XML for Software Engineers

Andrea Zisman and Anthony Finkelstein
{a.zisman, a.finkelstein}@cs.ucl.ac.uk
Software Systems Engineering Group
Department of Computer Science
University College London

1

Tutorial Outline

- Introduction
- XML Applications
- XML Documents & Processor
- Document Type Definition (DTD)
- XML Basics
- XML Related Technologies
  - . . XLink & XPointer
  - . . XSL
  - . . DOM
  - . . Namespace
  - . . XML-Data
  - . . XML-QL
- XML & Software Engineering

2

Contract / Pre-requisite

What are you going to get out of this tutorial ?

> ***Know about XML and its related technologies***

> ***Know how to use XML in Software Engineering***

Pre-requisites

> ***Software Engineering background***

> ***HTML and/or SGML***

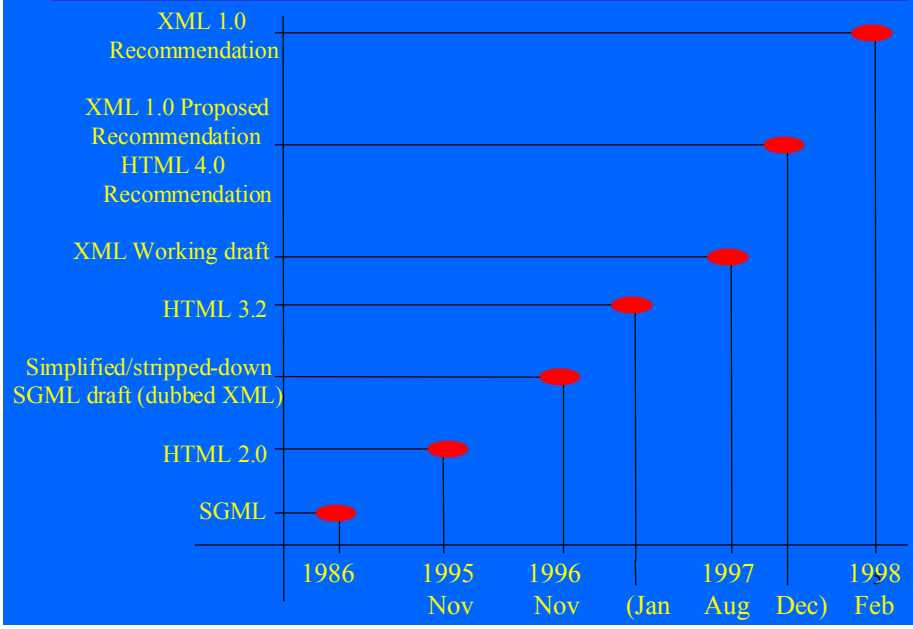
3

Introduction

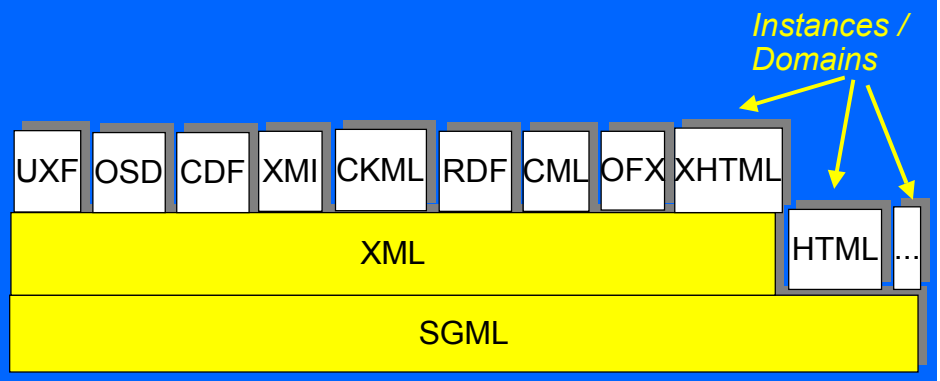
- **XML** - e**X**tensible **M**arkup **L**anguage
- **XML** - W3C (**W**orld **W**ide **W**eb **C**onsortium) Architecture Domain - headed by Dan Connolly
- **XML** - is based on **SGML** (**S**tandard **G**eneralized **M**arkup **L**anguage - ISO8879:1986)
- **XML** - allows developers to create their own markup languages
- **XML** - brings structured information to the Web and provides a data standard that can encode the content, semantics & schemata for a wide variety of cases

4

Development Timeline



Architectural Dependencies



XML vs. (HTML, SGML)

XML vs. HTML

- *Extensibility*
- *Structure*
- *Validation*
- Purity: it separates structure & presentation
- It allows fine-grained search facilities
- No fixed mark up tags
- It allows integration of data from diverse sources

XML vs. SGML

- Easier
- Simpler (simplified version)
- Less rigid
- Provides a small core set of "easy-to-learn-and-use" constructs
- It makes tool development simpler
- Useful on the Internet and not just for large corporate or research applications
- It supports easy-to-use style sheet languages

The 10 Commandments of XML (Goals)

- **Be usable over the Internet**
- **Support a wide variety of applications**
- **Be SGML compatible**
- **Be easy to write**
- **Be easy to process by program**
- **Have no optional features (minimum)**
- **Be human-legible and clear**
- **Be designed quickly**
- **Have a formal and concise design**
- **Unambiguous markup**

XML Applications

- **On line banking**
- **Push publishing Technology**
- **Web automation**
- **Database publishing**
- **Software distribution**
- **Scientific data**
- ***Software Engineering***

XML = *Grammar*
Applications =*Vocabulary*

Existing XML Applications

- **Horizontal-industry applications/vocabularies**

> *Channel Definition Format (CDF)*
> *Open Software Description (OSD)*
> *Web Interface Definition Language (WIDL)*
> *XML Metadata Interchange Format (XMI)*

- **Vertical-industry applications/vocabularies**

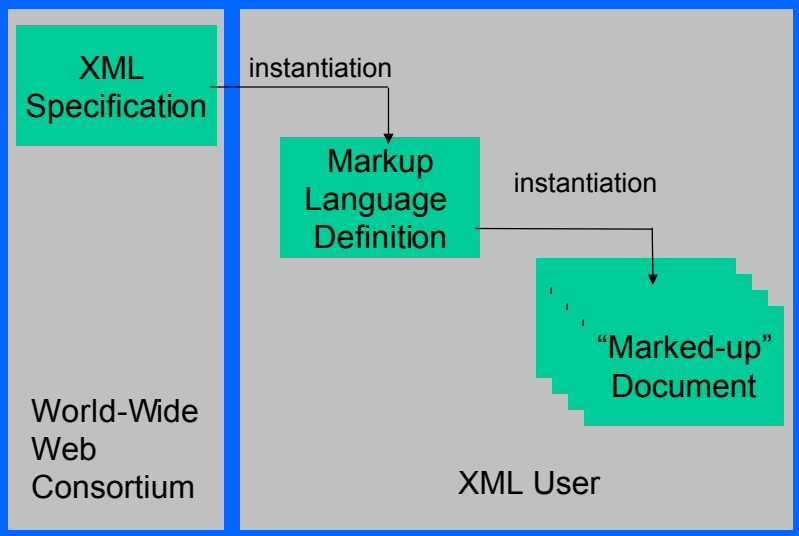
> *Chemical Markup Language (CML)*
> *Mathematical Markup Language (MathML)*
> *Open Financial Exchange (OFE or OFX)*
> *UML eXchange Format (UXF)*

Existing XML Applications *(Cont.)*

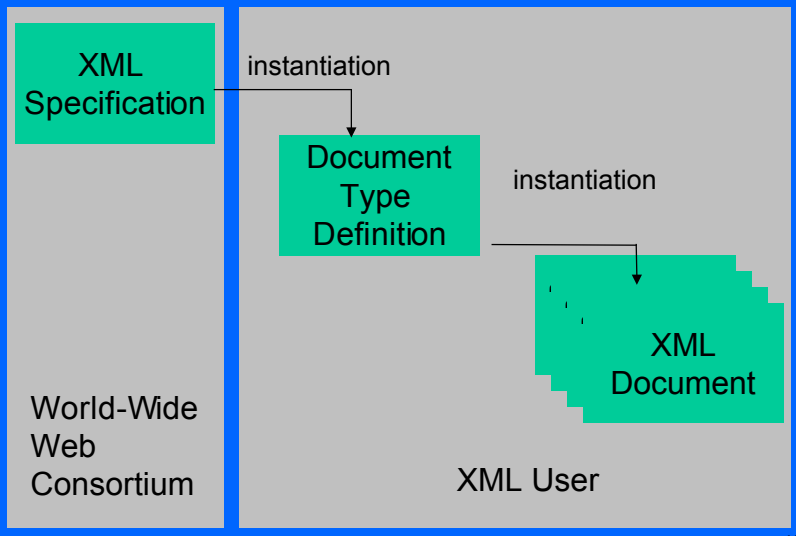
- **Internal applications/vocabularies**

