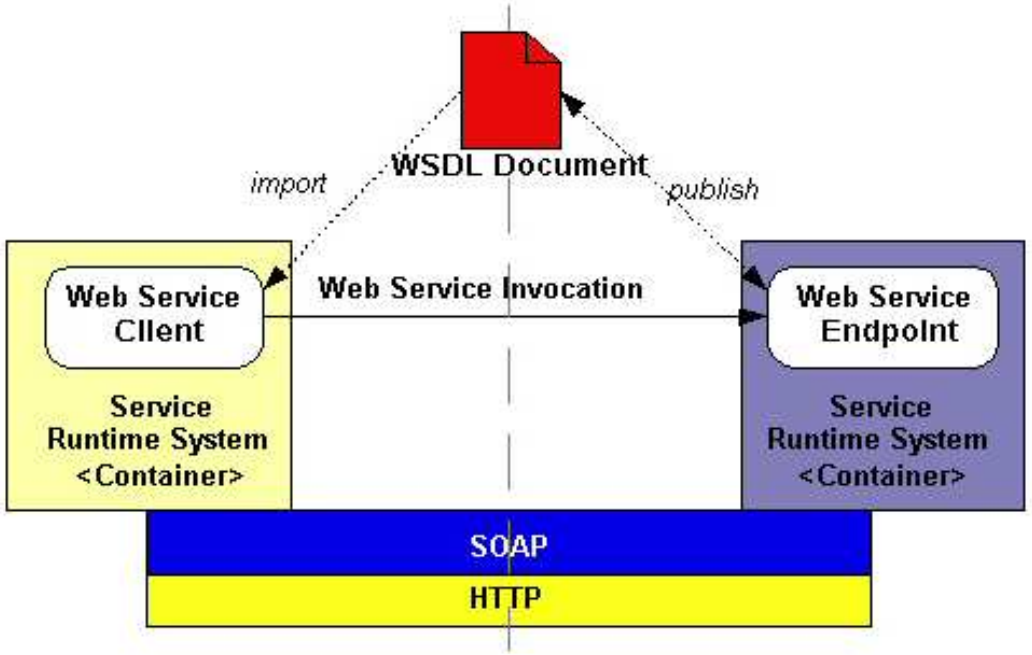
   a) SOAP and WSDL
   b) RPC encoded messaging
   c) SOAP with Attachments

WS and JAX-RPC

A Web service endpoint is deployed on either the Web container or EJB container based on the corresponding component model.

These endpoints are described using a WSDL document.

A client uses this WSDL document and invokes the Web service endpoint.



JAX-RPC requires SOAP over HTTP for interoperability.

JAX-RPC and SOAP

JAX-RPC provides support for SOAP message processing model through the SOAP message handler functionality.

JAX-RPC uses SAAJ API (SOAP with Attachments API for Java).

SAAJ provides a standard Java API for constructing and manipulating SOAP messages with attachments.

JAX-RPC Extensions

1) Message Handlers: they allow to manipulate SOAP header blocks as they flow in and out of JAX-RPC endpoint and the client applications.

2) Mappings from WSDL and XML to Java describing how:

   a) Java endpoint interfaces used by JAX-RPC web services are converted into WSDL
   b) WSDL documents are converted into JAX-RPC generated stubs and dynamic proxies
   c) method calls to JAX-RPC client APIs are converted into SOAP messages
   d) SOAP messages are mapped to JAX-RPC service endpoint methods

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

Message Context Class

`MessageContext` is the Axis implementation of the `SOAPMessageContext` class.

Is the core class for processing messages in handlers and other parts of the system.

This class also contains constants for accessing some well-known properties.

Some methods include:

- `getAllPropertyNames()`
- `getMessage()`
- `getService()`
- `getRequestMessage()`
- `getProperty()`
- `getOperation()`
- `getSOAPActionURI()`
- `getResponseMessage()`

Message Type

Asking the MessageContext for the request or response message will return a message of type `org.apache.axis.Message.`

The `Message` class extends `SOAPMessage`.

The message contains:

1) a `SOAPPart`

2) zero or more `AttachmentsParts`

Messages conforming to the simple SOAP HTTP binding will have only a `SOAPPart` and no attachments.

SOAPPart

The `SOAPPart` allows to get the `SOAPEnvelope`.

A client can access the `SOAPPart` object of a `SOAPMessage` object like:

```
void invoke(MessageContext context) ...

   SOAPMessage message = context.getMessage();
    SOAPPart soapPart = message.getSOAPPart();
```

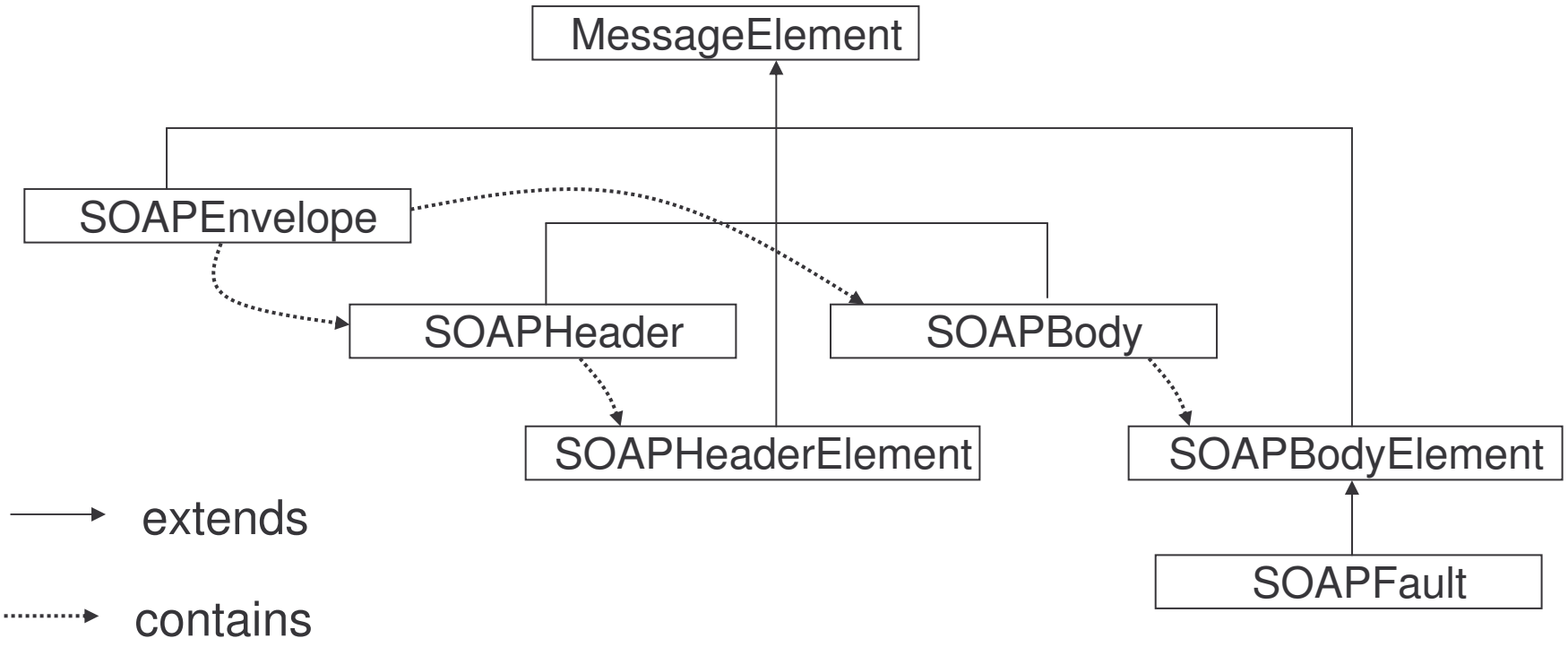
`SOAPPart` object contains a `SOAPEnvelope` object, which in turn contains a `SOAPBody` object and a `SOAPHeader` object.

The `SOAPPart` method `getEnvelope` can be used to retrieve the `SOAPEnvelope` object.

```
SOAPEnvelope env = soapPart.getEnvelope();
```

Accesing the Envelope Elements

Some important classes involving the SOAP envelope:



`MessageElement` is the base type of nodes of the SOAP message parse tree.

Accesing to Envelope Elements

Once, obtained the object `SOAPEnvelope` (env), it is possible to access `SOAPHeader` and `SOAPBody` objects:

```
void invoke(MessageContext context) ...
   SOAPMessage message = context.getMessage();
    SOAPPart soapPart = message.getSOAPPart();

        SOAPEnvelope env = soapPart.getEnvelope();

   SOAPHeader sh = env.getHeaders();
   SOAPBody sb = env.getBody();
```

Envelope Elements Example

Count the headers on a SOAP Envelope:

```
void invoke(MessageContext mc) ...

   /* Get the SOAP Envelope from the request message */
   Message requestMsg = mc.getRequestMessage();
   SOAPEnvelope env = requestMsg.getSOAPEnvelope();

   /* Obtain the headers */
   Vector headers = env.getHeaders();
```

Looking for a Fault Example

Analyze if the content of the body is a fault:

```
void invoke(MessageContext mc) ...

   /* Get the SOAP Envelope from the response message */
   Message responseMsg = mc.getResponseMessage();
   env = responseMsg.getSOAPEnvelope();
   SOAPBodyElement body = env.getFirstBody();

   /* controls whether the body contains a fault */
   if (body instanceof SOAPFault) {
     System.out.println ("First body element is a fault code,
       code =" + ((SOAPFault)body).getFaultCode().toString() );
```

Task 61: Envelope

Objective: Write the EnvelopeManager Java class which provides the invoke method. This method receives as argument an object of type `MessageContext`. The method provides the following functionality:

1) prints a message specifying how many headers has the request message
2) prints a message specifying if the body element of the response message contains a fault and what is the code of the fault

The EnvelopeManager class is used by:

1) envelopeExample1: the same example as FileTransferRequest
2) envelopeExample2: the same example as FileTransferSenderFault

These classes include the following code:

```
MessageContext mc = call.getMessageContext();
EnvelopeManager em = new EnvelopeManager();
em.invoke(mc);
```

Task 62: Envelope

```
1)  cd demos\axis\envelope

2)  dir
EnvelopeExample1.class
EnvelopeExample2.class
EnvelopeManagerTemplate.java
```

3) edit the class using: `EnvelopeManagerTemplate.java`

4) add the lines of code to get the headers of the envelope

5) add the lines of code to get the envelope

6) save the file as `EnvelopeManager,java`

Task 63: Envelope

Compile the class:

```
7)  javac EnvelopeManager.java
```

Execute the examples:

```
8)  java -cp \demos\Axis\Envelope envelopeExample1
          c:\WebServices\MacaoNews.txt macao.txt
9)  java -cp \demos\Axis\Envelope envelopeExample2
          c:\WebServices\MacaoNews.txt macao.txt
```

Replacing the Envelope Elements

It is possible to change the body or header of a SOAPEnvelope object by retrieving the current one, deleting it, and then adding a new body or header.

The javax.xml.soap.Node method detachNode detaches the XML element (node) on which it is called.

For example, to create a new header:

```
env.getHeader().detachNode();
SOAPHeader sh = env.addHeader();
```

Type Mapping

Axis has the ability to map XML to Java and vice versa.

A type mapping consists of:

1) an XML type (a QName)
2) a Java type (a class)
3) a serializer (to write Java to XML)
4) a deserializer (to write Java from XML)

To map types on the client side, use:

```
call.registerTypeMapping (Class javaType,
                          QName xmlType,
                          Class serializerFactoryClass,
                          Class deserializerFactoryClass)
```

Type Mapping Example

Recall the example that downloads the attributes of the file.

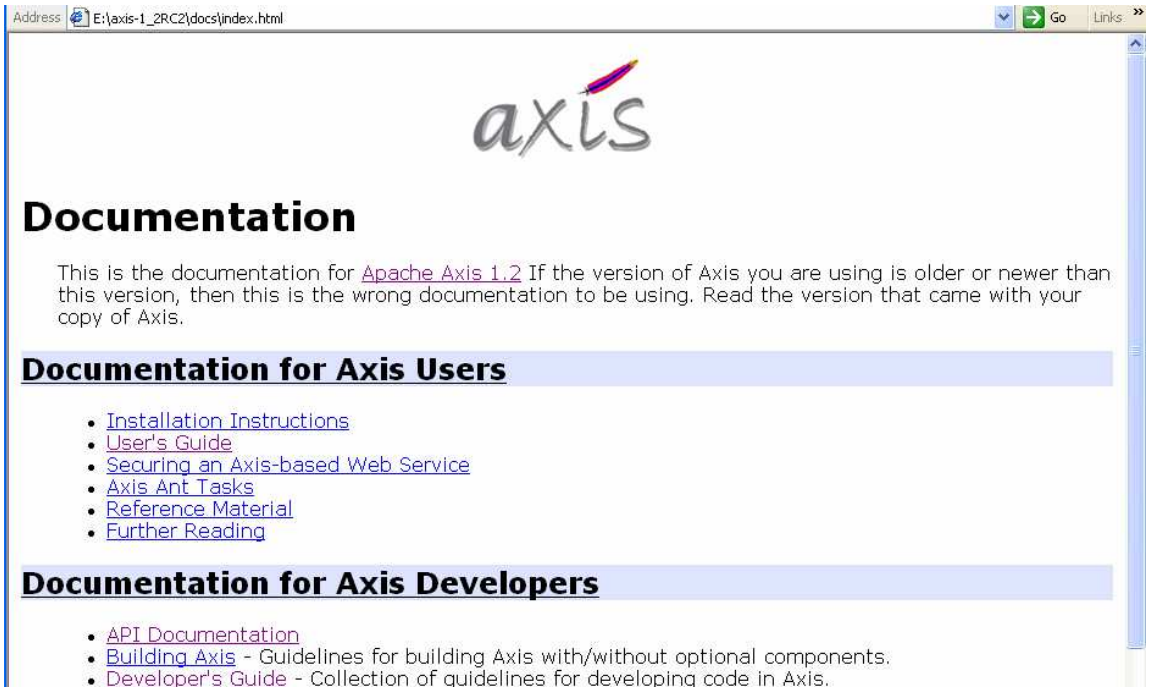
The code on the client, looks like:

```
QName qn = new QName("urn:FileAttribute","FileAttribute");

 call.registerTypeMapping(FileAttribute.class, qn,
   new org.apache.axis.encoding.ser.BeanSerializerFactory(
                                      FileAttribute.class, qn),
    new org.apache.axis.encoding.ser.BeanDeserializerFactory(
                                       FileAttribute.class, qn));
```

Task 64: Accessing Documentation

1) `cd axis-1_2RC2\docs`

2) `open index.html`



3) access API Documentation

Axis Client APIs

The client APIs can divided in two categories:

1) dynamic invocation: only pre-existing Java classes are used to do the work

2) stub generation: a tool generates code from the WSDL description

In order to invoke a web service, the client needs to use the Dynamic Invocation Interface (DII).

Service Object

The Service object acts as a factory for Call objects and it also stores meta-data about the service:

1) is the object representing the AxisClient instance that processes the client invocations
2) is where the type-mappings XML-Java are stored

The Service object can generate many Call objects.

Each Call object represents a single invocation of a service.

Since all these Call objects will talk to the same Web service, all the meta-data related to it is stored in the Service object.

Service Object and WSDL

A Service may or may not be associated with a WSDL description.

If it is related, it is possible to request:

1) a generic Call object,

2) a Call object that has been preconfigured with all the meta-data from the WSDL

The Service API as defined by JAX-RPC has no direct means to access WSDL documents for dynamic invocation.

The Service object has two constructors that allows the association with a WSDL document.

Service Object: WSDL Association

Two constructors for building a Service object with meta-data initialized from the WSDL:

1) `Service(URL wsdlLocation, QName serviceName)`: the WSDL is located at the specified URL.

2) `Service(String wsdlLocation, QName serviceName)`: the WSDLocation is a String that may be a URL or may also be a filename on the local filesystem, relative to the current directory.

The Axis Service object also has a no-argument constructor for use without a WSDL:

3) `Service()`

Call Object

Call objects are generated with the `createCall()` method of the Service object.

```
import org.apache.axis.client.Service;
import javax.xml.rpc.Call;

Service service = new Service();
Call call = service.createCall();
```

The no-arguments service constructor creates a blank service.

The `createCall()` factory method without arguments, creates a generic JAX-RPC Call object.

Setting Properties on the Call

The Call class has a setProperty() API:

```
void setProperty(String name, Object value)
```

This method allows to set properties on the Call object.

All the properties that are set on the Call will be available to every MessageContext that is created as a result of using the Call.

It is possible to use a Call object to make multiple invocations to a given service. Each time a new MessageContext will be created.

Generic Calls for WS

The development of the Web service client includes:

1) creating objects to manage the call and the service
2) defining the endpoint URL of the web service
3) defining the operation name of the web service
4) invoking the desired service passing the corresponding parameters

Creating Objects: Call and Service

Let's have a look at one of our Web Service client: FileTransferRequest:

```
public class FileTransferRequest {
   public static void main(String [] args) {
       . . .
      Service  service = new Service();
      Call     call    = (Call) service.createCall();
```

Two objects are created: Service and Call.

Defining the Endpoint

A variable endpoint is defined and initialized with the URL address of the desired web service.

```
String endpoint =
 "http://localhost:8080/axis/services/FileDownloadService";
```

This address containing the final destination of the SOAP message is passed to the newly created Call object:

```
call.setTargetEndpointAddress( new java.net.URL(endpoint) );
```

Axis also provides two constructors that allow to create a Call, pointing to a particular Web service endpoint:

```
1)  Call (String endpoint)
2)  Call(URL endpoint)
```

Defining the Operation Name

The name of the operation that the Call object is invoking is defined as:

```
call.setOperationName(new QName("http://soapinterop.org/",
 "downloadFile")
```

Invoking the Desired Service

The invoke() method allows to invoke a web service. It has several different forms.

The data for any given invocation is generally handed to the invoke method as an array of Java objects:

1) for RPC-Style services, these objects are parameters for a remote method call and each one maps to an XML element
2) for document-style services is generally a single object in the array and maps to the entire SOAP body for the operation

In our example, we have:

```
byte[] ret = (byte[])call.invoke( new Object[] { args[0] } );
```

args[0] is the name of the file to download, that is sent as a parameter

Invoke Method: Different Forms

`invoke()` Invokes this Call with its established MessageContext
`invoke(Message msg)` Invokes the service with a custom Message.
`invoke (java.lang.Object[] params)` Invokes the operation associated with this Call object using the passed in parameters as the arguments to the method.
`invoke (QName operationName, java.lang.Object[] params)` Invokes a specific operation using a synchronous request-response interaction mode.
`invoke(RPCElement body)` Invokes an RPC service with a pre-constructed RPCElement.
`invoke(SOAPEnvelope env)` Invokes the service with a custom SOAPEnvelope.

and more…

Task 65: Invoking the WS

Objective: Write the FileTransferClient Java class invoking the method downloadfile of the FileDownloadService. Use the FileTransferTemplate.java.

```
1) cd demos\Axis\Client

2) dir
FileTransferTemplate.java
```

3) edit `FileTransferTemplate.java` to generate `FileTransferClient.java` adding the corresponding lines of code

```
4) javac FileTransferClient.java

5) java -cp \demos\Axis\Client FileTransferClient
        c:\WebServices\MacaoNews.txt Macao.txt

6) dir
```

Naming Parameters

Axis automatically names the XML-encoded arguments in the SOAP message as “arg0”, “arg1”, etc.

For changing these names, we need to add the call `addParameter` for each parameter, and `setReturnType` for the return, before the invoke:

```
call.addParameter (“testParam”,
                   org.apache.axis.Constants.XSD_String,
                   javax.xnl.rpc.ParameterMode.IN);
call.setReturnType (org.apache.axis.Constants.XSD_String);
```

The testParam will be the first parameter on the invoke call. It also defines the type of the parameter and whether it is an input, output or inout parameter.

If names are added to the parameters, it is needed to add the type of the result.

Defining SOAP Version

It is possible to define the SOAP version in the Call object:

```
import org.apache.axis.soap.SOAPConstants;
. . .
. . .
call.setSOAPVersion(SOAPConstants.SOAP12_CONSTANTS);
```

The default version of SOAP is 1.1.

Task 66: Parameters and Version

Objective: Add to the FileTransferClient the name “fileName” to the argument of the method. The name of the result should be: “outputFile”. Generate a SOAP 1.2 envelope.

```
1) cd demos\Axis\ParametersVersion


2) copy:    \demos\Axis\Client\FileTransferClient.java
    to :    \demos\Axis\ParametersVersion
```

Task 67: Parameters and Version

3) edit FileTransferClient.java, adding:

```
  import org.apache.axis.soap.SOAPConstants;

/* Naming Parameters and Result */
call.addParameter("fileName",
      org.apache.axis.Constanst.XSD_STRING,
                  javax.xml.rpc.ParameterMode.IN);
call.setReturnType (org.apache.axis.Constants.XSD_BASE64);

  /* Defining SOAP Version */
call.setSOAPVersion(SOAPConstants.SOAP12_CONSTANTS);
```

```
4)  javac FileTransferClient.java

5)  java -cp \demos\Axis\ParametersVersion
         FileTransferClient
         c:\WebServices\MacaoNews.txt Macao.txt
```

Inserting a Header Example

Let us remember the header of our example:

```
<soapenv:Header>
 <ns1:authentication
   soapenv:actor="http://manager"
   soapenv:mustUnderstand="0"
    xmlns:n1="http://localhost:8080/axis/services/FileDownloadService">
   <ns1:username>admin</ns1:username>
   <ns1:password>admin</ns1:password>
 </ns1:authentication>
</soapenv:Header>
```

1) one header: authentication

2) two attributes: actor and mustUnderstand

3) two sub-elements: username and password

Implementing Headers 1

1) define a SOAP Header Element with the name of the header:

```
String nameSpace =
  "http://localhost:8080/axis/services/FileDownloadService";
SOAPHeaderElement she = new
   SOAPHeaderElement(XMLUtils.StringToElement(nameSpace,
                     "authentication", ""));
```

2) define the attributes:

```
she.setRole("http://manager");
she.setMustUnderstand(false);
```

Implementing Headers 2

3) define the sub-elements and its contents:

```
String nameSpace =
   "http://localhost:8080/axis/services/FileDownloadService";
MessageElement username = new
                MessageElement(nameSpace,"username");
username.addTextNode("admin");

MessageElement password = new
                 MessageElement(nameSpace,"password");
password.addTextNode("admin");
```

4) add the childs to the header element:

```
she.addChild(username);
she.addChild(password);
```

5) create the header:

```
call.addHeader(she);
```

Task 68: Inserting a Header

Objective: add a header to the request message.

1) `cd demos\Axis\Headers`

2) `copy: \demos\Axis\ParametersVersion\FileTransferClient.java`
   `to : \demos\Axis\Headers`

3) edit `FileTransferClient` and add the lines of code to generate the header

4) `javac FileTransferClient.java`

5) `java -cp \demos\Axis\ParametersVersion`
   `FileTransferClient`
   `c:\WebServices\MacaoNews.txt Macao.txt`

Dynamic Binding

Using one of the WSDL-aware Service() constructors, Axis extracts the meta-data (endpoint, parameters and return types) from the WSDL file and prepares the Call.

The only missing information is the port and the operation. Some possible formats include:

`service.createCall (QName portName)` Returns a Call that has been initialized with the endpoint address referred to by the named port.
`service.createCall (QName portName, QName operation)` Returns a Call that has been initialized with the endpoint address and also with all the parameters and return types.
`service.createCall (QName portName, String operation)` The same as the previous format, but it accepts the unqualified operation name.

