Managing Information (CSCU9T4) Lecture 2: XML Structure

Gabriela Ochoa

http://www.cs.stir.ac.uk/~goc/

Outline

- XML Elements vs. Attributes
- Well-formed vs. Valid XML documents
- Document Type Definitions (DTDs)
- XML Schemas
- Summary

XML Components

- Elements: can contain text, other elements, or be empty. Examples:
  - `<body>some text</body>`

  -
    ```
    <person>
        <firstname>Anna</firstname>
        <lastname>Smith</lastname>
    </person>
    ```
  - `<img src="computer.gif" />`
- Attributes: provide extra information about elements.
  - always placed inside the opening tag of an element.
  - always come in name/value pairs.
  - `<img src="computer.gif" />`

XML Elements vs. Attributes

- Data can be stored in child elements or in attributes. No rules about when to use one or the other.
- Two examples that provide the same information:

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

XML Elements vs. Attributes

- Suggestions:
  1. You should try to avoid attributes.
     - Cannot contain multiple values.
     - Cannot contain tree structures.
  2. Use child elements if the information feels like data.
  3. Use attributes if information feels like metadata.

```xml
<note date="2008-01-10">
  <from>Anna</from>
  <to>Smith</to>
</note>
```

```xml
<note id=“501">
  <from>Anna</from>
  <to>Smith</to>
</note>
```

```xml
<note>
  <date>
      <day>01</day>
      <month>10</month>
      <year>2008</year>
  </date>
  <from>Anna</from>
  <to>Smith</to>
</note>
```

Well-formed and Valid XML documents

- An XML document with correct syntax is *well formed*. Syntax rules were described in the previous lecture.
  - XML documents must have a root element
  - XML elements must have a closing tag
  - XML tags are case sensitive
  - XML elements must be properly nested
  - XML attribute values must be quoted
- A valid XML document is not the same as a well-formed XML document.
- A valid XML document must be well formed and must follow a defined structure.
- Two methods to define structure:
  - **Document Type Definition (DTD) or**
  - **XML Schema**

XML DTDs and Schemas

- They are optional but sometimes necessary. Why?
  - XML documents can have many different structures.
  - An application cannot tell if a particular document it receives is complete, missing data or ordered properly.
- Perform validity checks on XML documents
- Validity is Important. Why?
  - Web used for B2B (Business to Business)
  - Reader is not a person, but a program that extracts the content for further processing
  - To avoid unexpected troubles as errors in XML documents may stop your applications.

Document Type Definitions (DTD)

Document Type Definition (DTD)

- DTD constraints the structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What sub-elements can/must occur inside each element, how many times and in what order.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - `<!ELEMENT element (subelements-specification) >`
  - `<!ATTLIST element (attributes)  >`

Example: XML document with DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note
(to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
weekend!</body>
</note>
```

- !DOCTYPE note: root element
- !ELEMENT note: note element must contain four elements: "to, from, heading, body"
- !ELEMENT to: to element is of type "#PCDATA"
- !ELEMENT from: from element is of type "#PCDATA"
- !ELEMENT heading: heading element is of type "#PCDATA"
- !ELEMENT body: body element is of type "#PCDATA"

#PCDATA means parseable text data. Text will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded

External DTD: Simple example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

XML document with an external DTD

File "note.dtd" containing the Document Type Definition:

```
<?xml version="1.0"?>
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

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

<!-- Fig. 20.3: letter.xml                        -->
<!-- Business letter formatted with XML           -->

<!DOCTYPE letter SYSTEM "letter.dtd">

<letter>

   <contact type = "from">
      <name>John Doe</name>
      <address1>123 Main St.</address1>
      <address2></address2>
      <city>Anytown</city>
      <state>Anystate</state>
      <zip>12345</zip>
      <phone>555-1234</phone>
      <flag gender = "M"/>
   </contact>

   <contact type = "to">
      <name>Joe Schmoe</name>
      <address1>Box 12345</address1>
      <address2>15 Any Ave.</address2>
      <city>Othertown</city>
      <state>Otherstate</state>
      <zip>67890</zip>
      <phone>555-4321</phone>
      <flag gender = "M"/>
   </contact>
   <salutation> Good afternoon </salutation>
   <paragraph> Description of the letter </paragraph>
   <closing> Closing remarks </closing>
   <signature> Sender signature </signature>
</letter>
```

Element `flag` does not contain any text, just an attribute. It is said to be "empty"

Information for the person writing the letter.

Information for the person receiving the letter.

