Information Systems

DTD and XML Schema

Nikolaj Popov

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
popov@risc.uni-linz.ac.at

Outline

DTDs

- DTD stands for Document Type Definition.
- Used to identify the legal structure of an XML document.
- Describe the structure, content, and quantity of elements, attributes and entities that can exist within an XML document.

Why DTDs are Useful

Two reasons:

1. DTDs allow verification that an XML document is structurally consistent with a formal specification.
2. Independent parties sharing the same DTD can be sure that the XML documents they create will share the same structure.

DTDs and Schemas

- XML schemas serve the same purpose as DTDs.
- XML schemas are W3C's recommended replacement for DTDs.
- Schemas are likely to replace DTDs, but still DTDs are heavily used in the industry.

Outline

Document Type Declarations

1. DTDs must be declared with the Document Type Declaration.
2. The declaration is placed at the beginning of an XML document.
3. Indicates that the document's structure will be defined by a DTD.
4. Inline DTDs: Declared within an XML document.
5. External DTDs: Declared in separate entities (files) and identified in an XML document with an external URL reference.

Inline DTDs

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

<!DOCTYPE mydoc [
  <!-- DTD content goes here . . .  -->
]>

<mydoc>
  <!-- XML content goes here . . .  -->
</mydoc>
```

- An XML document with a Document Type Declaration for an DTD.
- The name that appears after `<!DOCTYPE` must match the name of the XML document's root element.

Reference to an External DTDs

```
<!DOCTYPE mydoc SYSTEM
   "http://www.mysite.com/mydoc.dtd">
```

- External reference to a DTD uses the SYSTEM command.
- External DTDs are more useful than inline ones, because a single DTD can be referenced by multiple XML documents.

Outline

Element Declarations

- Element declarations define contents of XML elements.
- Format of element declarations:
  `<!ELEMENT element_name (element_content)>`
- `element_name` defines the XML tag for the element.
- `element_content` lists the names of allowed child elements.

Element Declarations. Example

```
<!ELEMENT baseball_team (name,city,player*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT player (#PCDATA)>
```

- Every `<baseball_team>` element can contain only one `<name>` and `<city>` elements and an unlimited number of `<player>` elements.
- PCDATA: Parsed Character Data.
- `#PCDATA`: Term used in DTDs to indicate that an element contains character set.

Element Names

```
<!ELEMENT baseball_team (name,city,player*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT player (#PCDATA)>
```

- Fragment of an XML document illegal wrt the DTD:

```
<name>
<firstname>Bob</firstname>
<lastname>Smith</lastname>
</name>
```

- Legal version: `<name>Bob Smith</name>`

Element Names

- DTD allows to define both child elements, and their order.
- Child element names can appear on their own, or can include one of the suffixes “*”,“+”,“?”.
- Elements without suffix can appear only once within that element.
- Elements with “*” can appear zero or more times within that element.
- Elements with “+” can appear one or more times within that element.
- Elements with “?” can appear zero or one times within that element.

Element Names. Examples

```
<!ELEMENT baseball_team (name,city,player*)>
<!ELEMENT cast (actor+)>
<!ELEMENT name (first,middle?,last)>
```

Choices

- DTDs allow to choose between multiple child elements using (exclusive) choices.
- A choice is indicated by the pipe "|" character.
- Example. Clothing elements may contain either a `<pants>` or a `<skirt>` child element, but not both:

```
<!ELEMENT clothing (pants|skirts)>
```

Parentheses

- Child elements can be grouped together by parentheses.
- Useful when suffixes and choices are applied to multiple elements.
- Example. The `<clothes>` elements must begin with a `<glasses>` element, and can then contain any number of `<earrings>` and `<necklace>` elements, in any combination or order:

```
<!ELEMENT clothes
    (glasses?,(earrings|necklace)*)>
```

Mixed-Contents Elements

- Mixed-content elements can contain both character text and other elements.
- Example: `<!ELEMENT paragraph (#PCDATA|highlight|term)*>`
- DTD does not allow to specify, e.g., how many of character data elements, `<highlight>`, or `<term>` elements are allowed.

Empty and Any

- Defining an empty element:
  `<!ELEMENT shape EMPTY>`
- Empty elements are used by many XML applications.
- The `ANY` keyword is useful if one is defining an element for which it is not clear what the content might be:
  `<!ELEMENT additional_information ANY>`

Outline

Attribute List Declarations

- DTD attribute list declarations define what attributes can appear inside element start-tags.
- In addition to defining the attribute names, the attribute list declarations allow to define the type of an attribute, and whether or not it is required.
- The basic format:

```
<!ATTLIST element  att_name att_type att_default
                   att_name att_type att_default>
```

Attribute Names

- Attribute names must be legal XML names.
- Any number of attributes can be defined within an element.
- Each name must be unique within an element.
- The order does not matter.

Attribute Types

- Attribute types can be used to restrict the values of declared attributes.
- XML defines basic attribute types.
- Four of the most basic attribute types:
  - CDATA
  - NMTOKEN
  - ID
  - Enumerated

Attribute Types

- CDATA: Attribute values can contain any legal XML character.
- NMTOKEN: Attribute values can contain only characters legal for XML name. The only difference with XML name is that NMTOKENs can begin with numbers, dashes, or periods.
- ID: Attribute values must be a legal XML name, and must be unique within the document.
- Enumerated: Attribute values are given a specific list of values they are allowed to have.

Attribute Defaults

- Each attribute declaration must indicate whether the attribute is required, and whether it has a default value.
- Four possible attribute defaults:
  - #Required: Attribute must be included within the element in order the XML to be valid. No default value is provided.
    ```
    <!ATTLIST employee empid ID #REQUIRED>
    ```
  - #Implied: The attribute is optional. No default value is provided.
    ```
    <!ATTLIST employee shoesize NMTOKEN
    #IMPLIED>
    ```
  - #Fixed: The attribute can have only one value and that value is set.
    ```
    <!ATTLIST employee comp_name CDATA #FIXED
    "ABC Inc.">
    ```
  - A literal string can be included in place of the above keywords to provide a default value for the attribute. The default can be overwritten within the XML document.
    ```
    <!ATTLIST item price CDATA "0">
    ```

Outline

Entities

- XML defines five entity references to prevent certain characters from conflicting with reserved characters (<, >, &, ’, and ").
- DTDs allow to define other entity references.
- Example. Suppose we want to have the same copyright information on every XML-based page we create.
  - Create an entity reference in the DTD,
  - assign it a certain name,
  - and give it the appropriate XML markup for our copyright information as a value.
  - Use this name as an entity reference in our XML documents

Entities. Example

- Entity declaration within a DTD:

```
<!ENTITY copyright
  "<copyright id='123'>
    <company>Pop&apos;s Lecturing
Ltd</company>
    <date>2018</date>
  </copyright>">
```

- External entity declaration, with the copyright information in a separate XML file:

```
<!ENTITY copyright SYSTEM
  "http://www.mysite.com/copyright.xml">
```

- Using entity reference:

```
<footer>
  Copyright information:  &copyright;
</footer>
```

Summary

DTDs

- define the structure of XML document's elements and attributes,
- define the order in which they can occur,
- define (within limits) the values they can contain,
- allow consistency among multiple XML documents,
- ensure that XML documents created by different applications conform to the same structure.

Outline

XML Schema

- XML schemas are templates for XML documents.
- XML schemas are XML documents themselves that must follow strict syntactical rules.
- They can specify what structures the modeled XML documents may have and impose restrictions on their contents.
- There are also several software products available to validate schema definitions and XML documents against well defined schemas.

Properties of XML Schemas

- XML schemas are valid XML documents themselves, so there is no additional parser necessary in XML applications.
- XML schemas provide a wide range of data types (integers, dates, strings, etc.) for element content and attribute values.
- Schemas allow the users to define custom data types, and support inheritance between types.
- Schemas also provide powerful features to express element groupings, and other properties of elements like uniqueness.

Referencing Schemas

An XML document that references a schema is called an instance of the schema.

To reference a schema in an instance XML document one has to:

- If a target namespace is declared in the schema, this also has to be declared in the instance,
- define the schema instance namespace, which is the predefined namespace.
- reference the schema by setting the value of the `schemaLocation` attribute (belonging to the schema instance namespace) to the URI of the schema.

Outline

Element Declarations

- The structure of the instance documents of a schema is principally described by the structure of the schema itself.
- For instance if we want to prescribe the following structure in instances

```
<e1>
    <e2>
        <e3> ...  </e3>
    </e2>
        <e4> ...  </e4>
</e1>
```

we will encapsulate the declarations of these elements using this structure in the schema.

Complex Element Structures

- **sequence** This requires that the instance contains the elements that appear in the schema in the same order by default exactly once.
- **all** This allows that elements in the definition group appear in any order by default exactly once.
- **choice** This allows only one of the elements in the definition group to appear in an instance document inside the complex element.
- **minOccurs** Specifies the minimal number of occurrences of the declared element in its parent element.
- **maxOccurs** Specifies the maximal number of occurrences of the declared element in its parent element.

Declaration Complex Elements

Fixed structure:

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="id" type="xsd:unsignedInt"/>
      <xsd:element name="price" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Example: Complex Elements