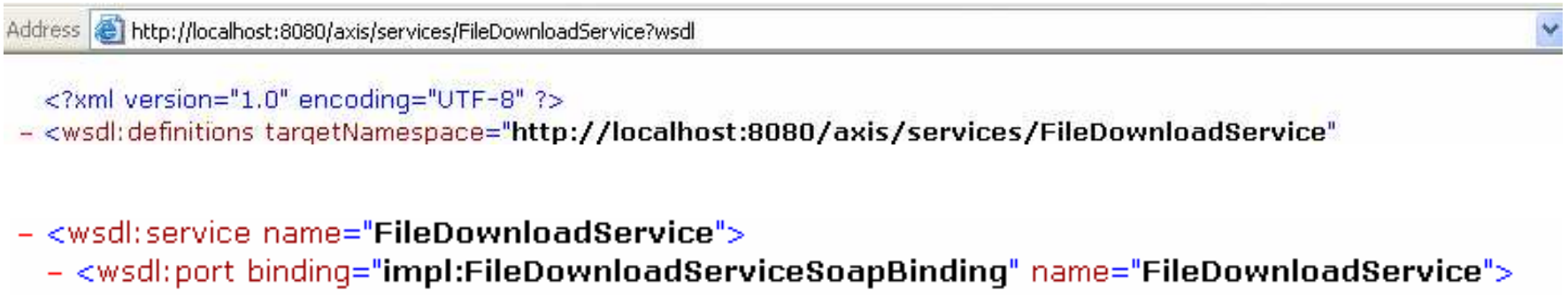
Dynamic Binding Example 1

Using the following constructor:

```
Service (java.lang.String wsdlLocation, QName serviceName)
```

The example looks like:

```
String wsdlLocation =
 "http://localhost:8080/axis/services/FileDownloadService?wsdl";
String nameSpace =
    "http://localhost:8080/axis/services/FileDownloadService";
QName serviceName = new QName(nameSpace, "FileDownloadService");
Service service = new Service(wsdlLocation, serviceName);
```



Dynamic Binding Example 2

Using the following method:

```
createCall (QName portName, java.lang.String operationName)
```

The example looks like:

```
QName portName = new QName(nameSpace, "FileDownloadService");
String operationName = "downloadFile";
Call call = (Call)service.createCall( portName, operationName);
```

```
- <wsdl:service name="FileDownloadService">
  - <wsdl:port binding="impl:FileDownloadServiceSoapBinding" name="FileDownloadService">
```

```
- <wsdl:binding name="FileDownloadServiceSoapBinding" type="impl:FileDownload">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="downloadFile">
```

Task 69: Dynamic Binding

Objective: Develop the FileTransferClient using a WSDL-aware Service() constructor. Use FileTransferTemplate.java

```
1) cd demos\Axis\Dynamic

2) dir
FileTransferTemplate.java
```

3) edit `FileTransferTemplate.java` and save it as `FileTransferClient.java`

Task 70: Dynamic Binding

4) add the following instructions:

```
String wsdlLocation =
"http://localhost:8080/axis/services/FileDownloadService?wsdl";
String nameSpace =
    "http://localhost:8080/axis/services/FileDownloadService";
QName serviceName = new QName(nameSpace, "FileDownloadService");
Service service = new Service(wsdlLocation, serviceName);

QName portName = new QName(nameSpace, "FileDownloadService");
String operationName = "downloadFile";
Call call = (Call)service.createCall( portName, operationName);
```

```
5)  javac FileTransferClient.java

6)  java -cp \demos\Axis\Dynamic FileTransferClient
        c:\WebServices\MacaoNews.txt Macao.txt
```

Using Sessions

In some cases, it is useful if the server is able to “remember” data related to various invocations of the same client.

Some kind of session is needed to associate some data with a given client.

Axis has some simple APIs for session support.

Session Implementations

Axis provides two built-in ways to maintain sessions across web services connections, using:

1) standard HTTP session mechanisms

2) SOAP headers

3) WS-Resource Framework

Sessions with HTTP

Uses HTTP cookies to store session state:

1) the server transmits to the client some kind of cookie
2) the server accepts the same cookie from the client on subsequent requests
3) the server realizes that the new requests are associated with the same client

The actual session data is kept by the servlet framework.

It is needed:

1) to use HTTP
2) to call `setMaintainSession(true)` on either the Call or the Service object

Sessions with SOAP Headers

To use this approach, the `SimpleSessionHandler` must be deployed.

The handler `org.apache.axis.handlers.SimpleSessionHandler` is included with Axis.

In order to work, this handler must be deployed in the global request and response flows of the client in order to work.

Sessions with WS-Resource

WS-Resource Framework consists of five specifications:

1) WS-ResourceProperties
2) WS-ResourceLifetime
3) WS-ServiceGroup
4) WS-RenewableReferences
5) WS-BaseFaults

and an approach to modeling statefull resource using web services.

The WS-Resource Framework was developed by Computer Associates, Fujitsu, Globus, Hewlett-Packard and IBM.

It was submitted to OASIS.

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

Axis Tools

Axis comes with the following tools:

1) WSDL2Java: based on WSDL descriptions generates Java code for the client and server side

2) Java2WSDL: is a command-line tool for taking Java interfaces and generating WSDL

3) generation of WSDL: at runtime, the Axis engine automatically generates WSDL for the deployed services

WSDL2Java

WSDL2Java automatically builds stubs.

A Stub is a Java class with a Java-friendly API that closely matches the Web service interface defined in a given WSDL document.

The tool is executed from the command line:

```
java org.apache.axis.wsdl.WSDL2Java  WSDL_Document_URL
```

For instance:

```
java org.apache.axis.wsdl.WSDL2Java
  http://localhost:8080/axis/services/FileDownloadService?wsdl
```

WSDL2Java Example 1

Executing WSDL2Java for our example, the following Java classes are generated:

1) FileDownload.java: this is the service interface . In JAX-RPC is known as Service Endpoint Interface (SEI).

```
/**
 * FileDownload.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST)
   WSDL2Java emitter.
 */

package localhost.axis.services.FileDownloadService;

public interface FileDownload extends java.rmi.Remote {
    public byte[] downloadFile(java.lang.String in0) throws
    java.rmi.RemoteException;
}
```

WSDL2Java Example 2

2) FileDownloadService: this interface allows type-safe access to the SEI from the locator class. Includes two methods:

   a) `getFileDownloadService()`: gets an implementation of the FileDownload interface that will call the endpoint specified in the WSDL
   b) `getFileDownloadService(URL url)`: gets a FileDownload stub that uses the same WSDL interface but points at a different endpoint

3) FileDownloadServiceLocator.java: this class implements the FileDownloadService interface and acts as the factory for stub instances

4) FileDownloadServiceSOAPBindingStub.java: the class that implements the FileDownload interface – the core of the client

Task 71: Execute WSDL2Java

Objective: execute WSDL2Java for the FileDownloadService

```
1) cd demos\Axis\WSDL2Java

2) java org.apache.axis.wsdl.WSDL2Java
  http://localhost:8080/axis/services/FileDownloadService?wsdl

3) cd localhost\axis\services\FileDownloadService

4) dir
FileDownload.java
FileDownloadService.java
FileDownloadServiceLocator.java
FileDownloadServiceSOAPBindingStub.java

5) notepad FileDownload*.java
```

Tools for Invoking a Service

Two ways for invoking a service from the client:

1) using a generic stub: instantiating a service and a call object

2) using a specific stub: invoking the interfaces generated by WSDL2Java based on the WSDL file

Task 72: Using a Specific Stub

Objective: testing the service with a client that uses the interfaces generated by WSDL2Java.

1) `cd \demos\Axis\TestStubs`

2) `dir`
`FileTransferTestTemplate.java`

3) generate the stubs:

```
java org.apache.axis.wsdl.WSDL2Java
http://localhost:8080/axis/services/FileDownloadService?wsdl
```

4) compile the generated classes:

```
cd localhost\axis\services\FileDownloadService
javac *.java
```

Task 73: Using a Specific Stub

5) `cd \demos\Axis\TestStubs`

6) edit `FileTransferTestTemplate.java`, adding:

```
import localhost.axis.services.FileDownloadService.*;

/* Get the stub from the locator object */
FileDownloadServiceLocator locator = new
  ileDownloadServiceLocator();
FileDownload stub = locator.getFileDownloadService();

/* Call the web service
 byte[] ret = stub.downloadFile(fileName);
```

save it as FileTransferTest.java

Task 74: Using a Specific Stub

Compile:

```
7)  javac -classpath \demos\Axis\TestStubs
                FileTransferTest.java
```

Execute:

```
8)  java -cp \demos\Axis\TestStubs FileTransferTest
              c:\WebServices\MacaoNews.txt Macao.txt


9)  dir
```

Using WSDL2Java for Services 1

It is possible to take a WSDL description of a Web Service and create a skeleton implementation of the service described by the WSDL.

Specifying the option `-s` to WSDL2Java, in addition to the client and data classes, will generate for the FileDownloadService:

1) FileDownloadServiceSOAPBindingImpl.java: the framework implementation of the service
2) deploy.wsdd: a pre-built deployment file for use with the AdminClient
3) undeploy.wsdd: a pre-built undeployment file

Using WSDL2Java for Services 2

The FileDownloadServiceSOAPBindingImpl.java looks like:

```
/**
 * FileDownloadServiceSoapBindingImpl.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2RC2 Nov 16, 2004 (12:19:44 EST)... */

package localhost.axis.services.FileDownloadService;

public class FileDownloadServiceSoapBindingImpl implements
 localhost.axis.services.FileDownloadService.FileDownload{
    public byte[] downloadFile(java.lang.String in0) throws
 java.rmi.RemoteException {
        return null;
    }

}
```

Task 75: Using WSDL2Java More

Objective: execute WSDL2Java for the FileDownloadService with –s option

```
1)   cd demos\Axis\WSDL2JavaWS

2)   java org.apache.axis.wsdl.WSDL2Java -s
     http://localhost:8080/axis/services/FileDownloadService?wsdl

3)  cd localhost\axis\services\FileDownloadService

4)  dir
deploy.wsdd
FileDownload.java
FileDownloadService.java
FileDownloadServiceLocator.java
FileDownloadServiceSOAPBindingImpl.java
FileDownloadServiceSOAPBindingStub.java
undeploy.wsdd

5) notepad deploy.wsdd, undeploy.wsdd,
           FileDownloadServiceSOAPBindingImpl.java
```

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

WSDD Overview

WSDD is an XML format that Axis uses to store its configuration and deployment information.

The Axis server keeps its configuration in a file: `server-config.wsdd`.

The Axis client has an equivalent file: `client-config.wsdd`.

These files have default versions stored in `axis.jar`

WSDD Structure

In order to deploy web services the root element of WSDD is `deployment`.

Inside the root element is a `<globalConfiguration>` element which contains options for the Axis engine plus definitions for the global request and response chains.

```
<globalConfiguration>
  <parameter name="defaultSOAPVersion" value="1.2" />
        <requestFlow>
            <handler type="requestHandler"/>
        </requestFlow>
        <responseFlow>
            <handler type="responseHandler"/>
        </responseFlow>
</globalConfiguration>
```

Global Configuration

The `parameter` declarations inside the `globalConfiguration` set options on the AxisEngine. For instance: SOAP version.

The `<requestFlow>` and `<responseFlow>` elements define the request and response global chains. They can contain either `<handler>` elements or `<chain>` elements.

The components inside `<requestFlow>` and `<responseFlow>` are invoked in exactly the same order as they are declared in the XML file.

Handler Declarations

Handler declarations tell Axis that a given Java class is a handler, allowing:

1) configure it with a set of options
2) name the configuraion, so it is possible to refer to it

```
<handler [name="name"] type="type">
   <parameter name="name" value="value />
</handler>
```

For example:

```
<handler name="requestHandler" type="java:MyRequestHandler">
    <parameter name="filename" value="SOAPRequest.log"/>
</handler>
```

This handler can be referenced:

```
<handler type="requestHandler"/>
```

Chain Definitions

It is possible to group a series of handlers into a chain.

```
<chain [name="name"] >
   <handler type="type">
     <parameter name="name" value="value" />
   </handler>
</chain>
```

For instance:

```
<chain name="logAndNotify" >
   <handler type="java:LogHandler" />
   <handler type="java:NotifyHandler" />
</chain>
```

This chain can be referenced as:

```
<requestFlow>
   <handler type="logAndNotify" />
</requestFlow>
```

Transport Declarations

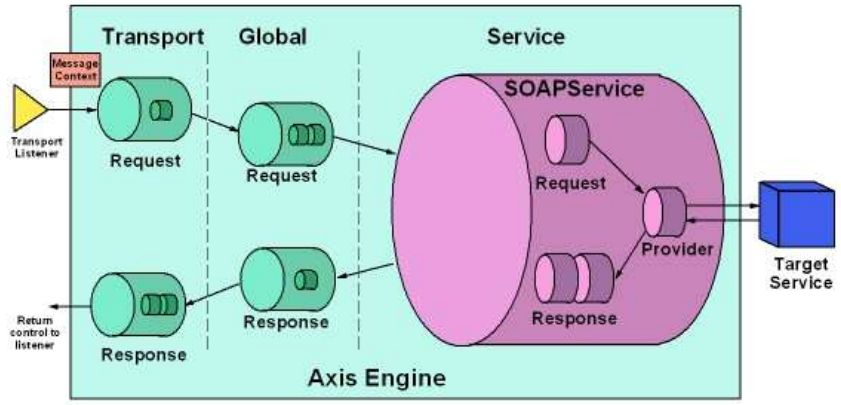
Transport declarations define a named, targeted chain, which has:

1) a requestFlow,
2) a responseFlow,
3) a pivot handler (only on the client)

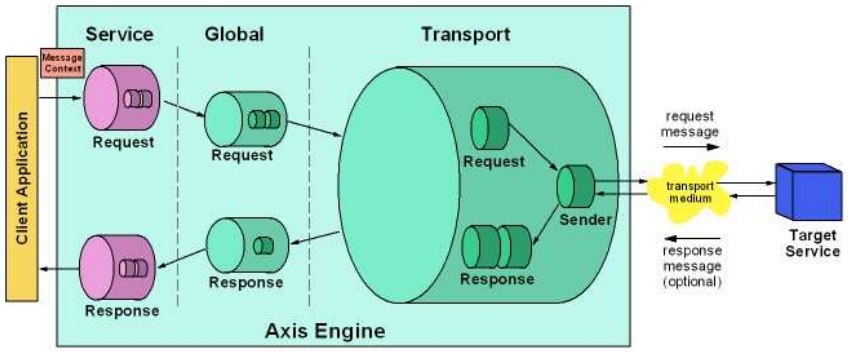
On the client, the pivot handler is the sender of the message.

On the server there is no need for a pivot handler.

Server Side



Client Side



Transport Handler: Client Example

Example from the client-config.wsdd:

```
<transport name="http"
   pivot="java:org.apache.axis.transport.http.HTTPSender" />
```

A pivot handler is defined using the `pivot` attribute.

Transport Handler: Server Example

Example from the server-config.wsdd:

```
<transport name="http">
 <requestFlow>
  <handler type="URLMapper"/>
  <handler
     type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
 </requestFlow>
...
```

This transport is named http and contains two-transport specific handlers:

1) URLMapper: sets the Axis service name in the MessageContext based on the HTTP URL
2) HTTPAuthHandler: takes the username and password out of the HTTP Basic authentication header and puts them in the MessageContext

Type Mapping

Type mappings control the mapping between Java classes and XML structures.

It is possible to tell the engine to map a particular Java class to a particular XML type, and even customize the serializer and deserializer classes.

```
<typeMapping qname="typeQName"
                        type="java:classname"
                        serializer="Serializer"
                        deserializer="DeserializerFactory"
                        encodingStyle="uri" />
```

Bean Mapping

The <beanMapping> tag is a shorthand for a <typeMapping>, which uses BeanSerializer, and BeanDeserializer classes to do the Axis's default data-mapping algorithms.

```
<beanMapping qname="typeQName"
             languageSpecificType="java:classname"
             encodingStyle ="url"/>
```

For instance, this mapping was used in the example of SOAP encoding, where the web service was returning the attributes of the file as an object:

```
<beanMapping qname="ns:FileAttribute"
                   xmlns:ns="urn:FileAttribute"
             languageSpecificType="java:FileAttribute"/>
```

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

WSDD for Services

The entire configuration of the Axis server is contained in: server-config.wsdd

Each deployed service in the server, has a `<service>` element in the WSDD file with the following syntax:

```
<service name="name"
                [style="rpc | wrapped | document | message"]
               [use="literal | encoded"]
               [provider="provider"] >
   <operation>*
   <typeMapping>* | <beanMapping>*
   <namespace>uri</namespace>*
   <wsdlFile>absolute-filename</wsdlFile>
   <endpointURL>uri</endpointURL>
   <handlerInfoChain>
   <parameter name="name" value="value" />
</service>
```

* means that the element may appear zero or more times.

Attributes 1

The `name` attribute contains the name of the service.

The `style` attribute specifies one of several different ways that Axis can map SOAP messages to and from Java method calls: rpc, document, wrapped or messsage.

If a style is specified, there is no need to specify the `use` or `provider` attribute, since they will have a default value based on the style chosen.

The `use` attribute specifies encoded or literal use. The default value is based on the `style` attribute.

Attributes 2

The `provider` attribute allows to specify a QName representing the particular provider:

1) Java:RPC: used for rpc, document, and wrapped styles
   a) this provider is automatically selected if the style is rpc, document or wrapped
   b) using RPC provider, the class name and the methods allowed must be specified:

```
<parameter name="className" value="class_name"/>
<parameter name="allowedMethods" value="m1 m2" />
```

2) Java:MSG: used for message style
   a) the message provider will dispatch raw XML to the service

3) Java:EJB: allows to use an Enterprise JavaBean as a web service

4) Handler: lets specify a user-defined handler for a particular service

Some Elements

If `<typeMapping>` or `<beanMapping>` are defined inside a service deployment, those XML/Java mappings will hold only for the service.

If the `<namespace>` element is present, the first one is the default namespace for the service.

The `<wsdlFile>` element allows to specify a custom WSDL file that the engine will return when asked about the WSDL for the service.

JAX-RPC Handlers Element

The <handlerInfoChain> element is used to enable deploying JAX-RPC style handlers:

```
<handlerInfoChain>
   <handlerInfo class="className" >
      <parameter name="name" value="value"/>
      <header qname="qname" />*
      <role SOAPActorName="uri" />
   </handlerInfo>*
</handlerInfoChain>
```

JAX-RPC handlers specified in this chain will run after the global chain, but before the request flow of the service.

JAX-RPC handlers have two methods: `handleRequest()` and `handleResponse()`, while Axis handlers only have `invoke()`.

Each <handlerInfo> defines a single JAX-RPC handler.

Operation Element 1

A service can contain zero or more `<operation>` elements.

The `<operation>` element is used when more fine-grained control of the options of a particular operation is desirable.

It handles the mapping from arbitrary XML QNames in the SOAP body to arbitrary Java methods, controlling:

1) how parameters to those methods map to XML elements
2) how the exceptions thrown by the Java methods are map to and from SOAP faults

Operation Element 2

```
<operation name="name" [qname="qname"] [returnQName="qname"]
        [returnType="qname"] [returnHeader="true" | "false"]>
 <parameter [qname="qname" | name="name"]
            [mode="in | out | inout"]
            type="qname"
            inHeader="true | false" outHeader="true | false" />*
 <fault name="name" qname="qname"
            class="classname" type="qname" />*
</operation>
```

The operation `name` is the name of the Java method this web service operation will invoke.

The `qname` is the QName of the XML element that will map to this operation.

Operation Element 3

Inside the `operation` element are zero or more `parameter` elements, each represents a parameter of the operation.

If the `inheader` or `outheader` attribute is specified for the parameter, then the serialization of the parameter will be in the SOAP header or in the SOAP body respectively.

The data related to the faults thrown by the service and specified in the `fault` element, will be serialized inside a fault class as a `<detail>` element with the specified QName and XML type.

Deploying Services

Two ways:

1) directly edit the server-config.wsdd

2) use the AdminClient tool

Admin Client

```
> java org.apache.axis.client.AdminClient
            [-u {username}] [-w {password}]
            [-p {port}] [-l {service-url}] {wsdd-file}
```

Executing this from the command line:

1) reads the WSDD file
2) attempts to deploy the service to the Axis engine

If authentication is required the `username` and `password` arguments are used.

If the WSDD has `<deployment>` as the root element, all the components in the WSDD are deployed.

All the classes referred in the WSDD must be available on the server's classpath before doing the deployment.

Undeploying Web Services

If the WSDD document has `<undeployment>` as its root element, all the referenced components will be removed from the running server.

For undeploying services, only the name of the components to undeploy are specified.

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/>
    <handler name="MyHandler">
    <service name="MyService">
</undeployment>
```

Task 76: Undeploying a Service

Objective: undeploy the FileUploadService service.

1) Make a copy of the file server-config.wsdd:

```
copy: \Tomcat 4.1\webapps\axis\WEB-INF\server-config.wsdd
  to: server-configbup.wsdd
```

2) Undeploy the service:

a) `cd \demos\Axis\Undeploy`

b) write the undeploy.wsdd file to undeploy the service:

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
    <service name="FileUploadService" />
</undeployment>
```

c)
```
java org.apache.axis.client.AdminClient
          undeploy.wsdd
```

d) browse: http://localhost:8080/Axis --> View

Task 77: Return to Previous State

Objective: return to the previous state (including the FileUploadService).

```
1)  cd \Tomcat 4.1\webapps\axis\WEB-INF

2)  java org.apache.axis.client.AdminClient
3)                server-configbup.wsdd

4)  browse: http://localhost:8080/Axis --> View
```

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

Service Lifecycle and Scope

Some questions that should be answered during design include:

1) how is the web service object created?
2) how many web service objects will exist?
3) can the objects be shared across multiple threads at once?

These issues can be specified when deploying a web service, using the `scope` option.

```
<service name="service_name">
  . . .
  <parameter name="scope"
             value = "[application | request | session]" />
</service>
```

Scope

The valid values for the scope option are:

1) application:
   a) only a single instance of the service class for the entire Axis engine
   b) all methods must be thread-safe – many active requests in parallel for the same code
   c) any state of the service object will be shared across all invocations

2) request:
   a) a new service will be created for every SOAP request
   b) constructors should not have expensive initialization code
   c) default value

3) session:
   a) Are created once per client session
   b) data fields hold state on a per-session basis

Creation and Destruction

JAX-RPC provides an interface called: javax.xml.rpc.server.ServiceLifecycle

This interface contains two methods:

```
1)  void init (Object context) throws ServiceException;
2)  void destroy();
```

When a service is created or destroyed by the Axis runtime, the engine checks to see if the objects implements one of these interfaces.

Implementing these methods, initialization or cleanup may be done.

When a new service object is created, `init` is called with a context object allowing the service to access the MessageContext.

Sessions on the Server Side

Axis provides an abstraction to manage sessions in the server side in the org.apache.axis.session package.

A given interaction can optionally be associated with a session.

The MessageContext has a slot in it for the currently active session.

A `Session` object lets you to store values in a library indexed by `String` keys – like a map or hash-table.

Data stored in a `Session` during one interaction will be available again on the next interaction of the same client.

Accessing Sessions

There are two built-in ways for accessing sessions on the Axis server:

1) using the servlet HTTPSession: the servlet engine will handle time out
2) using SimpleSession: the SimpleSessionHandler will periodically read expired sessions

It is possible to set the timeout on a session with:

```
session.setTimeout(int)
```

The session implementations included in Axis are not persistent – data will be lost in case of a server crash or restart.

AXIS Outline

1) Overview

2) Service Invocation
   a) data structures
   b) static binding
   c) dynamic binding
   d) sessions

3) AXIS Tools

4) AXIS Configuration

5) Service Deployment

6) Service Lifecycle

7) Summary

AXIS Summary 1

1) Axis is an engine for processing SOAP messages

2) the Axis engine invokes a series of handlers to process messages

3) handlers are built-in the engine or can be included in a module defined by the user

4) handlers may:

    1) receive a request and send a response
    2) process a request and produce a response – pivot handlers

5) handlers are grouped in chains.

AXIS Summary 2

1) Axis defines different processing levels:

   a) on the server: transport – global – service
   b) on the client: service – global – transport

2) the object passed through the different handles in all these levels is the MessageContext

3) the MessageContext structure includes:

   a) the request message
   b) the response message
   c) several properties
   d) other fields

AXIS Summary 3

1) JAX-RPC is Java API for XML-based RPC

2) JAX-RPC facilitates communication between Java and non-Java platforms

3) JAX-RPC is designed as a Java API for web services

4) JAX-RPC supports:

   a) SOAP and WSDL
   b) RPC encoded messaging
   c) SOAP with Attachments

AXIS Summary 4

1) Axis provides APIs for implementing SOAP classes.