> *The eXtensible HyperText Markup Language (XHTML)*
> *Resource Description Framework (RDF)*

XML

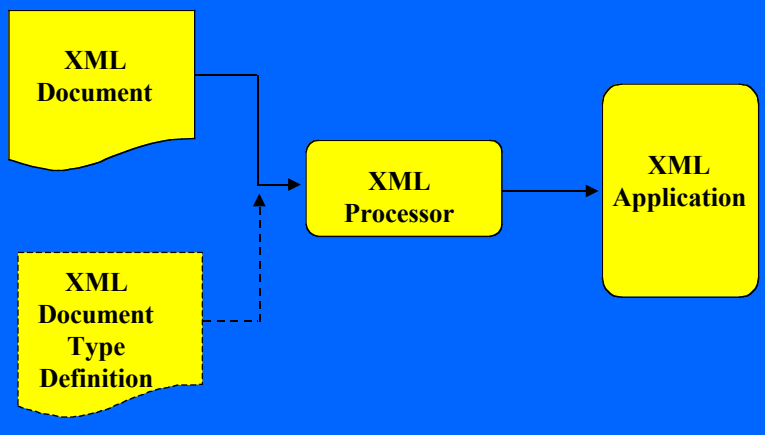


XML Documents



13

The Big Picture



14

XML Documents

- **Well-formed document:**

It obeys the rules for creating an XML document. The XML processor can build a *tree structure* representing the document. It does not necessarily have an associated ***document type definition*** (DTD).

- **Valid document:**

It has an associated DTD and each element in the document must conform to the rules that the DTD defines.

15

XML Documents

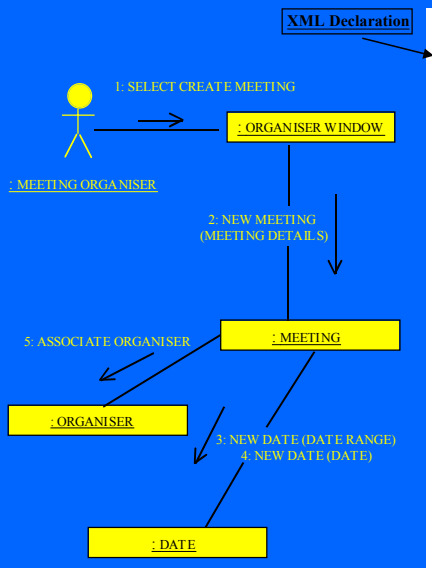
Rules for Well-formed documents:

1) Use a single root element;
2) Use a valid XML declarative statement;
3) Keep nesting order clear;
4) Do not overlap elements;
5) Match your start and end tags;
6) Close empty elements with the empty-element tag;
7) Attribute values are always in between " "

If rules for well-formed documents are violated
then
there is a fatal error !!!

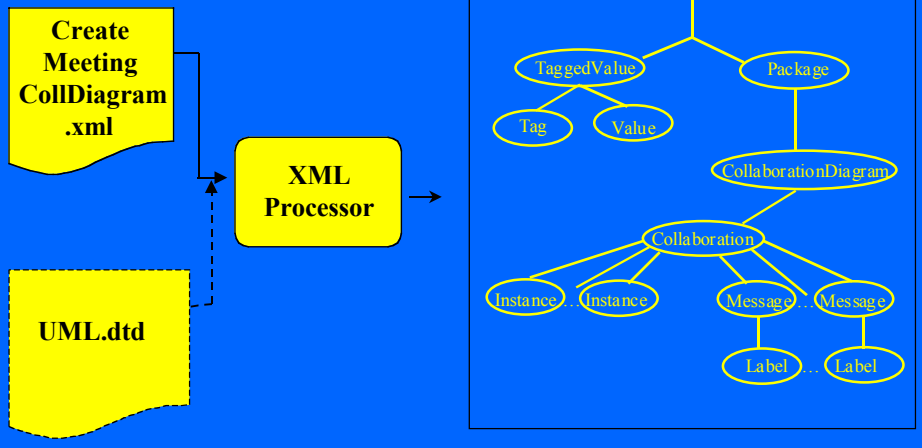
16

Example of a Well-Formed-Valid document




```
<!-- This is a XML document named CreateMeeting_CollDiagram-->
<?xml version="1.0"?>                          Document Type Declaration
<!DOCTYPE Model  SYSTEM "UML.dtd">

<Model>                                        Root
<TaggedValue>
        <Tag>Title<Value>Create Meeting Collaboration Diagram
                </Value></Tag>
        <Tag>Author<Value>Peter John</Value></Tag>
</TaggedValue>
<Package NAME="Create Meeting">
<CollaborationDiagram>
<Collaboration NAME="Create Meeting">
<Instance CLASS="Meeting Organiser" />
<Instance CLASS="Organiser Window" />
<Instance CLASS="Organiser" />
<Instance CLASS="Meeting" />
<Instance CLASS="Date" />
<Message TYPE="sync"
        SENDER="Meeting Organiser"
        RECEIVER="Organiser Window">
    <Label SEQUENCE_EX="1"
        MESSAGE_NAME="select create meeting" />
</Message>
. . .
</Collaboration>
</CollaborationDiagram>
</Package>
</Model>
```

XML Tree Structure



Document Type Definition (DTD)

UML.dtd

```
<!ELEMENT Model    (TaggedValue?, Package*)>
<!ELEMENT TaggedValue     (Tag*)>
<!ELEMENT Tag        (#PCDATA, Value*)>
<!ELEMENT Value      (#PCDATA)>
<!ELEMENT Note       (#PCDATA)>
<!ELEMENT Package  (TaggedValue?,Note*,
                    Dependency*,ClassDiagram?,
                    CollaborationDiagram?)>

<!ATTLIST Package NAME CDATA #REQUIRED>

<!ENTITY % ObjectElements "(TaggedValue?,
                   (Attribute|Operation|
                    Generalization|Association
                    |Dependency|Note)* )">
<!ENTITY % ClassDiagram SYSTEM
"class_diagram.dtd">
%ClassDiagram;
<!ENTITY % CollaborationDiagram SYSTEM

"collaboration_diagram.dtd">
%CollaborationDiagram;
```


Document Type Definition (DTD)

CollaborationDiagram.dtd

```
<!ELEMENT  CollaborationDiagram (TaggedValue?,
                                 Collaboration*)>
<!ELEMENT Collaboration (TaggedValue?, (Instance
                  | Interaction | Message | Note)*)>
<!ATTLIST Collaboration
          NAME          CDATA  #REQUIRED
          CLASSIFIER  CDATA  #IMPLIED
          OPERATION  CDATA  #IMPLIED>
<!ELEMENT Instance (Note*)>
<!ATTLIST  Instance
           NAME          CDATA  #IMPLIED
           CLASS         CDATA  #REQUIRED
           CONSTRAINT CDATA  #IMPLIED>
<!ELEMENT Interaction (Message)*>
<!ATTLIST  Interaction
           NAME       CDATA  #REQUIRED
           CONREXT  CDATA  #IMPLIED>
<!ELEMENT Message  (Label)>
<!ATTLIST   Message
            NAME          CDATA #IMPLIED
            TYPE          (sync|async|others) "sync"
            SENDER        CDATA  #REQUIRED
            RECEIVER    CDATA #REQUIRED
            ACTIVATOR CDATA #IMPLIED
            ACTION        CDATA #IMPLIED>
<!ELEMENT Label EMPTY>
<!ATTLIST   Label
            PREDECESSOR  CDATA #IMPLIED
            SEQUENCE_EX   CDATA #IMPLIED>
```

Why Create a DTD?

- To ensure that your documents conform to a given structure
- To provide visual XML editors with information needed to guide authors
- To enable the use of entities
- To enable others to make use of the vocabulary you have created

Where to find a DTD?