```
<product>
  <name>Disney Mouse Pad</name>
  <id>231069948</id>
  <price>3.49</price>
</product>
```

Declaration Complex Elements

Fixed content that can appear in any order:

```
<xsd:element name="product_extension">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="made" type="xsd:string"/>
      <xsd:element name="oem" type="xsd:boolean"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Example: Complex Elements

Instance elements of the previous schema fragment.

```
<product_extension>
  <made>Germany</made>
  <oem>false</oem>
</product_extension>

<product_extension>
  <oem>true</oem>
  <made>China</made>
</product_extension>
```

Declaration Complex Elements

Occurrence prescription:

```
<xsd:element name="product_sold_in">
 <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="country" type="xsd:string"
            minOccurs="1" maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Declaration of Elements with Mixed Content

Using the mixed attribute of the `complexType` schema element.

```
<xsd:element name="note">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="emp" type="xsd:string"
          minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

An instance:

```
<note>The exam will be <emp>on January 22th</emp>,
or it could also be <emp>on the 29th</emp>.
</note>
```

Outline

Attribute Declarations

- Attributes are declared similarly to child elements, but they always have simple content.
- Attributes can be defined only inside complex element declarations.
- The attribute declarations always come after the child element declarations.
- In an attribute definition the name and type attributes will prescribe the corresponding qualities of the attribute being declared.
- The use attribute controls requirements for the declared attribute and the value attribute can set default or fixed values for it.

Constraining Attributes with Use

- **required** The declared attribute must be explicitly defined every time its element occurs in the instance of the schema.
- **default** The declared attribute is optional and if not given the value attribute specifies its value.
- **optional** The declared attribute is optional, no default is given.
- **fixed** The declared attribute is optional, but if it appears, its value has to be the one given in the value attribute.
- **prohibited** The declared attribute must not appear in any occurrence of the corresponding element and it has no predefined value.

Example

- We want to create a schema for the following kind of instances:

```
<product>
    <id>231069948</id>
    <price currency="EUR"> 3.49</price>
</product>
```

  where the structure is fixed and the presence of the attribute of the price element is required.
- Unfortunately the definition is a bit lengthy.

Example

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" type="xsd:unsignedInt"/>
      <xsd:element name="price">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:float">
              <xsd:attribute name="currency"
                  type="xsd:string" use="required"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Using References

- Global declarations are the children of the root element in a schema.
- In local declarations we can refer to global ones via the ref attribute to use the global declaration to define the referencing element.
- For instance, assuming that the lengthy definition of the price element is available as a global declaration, `<xsd:element ref="price"/>` would suffice inside the declaration of the product element.

Outline

Type Declarations

- XML schema provides more than forty built-in data types and the possibility to define new types from the already existing ones.
- The previous examples already contained anonymous type definitions which we used only at the place of definitions:
  `<xsd:complexType> ... </xsd:complexType>`
- We can create named types by setting its name attribute.
  ```
  <xsd:complexType name="mytype"> ...
  </xsd:complexType>
  ```
- Then inside the scope of the above type definition we can use the named type in further element declarations.
  ```
  <xsd:element name="myelement"
  type="mytype"/>
  ```

Example

```
<xsd:complexType name="price_type">
  <xsd:simpleContent>
    <xsd:extension base="xsd:float">
      <xsd:attribute name="currency"
        type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" type="xsd:unsignedInt"/>
      <xsd:element name="price" type="price_type"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Type Restriction

Built-in data types provide one or more aspects through which one can create new types by restriction.

Example

- Strings that match a given regular expression.
- The new type contains strings format nnn-nn-nnnn where in place of an n any digit can stand (e.g. 323-90-0982).

```
<xsd:simpleType name="serial_number">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-\d{2}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Type Extension, Uniqueness, Keys and Keyrefs

For complex data types one can also use extension to add new elements and attributes to an already existing type.

We can prescribe that elements or attributes have unique values within a given scope in an instance of the XML schema.

Referential integrity can be enforced via the key and keyref schema elements. They are used similarly to the unique element.

Summary

- Schema declaration and referencing
- Declaration of elements
  - Simple elements
  - Complex elements (sequence, all, choice, . . .)
  - Complex elements with mixed content
- Declaration of attributes (required, default, optional, fixed, prohibited)
- Type declarations (restrictions, extensions)
- Uniqueness, Keys and Keyrefs