2) it is possible to access, add and delete SOAP Envelope elements

3) the client APIs can be divided in:
   a) dynamic invocation: using pre-existing Java classes
   b) stub generation: using code generated by WSDL2Java tool

4) dynamic invocation: uses a Service and a Call object, where the endpoint address and the operation name of the service are defined

5) the endpoint address and the operation name can be:
   a) directly hard-coded in the Java program
   b) extracted automatically by Axis from the WSDL file, using WSDL-aware service constructors

AXIS Summary 5

1) It is possible to manage sessions on web services, by:
   a) standard HTTP session mechanisms
   b) SOAP headers
   c) WS-Resource Framework

2) Axis provides several tools:
   a) WSDL2Java: generates Java code for the client and the server based on WSDL
   b) Java2WSDL: generates WSDL based on Java interfaces
   c) generation of WSDL: at runtime when deploying services

AXIS Summary 6

1) Axis uses an XML file called deployment descriptor to:
   a) deploy services
   b) undeploy services
   c) customize the engine

2) the scope parameter when deploying the service allows to define its lifecycle:
   a) application
   b) request
   c) session

3) JAX-RPC provides two methods that can be invoked when the service object is created or destroyed

4) Axis engine looks if these methods are implemented by the service, and invoke them accordingly.

Axis Summary 7

Accessing the envelope:

```
Message requestMsg = mc.getRequestMessage();
SOAPEnvelope env = requestMsg.getSOAPEnvelope();
Vector headers = env.getHeaders();
System.out.println("There are " + headers.size() + " headers.\n" );

Message responseMsg = mc.getResponseMessage();
env = responseMsg.getSOAPEnvelope();
SOAPBodyElement body = env.getFirstBody();
```

Example:
demos\SourceFiles\Axis\Envelope

Axis Summary 8

Adding a Header:

```
String nameSpace =  "http://localhost:8080/axis/services/FileDownloadService";
SOAPHeaderElement she = new
     SOAPHeaderElement(XMLUtils.StringToElement(nameSpace,"authentication", ""));

she.setRole("http://manager");
she.setMustUnderstand(false);

MessageElement username = new MessageElement(nameSpace,"username");
username.addTextNode("admin");

MessageElement password = new MessageElement(nameSpace,"password");
password.addTextNode("admin");

she.addChild(username);
she.addChild(password);

call.addHeader(she);
```

Example: demos\SourceFiles\Axis\Header

Axis Summary 9

Invoking a Service without referencing to the WSDL document:

```
/*  the creation of the Service and the Call */
Service service = new Service();
Call call = (Call)service.createCall();

/* the definition of a variable "endpoint" containing the address of the service */
String endpoint = "http://localhost:8080/axis/services/FileDownloadService";

/* the method to set the endpoint address in the Call object */
call.setTargetEndpointAddress( new java.net.URL(endpoint) );

'* the definition of the operation name */
call.setOperationName(new QName("http://soapinterop.org/", "downloadFile"));

/* the invocation */
byte[] ret = (byte[])call.invoke( new Object[] { args[0] } );
```

Example: \demos\SourceFiles\Axis\Client

Axis Summary 10

Invoking a Service referencing to the WSDL document:

```
String wsdlLocation = "http://localhost:8080/axis/services/FileDownloadService?wsdl";
String nameSpace = "http://localhost:8080/axis/services/FileDownloadService";
QName serviceName = new QName(nameSpace, "FileDownloadService");
Service service = new Service(wsdlLocation, serviceName);

QName portName = new QName(nameSpace, "FileDownloadService");
String operationName = "downloadFile";
Call call = (Call)service.createCall( portName, operationName);

byte[] ret = (byte[])call.invoke( new Object[] { args[0] } );
```

Example: \demos\SourceFiles\Axis\Dynamic

Axis Summary 11

Invoking a Service using a generated stub:

```
/* Get the stub from the locator object */
FileDownloadServiceLocator locator = new FileDownloadServiceLocator();
FileDownload stub = locator.getFileDownloadService();

/* Call the web service */
byte[] ret = stub.downloadFile(fileName);
```

Example: \demos\SourceFiles\Axis\TestStubs

Axis Summary 12

Developing a Handler:

```
public class MyRequestHandler extends BasicHandler{
  SOAPEnvelope env;
  public MyRequestHandler() {
  }

  public void invoke(MessageContext msgContext) throws AxisFault {
      try{
```

Example: \demos\SourceFiles\Axis\Handlers

UDDI

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

UDDI Outline

1) Introduction

2) Concepts

3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag

4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service

5) Summary

UDDI Motivation

Once services have been properly described, these descriptions should be made available to those interested in using them.

Service discovery is a process for locating service providers and retrieving service description documents that have been published in a service registry.

Two types of service discovery:

1) static: occurs at application design time and is done by a human designer
2) dynamic: occurs at runtime and is done by the application

For doing so, it is needed to standardize the web service registry.

Such standardization is done by UDDI project.

UDDI Project

UDDI project is an industry initiative attempting to create a platform-independent, open framework for:

1) describing,
2) discovering
3) integrating

business services.

UDDI specifications define a registry service for:

1) web services
2) other electronic and non-electronic services.

UDDI Registry Service

A UDDI registry service is a web service.

The UDDI registry service manages information about:

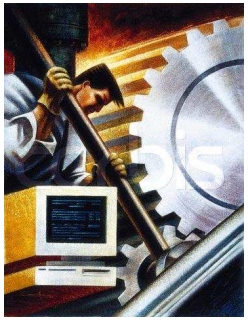
1) service providers
2) service implementations
3) and service metadata

UDDI Usage

UDDI is used by:

1) providers - use UDDI to advertise their services
2) consumers - use UDDI to:
    a) discover services that suit their requirements
    b) obtain the service metadata needed to consume them

UDDI Process



1) software companies populate the registry with descriptions of technical models

2) businesses populate the registry with descriptions of their services

UDDI Registry

Business registries

Service registries

3) UDDI assigns a unique identifier to each registry and stores them in an Internet registry

4) search engines and business applications query the registry to discover services

5) businesses use this data to facilitate business integration

UDDI Goals

Primary goal of UDDI:

- the specification of a framework for describing and discovering web services.

Two main goals for UDDI registry specifications:

1) support developers in finding information about services
2) enable dynamic binding

UDDI Data Structures and APIs

UDDI defines data structures and APIs for:

1) publishing service descriptions in the registry

2) querying the registry to look for published descriptions

UDDI History

The initiative was announced on 6th September 2000.

The first three UDDI versions were developed by uddi.org.

After completion of UDDI version 3.0, uddi.org submitted the specifications to the Organization for the Advancement of Structured Information Standards (OASIS).

In April 2003, the UDDI version 2.0 specifications were approved as a formal OASIS standard.

UDDI version 3.0.2 has been published as a Committee Draft in October 2004.

The specifications are available at:
http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

UDDI Services

UDDI defines a number of lookup services allowing clients to look up and retrieve information to access a web service.

Four services are provided to clients:

1) white pages to look up a web service by business identification
2) yellow pages to look up a web service by topic
3) green pages for searches through web services features

A service for providers:

4) UDDI business registry to publish/request information about web services

UDDI Members

UDDI members are companies that are committed to the:

1) enhancement,
2) evolution
3) world-wide acceptance

of the UDDI registry.

For instance, some UDDI members:

a) Cisco Systems
b) IBM
c) Intel
d) Microsoft
e) NEC Corporation
f) Oracle
g) SAP
h) Sun Microsystems
i) ...

UDDI Operators

A UDDI operator is an organization that hosts an implementation of the UDDI Business Registry (UBR).

Currently, there are four operators:

a) IBM UBR node: http://uddi.ibm.com/

b) Microsoft UBR node: http://uddi.microsoft.com/

c) SAP UBR node: http://uddi.sap.com/

d) NTT UBR node: http://www.ntt.com/uddi/

Different Operators – One Registry

A company needs to register its information with only one operator, called the custodian.

UBR is based on: “register once, publish everywhere”.

The information contained in one registry is replicated in the other registries, not instantaneously, but at least every 12 hours.

A company can update its information only through its custodian.

UDDI Specifications

The UDDI specifications define:

1) SOAP APIs that applications use to query and to publish information to a UDDI registry

2) XML Schema of the registry data model and the SOAP message formats

3) WSDL definitions of the SOAP APIs

4) UDDI registry definitions of various identifier/category systems that may be used to identify and categorize UDDI registrations

UDDI Outline

1) Introduction

2) Concepts

3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag

4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service

5) Summary

UDDI - UUID

A service registry maintains information about

1) businesses
2) services
3) technical information
4) specification of services

Instances of these data structures are kept separate and are identified by a Universally Unique Identifier (UUID).

UUIDs are assigned when the data structure is first inserted in the registry.

UUIDs are hexadecimal strings whose structure and generation algorithm is defined by the ISO/IEC 11578:1996 standard.

UDDI – Other Identifiers

UDDI allows to define additional international identifiers.

These identifiers are assigned to business and technical information in order to retrieve data according to them.

Two identifier types have been adopted and made core part of the UDDI operator registries:

1) D-U-N-S

2) Thomas Register

D-U-N-S Identifier

D-U-N-S number is a unique nine-digit identification sequence which provides unique identifiers of single business entities.

D-U-N-S is provided by Dun & Bradstreet.

Dun & Bradstreet is a company providing global business information, tools, and insights.

Reference: http://www.dnb.com

Thomas Register Identifier

Thomas Register identifiers are used in Thomas Global Register.

The Thomas Global Register is a directory of manufacturers and distributors from 28 countries.

The registry is classified by products and services categories.

Reference: http://www.thomasregister.com

UDDI Core Identifier Systems

The UDDI - UUIDs for these identifier systems are:

Name	UUID
D-U-N-S	uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823
Thomas Registry	uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039

In order to take advantage of these identification systems, businesses need to provide the relevant codes when publishing information.

This information is provided using the `identifierBag` element.

Categorization and Classification

In order to improve the search procedure UDDI provides a method to perform intelligent searches through categorization and classification.

Categorization: is the process of creating categories.

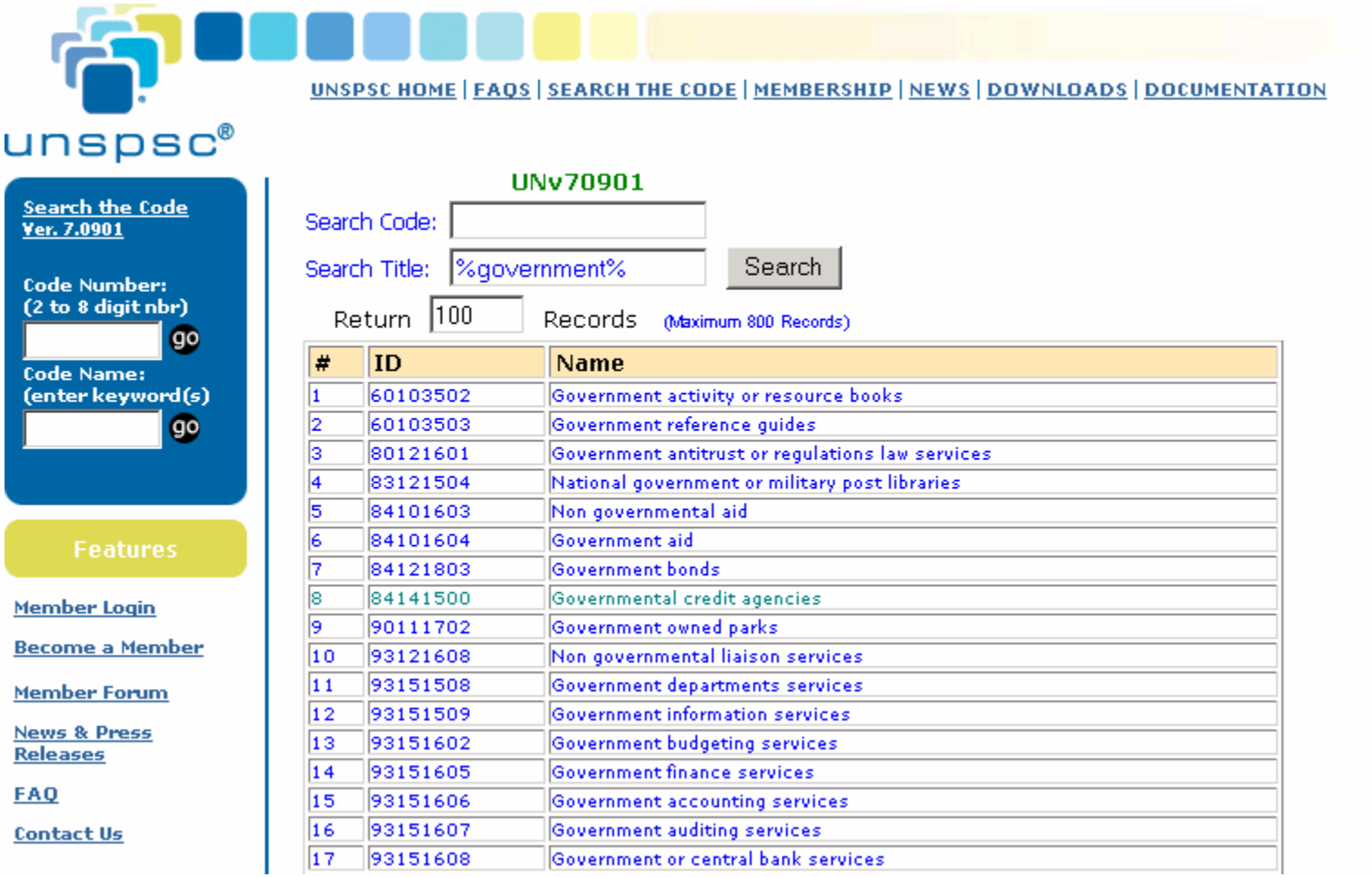
Classification: is the process of assigning objects to these predefined categories.

UDDI defines a set of built-in classification schemes (or taxonomies):

North American Industry Classification System (NAICS) http://www.census.gov/epcd/www/naics.html	classifies businesses by industry
United Nations Standard Products and Services Code (UNSPSC) http://www.unspsc.org/	classifies products and services
ISO 3166 http://www.iso.org/iso/en/prods-services/iso3166ma	classifies geographic locations

Classification Codes: UNSPSC

Examples of UNSPSC classification codes:



unspsc®

UNSPSC HOME | FAQS | SEARCH THE CODE | MEMBERSHIP | NEWS | DOWNLOADS | DOCUMENTATION

Search the Code Ver. 7.0901

Code Number: (2 to 8 digit nbr) go

Code Name: (enter keyword(s) go

Features

Member Login

Become a Member

Member Forum

News & Press Releases

FAQ

Contact Us

UNv70901

Search Code:

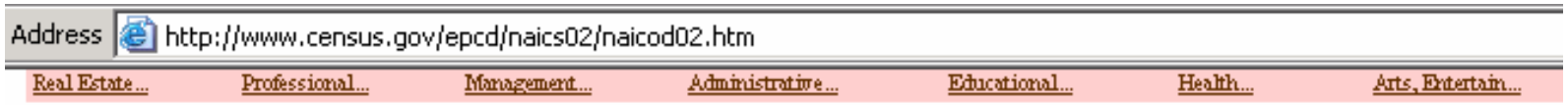
Search Title: %government% Search

Return 100 Records (Maximum 800 Records)

#	ID	Name
1	60103502	Government activity or resource books
2	60103503	Government reference guides
3	80121601	Government antitrust or regulations law services
4	83121504	National government or military post libraries
5	84101603	Non governmental aid
6	84101604	Government aid
7	84121803	Government bonds
8	84141500	Governmental credit agencies
9	90111702	Government owned parks
10	93121608	Non governmental liaison services
11	93151508	Government departments services
12	93151509	Government information services
13	93151602	Government budgeting services
14	93151605	Government finance services
15	93151606	Government accounting services
16	93151607	Government auditing services
17	93151608	Government or central bank services

Classification Codes: NAICS

Examples of NAICS classification codes for public administration:



Public Administration

2002 NAICS Code	2002 NAICS Title
92	Public Administration
921	Executive, Legislative, and Other General Government Support
9211	Executive, Legislative, and Other General Government Support
92111	Executive Offices
921110	Executive Offices
92112	Legislative Bodies
921120	Legislative Bodies
92113	Public Finance Activities
921130	Public Finance Activities
92114	Executive and Legislative Offices, Combined
921140	Executive and Legislative Offices, Combined
92115	American Indian and Alaska Native Tribal Governments
921150	American Indian and Alaska Native Tribal Governments
92119	Other General Government Support
921190	Other General Government Support

UDDI Classification Schemes

The UDDI - UUIDs for the classification schemes are:

Name	Type	UUID
NAICS	business	uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
UNSPSC	product and services	uuid:CD153257-086A-4237-B336-6BDCBDCC6634
ISO 3166	geographic	uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88

In order to take advantage of these classification schemes businesses need to provide the relevant classification information as they publish their entries.

This is done using the `categoryBag` element.

UDDI Core tModels

UDDI registries store `tModels`.

tModels represent technical specifications.

UDDI describes the classification schemes NAICS, UNSPSC and ISO 3166 as `tModels`.

UDDI Type Taxonomy

UDDI type taxonomy has been established to assist in general categorization of `tModels`.

UDDI type taxonomy is described as a tModel:

```
tModel name= uddi-org:types
tModel description = UDDI Type Taxonomy
tModel UUID: uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4
```

The categorization information for each tModel is added in the `categoryBag` element to indicate the type of `tModel`.

uddi-org:types: Values 1

Some of the values defined in uddi-org:types taxonomy include:

1) `namespace`: represents a scoping constraint or domain for a set of information. Similar to the namespace functionality used for XML

2) `specification`: is used for `tModels` that define interactions with a web service

3) `xmlSpec`: is used to indicate that the interaction with the service is via XML

4) `soapSpec`: is used to indicate that the interaction with the service is via SOAP

uddi-org:types: Values 2

5) `wsdlSpec`: is used to indicate that the web service is described using WSDL

6) `protocol`: is used for a `tModel` describing a protocol of any sort

7) `transport`: is used for a `tModel` specifying specific types of protocols: HTTP, FTP, and SMTP

UDDI Outline

1) Introduction

2) Concepts

3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag

4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service

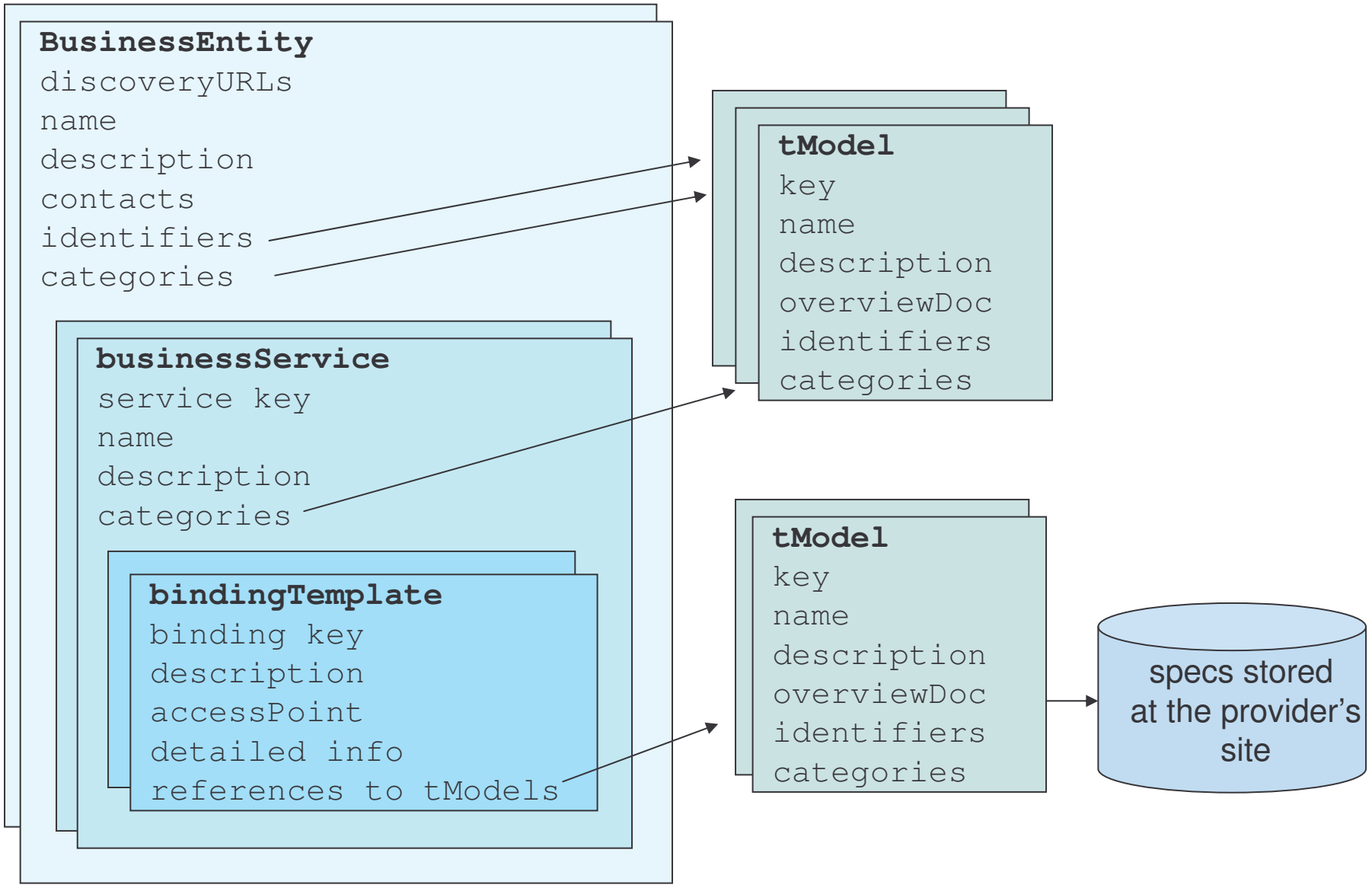
5) Summary

UDDI Data Types

UDDI version 2.0 is modeled using five data types:

1) businessEntity: describes an organization that provides web services
2) businessService: describes a group of related web services offered by a businessEntity
3) bindingTemplate: describes the technical information necessary to use a particular web service
4) tModel: (technical model) is a generic container for any kind of specification
5) publisherAssertion: is used to define a relationship between two or more businessEntity elements

UDDI Data Model



BusinessEntity

The structure is used to represent all known information about a business or entity that publishes descriptive information about the entity as well as the services that it offers.