Letter.xml

Letter.dtd

```
<!-- Fig. 20.4: let
<!-- DTD document

<!ELEMENT letter ( contact+, salutation, paragraph+,
   closing, signature )>

<!ELEMENT contact ( name, address1, address2,
   zip, phone, flag )>
<!ATTLIST contact type CDATA #IMPLIED>

<!ELEMENT name ( #PCDATA )>
<!ELEMENT address1 ( #PCDATA )>
<!ELEMENT address2 ( #PCDATA )>
<!ELEMENT city ( #PCDATA )>
<!ELEMENT state ( #PCDATA )>
<!ELEMENT zip ( #PCDATA )>
<!ELEMENT phone ( #PCDATA )>
<!ELEMENT flag EMPTY>
<!ATTLIST flag gender (M | F) "M">

<!ELEMENT salutation ( #PCDATA )>
<!ELEMENT closing ( #PCDATA )>
<!ELEMENT paragraph ( #PCDATA )>
<!ELEMENT signature ( #PCDATA )>
```

The **ELEMENT** element type declaration defines the rules for element **letter**.

The plus sign (**+**) occurrence indicator specifies that the DTD allows one or more occurrences of an element.

The **contact** element definition specifies that element **contact** contains child elements **name**, **address1**, **address2**, **city**, **state**, **zip**, **phone** and **flag**— in that order.

Keyword **#IMPLIED** specifies that if the parser finds a **contact** element without a **type** attribute, the parser can choose an arbitrary value for the attribute or ignore the attribute and the document will be valid.

The ***ATTLIST*** element type declaration defines an attribute (i.e., **type**) for the **contact** element.

Flag **#PCDATA** specifies that the element can contain parsed character data (i.e., text).

DTD SYNTAX

- DOCTYPE Declaration
  - Internal: `<!DOCTYPE root_element [ … ]>`
  - External: `<!DOCTYPE root_element SYSTEM "DTD_location">`
- ELEMENT Declaration
  - `<!ELEMENT name allowable_contents>`
- ATTLIST Declaration
  - `<!ATTLIST element_name attribute_name attribute_type default_value>`
  - *element_name*: name of the element to which the attribute applies
- ENTITY Declaration
  - Internal: `<!ENTITY name "entity_value">`
  - External: `<!ENTITY name SYSTEM "URI">`
  - Example: http://oak.cs.ucla.edu/cs144/reference/DTD.html

ELEMENT TYPE DECLARATION

- Set the rules for:
  - The type and number of elements
  - What elements may appear inside each other
  - Order of appearance
- Syntax: `<!ELEMENT name allowable_contents>`
- The *allowable_contents* are

Allowable Contents	Definition
EMPTY	Refers to tags that are empty (i.e with no text content). They can have attributes
ANY	Refers to anything at all, as long as XML rules are followed. ANY is useful to use when you have yet to decide the allowable contents of the element.
children elements	You can place any number of element types within another element type.
Mixed content	Refers to a combination of (#PCDATA) and children elements. PCDATA stands for parsed character data, that is, text that is not markup. Therefore, an element that has the allowable content (#PCDATA) may not contain any children.

ELEMENT TYPE DECLARATION: DECLARING CHILDREN

- Children element types are declared using parentheses in the parent element type's declaration.
- Syntax:

```
<!ELEMENT parent_name (child_name)>
<!ELEMENT child_name allowable_contents>
```

- Example:

```
<?xml version="1.0"?>
<!DOCTYPE student [
<!ELEMENT student (id)>
<!ELEMENT id (#PCDATA)>
]>
<student>
  <id>9216735</id>
</student>
```

Notes:
- 'student' must have one child element type 'id'
- 'id' is of type PCDATA (text)

Rules
- The child element must be declared in a separate element type declaration

ELEMENT TYPE DECLARATION: DECLARING MULTIPLE CHILDREN

- Multiple children are declared using commas (,).
- Commas fix the sequence in which the children are allowed to appear in the XML document..
- Syntax:

```
<!ELEMENT parent_name (child1_name,child2_name,child3_name)>
<!ELEMENT child1_name allowable_contents>
<!ELEMENT child2_name allowable_contents>
<!ELEMENT child3_name allowable_contents>
```

- Example:

```
<?xml version="1.0"?>
<!DOCTYPE student [
<!ELEMENT student (id,firstname,surname) )>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT surname (#PCDATA)>
]>
<student>
  <id>9216735</id>
 <firstname>Jo</firstname>
  <surname>Smith</surname>
</student>
```