External DTDs (***external DTD subset***):

- files with .dtd extension;
- private or public;

Ex.:

```
<?xml version="1.0"?>
<!DOCTYPE Model SYSTEM "UML.dtd">
```

```
<?xml version="1.0"?>
<!DOCTYPE RED-BOOK PUBLIC "-//W3C//DTD HTML 4.0//EN"
          "http://www.rivendell.org/bilbo/red-book.dtd">
```

Internal DTDs (***internal DTD subset***):

- it cannot be used by another XML document;
- it is more convenient for testing;

Ex.:

```
<?xml version="1.0"?>
<!DOCTYPE Model
[<!ELEMENT Model (TaggedValue?, Package*)>
 <!ELEMENT TaggedValue (Tag)*>
 . . . ]>
```

XML Basics

Elements:

• start tag, body (content), and end tag

```
<!ELEMENT Collaboration (TaggedValue?, (Instance
                                  | Interaction | Message | Note)*)>
<!ELEMENT Interaction EMPTY>
<!ELEMENT Note   ANY>
<!ELEMENT Process (#PCDATA))
```

```
<Collaboration> <TaggedValue>Create Meeting</TaggedValue>
                                                  </Collaboration>
<Interaction></Interaction>          <Interaction/>

<Note> Here you can write <Anything/> respecting the <well-formed><rules/>
                                                  </well-formed></Note>

<Process> Here we can write any text </Process>
```

XML Basics *(Cont.)*

Attributes:

• (attribute, value) pairs associated with elements

Types:
1) String attributes
2) Tokenized attributes
3) Enumerated attributes

```
<!ATTLIST   Message
            NAME        CDATA              #IMPLIED
            MESS_IDT    ID                 #REQUIRED
            TYPE        (sync|async|others)  "sync"
            SENDER      CDATA              #REQUIRED
            LINKED_EL   CDATA              #FIXED     "Instance" >
```

Name **Type** **Default value**

XML Basics *(Cont.)*

Tokenized Attributes:

- **ID:** unique identifier for the element within the document;
- **IDREF:** reference to an element with a specific ID;
- **IDREFS:** reference to multiple elements with specific IDs;
- **ENTITY(IES):** points to an external entity;
- **NMTOKEN(S):** takes a value that is any mixture of letters, digits, and punctuation characteres.

XML Basics *(Cont.)*

```
<!ENTITY instance_address SYSTEM "http://www.inst.com">
...
<!ATTLIST Instance
            INST_ID      ID          #REQUIRED
            ADDRESS      ENTITY      #REQUIRED
            DATE         NMTOKEN     #REQUIRED>
...
<!ATTLIST Message
            INST_REF     IDREFS      #REQUIRED>

<Instance
      INST_ID="inst01"
      ADDRESS="instance_address"
      DATE="11-06-99"/>
<Instance
       INST_ID="inst02"
       ADDRESS="instance_address"
       DATE="12-06-99"/>
...
<Message
     INST_REF="inst01 inst02"/>
```

XML Basics *(Cont.)*

Entities:

Internal: The *replacement text* of the entity is provided with the declaration;

External: The *replacement text* of the entity is in another storage unit; the content portion of the entity's declaration refers to this external storage unit;

Ex.:
```
<!ENTITY % String "CDATA">
<!ENTITY % Chapter1 SYSTEM "chap.dtd">
. . .
<!ELEMENT Book ANY>
<!ATTLIST   Book  Title          %STRING;   #REQUIRED
                  Chapter_list   %Chapter1; #IMPLIED>
```

XML Basics *(Cont.)*

Parameter: Entities that are used only within the DTD;

General: Entities that can be used anywhere in a document. This entities are referenced by **&** (ampersand).

Ex.: …
```
<!DOCTYPE Model SYSTEM "UML.dtd"[
    <!ENTITY uml "Class Diagram">]>

<Model>
   &uml;
</Model>
```

XML Basics *(Cont.)*

Other Definitions:

- **Comments** <!-- Attention, this is a comment -->
- **White Space:** space, tab, carriage return, and line feed
- **CDATA Section:** <![CDATA[This text is insulated from attention of the parser]]>

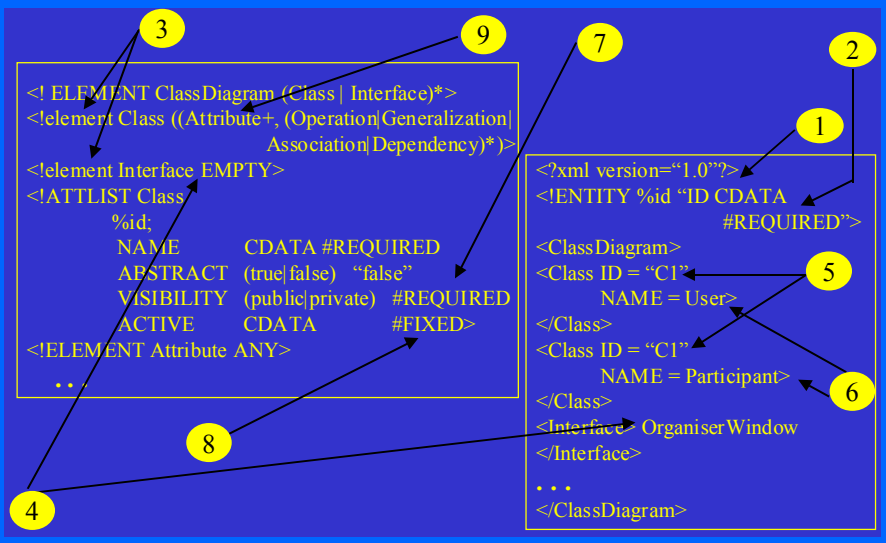
Exercise

1) In the extract of DTD and XML documents below there are 9 different types of syntax errors. Identify these errors and write the correct DTD and XML documents.

```
<! ELEMENT ClassDiagram (Class | Interface)*>
<!element Class ((Attribute+, (Operation|Generalization|
                    Association|Dependency)*)>
<!element Interface EMPTY>
<!ATTLIST Class
        %id;
        NAME        CDATA #REQUIRED
        ABSTRACT    (true|false)   "false"
        VISIBILITY  (public|private)  #REQUIRED
        ACTIVE      CDATA           #FIXED>
<!ELEMENT Attribute ANY>
  . . .
```

```
<?xml version="1.0"?>
<!ENTITY %id "ID CDATA
                    #REQUIRED">
<ClassDiagram>
<Class ID = "C1"
      NAME = User>
</Class>
<Class ID = "C1"
      NAME = Participant>
</Class>
<Interface> OrganiserWindow
</Interface>
. . .
</ClassDiagram>
```

Solution



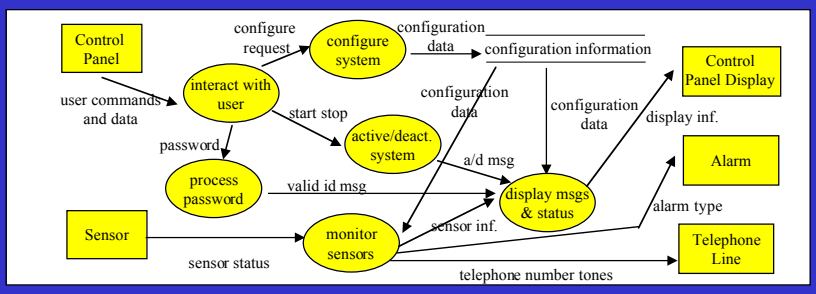
Solution

```
<!ENTITY %id "ID CDATA  #REQUIRED">
<!ELEMENT ClassDiagram (Class | Interface)*>
<!ELEMENT Class ((Attribute+, (Operation|Generalization|
                    Association|Dependency)*)>
<!ELEMENT Interface EMPTY>
<!ATTLIST Class
        %id;
        NAME        CDATA #REQUIRED
        ABSTRACT    (true|false)   "false"
        VISIBILITY  (public|private)  #REQUIRED
        ACTIVE      CDATA          #FIXED "true">
<!ELEMENT Attribute ANY>
   . . .
```

```
<?xml version="1.0"?>
<!DOCTYPE ClassDiagram
            SYSTEM "cldig.dtd">
<ClassDiagram>
<Class ID = "C1"
        NAME = "User"
        VISIBILITY = "public">
<Attribute>Password</Attribute>
</Class>
<Class ID = "C2"
        NAME = "Participant"
        VISIBILITY = "public">
<Attribute>Address</Attribute>
</Class>
<Interface> </Interface>
. . .
</ClassDiagram>
```

Exercise

2) Consider the DFD level 1 below. Construct a DTD for the DFD level 1 and the respective XML document for the given example.