The structure includes:

1) attributes:
   a) businessKey
   b) operator
   c) authorizedName
2) elements:
   a) discoveryURLs
   b) name
   c) description
   d) contacts
   e) businessServices
   f) identifierBag
   g) categoryBag

BusinessEntity Attributes

1) `businessKey`: UUID for a given instance of the `businessEntity` structure

2) `operator`: is the certified name of the UDDI registry site operator that manages the master copy of the `businessEntity` data

3) `authorizedName`: is the recorded name of the individual that published the `businessEntity` data.

```
<?xml version="1.0" encoding="utf-8" ?>
<businessEntity businessKey="5774466b-089d-4fa8-b8cb-fa2ead5329c5"
   operator="Microsoft Corporation"
   authorizedName="NiceWeather UDDI Publisher"
  . . .
```

BusinessEntity Elements 1

1) `discoveryURLs`: (optional) is used to hold pointers to alternate, file based service discovery mechanisms.

```
<discoveryURLs>
 <discoveryURL
   useType="businessEntity">http://uddi.microsoft.com/discovery?
   businessKey=5774466b-089d-4fa8-b8cb-fa2ead5329c5
 </discoveryURL>
</discoveryURLs>
```

2) `name`: (repeating element) are the human readable names recorded for the businessEntity, adorned with a unique xml:lang value

```
<name xml:lang="en">NiceWeather</name>
<name xml:lang="es">BuenTiempo</name>
```

BusinessEntity Elements 2

3) `description`: (optional repeating element) one or more short business descriptions. One description is allowed per language code supplied.

```
<description xml:lang="en">UDDI businessEntity for NiceWeather.
</description>
<description xml:lang="es">UDDI businessEntity para BuenTiempo.
</description>
```

BusinessEntity Elements 3

4) `contacts`: (optional) list of contact information including:
   a) `useType`: attribute describing the type of contact in freeform text
   b) `description`: (optional) descriptions in more than one language of the reasons for using the contact
   c) `personName`: is the name of the person or the name of the job role
   d) `phone`: (optional)
   e) `email`: (optional)
   f) `address`: (optional repeating element) this structure represents the printable lines suitable for addressing an envelope

```
<contact useType="Technical Information">
   <description>Contact for technical information</description>
   <personName>Susan Carroll</personName>
   <phone useType="Main Office">853.782.923</phone>
   <email useType="CTO">susancarroll@niceweather.com</email>
   <address useType="Main Office" sortCode="10001">
      <addressLine>2001 Kuong Building</addressLine>
      <addressLine>Macao</addressLine>
   </address>
</contact>
```

BusinessEntity Elements 4

5) `businessServices`: (optional) list of one or more logical business service descriptions

6) `identifierBag`: (optional) list of “name-value” pairs that can be used to record identifiers for a businessEntity

7) `categoryBag`: (optional) list of “name-value” pairs that are used to tag a `businessEntity` with specific taxonomy information. For instance: industry, product or geographic codes

BusinessEntity Example

```
<businessEntity businessKey="5441234-763E-11D5-B565-000782FD9C23">
  <name xml:lang="en">NiceWeather</name>
  <description xml:lang="en">UDDI businessEntity for NiceWeather.
  </description>
  <contacts>
    <contact useType="Technical Information">
     <description>Contact for technical information</description>
     <personName>Susan Carroll</personName>
     <phone useType="Main Office">853.782.923</phone>
     <email useType="CTO">susancarroll@niceweather.com</email>
     <email useType="General Information">info@niceweather.com</email>
     <address useType="Main Office" sortCode="10001">
        <addressLine>2001 Kuong Building</addressLine>
        <addressLine>Macao</addressLine>
     </address>
    </contact>
    <contact> ... </contact>
  </contacts>
 <businessServices> . . . </businessServices>
<identifierBag> . . . </identifierBag>
<categoryBag> . . . </categoryBag>
</businessEntity>
```

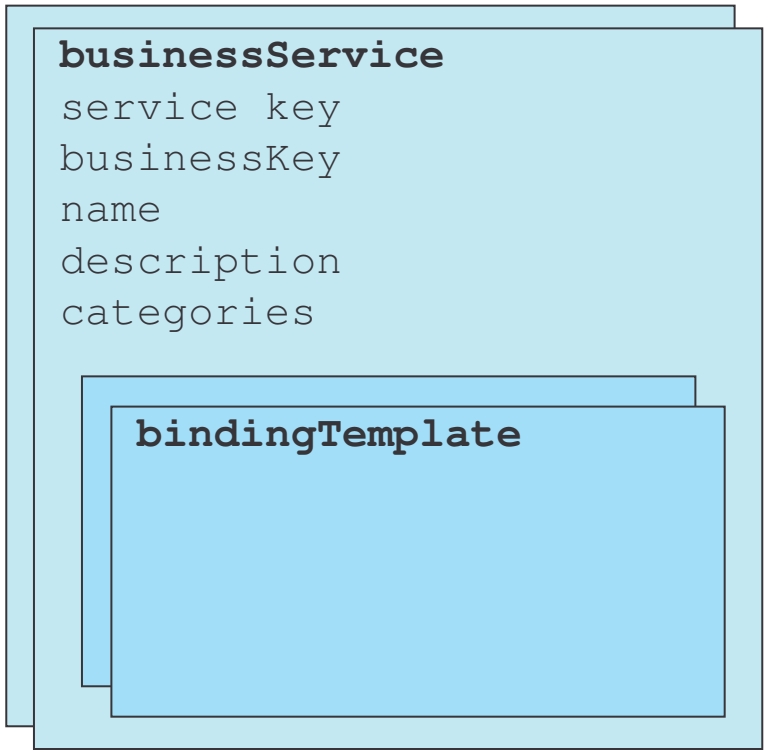
BusinessService

The `businessService` element is the root element for describing a logical business service.

The structure includes:

1) attributes:
   a) `serviceKey`
   b) `businessKey`

2) elements:
   a) `name`
   b) `description`
   c) `bindingTemplates`
   d) `categoryBag`

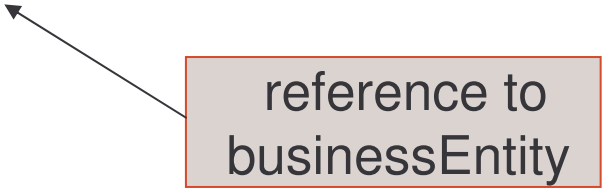


BusinessService Attributes

1) `serviceKey`: UUID for identifying a given `businessService.`

2) `businessKey`: is a direct reference to the `businessEntity` that is associated with it.

```
<businessService serviceKey="1T264170-2E3E-TE97-M9A8-F11111JE9WBH"
                 businessKey="5441234-763E-11D5-B565-000782FD9C2" >
```



BusinessService Elements

1) `name`: (optional repeating element) are the human readable names recorded for the `businessService`, adorned with a unique `xml:lang` value

2) `description`: (optional) descriptions in more than one language of the logical service family

3) `bindingTemplates`: this structure holds the technical service description information related to a given business service family

4) `categoryBag`: (optional) list of “name-value” pairs that are used to tag a `businessService` with specific taxonomy information. For instance: industry, product or geographic codes.

BusinessService Example

```
<businessServices>
   <businessService serviceKey="1T264170-2E3E-TE97-M9A8-F11111JE9WBH"
                    businessKey="5441234-763E-11D5-B565-000782FD9C2" >
   <name>ask Data Submission</name>
   <description>NiceWeather ask data submission service.</description>

 <bindingTemplates>
   . . .
 </bindingTemplates>

 <categoryBag>
   . . .
 </categoryBag>

</businessServices>
```

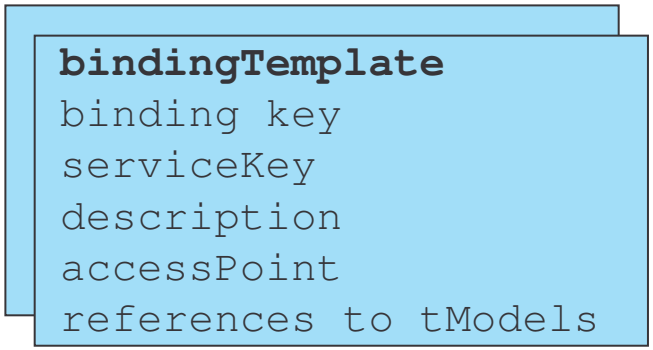
BindingTemplate

The `bindingTemplate` element contains the technical information necessary to invoke a specific web service.

The same logical service may have more than one type of binding (SOAP-HTTP, HTTP browser-based binding, etc.), each of them is described in a separate `bindingTemplate` element.

The structure includes:

1) attributes:
   a) `bindingKey`
   b) `serviceKey`
2) elements:
   a) `description`
   b) `accessPoint`
   c) `hostingRedirector`
   d) `tModelInstanceDetails`

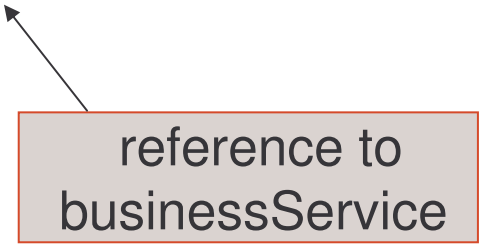


BindingTemplate Attributes

1) `bindingKey`: UUID for identifying a given `bindingTemplate`

2) `serviceKey`: is a direct reference to the `businessSevice` that is associated with it.

```
<bindingTemplate bindingKey="2T5FDS12-04T4-GTT6-08GF-Y67E454357T89"
                 serviceKey= "1T264170-2E3E-TE97-M9A8-F11111JE9WBH">
```



BindingTemplate Elements 1

1) `description`: (optional) descriptions in more than one language of the technical service entry point

2) `accessPoint`: is an attribute-qualified element that is used to convey the entry point address suitable for calling a particular web service. A single attribute named URLType is provided with the following values:

   a) `mailto`
   b) `http`
   c) `https`
   d) `ftp`
   e) `fax`
   f) `phone`
   g) `other`

3) `hostingRedirector`: used to designate that a `bindingTemplate` entry is a pointer to a different `bindingTemplate` entry. Is a required element if `accessPoint` is not provided. Is adorned with a `bindingKey` attribute, giving the redirected reference to a different bindingTemplate

BindingTemplate – Elements 2

3) `tModelInstanceDetails`: list of zero or more `tModelInstanceInfo` elements.

The `tModelInstanceInfo` is a distinct fingerprint used to identify services including one attribute (`tModelKey`) and two elements:

a) `tModelKey`: is a unique key reference to a `tModel` describing the implementation details of the service

b) `description`: (optional) descriptions in more than one language describing what role a `tModel` reference plays in the overall service description

c) `instanceDetails`: (optional) is a structure providing additional information required to understand the details relative to the `tModelKey` reference, or to provide further parameters and settings support

BindingTemplate Example

```
<bindingTemplate bindingKey="2T5FDS12-04T4-GTT6-08GF-Y67E454357T89"
                 serviceKey= "1T264170-2E3E-TE97-M9A8-F11111JE9WBH">
  <description>SOAP based ask data submission service.</description>
  <accessPoint URLtype="http:">
        http://www.example.com/WeatherService/askData
  </accessPoint>

  <tModelsInstanceDetails>
    <tModelsInstanceInfo
        tModelKey="uuid:67DF6F55-SG56-976F-9U77-570425RFD68J">
        <description>HTTP address</description>
    </tModelsInstanceInfo>
  </tModelsInstanceDetails>

</bindingTemplate>
```

BindingTemplate - instanceDetails

The `instanceDetails` element is added to the `tModelInstanceInfo` to specify additional service implementation details. It has the following structure :

a) `description`: (optional) descriptions in more than one language describing the purpose and/or the use of the particular instanceDetails entry

b) `overviewDoc`: (optional) is a structure referencing to remote descriptive information or instructions related to proper use of the `bindingTemplate` technical sub-element. The structure contains:
- `description`
- `overviewURL`

c) `instanceParms`: (optional) used to contain setting parameters or a URL reference to a file containing setting or parameters required to use a specific facet of a `bindingTemplate` description

instanceDetails Example

```
<tModelsInstanceDetails>
   <tModelsInstanceInfo
     tModelKey="uuid: 9OC7WMI1-0998-0375-NE00-0702IBA09049">
     <description>Reference to tModel with Web Service interface
                  definition
     </description>
     <instanceDetails>
       <overviewDoc>
         <description>
            Reference to additional semantic specification of the
            web service.
         </description>
         <overviewURL>
           http://www.example.com/WeatherService/additionalFeatures.xml
         </overviewURL>
       </overviewDoc>
     </instanceDetails>
</tModelInstanceInfo>
</tModelsInstanceDetails>
. . .
```

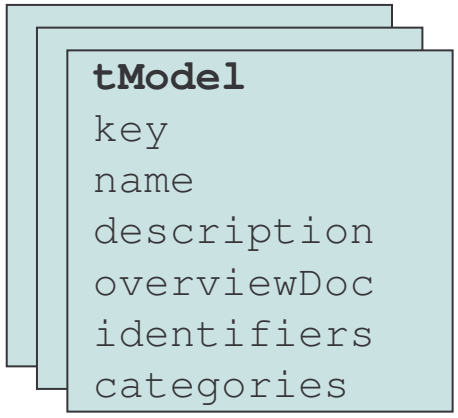
tModel Structure

The primary role that a tModel plays is to represent a technical specification.

The structure includes:

1) attributes:
   a) `tModelKey`
   b) `operator`
   c) `authorizedName`

2) elements:
   a) `name`
   b) `description`
   c) `overviewDoc`
   d) `identifierBag`
   e) `categoryBag`



tModel Attributes

1) `tModelKey`: is a unique key for a given `tModel` structure

2) `operator`: is the certified name of the UDDI registry site operator that manages the master copy of the `tModel` data

3) `authorizedName`: is the recorded name of the individual that published the `tModel` data

tModel Elements

1) `name`: the name recorded for the `tModel`

2) `description`: (optional repeating element) one description is allowed per national language code supplied

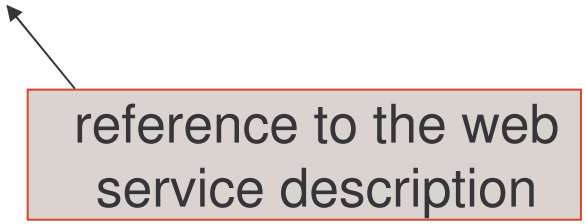
3) `overviewDoc`: is a structure referencing to remote descriptive information or instructions related to proper use of the `tModel`. The structure contains:
   - `description`
   - `overviewURL`

4) `identifierBag`: (optional) is an optional list of “name-value” pairs that can be used to record identification numbers for a `tModel`

5) `categoryBag`: (optional) is a list of “name-value” pairs that are used to tag a `tModel` with specific taxonomy information

tModel Example 1

```
<tModel tModelKey="uuid:67DF6F55-SG56-976F-9U77-570425RFD68J">
    <name>Ask Weather Data Service</name>
    <description xml:lang="en"> Service interface definition for ask
       weather data submission service.</description>
    <overviewDoc>
       <description xml:lang="en">Reference to the WSDL document that
         contains the service interface definition for the ask data
         submission service.</description>
       <overviewURL>
          http://www.example.com/WeatherService/askData.wsdl
       </overviewURL>
    </overviewDoc>
```



tModel Example 2

```
    <identifierBag>
      <keyedReference keyName="DUNS"
         keyValue="00-111-1111"
         tModelKey="uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823" />
    </identifierBag>


    <categoryBag>
      <keyedReference keyName="uddi-org:types"
          keyValue="soapSpec"
          tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4" />
      <keyedReference keyName="uddi-org:types"
          keyValue="wsdlSpec"
          tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB" />
      <keyedReference keyName="Weather"
          keyValue="84902934"
          tModelKey="uuid:J6LIJ84T-3874-8301-Y9JY-2JS76GFD60J8" />
     </categoryBag>

</tModel>
```

reference to DUNS tModel

reference to UDDI type taxonomy tModel

publisherAssertion 1

The `publisherAssertion` is a declaration done by two `businessEntity`s in order to establish a relationship between them.

The relation becomes visible when both `businessEntity`s published the same information.

The structure includes:

a) `fromKey`: UUID referencing the first `businessEntity`
b) `toKey`: UUID referencing the second `businessEntity`

c) `keyedReference`: designates the relationship between the business entities by three attributes:
   - `tModelKey`: reference the relationship type system
   - `keyName`
   - `keyValue`

   } are used to indicate the specific type of relationship

publisherAssertion 2

According to the relationship type system defined in the UDDI specification:

Values for the `keyValue` attribute are:

1) `parent-child`: the business referenced by the `fromKey` is the parent of the business referenced by the `toKey`
2) `peer-peer`: both businesses referenced are partners or affiliates
3) `identity`: both businesses referenced are the same. Is typically used to assert different divisions, units and departments of the same organization

Providing, publisher-assertion capabilities, UDDI allows large corporations to describe aspects of their businesses - such as: divisions, partners, and subsidiaries - to users of the UBR.

publisherAssertion Example

```
<publisherAssertion>
  <fromKey>5441234-763E-11D5-B565-000782FD9C23</fromKey>
  <toKey>U7O0J86-JU77-MN88-G86G-096YY8EV9JK1</toKey>
  <keyedReference
     keyName="subsidiary"
     keyValue="parent-child"
     tModelKey="uuid:807A2C6A-EE22-470D-ADC7-E0424A337C03" />
</publisherAssertion>
```

reference to UDDI relationships tModel

This declaration models the relationship between the company NiceWeather (`fromKey`) and its subsidiary in Hong Kong (`toKey`).

For this example, the `keyValue` attribute defines that NiceWeather is the parent-company of NiceWeather Hong Kong.

identifierBag

UDDI defines the notion of attaching identifiers to data using the `identifierBag`

Two of the core data types support attaching identifiers to data:

1) `businessEntity`

2) `tModel`

An `identifierBag` is an element that holds zero or more instances of something called `keyedReference`.

identifierBag Example

```
<identifierBag>
   <keyedReference keyName="DUNS"
      keyValue="00-111-1111"
      tModelKey="uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823" />
 </identifierBag>
```

Is a general-purpose structure for a "name-value" pair with one additional attribute referencing a `tModel` structure

The reference to a `tModel` structure makes the identifier scheme extensible, allowing `tModel`s to be used as a conceptual namespace qualifiers.

categoryBag

The `categoryBag` is the container that describes relevant classification information that businesses provide when publishing their services.

```
<categoryBag>
      <keyedReference keyName="uddi-org:types"
          keyValue="soapSpec"
          tModelKey="uuid:T6TGU76T-9876-3234-4J90-34J997T78TG5" />
</categoryBag>
```

Three of the core data types support attaching identifiers to data:

1) `businessEntity`
2) `businessService`
3) `tModel`

categoryBag Example

```
<categoryBag>
  <keyedReference keyName="Weather Station"
      keyValue="41114410"
      tModelKey="CD153257-086A-4237-B336-6BDCBDCC6634"/>
  <keyedReference keyName="Macao"
      keyValue="MO"
      tModelKey="4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88"/>
</categoryBag>
```

reference to UNSPSC tModel

reference to ISO 3166 tModel

This `categoryBag` corresponds to the `businessEntity` of NiceWeather.

We are defining that the business belongs to the “41114410” category that corresponds to “weather stations” according to UNSPSC codes, and we are locating the company in Macao according to ISO-3166.

UDDI Outline

1) Introduction

2) Concepts

3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag

4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service

5) Summary

Using a UDDI Registry

A UDDI is itself an instance of a web service.

Entries in the registry can be published and queried using SOAP-based service interface.

The WSDL service interface definitions for a UDDI registry can be found at:

UDDI Inquiry API v2.0	http://uddi.org/wsdl/inquire_v2.wsdl
UDDI Publication API v2.0	http://uddi.org/wsdl/publish_v2.wsdl
UDDI portTypes API v3.0	http://uddi.org/wsdl/uddi_api_v3_portType.wsdl
UDDI Bindings API v 3.0	http://uddi.org/wsdl/uddi_api_v3_binding.wsdl

UDDI Business Registry 1

Four organization hosts nodes in the UDDI Business Registry.

Most of them also host a test registry.

Three URLs are provided for each:

Host Organization	Access Type	URL
IBM Business Registry	Web	http://uddi.ibm.com
	Inquiry	http://uddi.ibm.com/ubr/inquiryapi
	Publish	http://uddi.ibm.com/ubr/publishapi
IBM Test Registry	Web	http://uddi.ibm.com/testregistry/registry.html
	Inquiry	http://www-3.ibm.com/services/uddi/testregistry/inquiryapi
	Publish	https://uddi.ibm.com/testregistry/publishapi

UDDI Business Registry 2

Host Organization	Access Type	URL
Microsoft Business Registry	Web	http://uddi.microsoft.com
	Inquiry	http://uddi.microsoft.com/inquire
	Publish	https://uddi.microsoft.com/publish
Microsoft Test Registry	Web	http://test.uddi.microsoft.com
	Inquiry	http://test.uddi.microsoft.com/inquire
	Publish	https://test.uddi.microsoft.com/publish
SAP Business Registry	Web	http://uddi.sap.com
	Inquiry	http://uddi.sap.com/uddi/api/inquiry
	Publish	https://uddi.sap.com/uddi/api/publish
SAP Test Registry	Web	http://udditest.sap.com
	Inquiry	http://udditest.sap.com/UDDI/api/inquiry
	Publish	https://udditest.sap.com/UDDI/api/publish

UDDI Business Registry 3

Host Organization	Access Type	URL
NTT Business Registry	Web	http://www.ntt.com/uddi
	Inquiry	http://www.uddi.ne.jp/ubr/inquiryapi
	Publish	https://www.uddi.ne.jp/ubr/publishapi

Task 78: UDDI Registry

1) access the UDDI v2 IBM Business Registry: http://uddi.ibm.com
2) select option “Search UDDI Business Registry”
3) search for “Business” – starting with: “Public Administration”
4) select Administration Division
5) access the file in the DiscoveryURL
6) save this file in your PC

Task 79: Test UDDI Registry

7) open the file:

   a) where is the master copy of the businessEntity data?

   b) who registered the information for this businessEntity?

   c) access the business contact on the web page containing the businessEntity information

   d) what is the information described for the contact? check the address details with the information on the XML file

   e) how many services related does the business have?

   f) what is the information provided for this service and how can you access this service?

Task 80: Test UDDI Registry

8) go back to the page where the results of step 3 where shown

9) select “services” for the Administration Division

10) what is the information shown in the page?

11) what links are provided?

12) select “home page”

13) what information is provided?

Publishing Service Descriptions 1

Most UDDI operators require the user to register before publishing any UDDI entries.

The registration process provides a publisher account consisting on a user-id and a password.

Entries in the registry are owned by the publisher who created them, and only the owner can update or delete a registry entry.

The UDDI publication APIs provide support for creating, updating, and deleting the following entries:

1) `businessEntity`
2) `businessService`
3) `bindingTemplate`
4) `tModel`
5) `publisherAssertions`

Authentication Token

Before using a publication API, an authentication token must be obtained, using the `get_authToken` API call.

Authentication tokens are required for all publication APIs. They represent an active session with the registry.

Authentication tokens are valid for a period of time defined by the registry.

The `discard_authToken` message is used to indicate the registry that the token can be discarded.

APIs for Publishing

APIs for publishing the four primary data types:

Datatype	Save API	Delete API
bindingTemplate	save_binding	delete_binding
businessEntity	save_business	delete_business
serviceBusiness	save_service	delete_service
tModel	save_tModel	delete_tModel

The `get_registeredInfo` API call is used to obtain a complete list of `businessEntity` and `tModel` entries owned by the publisher.

Publishing Publisher-Assertions

Five APIs are used to process publisher assertions:

1) `add_publisherAssertions`

2) `delete_publisherAssertions`

3) `get_publisherAssertions`: gets the full list of publisher assertions associated with a publisher's assertion collection

4) `get_assertionsStatusReport`: determines the status of current assertions

5) `set_publisherAssertions`: adds new assertions or updates existing assertions

WSDL and UDDI

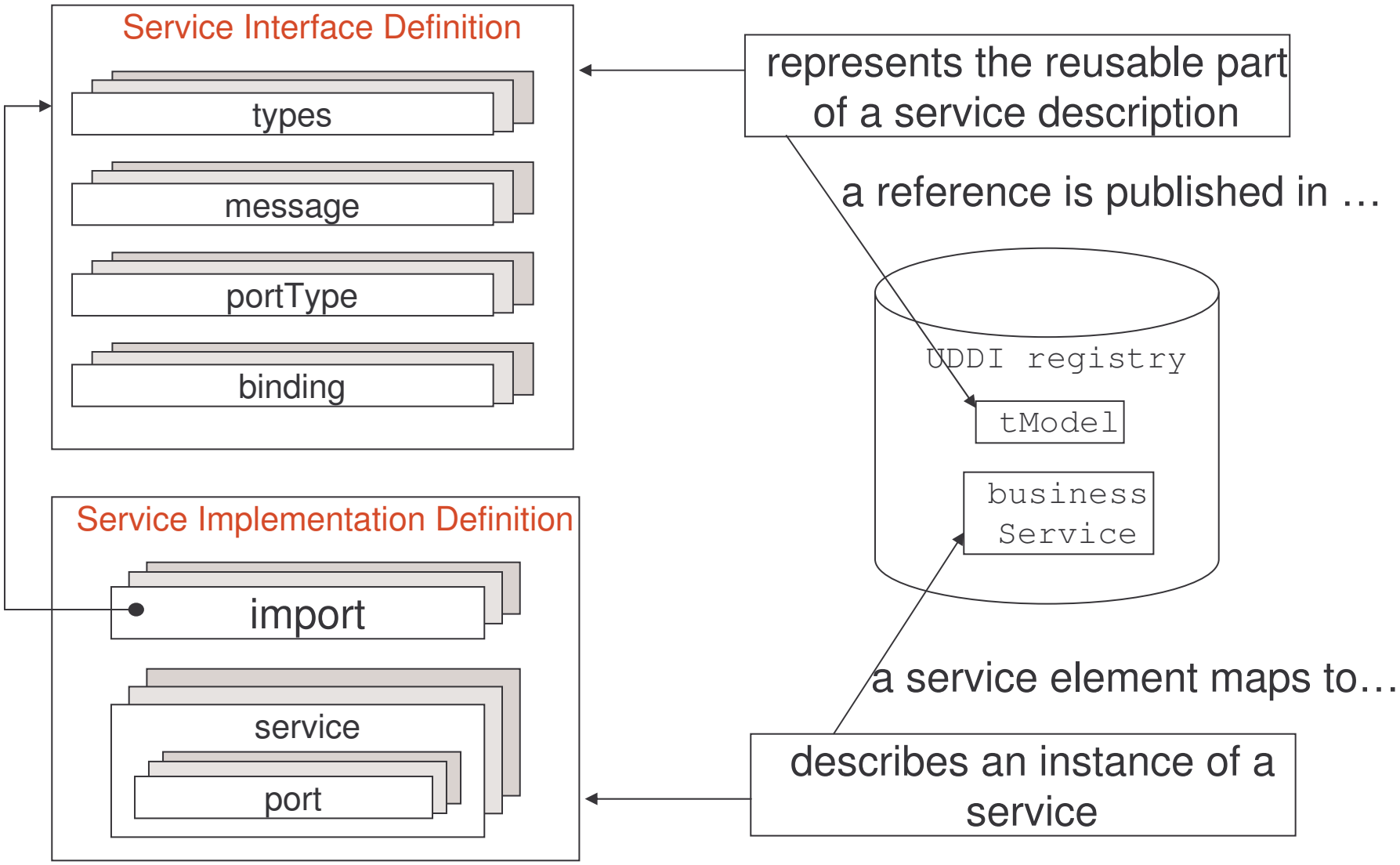
UDDI provides a method for publishing and finding businesses and services information.