Elements in the order listed

ELEMENT TYPE DECLARATION: OTHER ASPECTS OF DECLARING CHILDREN

- Declaring optional children
  - Use the (?) operator. Optional means zero or one times.
  - Syntax: `<!ELEMENT parent_name (child_name?)>`
- Declaring zero or more children:
  - Use the (*) operator.
  - Syntax: `<!ELEMENT parent_name (child_name*)>`
- Declaring one or more Children:
  - Use the (+) operator.
  - Syntax: `<!ELEMENT parent_name (child_name+)>`
- Combinations of children (choice)
  - A choice between children element types is declared using the (|) operator
  - Syntax: `<!ELEMENT parent_name (child1_name|child2_name)>`

More complex example

```
<?xml version="1.0"?>
<!DOCTYPE student [
  <!ELEMENT student (surname,name+,dob?,(origin|gender)?)>
  <!ELEMENT surname (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT dob (#PCDATA)>
  <!ELEMENT origin (#PCDATA)>
  <!ELEMENT gender (#PCDATA)>
]>
```

```
<student>
  <surname>Smith</surname>
  <name>Anne</name>
  <name>Marie</name>
  <gender>female</gender>
</student>
```

```
<student>
  <surname>Smith</surname>
  <name>Anne</name>
  <name>Marie</name>
  <origin>Stirling</origin>
</student>
```

```
<student>
  <surname>Smith</surname>
  <name>Anne</name>
  <name>Marie</name>
</student>
```

```
<student>
  <surname>Smith</surname>
  <name>Anne</name>
  <name>Marie</name>
  <dob>01-01-1998</dob>
</student>
```

Attribute list (ATTLIST) declaration

- Set of rules for
  - Which elements types may have attributes
  - What type of attributes they may be
  - The default values for attributes
- Syntax:

  `<!ATTLIST` *`element_name attribute_name attribute_type default_value`*`>`
- Example:

```
<?xml version="1.0"?>
<!DOCTYPE image [
  <!ELEMENT image EMPTY>
  <!ATTLIST image height CDATA #REQUIRED>
  <!ATTLIST image width CDATA #REQUIRED>
]>
<image height="32" width="32"/>
```

PCDATA: is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.

CDATA: is text that will *not* be parsed by a parser. Tags inside the text will *not* be treated as markup and entities will not be expanded.

Attribute list (ATTLIST) declaration

- `<!ATTLIST` *`el_name att_name att_type default_value`*`>`
- Main attribute types
  - **`CDATA:`** character data, that is, text that does not form markup (no **`<, >, &,'`** or **`"`** allowed)
  - **`ID:`** a unique identifier of the attribute. IDs of a particular value should not appear more than once in an XML document.
  - **`IDREF:`** used to establish connections between elements. The IDREF value of the attribute must refer to an ID value declared elsewhere in the document.
- Main default values
  - **`#REQUIRED:`** the attribute must always be included
  - **`#IMPLIED:`** the attribute does not have to be included.
  - **`#FIXED:`** The attribute must always have the default value that is specified
- Examples: http://xmlwriter.net/xml_guide/attlist_declaration.shtml

XML Schema

XML Namespaces

- Major goal of XML is to add uniformity to tags.
- Potential problem:
  - When different XMLs are combined, or
  - When tags created by several people are combined
- XML Namespace – solve ambiguity problem.
- Allow elements and attributes to be distinguish.
- Provides a unique prefix at the beginning of every elements and attributes.
- Technically, the prefix can be any URI (Uniform Resource Identifier).
  - **URI** is a string of characters which identifies an Internet resource.

A Simple example: XML Namespaces

HTML table information

```
<table>
   <tr>
      <td>Apple</td>
      <td>Banana</td>
   </tr>
</table>
```

A furniture table information

```
<table>
   <name>Coffee table</name>
   <width>80</width>
   <length>120</length>
</table>
```

```
<root>
    <table>
        <tr>
            <td>Apple</td>
            <td>Banana</td>
        </tr>
    </table>
    <table>
        <name>Coffee table</name>
        <width>80</width>
        <length>120</length>
    </table>
</root>
```

A Simple example: XML Namespaces

```
<root>
     <h:table xmlns:h=“http://www.w3.org/TR/html4/">
          <h:tr>
               <h:td>Apple</h:td>
               <h:td>Banana</h:td>
          </h:tr>
     </h:table>
     <f:table xmlns:f=“https://www.w3schools.com/furniture/">
          <f:name>Coffee table</f:name>
          <f:width>80</ f:width>
          <f:length>120</f:length>
     </f:table>
</root>
```

```
<root xmlns:h=“http://www.w3.org/TR/html4/”  xmlns:f=“https://www.w3schools.com/furniture/">
     <h:table>
          <--- Child elements --- >
     </h:table>
     <f:table>
          <--- Child elements --- >
     </f:table>
</root>
```