Solution (DTD)

```
<!ELEMENT DFD (TaggedValue?, Level*)>
<!ELEMENT TaggedValue (Tag*)>
<!ELEMENT Tag    (#PCDATA, Value*)>
<!ELEMENT Value (#PCDATA)>
<!ELEMENT Level (Level0?, Level1?, Level2?)>
<!ELEMENT Level1 (TaggedValue?,(DataSource
                  |DataDestination| DataFlow
                  |StoredData| Function) *)
<!ATTLIST Level 1
          NAME    CDATA   #REQUIRED
          TEAM    CDATA   #IMPLIED
          DATE    CDATA   #IMPLIED>
<!ELEMENT DataSource (DataFlow*)>
<!ATTLIST DataSource
          NAME   CDATA   #REQUIRED>
<!ELEMENT DataDestination (DataFlow*)>
<!ATTLIST DataDestination
          NAME   CDATA  #REQUIRED>
<!ELEMENT DataFlow EMPTY>
<!ATTLIST DataFlow
          LABEL      CDATA  #REQUIRED
          RECEIVER CDATA #REQUIRED
          SENDER     CDATA #REQUIRED>
<!ELEMENT StoredData (DataFlow*)>
<!ATTLIST   StoredData
          NAME   CDATA   #REQUIRED>
<!ELEMENT Function (DataFlow*)>
<!ATTLIST Function
          NAME    CDATA   #REQUIRED>
```

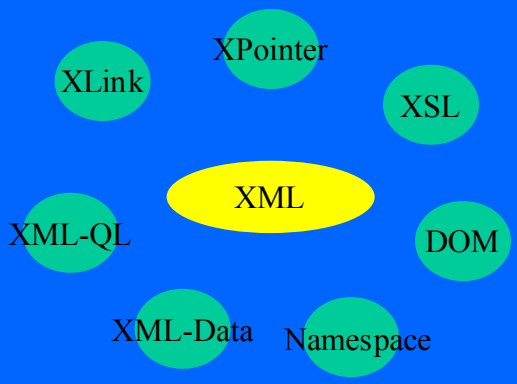
Solution (XML)

```
<?xml version="1.0"?>
<!DOCTYPE DFD System "DFD.dtd">
<DFD>
<TaggedValue>
 <Tag>Title<Value>Alarm System</Value></Tag>
 <Tag>Author<Value>R.Pressman</Value></Tag>
</TaggedValue>
<Level>
<Level1 NAME="SafeHome">
<DataSource NAME="Control Panel">
<DataFlow LABEL="user commands and data"
          RECEIVER="interact with user"
          SENDER="control panel"/>
</DataSource>
<DataSource NAME="Sensors">
<DataFlow LABEL="sensor status"
          RECEIVER="monitor sensors"
          SENDER="Sensors"/>
</DataSource>
<DataDestination NAME="Control Panel Display">
<DataFlow LABEL="display information">
          RECEIVER="Control Panel Display"
          SENDER="display msgs & status"/>
 </DataDestination>
 <DataDestination NAME="Alarm">
 <DataFlow LABEL="alarm type"
           RECEIVER="Alarm"
           SENDER="monitor sensors"/>
</DataDestination>
<DataDestination NAME="Telephone Line">
<DataFlow LABEL="telephone number tones"
          RECEIVER="Telephone Line"
          SENDER="monitor sensors"/>
</DataDestination>
<StoredData  NAME="configuration inf.">
<DataFlow LABEL="configuration data"
          RECEIVER="configuration inf."
          SENDER="configure system"/>
<DataFlow LABEL="configuration data"
          RECEIVER="display msgs & status"
          SENDER="configuration inf"/>
<DataFlow LABEL="a/d msg"
```


Solution (XML)

```
          RECEIVER="monitor sensors"
          SENDER="configuration inf."/>
</StoredData>
<Function   NAME="interact with user">
<DataFlow LABEL="password"
          RECEIVER="process password"
          SENDER="interact with user"/>
<DataFlow LABEL="start stop"
          RECEIVER="activate/deac. system"
          SENDER="interact with user"/>
<DataFlow LABEL="configure request"
          RECEIVER="configure system"
          SENDER="interact with user"/>
</Function>
<Function   NAME="configure system">
<DataFlow LABEL="configure request"
          RECEIVER="configure system"
          SENDER="interact with user"/>
<DataFlow LABEL="configuration data"
          RECEIVER="configuration inf."
          SENDER="configure system"/>
</Function>
<Function   NAME="process password">
<DataFlow LABEL="password"
          RECEIVER="interact with user"
          SENDER="process password"/>
</Function>
<Function   NAME="activate/deac. system">
<DataFlow LABEL="start stop"
          RECEIVER="activate/deac. system"
          SENDER="interact with user"/>
<DataFlow LABEL="a/d msg"
          RECEIVER="display msgs and status"
          SENDER="activate/deac. system"/>
</Function>
...
</Level1>
</Level>
</DFD>
```

XML and Related Technologies

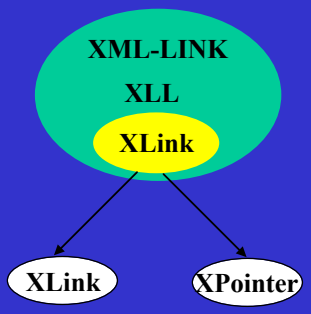


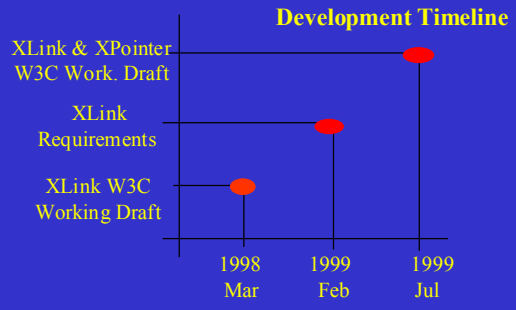
37

Linking with XML

Link: is an explicit relationship between two or more data objects or portions of data objects.

Address: is the description of how to find the two objects being linked





XLink

Characteristics:

1) It provides a mechanism for signaling the presence of a link (recognition)

2) It defines, partially, the intended behavior of the link

3) It extends the types of hypertext links that can be used beyond the simple hypertext model of HTML

39

XLink Terminology

- ***Linking element:*** An XML element that asserts the existence and describes the characteristics of a link.
- ***Resource:*** The addressable unit of information; anything that you can point at on the Web (e.g. file, document, program, image, query results, sound file).
- ***Locator:*** Character string associated with a link which identifies a resource.
- ***Traversal:*** The action of using a link (accessing a resource).
- ***Arc:*** A symbolic representation of traversal behavior in links (direction, context, and timing of traversal)

40

XLink Terminology *(Cont.)*

- ***Inline Link:*** A link that serves as one of its own resources. In an inline link the content of the linking element acts as a resource.

- ***Out-of-line Link:*** A link whose content does not serve as one of the resources of the link.

- ***Multidirectional Link:*** A link that can be traversed from more than one of its resources. A link that points the user in more than one direction. It gives the ability to move up, down, left, right, or backward.

41

XLink vs. HTML

- Any element can be a link element
- Control link behavior
- Multi-directional links
- Multiple destinations
- Indirect links via link files
- Links to specific unit of information

- Only A and IMG as link elements
- One behavior for A and one for IMG
- One-way links
- One link destination
- Link destination "lives" in source
- Links to the entire document

```
<A  xlink:type="SIMPLE"
xlink:href="www.html#root()/id("exp")"
    . . . >
W3C  <\A>
```

```
<A HREF="www.html#example">
W3C </A>
```

42

Goals of XLink

- Be usable over the Internet
- Be usable by a wide variety of link usage domains and classes of linking application software
- Support HTML 4.0 linking constructs
- Expression language must be XML
- Be feasible to implement
- Be informed by knowledge of established hypermedia systems and standards

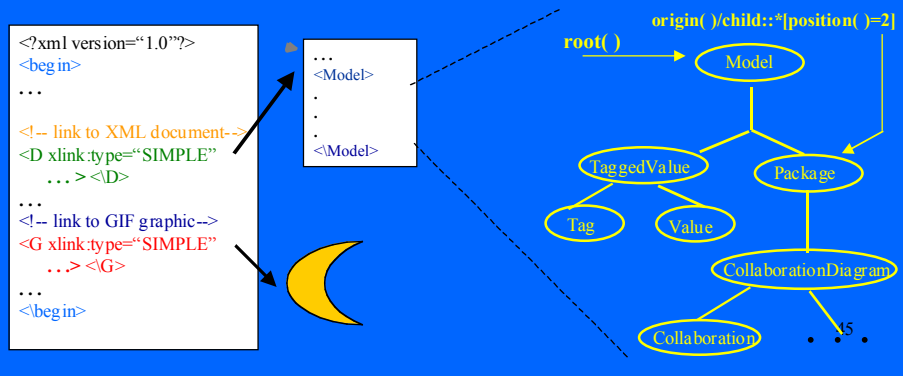
Goals of Xlink *(Cont.)*

- Be human-readable and human-writeable
- Be designed quickly
- Have a formal, concise and illustrative design
- Reside within or outside the documents in which the participating resources reside
- Represent the abstract structure and significance of links

XLink vs. XPointer

XLink: Is the syntax used to assert link existence and to describe link characteristics

XPointer: Is a language that supports addressing into the internal structures of XML documents. It points to subparts of XML resources *(fragment identifier)*