UDDI provides support for many different types of service descriptions:

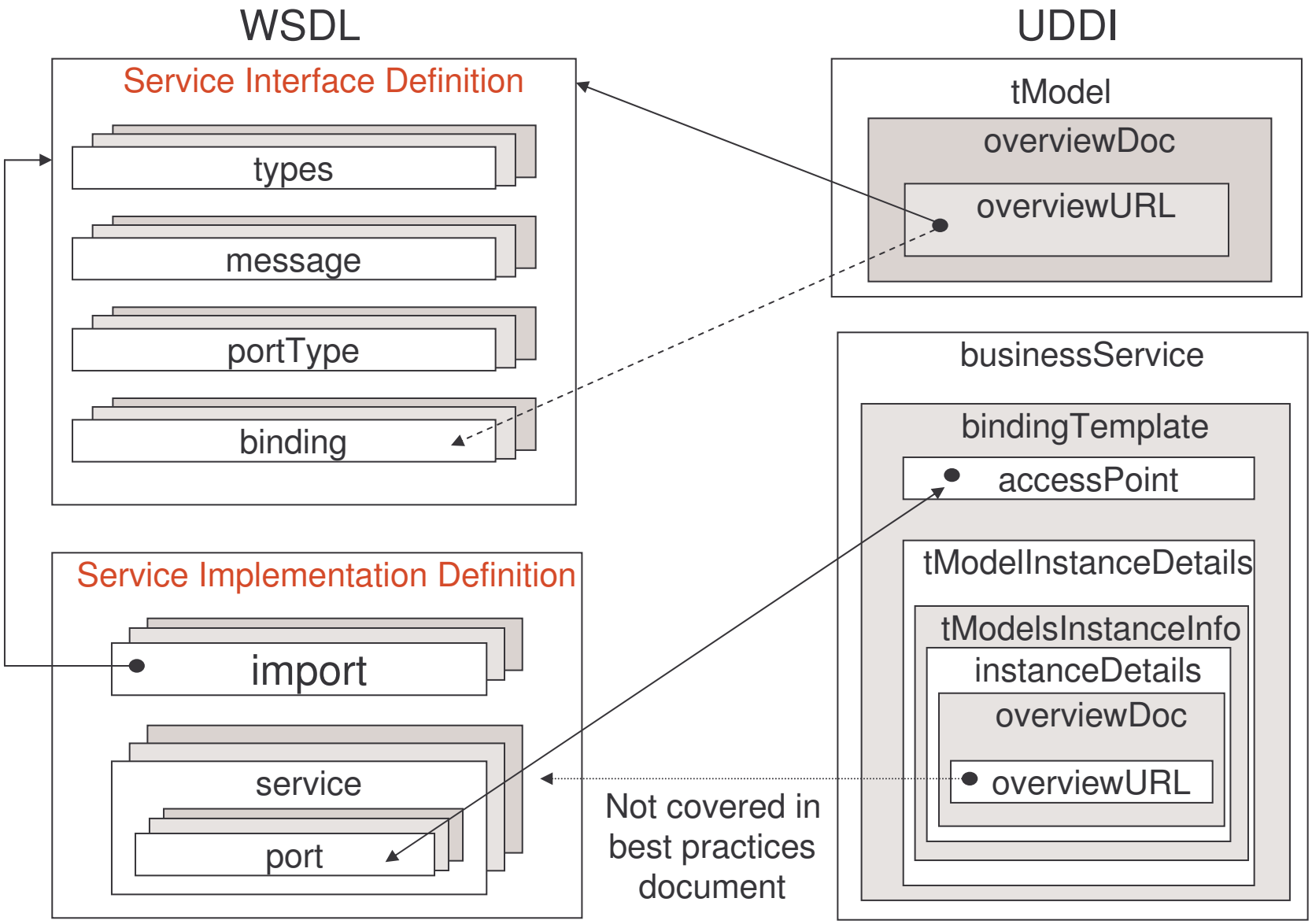
1) WSDL
2) plain ASCII text
3) RDF
4) others

The service description information defined in WSDL is complementary to the information found in a UDDI registry.

Mapping from WSDL to UDDI 1



Mapping from WSDL to UDDI 2



Mapping Co-Relations

1) Service Interface Definition - UDDI-`tModel`:

    a) the `overviewDoc` field in each new `tModel` will point to the corresponding WSDL document

2) Service Implementation Definition – UDDI-`businessService`:

    a) a `bindingTemplate` is created for each access endpoint. The network address of the access point is the `accessPoint` element
    b) one `tModelInstanceInfo` is created in the `bindingTemplate` for each wsdlSpec `tModel` that defines interfaces and bindings supported by the service

Procedure for Publishing Services

1) the WSDL service interface definition is created

2) the WSDL service interface definition is registered as UDDI `tModels`. Such models are called `wsdlSpec tModels`

3) programmers will build services conforming to the service definitions

4) the new service must be deployed and registered in the UDDI registry. A UDDI `businessService` data structure is created and registered

Publishing a Service Example 1

1) the WSDL service interface definition is created

2) the WSDL service interface definition is registered as UDDI `tModel`.

```
<tModel authorizedName="…" operator="…"
        tModelKey="9OC7WMI1-0998-0375-NE00-0702IBA09049">
  <name>Ask Weather Data Service</name>
  <description xml:lang="en">
       WSDL description of a standard ask data about weather service.
  </description>
  <overviewDoc>
     <description xml:lang="en">WSDL source document.</description>
     <overviewURL> "http://www.example.com/WeatherService/askData.wsdl"
     </overviewURL>
  <overviewDoc>
  <categoryBag>
     <keyedReference
        tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
        keyName="uddi-org:types"
        keyValue="wsdlSpec" />
  </categoryBag>
</tModel>
```

reference WSDL binding element

tModel is categorized as a WSDL specification

Publishing a Service Example 2

3) after the service is deployed, is registered as a `businessService`:

```
<businessService serviceKey="1T264170-2E3E-TE97-M9A8-F11111JE9WBH"
                 businessKey="5441234-763E-11D5-B565-000782FD9C23">
  <name>Ask Data Service</name>
  <description>Ask data submission service.</description>

 <bindingTemplates>
  <bindingTemplate bindingKey="2T5FDS12-04T4-GTT6-08GF-Y67E454357T89"
                   serviceKey= "1T264170-2E3E-TE97-M9A8-F11111JE9WBH">
    <description>SOAP based ask data submission service.</description>
    <accessPoint URLType="http:">
        http://www.example.com/WeatherService/askData
    </accessPoint>

  <tModelsInstanceDetails>
    <tModelsInstanceInfo
       tModelKey="uuid: 9OC7WMI1-0998-0375-NE00-0702IBA09049">
       <description>Reference to tModel with Web Service interface
            definition
       </description>
```

URL where the WS can be invoked

reference to tModel (wsdlSpec)

Publishing a Service Example 3

```
  <instanceDetails>
      <overviewDoc>
         <description>
             Reference to WSDL service implementation document.
         </description>
         <overviewURL>
        http://www.example.com/WeatherService/askDataImplementation.wsdl
    </overviewURL>
      </overviewDoc>
 </instanceDetails>
</tModelInstanceInfo>
</tModelsInstanceDetails>
</bindingTemplate>
</bindingTemplates>
. . .
</busienessService>
```

reference to service implementation definition (not defined in best practices)

Service with Multiple Bindings 1

The WSDL service interface definition contains multiple bindings:

```
<portType name="CancelUserPortType">
. . .
</portType>

<portType name="AskDataPortType">
. . .
</portType>

<binding name="CancelUserSoapBinding">
. . .
</binding>

<binding name="AskDataSoapBinding">
. . .
</binding>
```

Service with Multiple Bindings 2

Create a `tModel` for each binding definition using an `XPointer` in the `overviewURL` element

```
. . .
  <overviewDoc>
     <description>
             Reference to WSDL service implementation document.
     </description>
     <overviewURL>
        http://www.example.com/WeatherService/MultipleBindings.wsdl
        xmlns(wsdl=http://schemas.xmlsoap.org/wsdl/)
        xpointer (//wsdl:binding[@name="AskDataSoapBinding"])
     </overviewURL>
  </overviewDoc>
. . .
```

Finding Service Description 1

When finding service descriptions, two types of inquiry APIs are available:

1) find APIs:

   a) `find_binding`: returns the contents of a `bindingTemplate`

   b) all others (business, service, tModel): retrieve a list of references (UDDI keys) to UDDI data entries matching the specified search criteria

2) get APIs: return the actual contents of a data entry

Finding Service Description 2

APIs for inquiring the four primary data types:

Datatype	Find API	Get API
bindingTemplate	find_binding	get_bindingDetail
businessEntity	find_business	get_businessDetail
serviceBusiness	find_service	get_serviceDetail
tModel	find_tModel	get_tModelDetail

Task 81: Find Business

Objective: find businesses in the IBM test UUDI registry. The name must match “Business”, case sensitive match, 100 instances.

```
1)  cd demos\UDDI\FindBusiness

2)  dir
FindBusinessExample.class
FindBusinessExample.java

1)  notepad FindBusinessExample.java

2)  java -cp demos\UDDI\FindBusiness FindBusinessExample
```

Task 82: Find Service

Objective: find a service in the IBM test UUDI registry. The tModel key is: “AFFC30D0-D83E-11D5-8055-0004AC49CC1E”.

```
1)  cd demos\UDDI\FindService

2)  dir
FindServiceExample.class
FindServiceExample.java

1)  notepad FindServiceExample.java

2)  java -classpath demos\UDDI\FindService
        FindServiceExample
```

UDDI Outline

1) Introduction

2) Concepts

3) Data Types
   a) businessEntity
   b) businessService
   c) bindingTemplate
   d) tModel
   e) publisherAssertion
   f) identifierBag
   g) categoryBag

4) UDDI Registry
   a) registry implementations
   b) publishing a service
   c) finding a service

5) Summary

UDDI Summary 1

UDDI is a platform independent, open framework for describing, discovering and integrating business services

Two types of service discovery:

1) static
2) dynamic

UDDI Summary 2

How it works:

a) service providers publish information about businesses and services

b) UDDI assigns unique identifiers to the information provided

c) service requestors query the registry

d) data returned is used to invoke web services

UDDI Summary 3

Four services provided:

a) white pages to look up a web service by the business

b) yellow pages to look up a web service by topic

c) green pages to look up a service through web services features

d) publish information

UDDI Summary 4

Global registry hosted by UDDI operators: IBM, Microsoft, SAP, NTT

UDDI principle: “register once – publish everywhere”

UDDI provides two core systems for identifying businesses and services:

1) D-U-N-S
2) Thomas Register

UDDI provides three classification schemes:

1) NAICS
2) UNSPSC
3) ISO-3199

UDDI Summary 5

UDDI is modeled using five data types:

1) `businessEntity`

2) `businessService`

3) `bindingTemplate`

4) `tModel`

5) `publisherAssertion`

UDDI Summary 6

1) `businessEntity`: represents all the information about a business

2) `businessService`: describes a logical business service

3) `bindingTemplates`: contains the technical information needed to invoke a web service

4) `tModel`: represents a technical model specification

5) `publisherAssertion`: establishes a relation between two `businessEntitys`

UDDI Summary 7

1) `identifierBag` and `categoryBag` can be defined for `businessEntity` and `tModels`.

2) `categoryBag` can also be defined for businessService

3) UDDI provides the UDDI type taxonomy for assisting in general categorization of the `tModel`s themselves

UDDI Summary 8

1) APIs are provided for publishing the four core data types:
   a) `save_business`
   b) `save_service`
   c) `save_binding`
   d) `save-tModel`

2) four APIs are also provided to delete the information related to these core data types.

3) five APIs are used to process publisher assertions:
   `add` / `delete` / `get_publisherAssertions`, `get_assertionsStatusReport` and `set_publisherAssertions`

4) for retreving information, two types of APIS are available:
   1) `find_business` - `find_`(`service`/`binding`/`tModel`)
   2) `get_businessDetail` - `get_`(`service`/`binding`/`tModel`)`Detail`

UDDI Summary 9

The WSDL service interface definition of a web service is created and published as a `tModel`.

The `overviewURL` of the `tModel` points to the WSDL document.

After the service implementation is built and deployed, it is published as a `businessService`.

A `bindingTemplate` is created for each access endpoint.

The `accessPoint` contains the network address of the service Implementation.

If the WSDL service interface definition contains multiple bindings, a `tModel` is created for each binding definition using an XPointer in the overviewURL element.

Security

Course Outline

1) Introduction

2) SOAP
   a) introduction
   b) messaging
   c) data structures
   d) protocol binding
   e) binary data

3) WSDL
   a) introduction
   b) the language
   c) transmission primitives
   d) WSDL extensions
   e) WSDL and Java

4) AXIS
   a) concepts
   b) service invocation
   c) tools and configuration
   d) service deployment
   e) service lifecycle

5) UDDI
   a) introduction
   b) concepts
   c) data types
   d) UDDI registry

6) Security
   a) security basics
   b) web service security
   c) digital signatures

Security Outline

1) Security Basics
2) Web Service Security
3) Digital Signatures

Security Context

e-Business as well as e-Government relies on the exchange of information between partners over insecure networks.

Sending messages over insecure networks implies risks.

Messages could be:

a) stolen

b) lost

c) modified

Security Requirements

Four security requirements must be addressed to ensure the safety of information exchanged among partners:

a) confidentiality
b) integrity
c) authentication
d) non-repudiation

} protect messages

One security requirement to assure resources:

1) authorization } protects resources

Confidentiality

Guarantees that exchanged information is protected against eavesdroppers.

For example:

a) license's information should not be exposed to outsiders

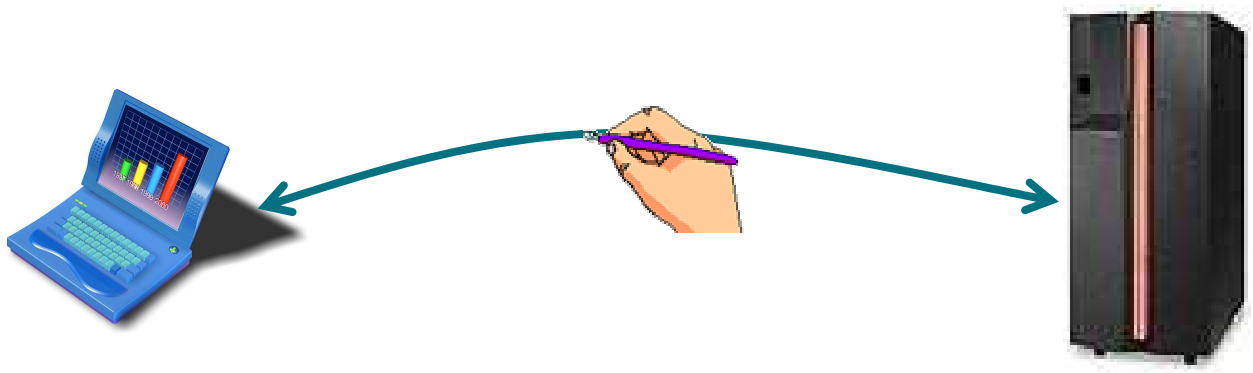
b) credit card information should not be wiretapped by third parties

Integrity

Assures that a message is not accidentally or deliberately modified in transit.

For example:

a) social security benefits's information should not be modified as it moves between citizens and government



Authentication

Guarantees that access to e-applications and data is restricted to those who can provide appropriate proof of identity.

For example:

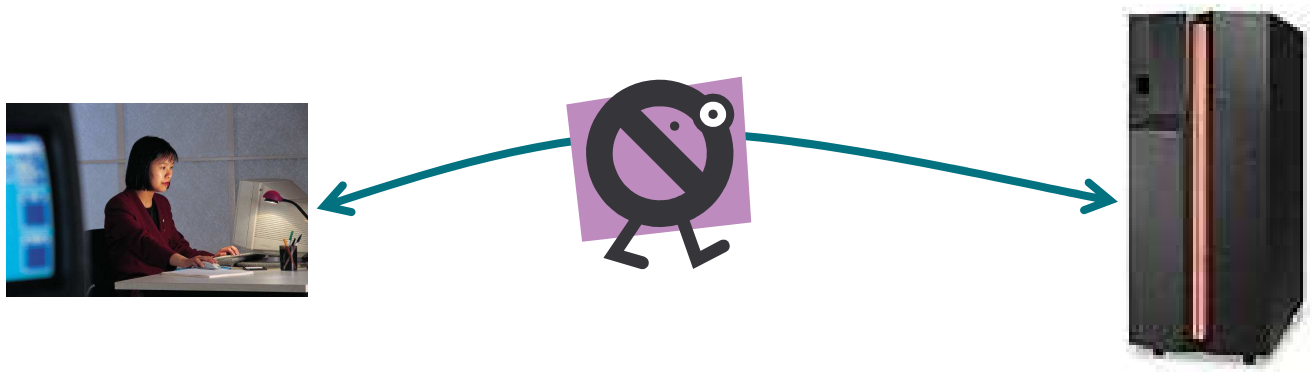
a) in order to track the status of a license application, subscribers are required to provide an ID and password as proof of their identity

Non-Repudiation

Guarantees that the message's sender cannot deny having send it.

For example:

a) with non-repudiation, once an application license is submitted, the business cannot repudiate it

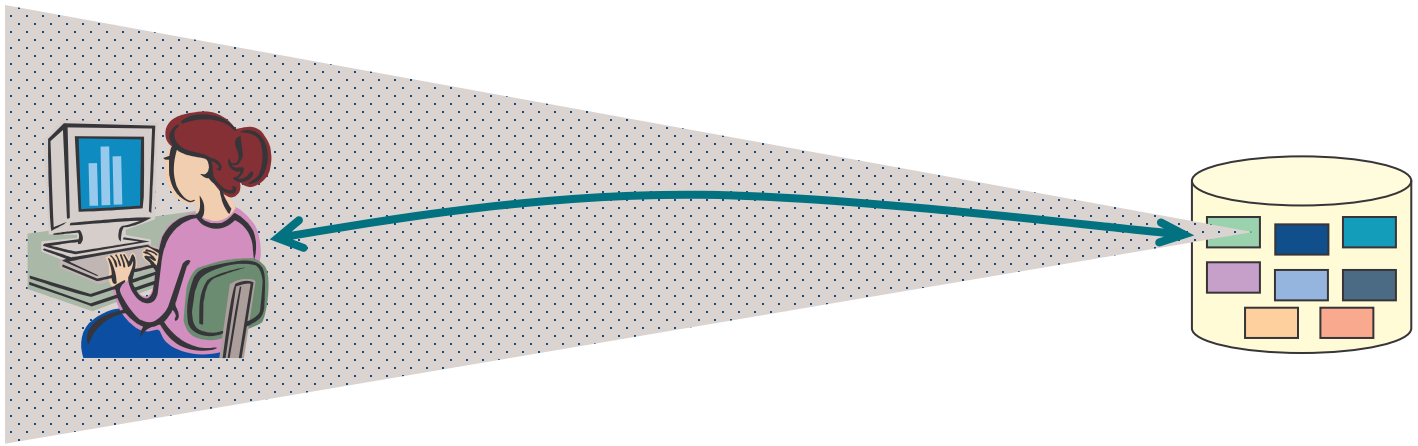


Authorization

Decides whether an entity with a given identity can access a particular resource

For example:

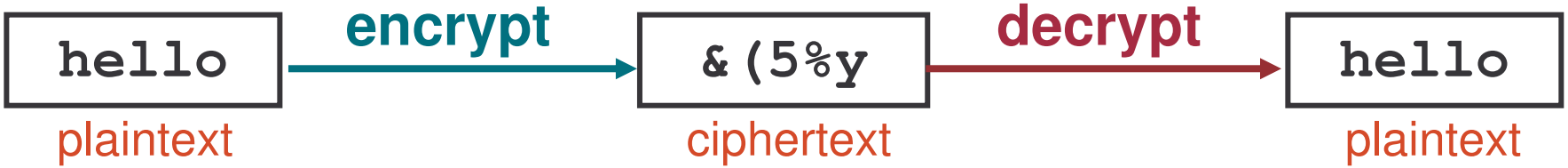
a) a particular citizen can only view the information related to him



Cryptography

Cryptography technologies provide a basis for protecting messages exchanged between partners.

A process based on algorithms transforming a clear text message – plaintext, in encrypted data - ciphertext.



Cryptography Algorithms

Most of the algorithms use a key to encrypt and decrypt.