XML Schema

- Describe structure of an XML document, like a DTD
- Became a W3C Recommendation in May 2001
- More powerful than DTD because they:
  - are written in XML
  - are extensible to additions
  - support data types (easier to define restriction on data)
  - support namespaces
- Drawback of Schemas: they are more complex and verbose than DTD

XML Schema- Simple Example

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="note">
     <xs:complexType>
       <xsd:sequence>
         <xs:element name="to" type="xs:string"/>
         <xs:element name="from" type="xs:string"/>
         <xs:element name="heading" type="xs:string"/>
         <xs:element name="body" type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

note.xsd

<xs:element name="note"> defines the element called "note"

<xs:complexType> the "note" element is a complex type

<xs:sequence> the complex type is a sequence of elements

<xs:element name="to" type="xs:string"> the element "to" is of type string (text)

<xs:element name="from" type="xs:string"> the element "from" is of type string

<xs:element name="heading" type="xs:string"> the element "heading" is of type string

<xs:element name="body" type="xs:string"> the element "body" is of type string

XML Schema: Simple Example

```
<?xml version="1.0" encoding="UTF-8"?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=“note.xsd">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

THE <SCHEMA> ELEMENT

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

<xs:schema>
...
...
</xs:schema>
```

- The root element of every XML Schema
- It may contain some attributes

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
...
...
</xs:schema>
```

`<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>`

- Elements and data types come from the "http://www.w3.org/2001/XMLSchema" namespace.
- Elements and data types in the scheme should be prefixed with `xs`:

`targetNamespace=http://www.w3schools.com`

- Elements defined by this schema come from the "http://www.w3schools.com" namespace.

`xmlns=http://www.w3schools.com`

- The default namespace is "http://www.w3schools.com".

Elements

- Elements are the main building block of all XML documents, containing the data and determine the structure of the instance document.
- Syntax: `<xs:element name="x" type="y"/>`
  - `Name` property: tag name that will appear in the XML document
  - `Type` property: provides the description of what type of data can be contained within the element when it appears in the XML document.
- Element content type can be:
  - Simple: simple predefined data types (such as: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time)
  - Complex: other elements
  - User defined types: using the `<xs:simpleType>` and `<xs:complexType>` constructs, which we will cover later.

Examples of simple elements and their XML data

Sample XSD	Sample XML
`<xs:element name="Customer_dob“ type="xs:date"/>`	`<Customer_dob> 2000-01-12T12:13:14Z </Customer_dob>`
`<xs:element name="Customer_address "type="xs:string"/>`	`<Customer_address> 99 London Road </Customer_address>`
`<xs:element name="OrderID“ type="xs:int"/>`	`<OrderID> 5756 </OrderID>`
`<xs:element name="Body“ type="xs:string"/>`	`<Body> (a type can be defined as a string but not have any content, this is not true of all data types however).</Body>`

Note: the corresponding value in the XML document must be in the correct format for its given type otherwise this will cause a validation error.

FIXED AND DEFAULT PROPERTIES

- The valid data values for the element in the XML document can be further constrained using the fixed and default properties.
  - **Default**: if no value is specified in the XML document then the application reading the document, should use the default specified in the XSD.
  - **Fixed**: the value in the XML document can only have the value specified in the XSD
- Example:

```
<xs:element name="Customer_name" type="xs:string"
default="unknown"/>
<xs:element name="Customer_location" type="xs:string"
fixed="UK"/>
```

Cardinality constraints

- Use to indicate the specific number of elements that can appear in an XML document
- Use `minOccurs` and `maxOccurs` attributes
- Specify whether an element is mandatory, optional, or can appear up to a set number of times.
- Default values for `minOccurs` and `maxOccurs` is 1. So if not present, the element must appear once.
- `minOccurs`
  - Any non-negative integer value (e.g. 0, 1, 2, 3… etc.)
- `maxOccurs`
  - Any non-negative integer value
  - Special string constant `unbounded` meaning there is no maximum so the element can occur an unlimited number of times.

Cardinality constraints: Examples

Sample XSD	Description
`<xs:element name="Customer_dob" type="xs:date"/>`	If we do not specify minOccurs or maxOccurs, then the default values of 1 are used. This means there has to be one and only one occurrence of Customer_dob, i.e. it is mandatory.
`<xs:element name="Customer_order" type="xs:integer" minOccurs ="0" maxOccurs="unbounded"/>`	If we set minOccurs to 0, then the element is optional. Here, a customer can have from 0 to an unlimited number of Customer_orders.
`<xs:element name="Customer_hobbies" type="xs:string" minOccurs="2" maxOccurs="10"/>`	Setting both minOccurs and maxOccurs means the element Customer_hobbies must appear at least twice, but no more than 10 times.

Complex Types

- A complex type is a container for other element definitions, this allows you to specify which child elements an element can contain.
- E.g. The following XML element