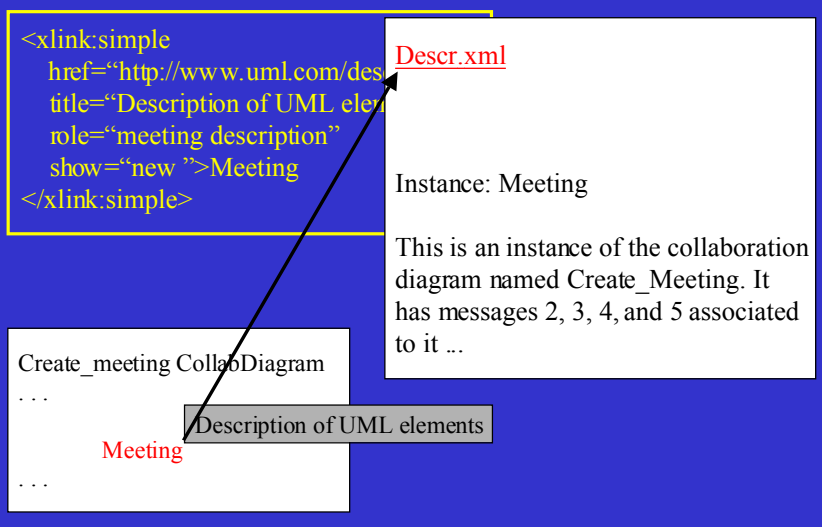


Simple Link

```
<!ELEMENT xlink:simple ANY>
<!ATTLIST   xlink:simple
        href      CDATA                 #REQUIRED
        role      CDATA                 #IMPLIED
        title     CDATA                 #IMPLIED
        show      (new|parsed|replace)  "replace"
        actuate   (user|auto)           "user">
```

```
<!ELEMENT Simple-Link ANY>
<!ATTLIST   Simple-Link
   xmlns:xlink   CDATA                          #FIXED   "http://www.w3..."
   xlink:type    (simple|extended|locator|arc)  #FIXED   "simple"
   xlink:role    CDATA                          #IMPLIED
   xlink:href    CDATA                          #REQUIRED
   xlink:title   CDATA                          #IMPLIED
   xlink:show    (new|parsed|replace)           "replace"
   xlink:actuate (user|auto)                    "user">
```

Simple Link (Example)



Extended Link

```
<!ELEMENT xlink:extended ((xlink:arc | xlink:locator)*)>
<!ATTLIST   xlink:extended
        role              CDATA              #IMPLIED
        title             CDATA              #IMPLIED
        showdefault       (new|parsed|replace)  #IMPLIED
        actuatedefault    (user|auto)        #IMPLIED >
```

```
<!ELEMENT Extended-Link ((xlink:arc | xlink:locator)*)>
<!ATTLIST Extended-Link
    xmlns:xlink           CDATA                        #FIXED   "http://..."
    xlink:type            (simple|extended|locator|arc) #FIXED   "extended"
    xlink:role            CDATA                        #IMPLIED
    xlink:title           CDATA                        #IMPLIED
    xlink:showdefault     (new|parsed|replace)         #IMPLIED
    xlink:actuatedefault  (user|auto)                  #IMPLIED>
```

Extended Link *(Cont.)*

```
<!ELEMENT xlink:locator ANY>
<!ATTLIST   xlink:locator
    id          ID          #REQUIRED
    href        CDATA       #REQUIRED
    role        CDATA       #IMPLIED
    title       CDATA       #IMPLIED >


<!ELEMENT Reference ANY>
<!ATTLIST   Reference
   xmlns:xlink  CDATA  #FIXED
                       "http://www.w3.org..."
   xlink:type   (locator)  #FIXED "locator"
   id           ID         #REQUIRED
   xlink:href   CDATA      #REQUIRED
   xlink:role   CDATA      #IMPLIED
   xlink:title  CDATA      #IMPLIED >
```

```
<!ELEMENT xlink:arc ANY>
<!ATTLIST   xlink:arc
   from     IDREF    #REQUIRED
   to       IDREF    #REQUIRED
   show     (new|parsed|replace) "replaced"
   actuate          (user|auto)   "user" >


<!ELEMENT generic_arc ANY>
<!ATTLIST   generic_arc
   xmlns:xlink   CDATA  #FIXED
                        "http://www.w3.org..."
   xlink:type    (arc)      #FIXED  "arc"
   xlink:from    IDREF      #REQUIRED
   xlink:to      IDREF      #REQUIRED
   xlink:show    (new|parsed|replace)
                                    "replaced"
   xlink:actuate  (user|auto)   "user" >
```

Extended Link (Example)

```
<xlink:extended
   showdefault="replace"
   actuatedefault="user">
<xlink:locator  id="inst1"
                href="http://www.uml
                title="Meeting Organi
<xlink:locator  id="inst2"
                href="http://www.uml
                                    " >

                                    ww.uml
                                    er" >
                                    nst3  show
                                    nst2"  show
```

Descrorg.xml

Instance: Organiser

This is an instance of the collaboration diagram named Create_Meeting. It has messages 3, 4, and 5 associated to it ...

Descrmorg.xml

Instance: Meeting Organiser

...

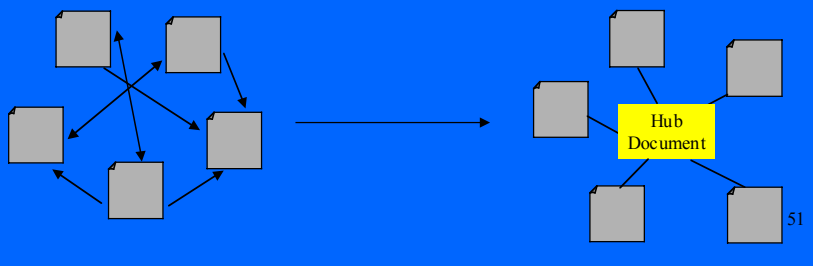
Descrmeet.xml

Instance: Meeting

...

Extended Link Group

```
<!ELEMENT xlink:group (xlink:document*)>
<!ATTLIST    xlink:group
     steps                CDATA                    #IMPLIED>

<!ELEMENT xlink:document  EMPTY>
<!ATTLIST    xlink:document
     href                 CDATA                    #REQUIRED>
```

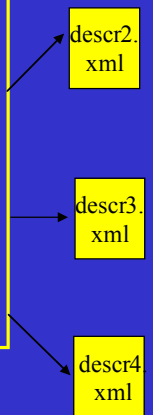


51

Extended Link Group (Example)

descr1.xml

```
<xlink:group
   step = 1 >

  <xlink:document
              href="http://www.uml.com/descr2.xml"/>
  <xlink:document
              href="http://www.uml.com/descr3.xml"/>
  <xlink:document
              href="http://www.uml.com/descr4.xml"/>

</xlink:group>
```



XPointer

XPointer allows you to target a given element by number, name, type, or relation to other elements in the document.

XPointer avoids modifications in the target document; i.e. unlike HTML it is not necessary to insert a named anchor in the target document.

HREF = "***url#XPointer***" or HREF = "***url\|XPointer***"

URL

HREF = "http://www.cs.ucl.ac.ul/~Staff/a.zisman/uml.xml#
root()/child::Package[position()=1]/child::ClassDiagram[position()=1]

XPointer

XPointer (Location Path)

- Based on **XPath** - XML Path Language
(W3C working draft - http://www.w3.org/1999/07/WD-xpath-19990709)

basis

LocationPath: ***axis-name :: node-test[predicate]****
(/ axis-name :: node-test[predicate]*)*

XPointer (Location Path - *Cont.*)

Absolute axes:

a) root( ) - /
b) origin( )
c) id( )
d) here( )
e) unique( )

Relative axes:

a) ancestor
b) ancestor-or-self
c) attribute
d) child
e) descendant
f) descendant-or-self
g) following
h) following-sibling
i) parent
j) preceding
k) preceding-sibling
l) self

XPointer (Location Path - *Cont.*)

Node Tests:

a) *
b) node( )
c) text( )
d) comment( )
e) processing-instruction( )

Predicates:

a) position ( ) = integer
(<, >, !=, >=, <=)
b) position( ) = last( )
c) Count(Location Path)
d) attribute::attr_name