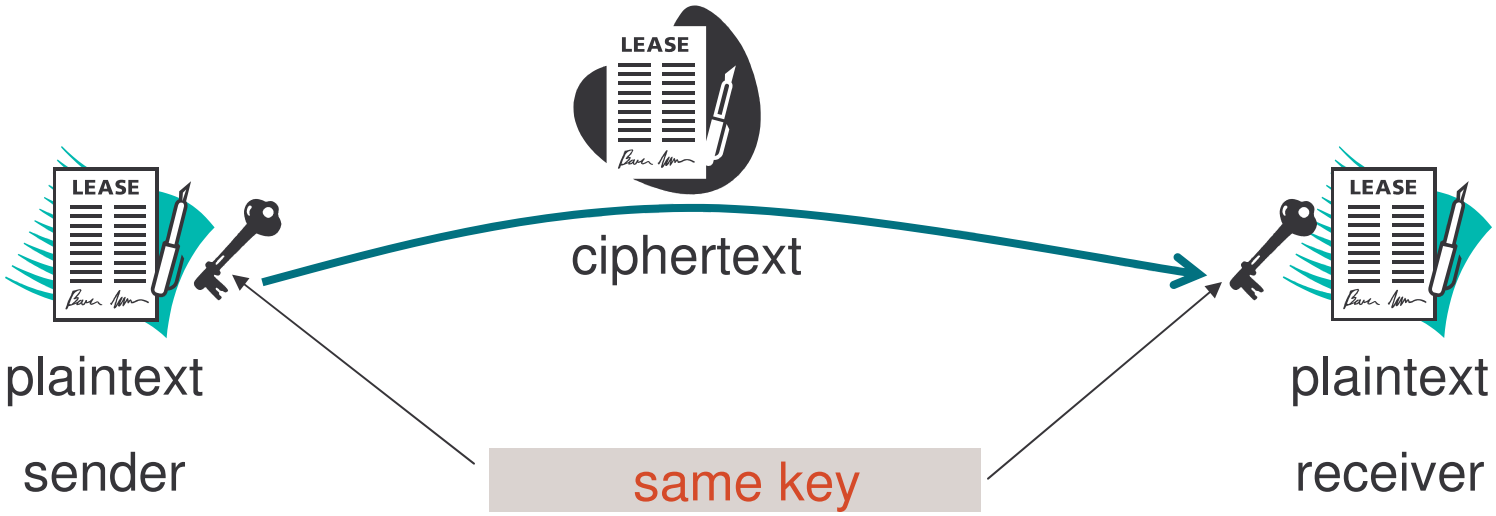
According to the keys used, encrypting algorithms can be classified in:

1) symmetric
2) asymmetric

Different technologies	Symmetric key	Asymmetric key
Encryption	3DES – AES – RC4	RSA15
Digital signature	HMAC-SHA1 HMAC-MD5	RSA-SHA1

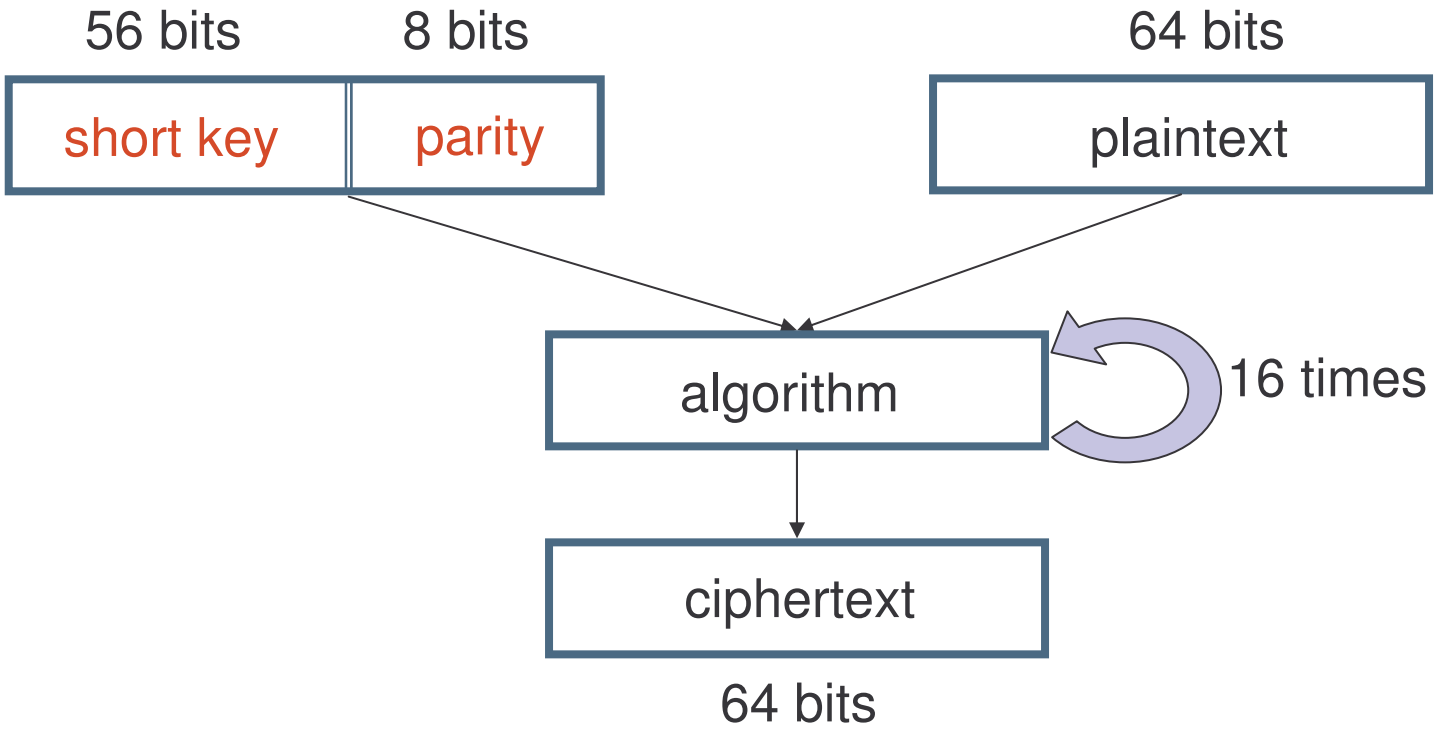
Symmetric Encryption

Requires the use of the same key for encryption and decryption.



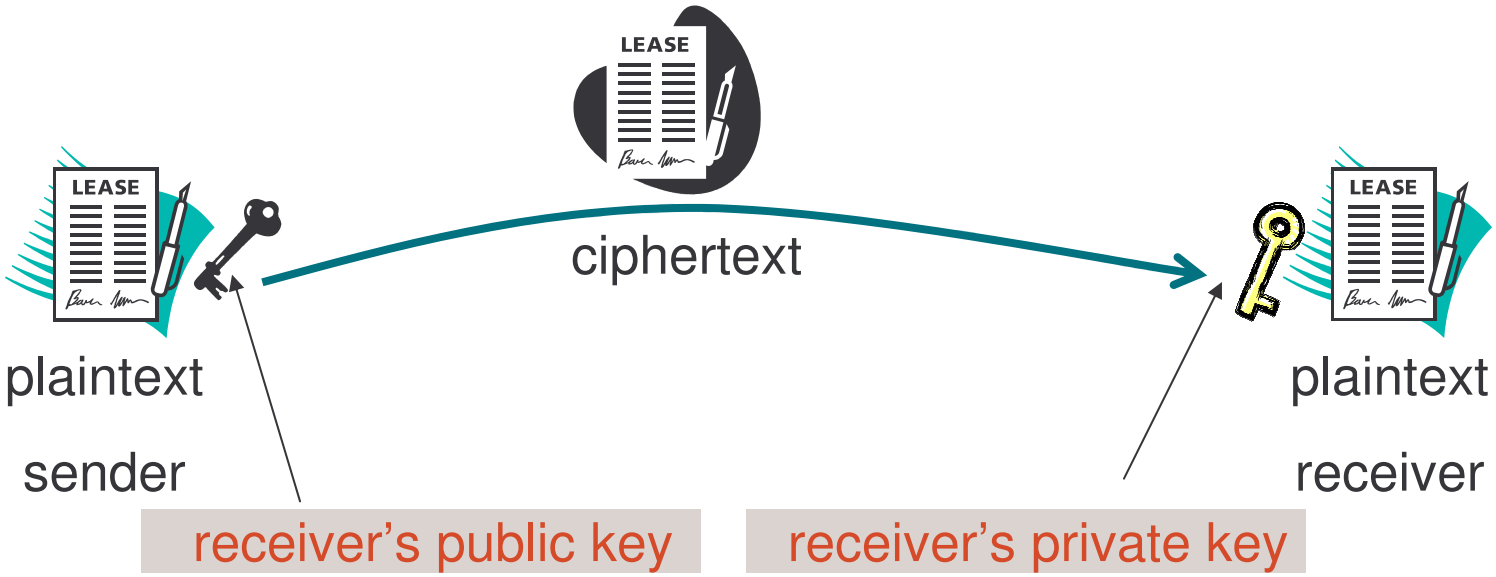
DES Algorithm

DES – Data Encryption Standard: developed by IBM and approved by the National Bureau of Standards (NBS).

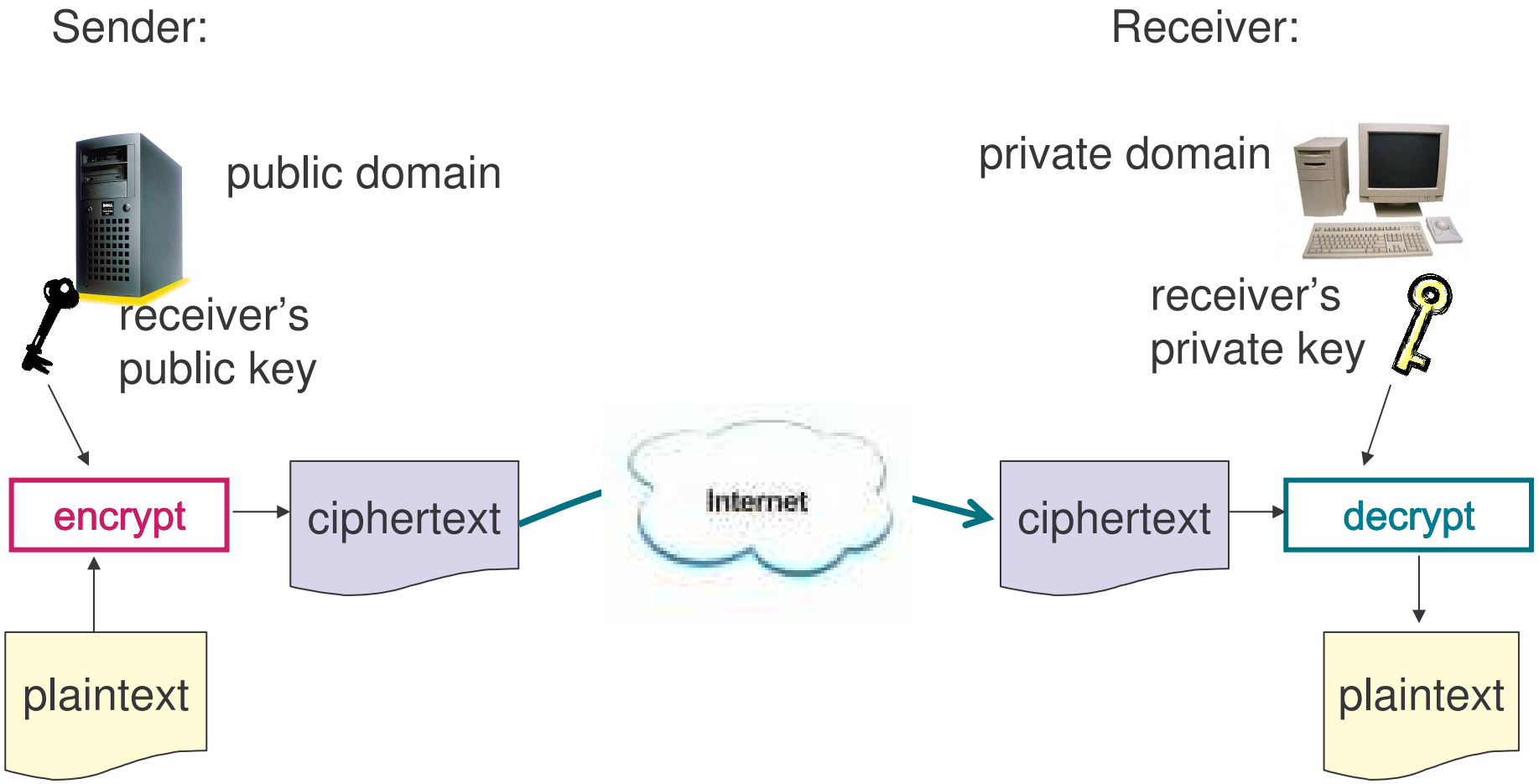


Asymmetric Encryption

Uses two keys: public and private key



Asymmetric Encryption Process

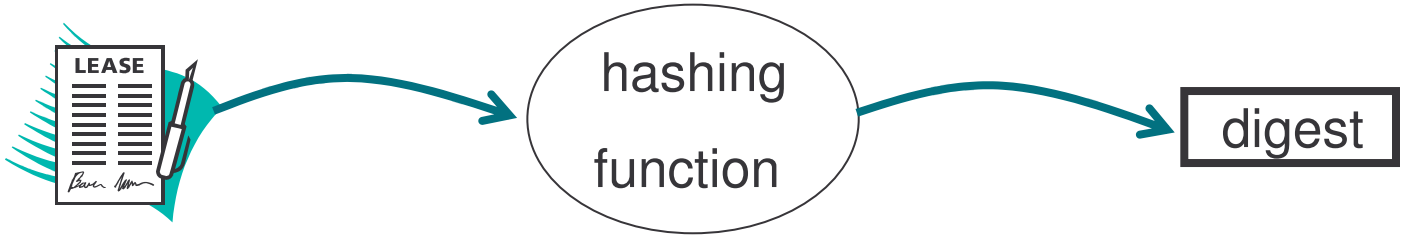


Symmetric Digital Signature

Symmetric digital signature technology is called Message Authentication Code (MAC) technology.

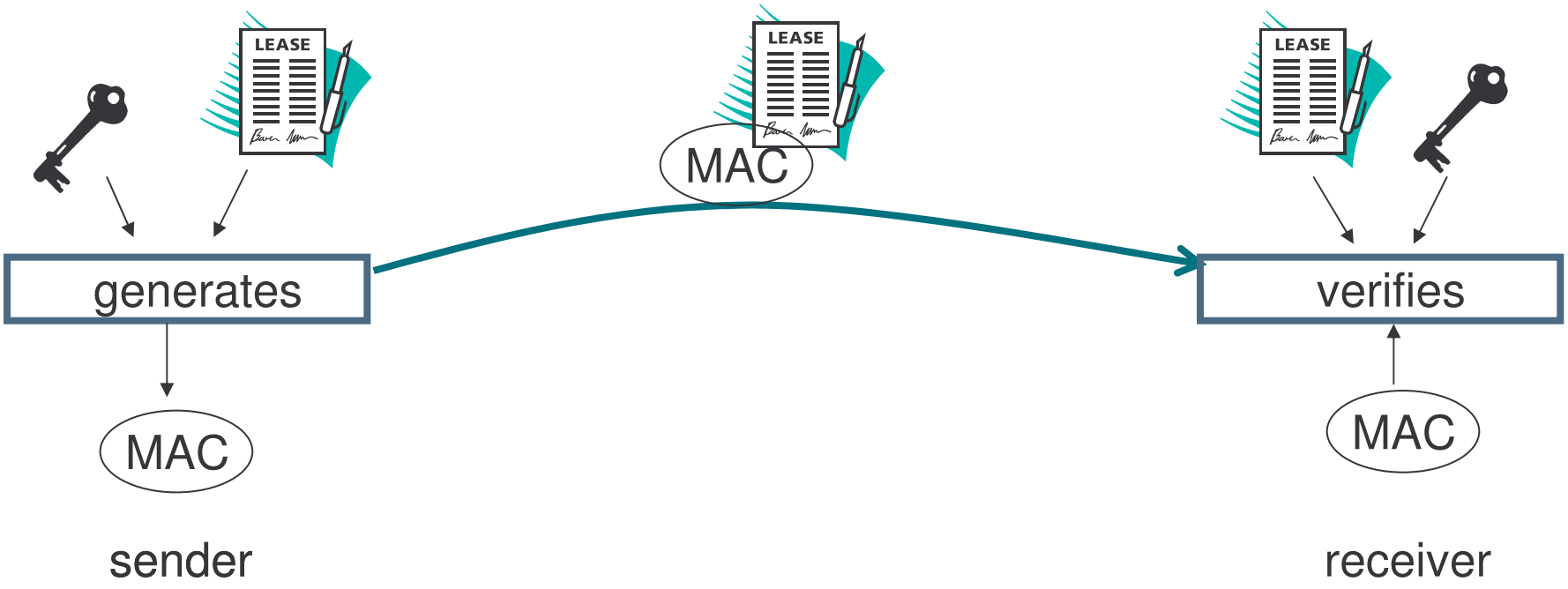
MAC relies on mathematical algorithms known as hashing functions to ensure data integrity.

A hashing function takes data as input and produces smaller data called digest.



In MAC, the digest is created with a key in addition to the input data.

Symmetric Digital Signature



Keyed-Hashing for Message Authentication Code (HMAC) is an example of MAC.

HMAC is combined with hashing functions such as MD5 and SHA-1. Therefore, the algorithm names: HMAC-SHA1 and HMAC-MD5.

Digital Signature

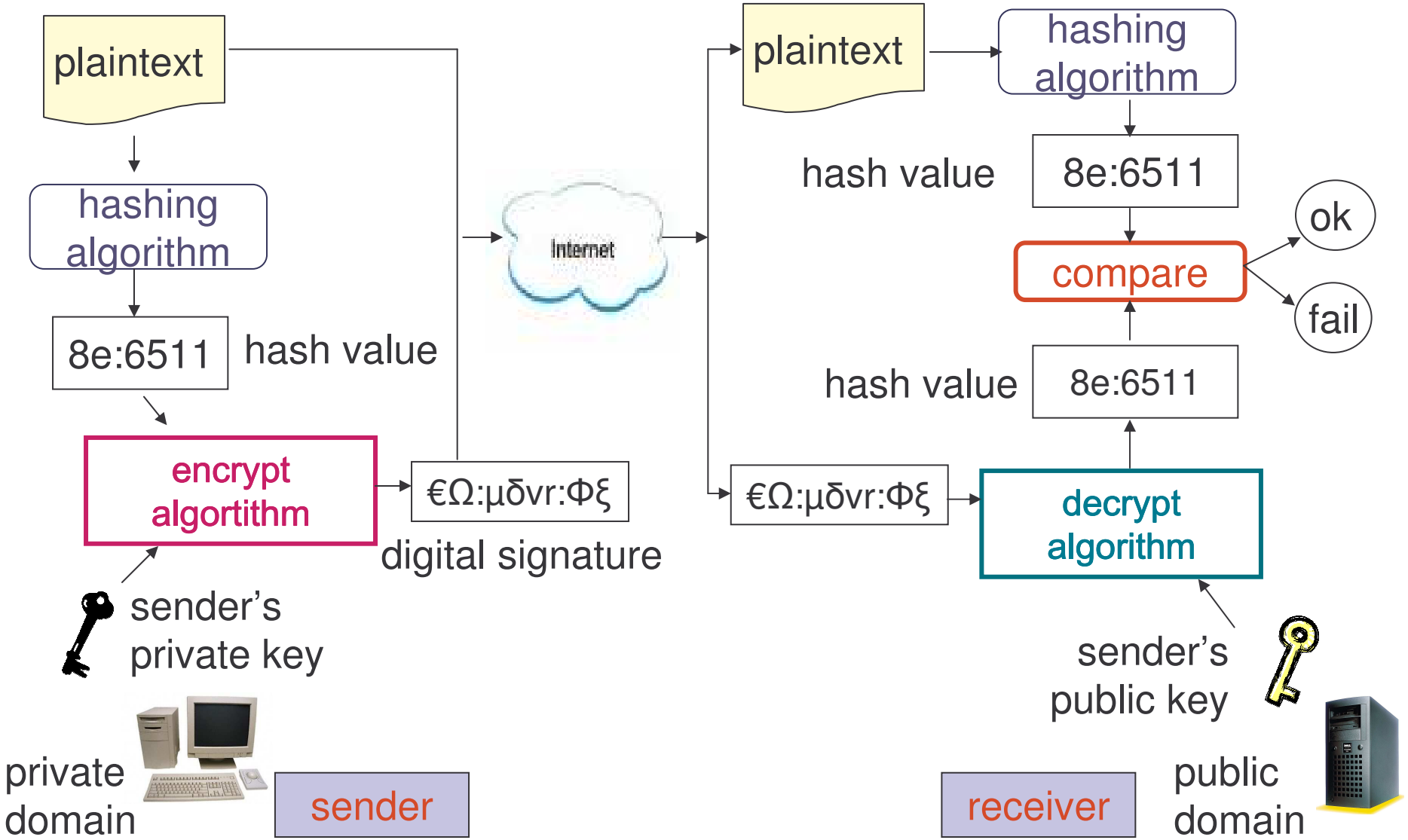
The asymmetric digital signature technology is called digital signature.

The sender signs the plaintext with his private key.

Signing means creating a signature value that is sent with the original plaintext.

Like MAC algorithms, digital signature algorithms are also combined with hashing functions such as SHA-1 (SHA: Secure Hash Algorithm).

Digital Signature Process



Use of Digital Signature

The digital signature technology ensures:

a) non-repudiation: the receiver can assure the message is signed by the sender because he is using the sender's public key to decrypt the message

b) integrity: the receiver can assure that the message was not changed during the transmission

Public Key Infrastructure

How can the receiver know that “Person A” is the holder of the public key?

Public Key Infrastructure (PKI) provides a solution.

In PKI an “authority” issues digital certificates.

Digital certificates are used to bind a party to a public key.

Different Solutions

Different solutions may be applied to assure security:

1) password authentication
2) HTTP Basic Authentication
3) digital signature authentication
4) secure protocols – SSL

Password Authentication

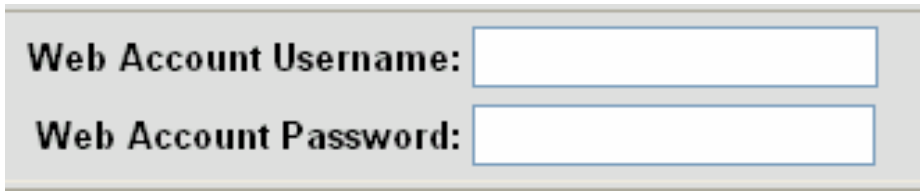
Password authentication is the most commonly used authentication method on the Internet:

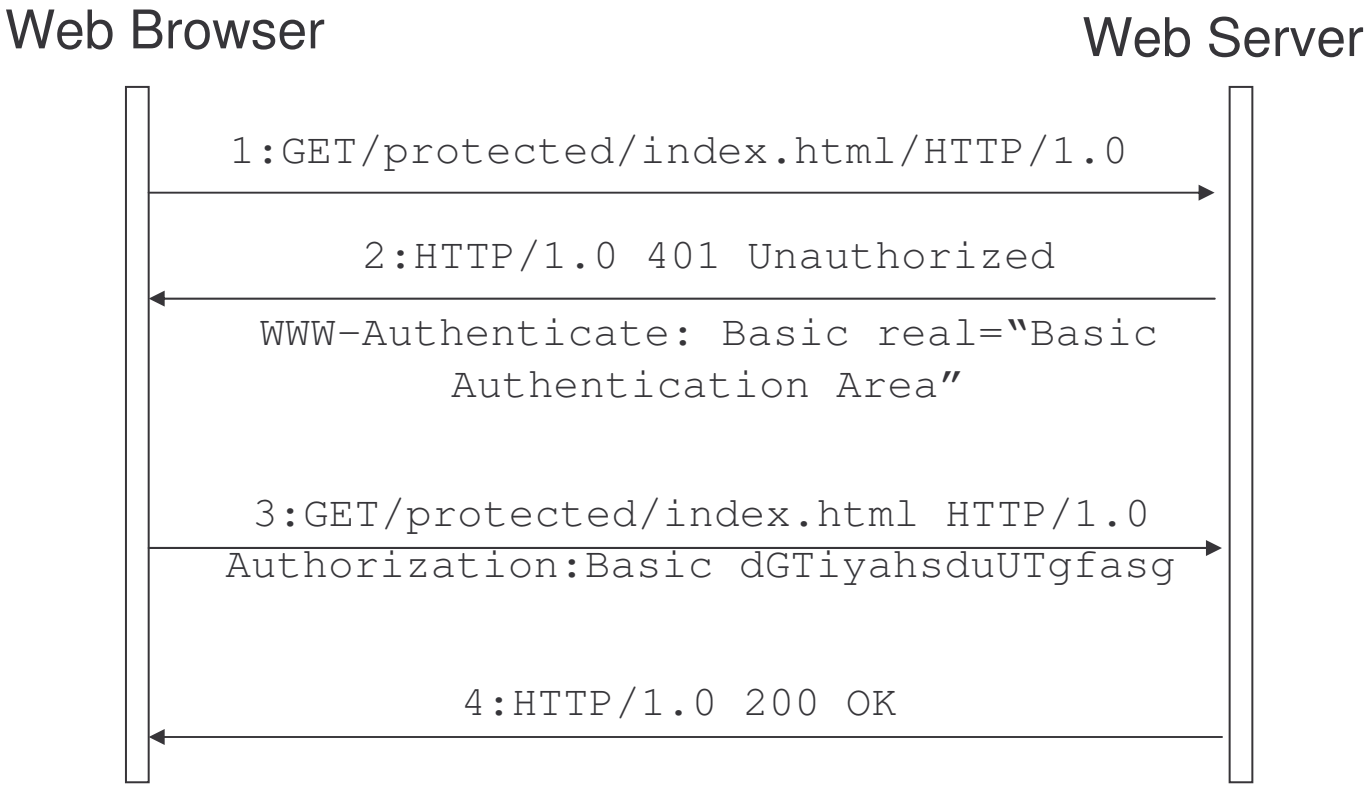
1) a client shows its ID or username and password
2) the server checks the ID and password in a user registry

Password authentication to access Web servers over HTTP is called HTTP Basic Authentication (BASIC-AUTH) and its defined in RFC 2617

HTTP Basic Authentication

Is an interaction protocol between a Web browser and a Web server.





Digital Signature Authentication

Digital signatures can also be used for authentication.

Is more convenient than password authentication – since a certificate authority manages certificates.

Use of certificates:

1) client certificates are not widely used - not easy for users to install certificates.

2) server certificates are commonly used - when web browsers need to authenticate servers.

