HELSINKI UNIVERSITY OF TECHNOLOGY 04.12.2000
Telecommunications Software and Multimedia Laboratory
Tik-111.590 Research Seminar on Digital Media
Fall 2000: XML

XML Schema

Jarkko Ansamaa

42840S

XML Schema

Jarkko Ansamaa
Helsinki University Of Technology

jansamaa@cc.hut.fi

Abstract

This paper describes the concepts of XML Schema or Xschema as mentioned in some resources. The Extensible Markup Language (XML) is a format for structured data as specified by The World Wide Web Consortium (W3C). XML Schema is a model for describing the structure of information in an XML document. Schemas specify the possible arrangement of tags and text (structure), possible values of the information elements (data-types) and allow support for namespaces, attribute grouping and inheritance. If the structure of a given XML document confirms to given schema it is said to be valid in context of that particular schema. Applications exchanging a class of XML documents use Schema as an agreement on common vocabulary; the validity of the documents is easy to verify using automated tools. This paper lists a short introduction to XML Schema concept, gives a general view on main characteristics and shows some technical definitions of the Schema language with a few examples. This paper was written for the Helsinki University of Technology course "Tik-111.590, Research Seminar on Digital Media".

1 INTRODUCTION

XML is a mark-up language that is used for describing structured data. The basis of the logical structure of a XML document consists of document declaration followed by elements and their attributes. Each element has a name and they are declared by start and end tag. Element's content resides between the tags and may consist of other elements, data or be empty. Each element may have attributes that consist of name and value. XML enables users to introduce elements and attributes, their names and their relations in the document. To be XML, documents only have to conform to the XML syntax.

The XML document's meaning has still be understood. An arbitrary structure with arbitrary named elements and attributes does not really tell what the document is all about. For this the XML document declaration may point or contain mark-up declaration that provides the grammar for a class of document. This grammar specifies the structure of the document by defining the attributes and elements and their hierarchy and granularity. It enables the common vocabulary to be agreed on and to be used for specialised domains. One could design new mark-up languages for mathematics,

astronomy or just for simple forms or for any purpose imagined. The grammar may then be shared with others in that domain. By agreeing to that, a group of people has accepted a set of rules about document vocabulary and structure.

All the above leads to two basic qualities of XML documents. Document can be *well-formed* and *valid*. Well-formed document is a document where the XML syntax is correct. Technically it also means that a XML parser is capable or processing the XML document. Valid XML document is a well-formed document that structure of information (tags, elements, attributes) conforms to the defined grammar. Documents that conform the defined grammar are said to be *instance documents* of that grammar.

The concept of valid XML document has many other advantages. Validation ensures that the information is structured in a way that is sensible for applications that use it. Before applications use the information on a XML document, a XML parser evaluates the document. Most of the parsers also validate XML documents automatically. This is a considerable function since most applications use a lot of processor time just for checking the validity of their input data or coping the consequences of corrupted data. For example company A receiving XML data from company B may automatically check the data before entering it into their systems avoiding the corruption of their databases. Applications producing XML documents can also use validation so that they check the data before sending it further. XML editing applications can use these as frameworks, letting users create documents that conform to the grammar.

The original way for describing the grammar of a XML document is called *Document Type Declaration* (DTD). It was defined by XML 1.0 recommendation.

2 DOCUMENT TYPE DECLARATION

Document Type Declaration (DTD) declares the grammar for a class of XML documents. DTD can point to an external or internal subset. Internal subset means that XML document embeds the DTD. External subset means that XML document points to an external DTD. XML document may have both the internal and external subset where the internal is processed before the external.

DTD has its own syntax. Syntax defines elements and attributes. Elements may consist of child elements and of character data. Elements may be optional, appear zero or multiple or one or multiple times, they may follow each other or be alternatives to each other. Attributes may for example be unique, have list of accepted values or default value and so on.

The following example defines an address book that consists of zero or more persons. Each person has either nickname or full name. Full name consists of first name, middle name and last name where middle name is optional. Each person also has one or more email addresses and phone numbers. An attribute is used to identify whether the person is a friend or work associate. Friend is the default value. As an external document type definition this could look like:

```
<!ELEMENT addressbook (person)* >
<!ELEMENT person ((nickname | fullname), email+, phonenumber+)>
```

```
<!ATTLIST person
                category ( friend |work ) “friend” >
<!ELEMENT fullname (givenname, middlename?, familyname)>
<!ELEMENT givenname (#PCDATA)>
<!ELEMENT middlename (#PCDATA)>
<!ELEMENT familyname (#PCDATA)>
<!ELEMENT nickname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phonenumber (#PCDATA)>
```

If this document type definition were named addressbook.dtd the following XML document that includes the information of two persons would be a valid address book:

```
<?xml version="1.0"?>
<!DOCTYPE addressbook SYSTEM "book.dtd" >
<addressbook>
  <person category=”friend”>
    <nickname>Tommy</nickname>
    <email>tom.jones@home.com</email>
    <email>tom.jones@work.com</email>
    <phonenumber>0501234567</phonenumber>
  </person>
  <person category=”work”>
    <givenname>Susan</givenname>
    <familyname>Smith</familyname>
    <email>susan.smith@company.com</email>
    <phonenumber>0507654321</phonenumber>
  </person>
</addressbook>
```

With this simple example it is easy to point out some limitations on DTD. Firstly the elements in the document must be exactly in the defined order. Secondly lets assume that we would like to restrict the element `phonenumber` only to contain digits and the element `email` to have the @ character surrounded by other characters. DTD lacks the concept of strong data typing so this is impossible. Thirdly to restrict that a person may have for example a maximum of five email addresses is not possible.