Examples

```
<?xml version="1.0"?>
<!DOCTYPE Model SYSTEM "UML.dtd">

<Model>
<TaggedValue>
      <Tag>Title<Value>Create Meeting Collaboration Diagram
                  </Value></Tag>
      <Tag>Author<Value>Peter John</Value></Tag>
</TaggedValue>
<Package NAME="Create Meeting">
<CollaborationDiagram>
<Collaboration NAME="Create Meeting">
<Instance CLASS="Meeting Organiser" />
<Instance CLASS="Organiser Window" />
<Instance CLASS="Organiser" />
<Instance CLASS="Meeting" />
<Instance CLASS="Date" />
<Message TYPE="sync"
          SENDER="Meeting Organiser"
          RECEIVER="Organiser Window">
  <Label SEQUENCE_EX="1"
          MESSAGE_NAME="select create meeting"/>
</Meesage>
. . .
</Collaboration>
</CollaborationDiagram>
</Package>
</Model>
```

root( )

child::TaggedValue/child::Tag[position( )=2]/child::Value[position( )=1]

descendant::Instance[position( )=3]

descendant::Message[attribute::SENDER="Meeting Organiser"]

origin( )/parent::*

XPointer (*Cont.*)

Axes:

a) *range :: LocationPath , LocationPath*

ex.: range::/descendant::Instance[position( )=1], /descendant::Instance[position( )=last( )]

b) *string :: 2,"Meeting Organiser", 5*

(selects the position before the letter "i" in the second occurrence of the string "Meeting Organiser")

Goals of XPointer

- Be usable over the Internet
- Address into XML documents
- Be usable in URIs
- Be feasible to implement
- Be optimized for usability
- Be human-readable and reasonably compact
- Be designed quickly
- Be prepared quickly
- Have a formal and concise design

Exercise

3) Consider part of an XML document and the XPointer expressions below and identify the respective elements.

```
<Model>
<TaggedValue>
<Tag>Title<Value>Business Entities Class
            Diagram</Value></Tag>
<Tag>Date<Value>10/09/99</Value></Tag>
</TaggedValue>
<Package NAME="BusinessEntities">
<ClassDiagram>
<Class NAME="Meeting">
<Attribute NAME="status"
      ID="attribute6"
      TYPE="String"> </Attribute>
<Attribute NAME="name"
      ID="attribute7"
      TYPE="String"> </Attribute>
</Class></ClassDiagram> </Package></Model>
```

a) root( );
b) origin( )/child::Package/
   child::ClassDiagram/
   child::Class[position()=1]/
   child::Attribute[position()=2);
c) id("attribute7");
d) origin( )/parent::*;
e) range::/descendant::Attribute
   [position( )=1],/descendant::
   Attribute[position( )=2];
f) /descendant::Value[position( )=2];

Solution

```
<Model>
<TaggedValue>
<Tag>Title<Value>Business Entities Class
        Diagram</Value></Tag>
<Tag>Date<Value>10/09/99</Value></Tag>
</TaggedValue>
<Package NAME="BusinessEntities">
<ClassDiagram>
<Class NAME="Meeting">
<Attribute NAME="status"
      ID="attribute6"
      TYPE="String"> </Attribute>
<Attribute NAME="name"
      ID="attribute7"
      TYPE="String"> </Attribute>
</Class></ClassDiagram> </Package></Model>
```

a) root( );
b) origin( )/child::Package/
child::ClassDiagram/
child::Class[position()=1]/
child::Attribute[position()=2);
c) id("attribute7");
d) origin( )/parent::*;
e) range::/descendant::Attribute
[position( )=1],/descendant::
Attribute[position( )=2];
f) /descendant::Value[position()=2];

61

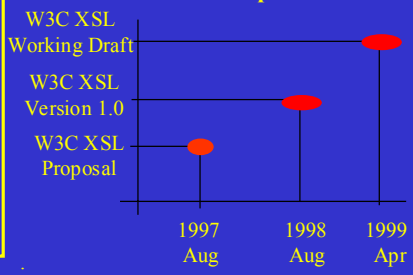
XSL

XSL - eXtensible Style Language

Based on ISO/IEC 10179 Document Style Semantics and Specification Language (DSSSL)

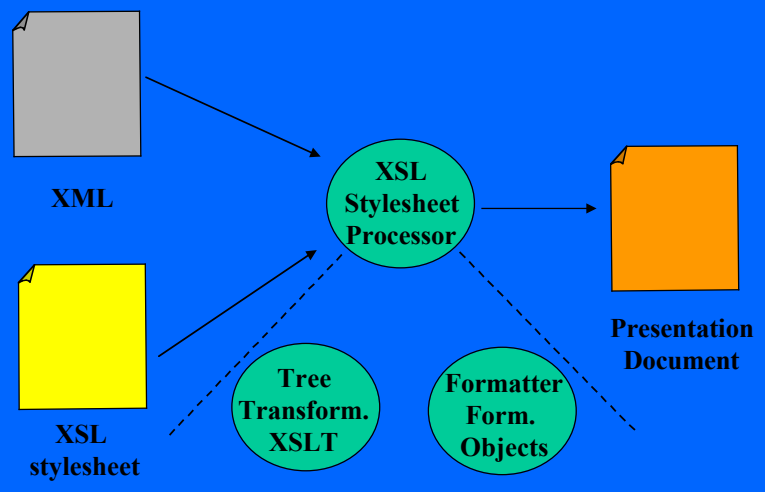
Based on W3C's Cascading Style Sheet (CSS) Language

Development Timeline



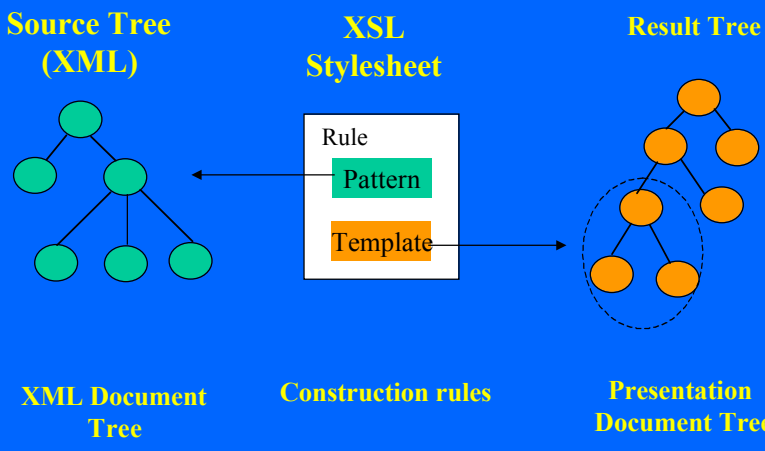
- **XSL is a language for expressing stylesheets;**
- **XSL allows authors to transform the structure of an XML source tree into an XML result tree (XSLT);**
- **XSL allows authors to apply formatting and presentation operations to XML elements (formatting objects vocabulary);**

XSL Processor



63

XSL *(Cont)*



64

XSL *(Cont.)*

Tree Construction Rules ***(Template Rules)***

- **Pattern**: identifies the element from the XML source document tree to which the construction rule applies **(<xsl:template>).**
- **Template:** it is instantiated to create the result tree.
- **Formatting Objects:** the typographic units from which a presentation is constructed.

Examples

XML:

```
<TaggedValue>
   <Tag>Title<Value>Create Meeting Collaboration Diagram</Value></Tag>
   <Tag>Author<Value>Peter John</Value></Tag>
</TaggedValue>
```

XSL:

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
   xmlns:fo="http://www.w3.org/XSL/Format/1.0">
   <xsl:template match="Tag">
     <fo:block font-size="18pt" color="green">
       <xsl:apply-templates/>
     </fo:block>
   </xsl:template>
   <xsl:template match="TaggedValue">
     <xsl:apply-templates/>
      Date: 11-06-99
   </xsl:template>
</xsl:stylesheet>
```

Presentation Document

```
<fo:block font-size="18pt" color="green"
 xmlns:fo="http://…Format/1.0">Title Create
Meeting Collaboration Diagram</fo:block>
<fo:block font-size="18pt" color="green"
 xmlns:fo="http://…Format/1.0">Author Peter
John</fo:block>
Date: 11-06-99
```

Title Create Meeting Collaboration Diagram
Author Peter John
Date: 11-06-99

Examples

XML:

```
<TaggedValue>
   <Tag>Title<Value>Create Meeting Collaboration Diagram</Value></Tag>
   <Tag>Author<Value>Peter John</Value></Tag>
</TaggedValue>
```

XSL:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
   <xsl:template match="Tag">
    <DIV style="font-weight:bold; color:blue">
      <xsl:apply-templates/>
    </DIV>
   </xsl:template>
   <xsl:template match="TaggedValue">
     <xsl:apply-templates/>
      <DIV style="color:red">
       Date: 11-06-99</DIV>
   </xsl:template>
</xsl:stylesheet>
```

Presentation Document (HTML)

```
<DIV style="font-weight:bold; color:blue">
Title Create Meeting Collaboration Diagram
</DIV>
<DIV style="font-weight:bold; color:blue">
Author Peter John </DIV>
<DIV style="color:red">
Date: 11-06-99</DIV>
```

Title Create Meeting Collaboration Diagram
Author Peter John
Date: 11-06-99

Other Examples

XML:

```
<ClassDiagram><Class>Meeting
   <Attribute>status</Attribute>
   <Attribute>name</Attribute>
   <Method>add</Method>
   <Method>remove</Method>
</Class></ClassDiagram>
```

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
   <xsl:template match="ClassDiagram">
    <DIV style="font-weight:bold; color:blue">
      <xsl:apply-templates select="Method"/>
    </DIV>
   </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
   <xsl:template match="ClassDiagram">
     <ul>
      <xsl:apply-templates select="Class">
        <xsl:sort select="Attribute"/>
      </xsl:apply-templates>
     </ul>
   </xsl:template>
</xsl:stylesheet>
```

Exercise

4) Write XSL tree construction rules for part of the XML document below
   a) Element TaggedValue should be bold and red;
   b) Organise element Instance in alphabetical order;
   c) Element Message should be of size 20pt and green;
   d) Element Label should be yellow.