Security Protocols - SSL

Security protocols allows to share symmetric keys between partners in a secure manner.

SSL – Secure Socket Layer defined by Netscape for Web browsers is the most widely used protocol on the Internet:

1) enables to share symmetric keys

2) performs authentication

SSL Protocol

Client:	Server:
1) accesses the server	
	2) returns its certificate
3) prepares a random number (a seed for generating a symmetric key)	
4) encrypts the seed number with a public key contained in the server certificate	
5) sends the encrypted data to the server	
	6) decrypts the received data to extract the seed number
7) generates a symmetric key based on the seed number	7) generates a symmetric key based on the seed number

SSL Authentication

SSL authentication is based on encryption.

After the negotiation has been completed, the client and the server can authenticate themselves:

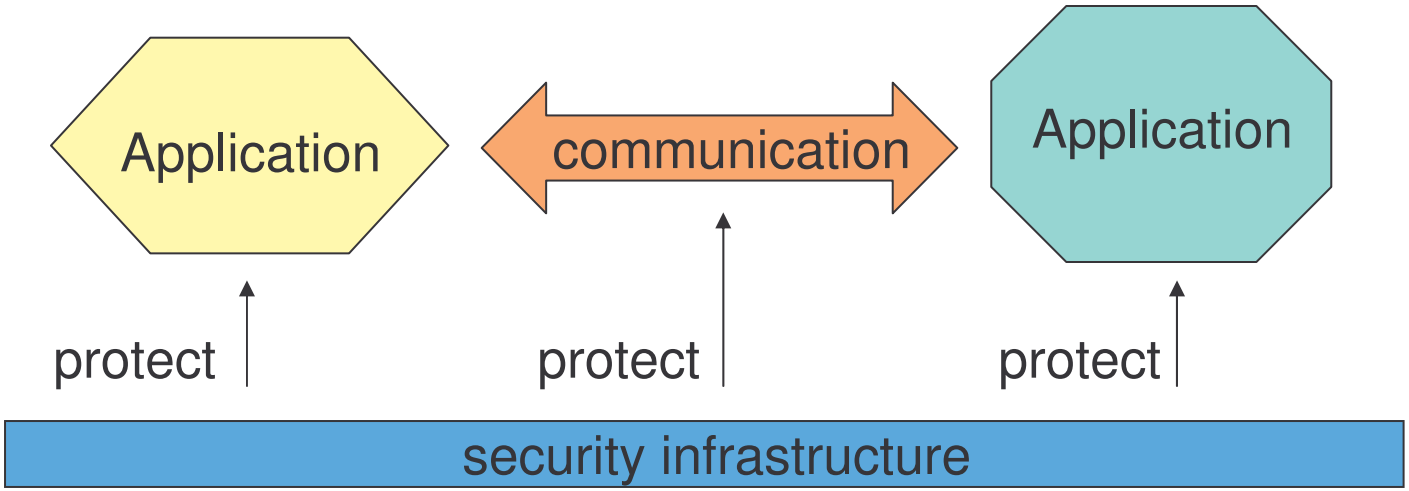
1) the client authenticating the server:
    a) client sends application data encrypted with the symmetric key
    b) server sends response encrypted with the symmetric key
    c) client can authenticate the server

2) the server authenticating the client – two ways:
    1) HTTP Basic Authentication
    2) the client is required to decrypt a random number encrypted with its public key

Security Infrastructure

To solve the difficulties of combining security technologies properly, security infrastructures have been developed and are use in real systems.

A security infrastructure is a basis on which applications can interact with each other securely.



Different Security Infrastructures

Each security infrastructure has different design requirements and vary in terms of their design and architecture.

Three security infrastructures will be presented:

a) user registries

b) Public Key Infrastructure

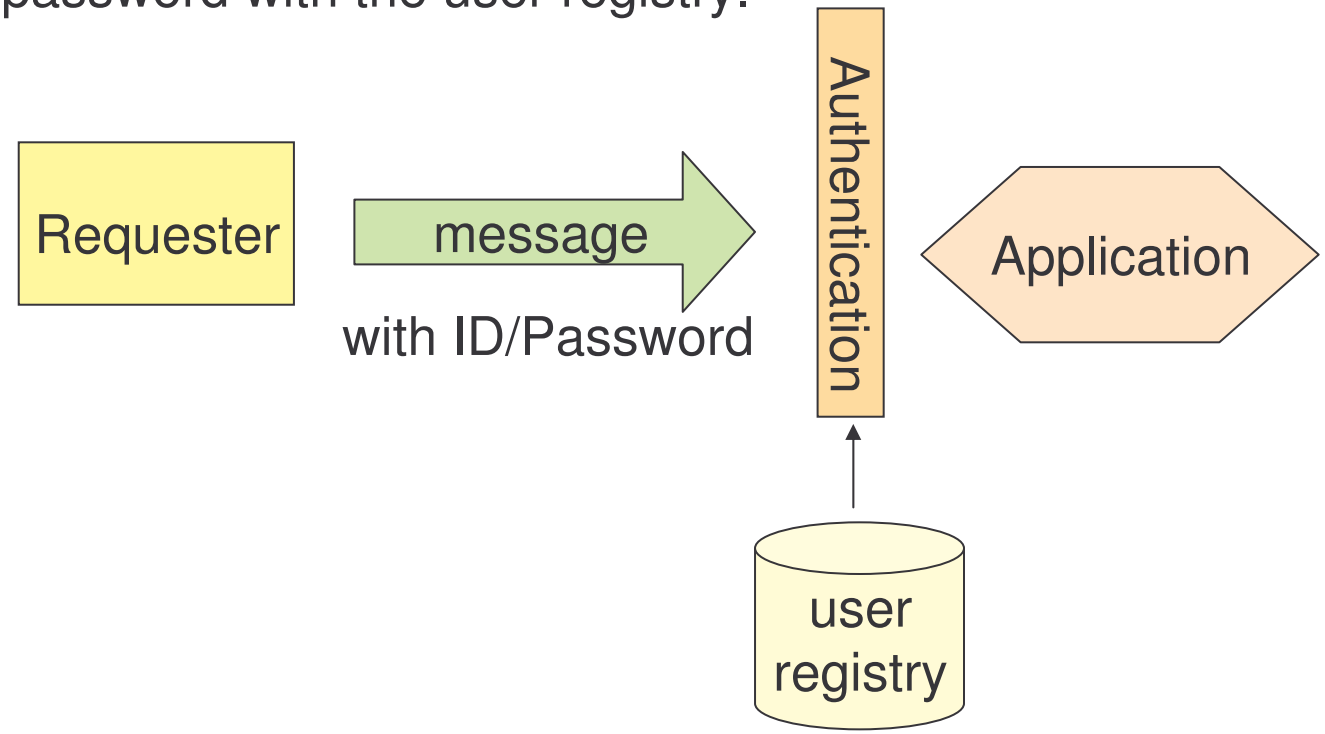
c) Kerberos

User Registries

The most basic security infrastructure used for authentication.

User registries manages users identification and password.

An authentication module is placed “in front” of applications checking each ID/password with the user registry.



User Registries Usage

The advantage is its simplicity.

The disadvantage is that they may be cracked. Once a password is stolen, the attacker can easily access a system.

Operating systems, Data Base Management Systems (DBMS) and HTTP servers incorporate user registries.

The cost for development and management of user registries is cheaper than other mechanisms.

Public Key Infrastructure

Public Key Infrastructure provides a basis to certify holders of public keys.

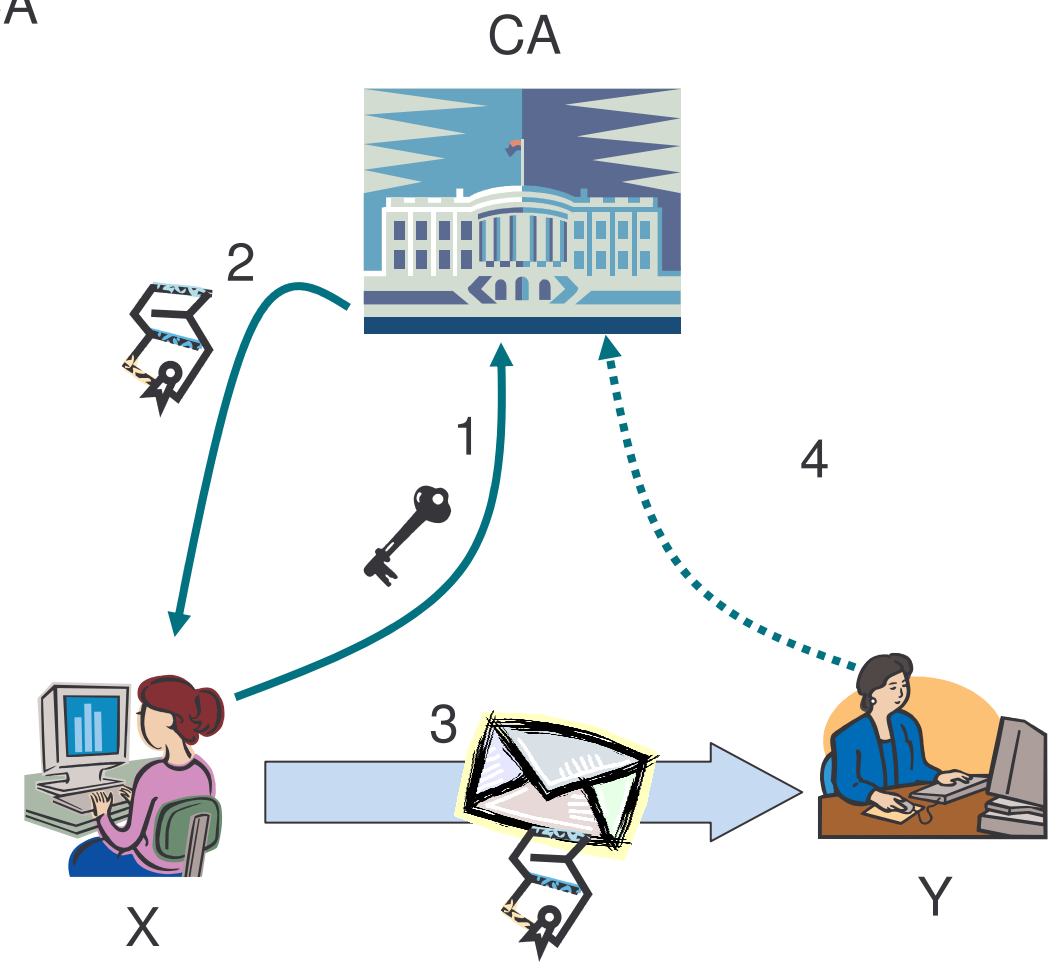
The key constructs of PKI are:

1) certificate: a proof of identity

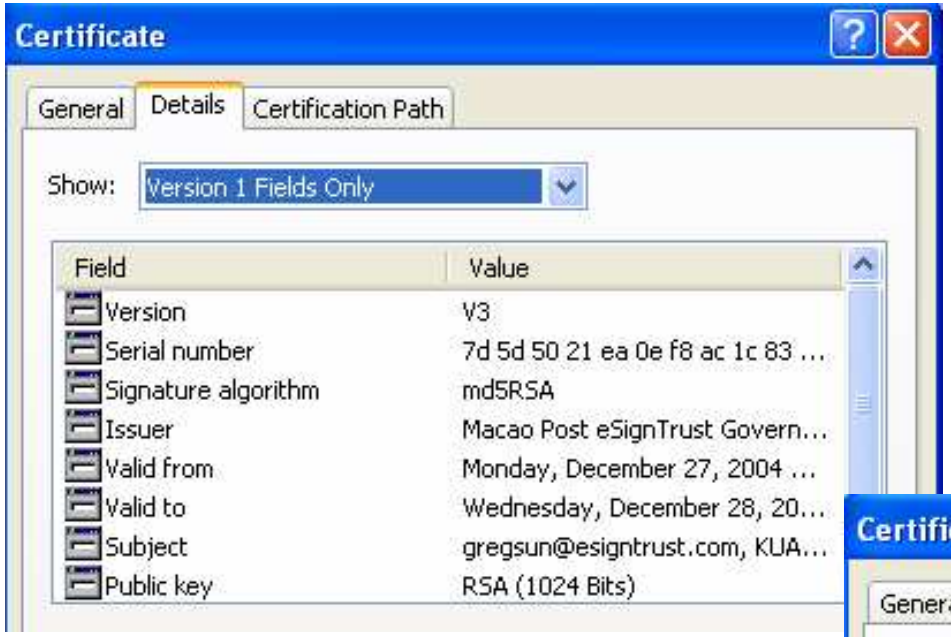
2) certificate authority: an entity that issues certificates.

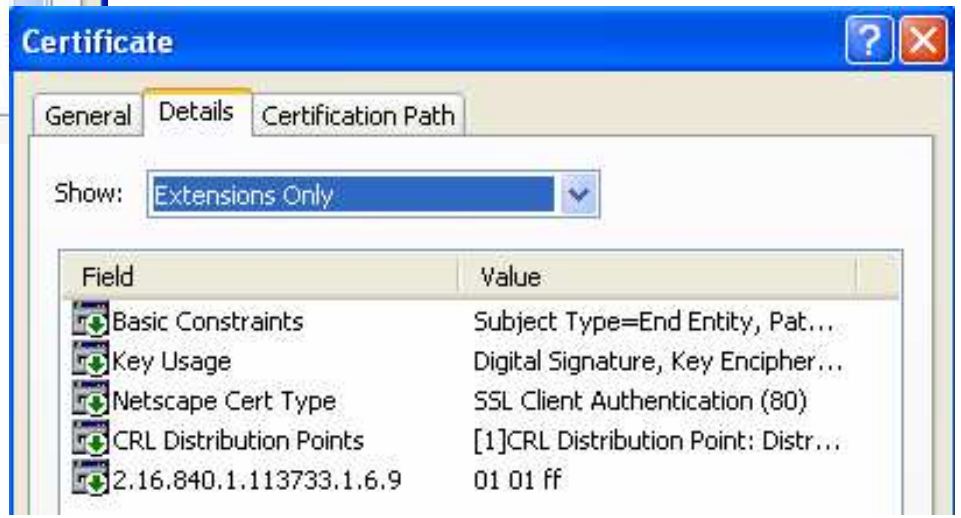
Use of Certificates

1) X registers its public key in CA
2) CA issues a certificate to X
3) X signs a message with the private key and sends it to B, attaching the certificate
4) Y verifies the signature using a public key included in the certificate
5) Y verifies if the certificate is signed by CA

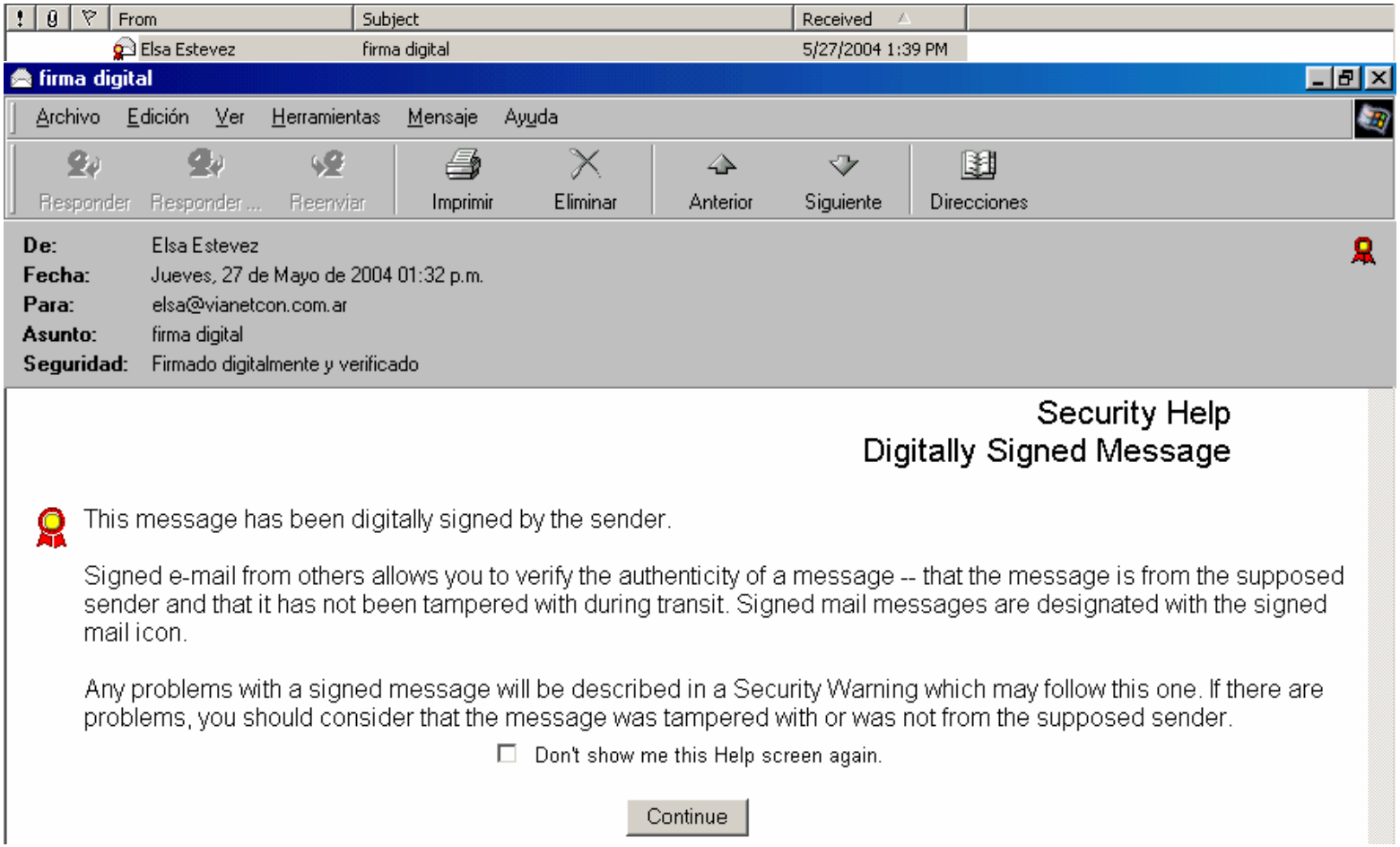


Certificate Example

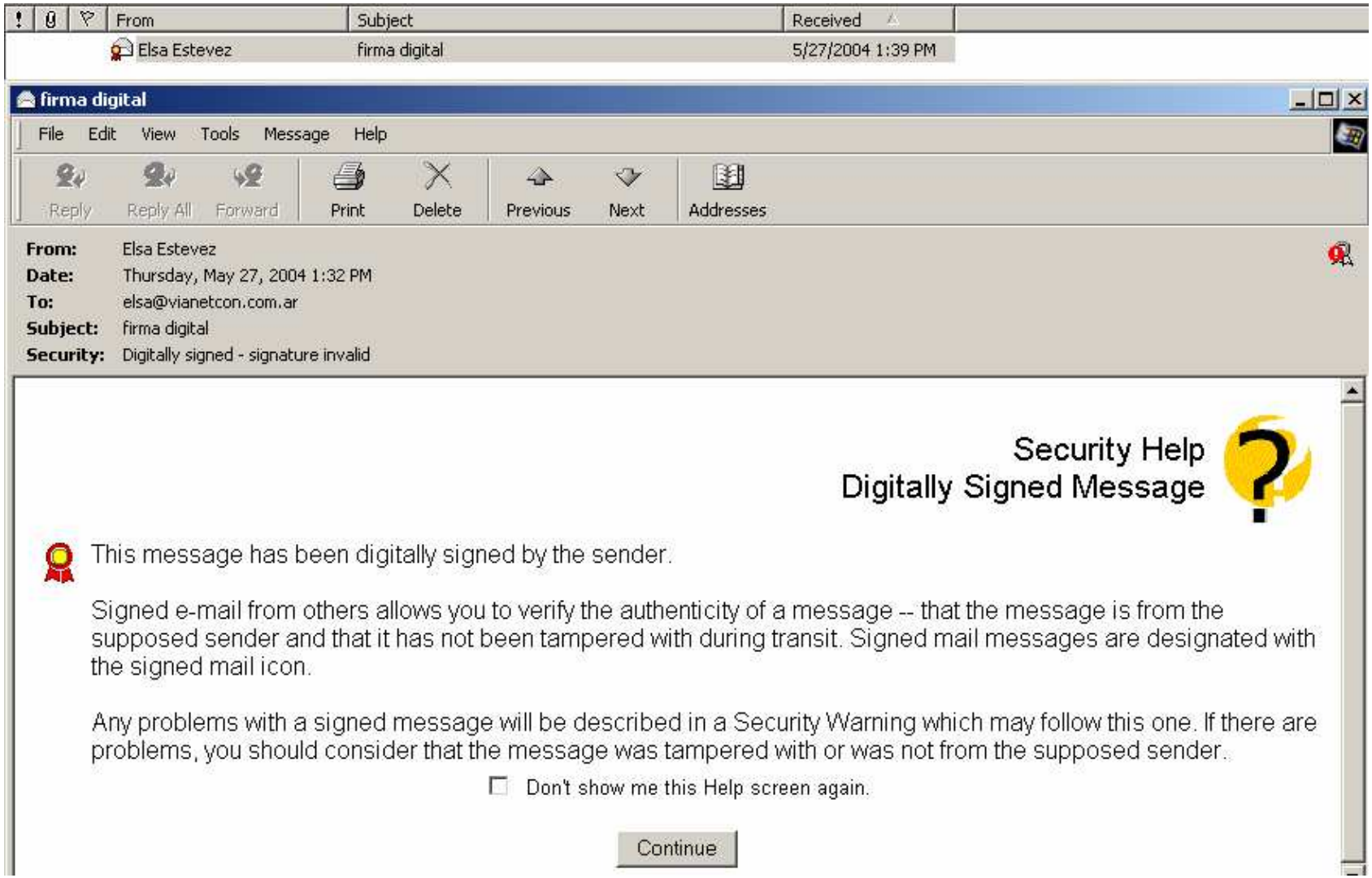




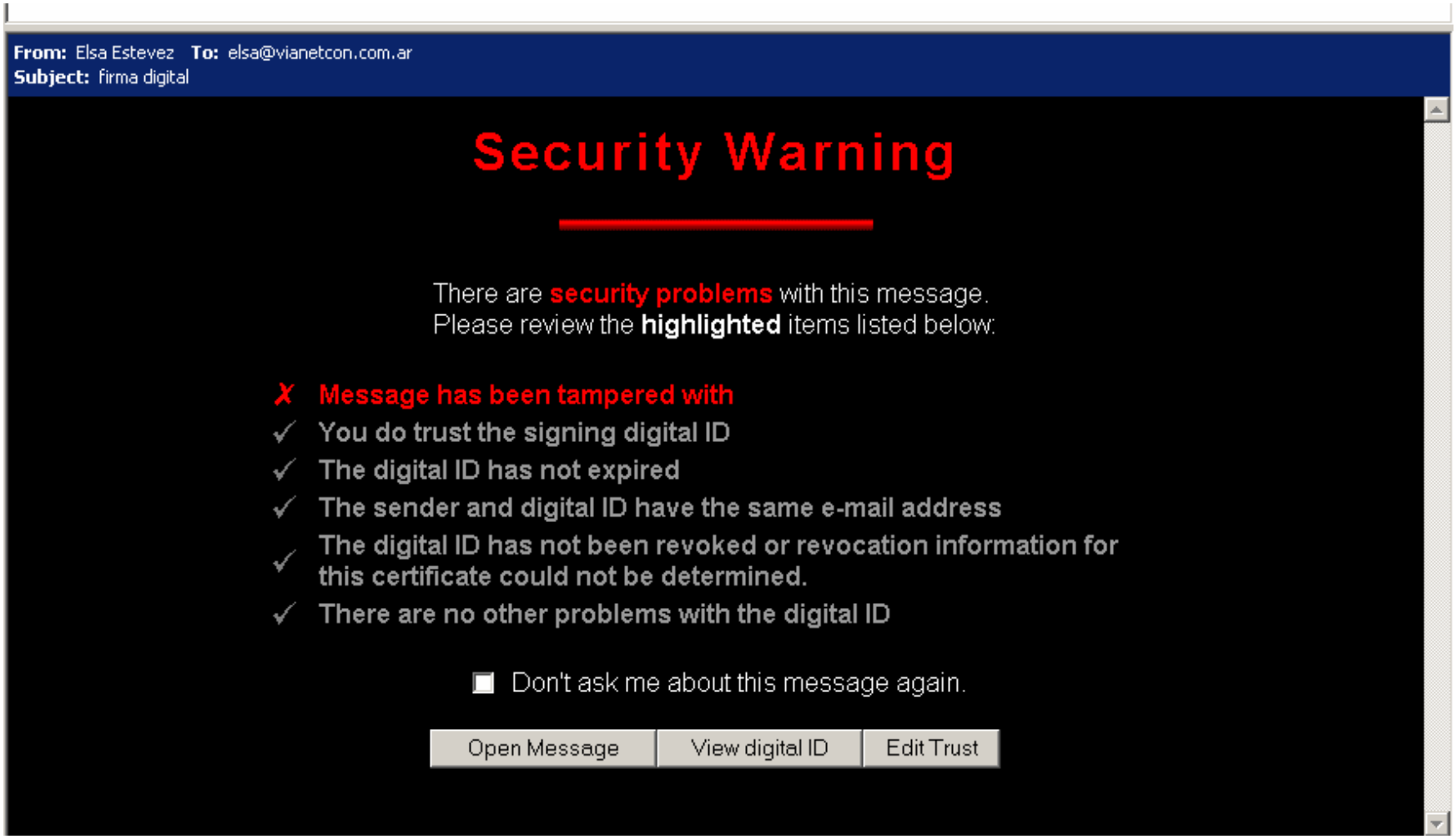
Valid Signed Message Example



Invalid Signed Message Example 1



Invalid Signed Message Example 2



Kerberos

Kerberos was initially developed for workstation users who wanted to access a network.