```
<customer>
   <firstname>John</firstname>
   <lastname>Smith</lastname>
   <address>House x, st y, Stirling</address>
   <phone>Hxyz-000</phone>
</customer>
```

Other possibilities include xs:choice and xs:all

```
<xs:element name="Customer">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="firstname" type="xs:string" />
         <xs:element name="lastname" type="xs:string" />
         <xs:element name="address" type="xs:string" />
         <xs:element name="phone" type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Complex Types – Order Indicators

- xs:sequence
  - use when all child element must present and the order of the elements does matter.
- xs:choice
  - use when one of the child elements must be present.
- xs:all
  - use when all child elements must present and the order doesn’t matter

Complex Types: Another Example

```xml
<customer>
     <name>
          <firstname>John</firstname>
          <middlename>M</middlename>
          <lastname>Smith</lastname>
     </name>
     <address>
          <city>Stirling</city>
          <postcode>FK9 4LA</postcode>
     </address>
     <carddetails>
          <cardNo>Dave Smith</cardNo>
          <pinCode>7445</pinCode>
          <cardType>visa</cardType>
     </carddetails>
</customer>
```

1. Customer can have zero or multiple middle names.
2. City or post code
3. Elements must appear in the order as displayed.
4. The child elements of carddetails can occur in any order but all must be present.

Complex Types: A slightly complicated Example

```
<xs:element name="customer">
     <xs:complexType>
          <xs:sequence>
               <!--- Definition of name element -->
               <!--- Definition of address element -->
               <!--- Definition of carddetails element -->
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

```
<xs:element name="name">
     <xs:complexType>
        <xs:sequence>
     <!--- Definitions of child elements -->
        </xs:sequence>
     </xs:complexType>
</xs:element>
```

```
<xs:element name="cardetails">
     <xs:complexType>
          <xs:all>
           <!--- Definitions of child elements -->
          </xs:all>
     </xs:complexType>
</xs:element>
```

```
<xs:element name="address">
     <xs:complexType>
          <xs:choice>
          <!--- Definitions of child elements -->
           </xs:choice>
      </xs:complexType>
</xs:element>
```

User defined data types: Example

```
<trade>
	<customer>
		<name>John Smith</name>
		<address>Stirling, FK9 4LA</address>
		<phone>yyyy</phone>
		<email>c@test.com</phone>
	</customer>
	<supplier>
		<name>William scot</name>
		<address>Stirling, FK9 4LA</address>
		<phone>xxxx</phone>
		<email>s@test.com</phone>
	</supplier>
	<--- Some other details --->
</trade>
```

```
<xs:element name="trade">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="customer">
				<xs:complexType>
					<xs:sequence>
						<!--- Definition for each child element -->
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="supplier">
				<xs:complexType>
					<xs:sequence>
						<!--- Definition for each child element -->
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<!--- Definition of carddetails element --->
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

USER DEFINED DATA TYPES: EXAMPLE

```
<xs:element name=“trade”>
     <xs:complexType>
          <xs:sequence>
                    <xs:element name=“customer” type=“personType”>
                    <xs:element name=“supplier” type=“personType”>

                    <!--- Definition of carddetails element --->
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

```
<xs:complexType name=“personType”>
     <xs:complexType>
          <xs:sequence>
                    <!--- Definition of each child element --->
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

SUMMARY

- Two ways of describing the structure of XML docs.
  - Document Type Definitions (DTDs)
    - <!DOCTYPE *root_element* >
    - <!ELEMENT *name allowable_contents*>
    - <ATTLIST *el_name att_name att_type default_value* >
  - XML Schemas
    - <schema>
    - <xs:element name="x" type="y"/>
    - <xs:attribute name="x" type="y"/>
- DTDs are the older standard
  - Gradually succumbing to XML Schemas
  - Are still in widespread use particularly with (X)HTML.
- XML Schemas offer more functionality
  - Written in XML so the same tools can be used with both the data and its schema.
  - More complex and verbose