```
<TaggedValue>
  <Tag>Title<Value>Create Meeting
      Collaboration Diagram</Value></Tag>
  <Tag>Author<Value>Peter John</Value>
                                  </Tag>
</TaggedValue>
<CollaborationDiagram>
<Collaboration NAME="Create Meeting">
<Instance>Meeting Organiser</Instance>
<Instance>Organiser Window</Instance>
<Instance>Organiser</Instance>
<Instance>Meeting</Instance>
<Instance>Date</Instance>
```

```
<Message TYPE="sync"
         SENDER="Meeting Organiser"
         RECEIVER="Organiser Window">
 <Label SEQUENCE_EX="1"
  MESSAGE_NAME="select create meeting" />
</Message>
. . .
</Collaboration>
</CollaborationDiagram>
</Package>
</Model>
```

Solution

a)

```
<xsl:template match="TaggedValue">
  <DIV style="font-weight:bold; color:red">
     <xsl:apply-templates/>
  </DIV>
 </xsl:template>
```

b)

```
<xsl:template match="Instance">
      <xsl:apply-templates select="Instance">
         <xsl:sort select="Instance"/>
       </xsl:apply-templates>
     </xsl:template>
```

d)

```
<xsl:template match="Message">
    <DIV style="color:yellow">
      <xsl:apply-templates select="Label">
    </DIV>
 </xsl:template>
```

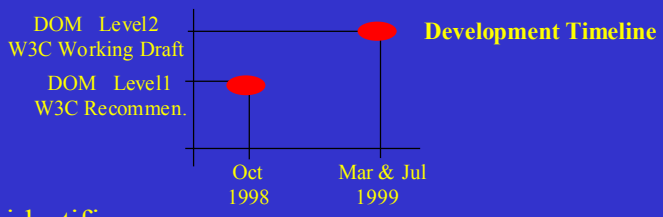
c)

```
<xsl:template match="Message">
      <DIV style="font-size:20pt; color:green">
         <xsl:apply-templates/>
       </DIV>
     </xsl:template>
```

Document Object Model (DOM)

- DOM is an application programming interface (API) for documents, i.e. HTML and XML documents.
- DOM is a way to describe an XML document to another application or programming language in an effort to manipulate the information the way it is wanted.
- DOM defines the logical structure of documents and the way a document is accessed and manipulated.
- DOM allows programmers to build documents, navigate their structures, and add, modify, or delete elements and contents.

DOM *(Cont.)*



DOM identifies:

(a) the interfaces and objects used to represent and manipulate a document;

(b) the semantics of these interfaces and objects (behavior and attributes);

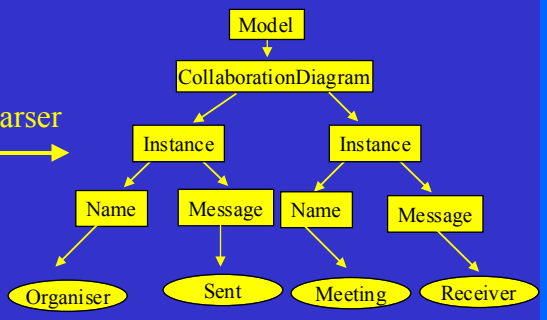
(c) the relationships and collaborations among these interfaces and objects.

DOM *(Cont.)*

DOM representation (logical structure)

```
<Model>
<CollaborationDiagram>
<Instance>
<Name>Organiser</Name>
<Message>Sent</Message>
</Instance>
<Instance>
<Name>Meeting</Name>
<Message>Receive</Message>
</Instance>
</CollaborationDiagram>
</Model>
```

Parser



DOM Objects

The nodes in the tree represent objects and not data structure.

DOM *(Cont.)*

- **Level 1:** Methods to represent and manipulate document structure and content.

> **Core:** interfaces for accessing and manipulating XML contents

> **XML:** interfaces for accessing and manipulating XML DTDs

> **HTML:** interfaces for accessing and manipulating HTML contents

DOM *(Cont.)*

• **Level 2:**

> **Stylesheets:** interfaces for associating stylesheets with a document

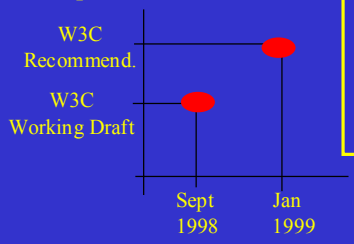
> **Event model:** interfaces for the design of an event system which allows registration of event handlers, describes event flow and provides contextual information for each event

> ***Query*, Filters, Iterators, TreeWalker:** interfaces to allow traversal of document subtrees, node lists, or *query results*

> **DOM Range:** interfaces for accessing and manipulating a range of content in a document.

Namespace

Development Timeline



It is a collection of names, identified by a URI, which are used in XML documents for qualifying names used as ***element types*** and ***attribute names***.

Motivation:

• Applications where a single XML document contains elements and attributes that are defined for and used by multiple software modules.

• Documents containing markup from multiple sources pose problems of recognition and collision.

Namespace *(Cont.)*

Declaring and Using Namespaces:

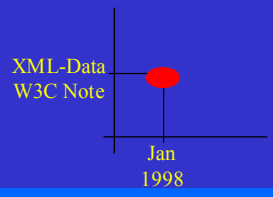
```
<?xml version="1.0"?>
  <uml:diagram  xmlns:uml='http://uml.org'>
        <uml:class_diagram>This is a universally unique element
                                    </uml:class_diagram>
   </uml:diagram>

<?xml version="1.0"?>
  <uml:diagram xmlns:uml='http://uml.org'
                    xmlns:web='http://web.org'>
         <uml:class_diagram
                  web:address="www.classdiagram.com />
  </uml:diagram>
```


XML-Data

- **XML-Data** is an XML vocabulary (syntax) for schemas; i.e. for describing and documenting object classes.
- **XML-Data** can describe syntax of XML documents and conceptual relationships.
- **XML-Data** provides developers to further specify particular elements (a model for extending XML elements).

Schemas describe the rules of an XML document, such as element names, elements ordering, available attributes.

Development Timeline



XML-Data *(Example)*

UML.dtd

```
<!ELEMENT Model (Package+)>
<!ELEMENT Package (#PCDATA)>
```

UML.xml

```
<?xml version="1.0"?>
<!DOCTYPE Model SYSTEM UML.dtd>
<Model>
<Package>Collaboration Diagram</Package>
<Package>Class Diagram</Package>
<Package>StateChart Diagram</Package>
</Model>
```

UML.xml

```
<?xml version="1.0"?>
<?xml:namespace name="http://data.org/"
                as="dt"/?>

<dt:schema>
  <elementType id="Package">
    <string/>
  </elementType>
  <elementType id="Model">
    <element type="#Package"
             occurs="ONEORMORE"/>
  </elementType>
</dt:schema>

<Model>
<Package>Collaboration Diagram</Package>
<Package>Class Diagram</Package>
<Package>StateChart Diagram</Package>
</Model>
```

XML-QL

Development Timeline

Who?

AT&T Labs
University of Pensilvania

- **How can data be extracted from large XML documents?**
- **How can XML data be exchanged between user communities using different ontologies?**
- **How can XML data from multiple sources be integrated?**

Data extraction
Data transformation → **XML-QL**
Data integration

XML-QL *(Cont.)*

- XML-QL can express *queries* which can extract pieces of data from XML documents.
- XML-QL can express *transformations* which can map XML data between DTDs and can integrate XML data from different sources.
- XML-QL supports joins and aggregates.

XML-QL *(Examples)*

<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book
<!ELEMENT artic

<!ATTLIST article
<!ELEMENT publ
<!ELEMENT auth

<bib>
<book year="1995"
<title>An Introdu
<author> <lastnan
<publisher><nam

<book year="1998"
<title>Foundatio
The Third
<author><lastnan
<author><lastnan
<publisher><nam
</publisher>
</book> </bib>

<result>
<author><lastname>Date</lastname></author>
<title>An Introduction to DB Syst</title>
</result>
<result>
<author><lastname>Date</lastname></author>
<title>Foundation for Object/Relational Dbs: The
Third Manifesto</title>
</result>
<result>
<author><lastname>Darwen</lastname></author>
<title>Foundation for Object/Relational Dbs:The
Third Manifesto</title>
</result>

dison-Wesley"
></publisher>

hor> $a </>
e> $t </> </result>

XML-QL *(Examples)*

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<!ELEMENT article (author+, title, year?,
                   (short|long))>
<!ATTLIST article type CDATA)>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname)>
```

```
<!ELEMENT person (lastname, firstname, address?,
                  phone?, publicationtitle*)
```