Key requirements:

1) Single Sign ON (SSO): a user provides an ID and a password only once to access various applications within a certain interval

2) no use of public key cryptography

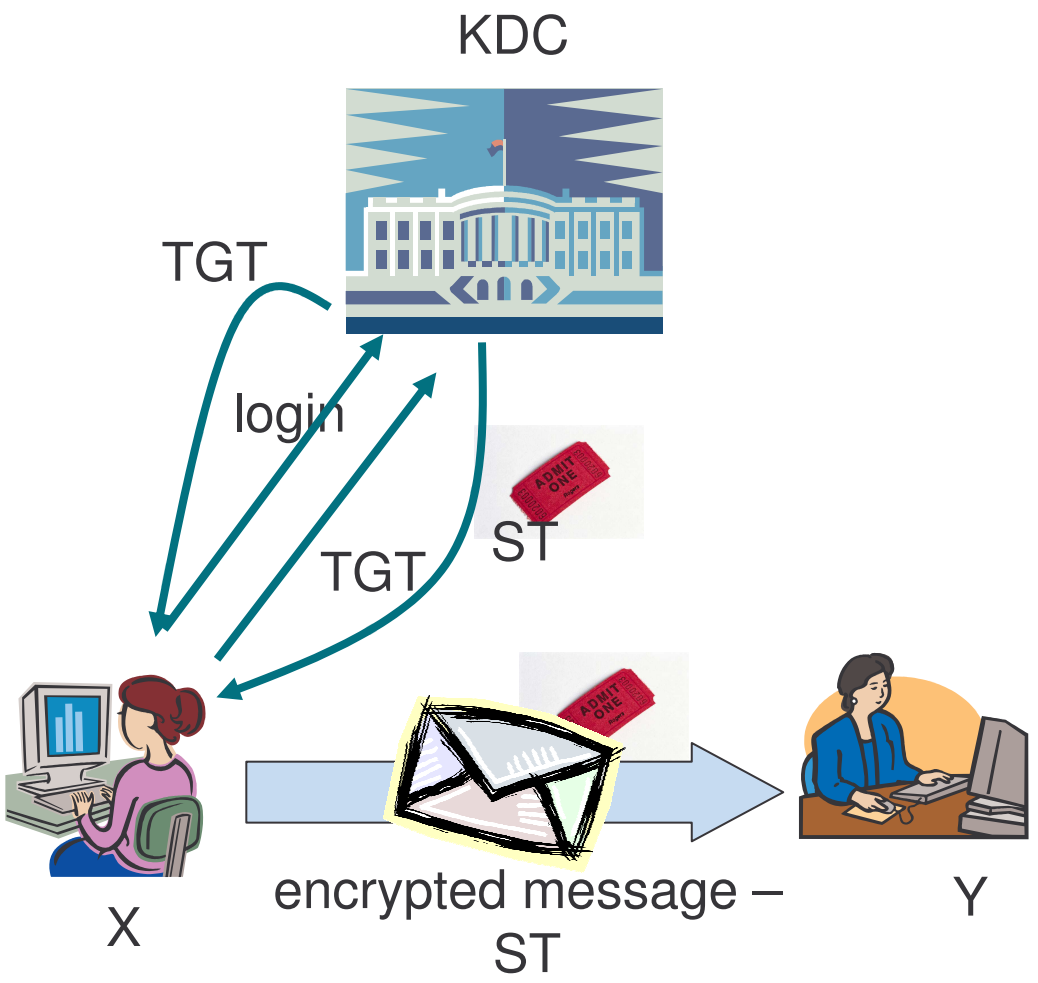
Use of Kerberos

1) X logins in KDC (Key Distribution Center) using password authentication and requests a Ticket-Granting Ticket

2) X receives TGT from KDC

3) X requests and receives a service ticket (ST) for Y, showing the TGT to the KDC

4) X can now access Y by including the ST in a request message



Ticket-Granting Ticket

The TGT contains:

1) user's ID
2) session key
3) TGT expiration time

As long as the TGT is valid, X can get various STs without giving its ID and password. Therefore, single sign on is achieved.

When issuing the ST, KDC encrypts X's information with Y's information. Thus, only Y can decrypt X's ID in the ST.

ST contains a session key between X and Y, so they can securely exchange messages with encryption and digital signatures.

Security Domains

Each security infrastructure has a scope.

The scope determines the participants and resources managed by the security infrastructure.

User registries and Kerberos have explicit databases defining their scope.

CA implicitly prescribes a set of participants in PKI.

The scope of the security infrastructure is called security domain.

Multiple Security Domains

Multiple security domains exist in the real world.

Is out of question considering a single security infrastructure to integrate them.

Web services security addresses how to integrate security domains based on different security infrastructure.

Security Outline

1) Security Basics
2) Web Service Security
3) Digital Signatures

Web Services Security

Each business has its own security infrastructure.

Web services need to interoperate over different security domains.

The security model for web services defines:

1) concepts

2) architecture

WS Security Model Concepts

Some concepts used in the WS security model include:

1) security token
2) subject
3) claim
4) web service endpoint policy
5) security token service

Security Token

A security token is a piece of information related to security.

For instance a security token can be:

1) a X.509 certificate,
2) Kerberos ticket,
3) username,
4) mobile device security token from a SIM card,
5) etc.

Subject

A subject is an entity about which the claims expressed in the security token apply.

For instance:

1) a person

2) an application

3) business

Claim

A claim is a statement about a subject.

A claim can be done by:

1) the subject itself

2) a third party that associates a subject with a claim

For instance, claims:

1) may be about keys that may be used to sign or encrypt messages

2) may be statements of the security token itself

3) may be used to assert the user's identity or an authorized role

Web Service Endpoint Policy

A Web service endpoint policy are the claims and related information that web services require in order to process messages.

Endpoint policies may be expressed in XML.

Endpoint policies can be used to indicate requirements related to:

1) authentication – proof of user

2) authorization – proof of execution capabilities

3) other requirements

Security Token Service (STS)

A security token service is a third party issuing security tokens.

For instance:

1) the certificate authority in PKI

2) Key Distribution Center in Kerberos

A security token service is a web service.

WS Security Architecture

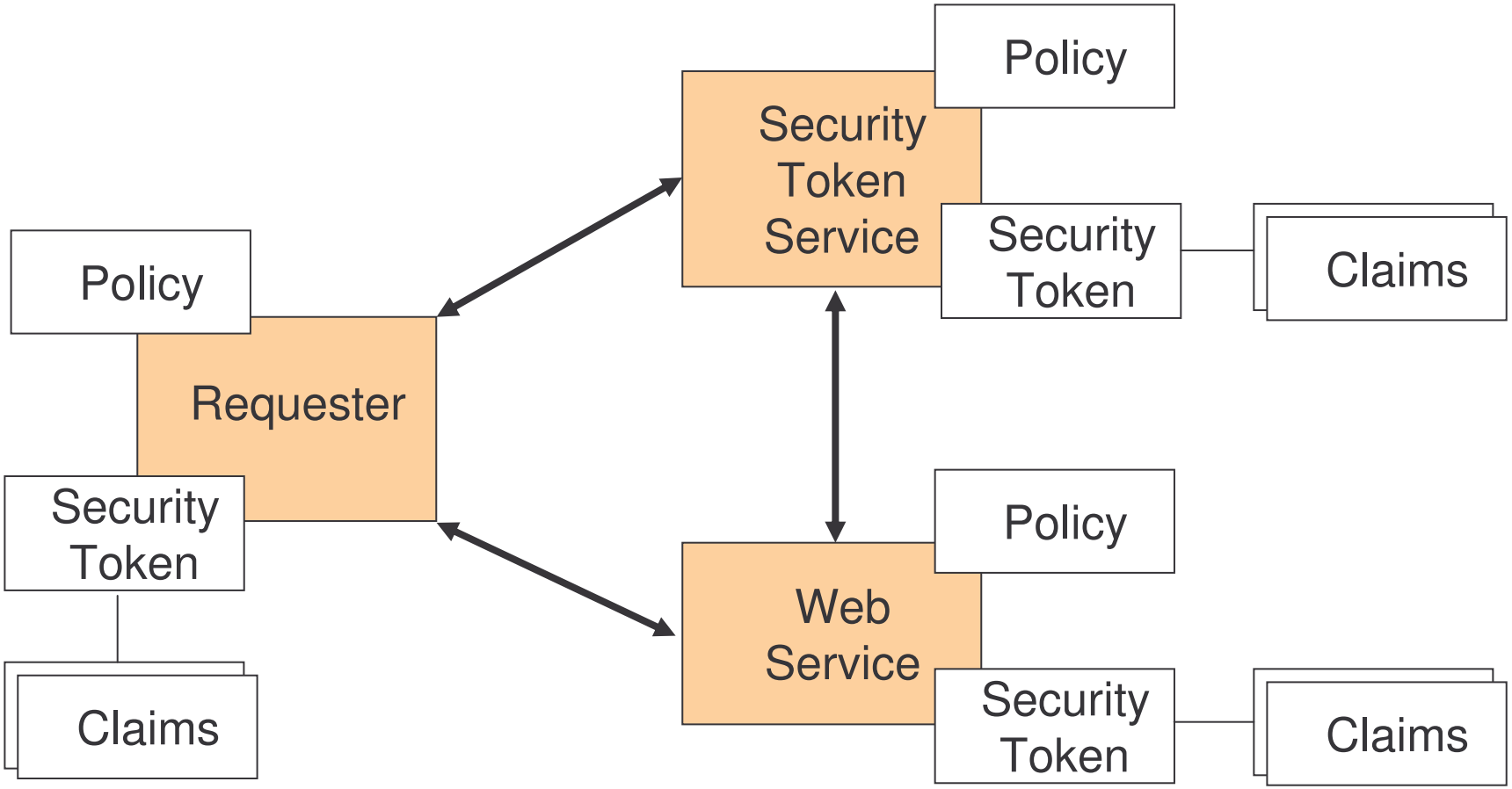
The web services security architecture defines an abstract model for managing security based on three parties:

a) requestor

b) web service

c) security token service

Security Model for WS



Each party has its own claims, security token and policy.

Scenario for the Security Model

1) One end - the requestor:

    a) wants to invoke a web service

    b) has claims, such as identity and privileges

2) other end - the web service:

    a) has a policy – requires encryption of messages and authentication of requestors

Sending Security Claims

How security claims can be represented in messages?

WS security model suggest that all security claims should be included in the security token that is attached to the request message.

For instance:

1) identification via password

2) X.509 certificate

are security claims, therefore they are represented as security tokens attached to the message.

WS Security Specifications

On April 2004 OASIS officially announced Web Services Security v1.0, composed of:

1) Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

2) Web Services Security Username Token Profile 1.0

3) Web Services Security X.509 Certificate Token Profile

4) two XML schema documents:
   a) secext.xsd
   b) utility.xsd

Reference: http://www.**oasis**-open.org/committees/wss/

WSS - SOAP Message Security 1

Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) specification proposes a standard set of SOAP (1.1 and 1.2) extensions.

It can be used when building secure web services to implement message content integrity and confidentiality.

Provides support for:

1) multiple security token formats
2) multiple trust domains
3) multiple signature formats
4) multiple encryption technologies

WSS - SOAP Message Security 2

The specification provides three main mechanisms:

1) ability to send security tokens as part of a message
2) message integrity
3) message confidentiality

They do not provide a complete security solution for WS.

This specification is a building block that can be used in conjunction with other WS extensions.

SOAP Security Namespaces

The XML namespaces URI that must be used by implementations are:

wsse: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

wsu: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

The XML namespaces URI for digital signature and encryption are:

ds: http://www.w3.org/2000/09/xmldsig#

xenc: http://www.w3.org/2001/04/xmlenc#

SOAP Message Security Model

The specification uses security tokens combined with digital signatures to protect and authenticate messages.

Security tokens:

1) state claims
2) can be used to declare the binding between authentication keys and security identities

Signatures are used to verify the message origin and integrity:

1) bind the identity of the sender with the message
2) confirm the claims in a security token

Message Protection

The specification provides a means to protect a message by encrypting and/or digitally signing a body, a header, or any combination of them.

Message integrity is provided by XML Signature [XMLSIG] in combination with security tokens to ensure that modifications to messages are detected.

Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens to keep portions of a SOAP message confidential.

The specification defines syntax and semantics of signatures within `<wsse:Security>` element.

Rules for Invalid Messages

A message recipient should reject messages:

a) containing invalid signatures

b) messages missing necessary claims

c) messages whose claims have unacceptable values

These are unauthorized or malformed messages.

SOAP Security Example 1

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
              xmlns:ds="...">
  <S11:Header>
   <wsse:Security  xmlns:wsse="...">
        <xxx:CustomToken wsu:Id="MyID"
             xmlns:xxx="http://www.example.com/token">FHUIORv...
        </xxx:CustomToken>

        <ds:Signature>

          <ds:SignedInfo>
            <ds:CanonicalizationMethod
               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
            <ds:Reference URI="#MsgBody">
              <ds:DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
```

1) what is being signed

2) type of canonicalization being used

SOAP Security Example 2

```
            <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
            <ds:KeyInfo>
                 <wsse:SecurityTokenReference>
                    <wsse:Reference URI="#MyID"/>
                 </wsse:SecurityTokenReference>
            </ds:KeyInfo>
          </ds:Signature>

    </wsse:Security>
  </S11:Header>

  <S11:Body wsu:Id="MsgBody">
     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">QQQ
     </tru:StockSymbol>
  </S11:Body>

</S11:Envelope>
```

Id Attribute

There are many situations where elements within SOAP messages need to be referenced.

The specification introduces `wsu:ID` attribute to reference elements.

Example:

```
...
<ds:Reference URI="#MsgBody">
...
<S11:Body wsu:Id="MsgBody">
```

Security Header

The <wsse:Security> header block provides a mechanism for attaching security-related information targeted at a specific recipient.

```
<S11:Envelope>
   <S11:Header>
        . . .
        <wsse:Security soap:role="…" soap:mustaunderstand=".." >
            . . .
        </wsse:Security>
             . . .
   </S11:Header>
    . . .
</S11:Envelope>
```

This header is extensible by design -it supports many types of security information.

Security Headers Rules 1

A message may have multiple `<wsse:Security>` headers if they are targeted for separate recipients.

Only one `<wsse:Security>` header may omit the `role` attribute (`actor` in SOAP 1.1).

Two `<wsse:Security>` headers must not have the same value for the `role` attributes (`actor` in SOAP 1.1).

Security Headers: Rules 2

Message security information targeted for different recipients must appear in different `<wsse:Security>` header blocks.

The `<wsse:Security>` header block without a specified `role` (or `actor`) may be processed by anyone, but must not be removed prior to the final destination or endpoint.

Security Tokens

The extensibility of the `<wsse:Security>` header allows to insert security tokens based on XML into the header.

The security token may be:

a) user name token

b) binary security token

c) XML token

User Name Token

The `<wsse:UsernameToken>` element is introduced as a way of providing a username.

This element is optionally included in the `<wsse:Security>` header.

It contains a mandatory `UserName` and an optional `Password` sub-elements.

It is used for password authentication, such as HTTP Basic Authorization.

Disadvantage - the plaintext representation is extremely insecure.

User Name Token Example

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." >
  <S11:Header>
     <wsse:Security>
          <wsse:UsernameToken wsu:ID="…" >
              <wsse:Username>Mary</wsse:Username>
          </wsse:UsernameToken>
     </wsse:Security>
      . . .
  </S11:Header>
  . . .
</S11:Envelope>
```

Binary Security Token

Binary security tokens, such as: X.509 certificates or Kerberos tickets, or other non-XML formats require a special encoding format for inclusion.

The `<wsse:BinarySecurityToken>` element defines two attributes:

1) `<ValueType>`: indicates what is the security token:

X509v3	X.509 v3 digital certificate
Kerberos5TGT	Kerberos ticket-granting ticket
Kerberos5ST	Kerberos service ticket

2) `<EncodingType>`: specifies how the security token is encoded, using a URI

Binary Security Token Example

```
<wsse:BinarySecurityToken wsu:ID="Kerberosv5ST"
                  ValueType="Kerberosv5ST"
                  EncodingType="wsse:Base64Binary" />
```

XML Token

The specification also defines multiple mechanisms for identifying and referencing security tokens using:

1) `wsu:ID` attribute

2) `wsse:SecurityTokenReference` element

Security Token Reference

A security token conveys a set of claims.

Sometimes these claims reside somewhere else and need to be retrieved by the receiving application.

The `<wsse:SecurityTokenReference>` element provides an extensible mechanism for referencing security tokens.

```
<wsse:SecurityTokenReference wsu:ID="...">
     . . .
</wsse:SecurityTokenReference>
```

Use of Security Token Reference

The <wsse:SecurityTokenReference> element can be used as a direct child element of <ds:KeyInfo> to indicate a hint to retrieve the key information from a security token placed somewhere else.

It is recommended to use it when applying XML Signature and XML Encoding to reference the security token used for the signature or encryption.

```
<wsse:SecurityTokenReference>
    <wsse:Reference URI="#MyID"/>
</wsse:SecurityTokenReference>
```

Security Outline

1) Security Basics

2) Web Service Security

3) Digital Signatures

Signatures

Signatures are used to:

1) enable message recipients to determine whether the message was altered in transit

2) verify that the claims in a particular security token apply to the producer of the message

The specification allows multiple signatures and signature formats to be attached to a message.

Each signature may refer to different or overlapping parts of a message.

Signature Algorithms

The specification builds on XML Digital Signature Specification.

XML Digital Signature specification (XML Signature) defines how to sign part of an XML document in a flexible manner, using two canonicalization algorithms:

1) XML Canonicalization (Inclusive Canonicalization)
2) Exclusive XML Canonicalization

Neither one solves all possible problems that can arise.

Signature Algorithms: Problems

Two problems:

1) XML allows different documents to be considered equivalent. For instance: duplicate namespace declaration can be removed or created

2) if the signature covers something like “nms:abc”, its meaning may change if nms is redefined

Related to the second problem:

a) it could be solved by expanding all the values

b) mechanisms like XPATH considers nms1=http://example.com to be different from nms2=http://example.com

Canonicalization Example

Document 1:

```
<?xml version =“1.0” encoding=“us-ascii” ?>
<example
     a=“a”
     b=“b”
></example>
```

Document 2:

```
<?xml version =“1.0” encoding=“us-ascii” ?>
<example a=“a” b=“b” />
```

These two documents appear quite different, although with canonicalization are translated both to:

```
<?xml version =“1.0” encoding=“us-ascii” ?>
<example a=“a” b=“b”></example>
```

Inclusive Canonicalization

The fundamental difference between Inclusive and Exclusive Canonicalization is the namespace declarations.

Inclusive Canonicalization copies all the declarations that are currently in force, even if they are defined outside the scope of the signature.

Problem: if the file is moved into another XML document which has other declarations, the signature will be invalid.

XML-C14N specifies inclusive canonicalization.

Exclusive Canonicalization

Exclusive Canonicalization copies only the namespaces that are “visibly used”, those that are part of the XML syntax.

It does not look into attributes values or element content, so the namespaces declarations required to process these are not copied.

It allows you to create a list of the namespaces that must be declared.

Exclusive canonicalization is useful when you have a signed XML document that you wish to insert into other XML documents.

EXC-C14N specifies exclusive canonicalization.

The specification strongly recommends the use of exclusive canonicalization.

Signing Messages

The `<wsse:Security>` header block may be used to carry a signature compliant with XML Signature specification within a SOAP envelope.

Multiple signature entries may be added within one `<wsse:Security>` header block.

To add a signature, a `<ds:Signature>` element must be inserted at the top of the existing content of the `<wsse:Security>` header block.

XML Signature

In XML Signature, an element `Signature` is defined with its descendants under the namespace http://www.w3.org/2000/09/xmldsig#

The WS-Security defines how to embed the `Signature` element in SOAP messages as a header entry.

Signature Example

```
<S11:Header>
 <wsse:Security  xmlns:wsse="...">
  <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
      <ds:Reference URI="#MsgBody">
         <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
   <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
  </ds:Signature>
 </wsse:Security>
</S11:Header>
```

Signing Messages 1

The part signed is specified by `Reference` and is transformed by the method specified in `CanonicalizationMethod.`

The digest value is calculated with an algorithm specified by the `DigestMethod` element.

The value is inserted in the `DigestValue` element represented in Base64.

```
<ds:SignedInfo>
   <ds:CanonicalizationMethod
         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
   <ds:Reference URI="#MsgBody">
       <ds:DigestMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
   </ds:Reference>
</ds:SignedInfo>
```

Signing Messages 2

The value of the part is not signed directly. The `SignedInfo` element is signed.

The `SignedInfo` element is canonicalized and signed with the algorithm specified in the `SignatureMethod` element.

The calculated value is inserted in `SignatureValue` element with Base64 format.

```
<ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
```

Security Summary 1

Security requirements include:

1) confidentiality: protect messages against eavesdroppers

2) integrity: protect messages against deliberate or accidental modifications

3) authentication: guarantees access to those who provide proof of identity

4) non-repudiation: guarantees that the sender not deny the message

5) authorization: decides whether an entity can access a particular resource

Security Summary 2

Cryptography technologies provide a basis for protecting messaged exchanged between partners.

Cryptography algorithms are classified in: symmetric and asymmetric.

Symmetric algorithms require the use of the same key for encryption and decryption.

Asymmetric algorithms uses a public and a private key.

Digital signatures assures integrity and non-repudiation of messages.

Security Summary 3

Different technologies can be applied:

1) Password authentication

2) HTTP Basic Authentication

3) Digital Signature Authentication

4) Security protocols – SSL

To solve the difficulties of combining these technologies security infrastructure have been developed.

Security Summary 4

Some security infrastructure:

1) User registries – managing user identifications and passwords

2) Public Key Infrastructure – certificate authority providing certificates

3) Kerberos – a key distribution center provides tickets based on a single sign on

Security Summary 5

To manage different security infrastructures, web services define a security model based on three parties:

1) requester
2) web service
3) security token service

Each of them posses:

1) policy
2) security token
3) claims

Security Summary 6

On April 2004 OASIS announced Web Services Security v1.0 composed by a set of specifications.

WS Security SOAP Message Security 1.0 propose a set of SOAP extensions to assure integrity and confidentiality of the message contents.

Security headers may include:

1) security tokens:
   a) user name token
   b) binary security token
   c) XML token
2) digital signatures

Acknowledgements

I would like to thank to:

1) Tomasz Janowski and Adegboyega Ojo - for their valuable comments

2) Gabriel Oteniya - for his help in developing the examples

3) Audience - for your presence and valuable comments

Web Services and e-Government

1) promotes the development of seamless services grouping services provided by different agencies

2) do not require expensive technologies

3) based on open-source standards

4) facilitates the integration of legacy systems

Many Thanks!