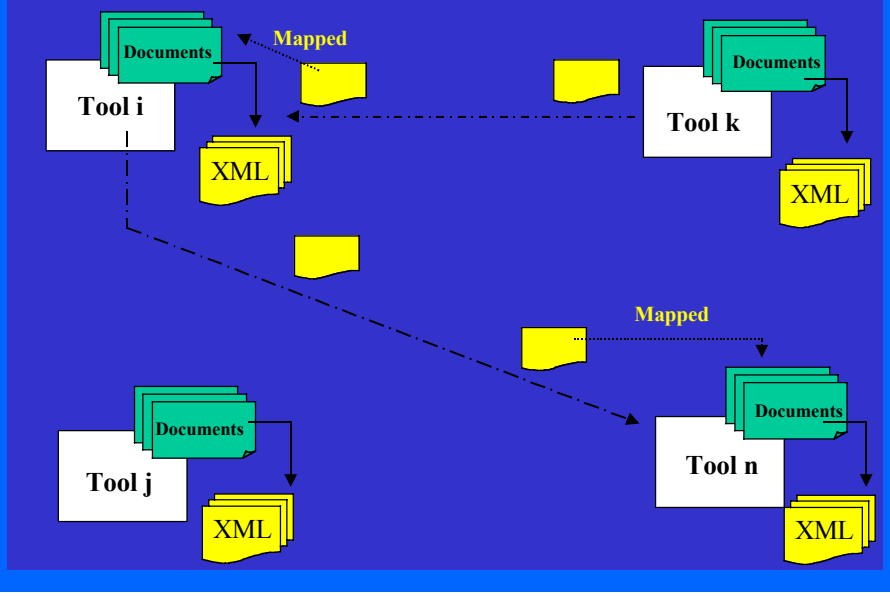
Transforming XML Data

```
WHERE <$> <author><firstname> $fn</>
                  <lastname> $ln</></>
          <title> $t </>
        </>  in "bib.xml"
CONSTRUCT <person ID=PersonID($fn,$ln)>
            <firstname> $fn</>
            <lastname> $ln</>
            <publicationtitle> $t</>
            </person>
```

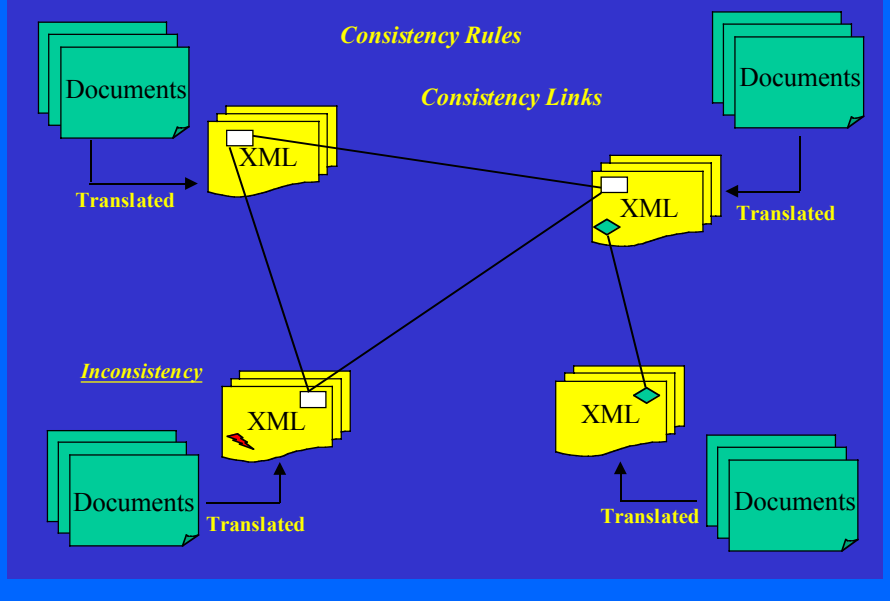

Software Engineering Applications

- **Structured information on the Web (repositories)**
- **Information interchange**
- **Consistency Management**
- **Generation of specialised mark-up languages**
- **Document templates**
- **Multiple document views & Graphical views**
- **Validation of semi-structured documents**

Information Interchange



Consistency Management



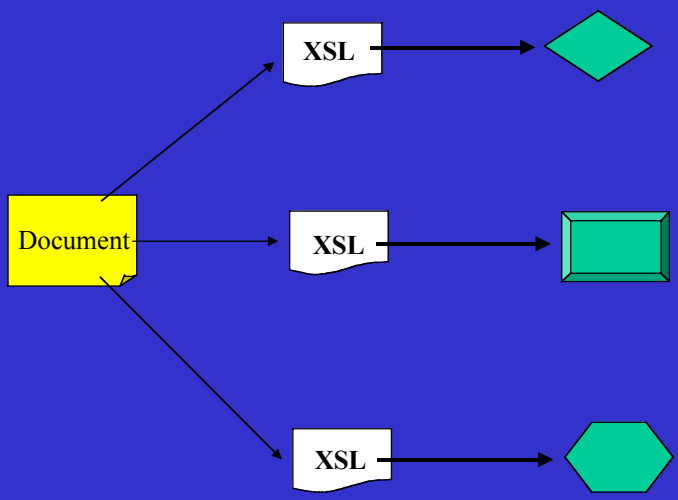
Generation of Specialised Mark-up Lgs.

- Mark-up language for UML Model *(XMI, UXF);*
- Mark-up language for Z specification *(ZIF);*
- Mark-up language for Data Flow Diagrams;
- Mark-up language for other design methods;

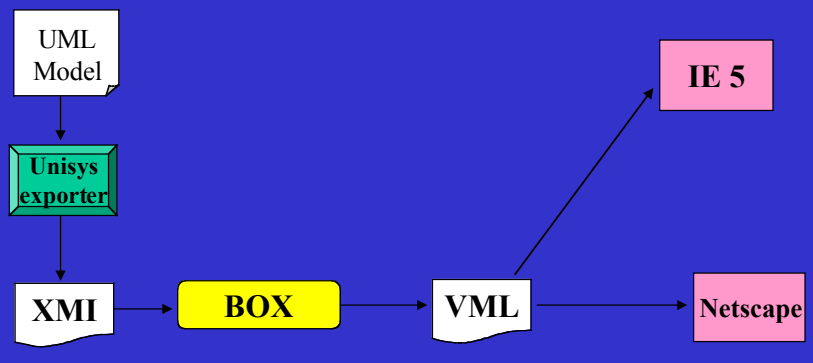
Document Templates

- User requirements documents;
- Software Requirements Specification *(SRS IEEE);*
- Architectural Design Document;
- Detailed Design Document;
- User Manual;
- Software Verification and Validation Plan;
- Software Engineering Standards *(PSS-05-0);*

Multiple Document Views



Graphical View



Exercise/Solution

Other Software Engineering applications ?

Final Remarks

- XML is a data description language, subset of SGML;
- XML is designed to bring structured information to the Web;
- XML provides a data standard that can encode the content, semantics and schemata for a wide variety of cases;
- XML allows identification, exchange and processing of distributed data in a manner that is mutually understood;
- XML provides *extensibility*, *structure*, and *data checking* needed for a large-scale commercial document distribution and publishing.

References

Books:

1) *The XML Handbook*; C.F. Goldfarb & P. Prescod; 1998; Prentice Hall

2) *XML by Example, Building E-Commerce Applications*; S. McGrath; 1998; Prentice Hall

3) *XML A Primer*; S. St. Laurent; 1998

4) *XML Black Book;* N. Pitts-Moultis & C. Kirk; 1999; Coriolis

5) *XML Specification Guide*; I.S. Graham & L. Quin; 1999; John Wiley & Sons, Inc.

References *(Cont.)*

URLs:

1) List of Books
http://www.able-consulting.com/books_xml.htm

2) Technical reports, working drafts, recommendations & notes
http://www.w3.org/TR

3) Software
http://www.xmlsoftware.com/