The combined list of most of the limitations include:

- DTD has different syntax than XML which is inconsistent
- DTD has very limited data type support
- Expressing complex structures is limited
- No support for XML namespaces

Due the limitations the World Wide Web Consortium (W3C) started to work on new model for describing the structure of XML documents. This work resulted to the specification of XML Schema.

3 XML SCHEMA

3.1 General

Like DTD, XML schema describes a model for a whole class of documents. The model describes the possible arrangement of tags and text in a valid document. A

schema might also be viewed as an agreement on a common vocabulary for a particular application that involves exchanging documents. The XML Schema specification is written in three parts:

- XML Schema Part 0: Primer is an introduction to XML Schemas with examples.
- XML Schema Part 1: Structures specifies the XML Schema definition language for describing the structure and constraining the contents of XML documents.
- XML Schema Part 2: Datatypes specifies a system of data categories for XML.

The first Working Draft of the specification was published in May 1999 at the end of October 2000 the XML Schema language was given a Candidate Recommendation status.

An instance document that is valid against schema is *schema-valid.* Schema processor(s) performs the validation. XML Schema itself is well-formed XML. Schemas are valid to Schema DTD.

3.2 New Features and Improvements

XML Schema offers a range of improvements compared to DTD:

- The syntax is written in XML allowing the use of XML tools like XML editors
- Many in-built datatypes like booleans, numbers, different types of dates and times, URIs, integers, decimal numbers, real numbers, intervals of time, etc.
- User may define almost any kind of own datatypes
- Inheritance allows types to be derived from other types by extension or restriction
- Element contents may be unique within the whole document or just region
- Set expression allows element's child elements occur in any order
- Support for XML namespaces
- Regular expressions
- Documentation mechanism

In overall, XML Schema is very powerful in XML document modelling.

3.3 Basic features of XML Schema

XML Schema is written in XML. The whole schema definition is inside a `schema` element, which is associated with XML Schema namespace. This namespace identifies element and attribute names such as built-in data types to belong to the vocabulary of the XML Schema language. `Schema` element has a number of child elements. The most common child elements are `element`, `complexType` and `simpleType` that determine the appearance of elements and their content in instance documents. Attributes in instance documents are declared using `attribute` element.

Let's use the address book example introduced earlier in chapter 2. The Schema for an address book could look like the following:

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

```
<!--Generated by XML Authority. Conforms to w3c
http://www.w3.org/1999/XMLSchema-->
<schema xmlns="http://www.w3.org/1999/XMLSchema">
  <element name="addressbook">
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element ref="person"/>
      </sequence>
    </complexType>
  </element>

  <element name="person">
    <complexType>
      <sequence>
        <choice>
          <element ref="nickname"/>
          <element ref="fullname"/>
        </choice>
        <element ref="email" minOccurs="1" maxOccurs="unbounded"/>
        <element ref="phonenumber" minOccurs="1"
maxOccurs="unbounded"/>
      </sequence>
      <attribute name="category" use="default" value="friend">
        <simpleType base="ENUMERATION">
          <enumeration value="friend"/>
          <enumeration value="work"/>
        </simpleType>
      </attribute>
    </complexType>
  </element>

  <element name="fullname">
    <complexType>
      <sequence>
        <element ref="givenname"/>
        <element ref="middlename" minOccurs="0" maxOccurs="1"/>
        <element ref="familyname"/>
      </sequence>
    </complexType>
  </element>

  <element name="givenname" type="string"/>
  <element name="middlename" type="string"/>
  <element name="familyname" type="string"/>
  <element name="email" type="string"/>
  <element name="phonenumber" type="string"/>
  <element name="nickname" type="string"/>
</schema>
```

The schema above declares one structure level at the time. It includes references to elements on the next structure level. The final elements are all declared global which means that they appear the on next level from the `schema` element. This is one way of writing a schema.

3.3.1 Elements

Schema element `element` declares an element in the document instance. For example

```
<element name="middlename" type="string"/>
```

declares an element `middlename` which content is of built-in type `string`. Since

`middlename` is now defined there it can be referenced:

```
<element ref="middlename" minOccurs="0" maxOccurs="1"/>
```

By reference we avoid defining a repeatable item again. In general the `ref` attribute must point to global element which is declared as a child of `schema` element.

By default, elements occur once in the instance document. Using `minOccurs` and `maxOccurs` attributes we can restrict the occurrence of an element to any combination. In the above `middlename` may occur once or not at all.

In our addressbook example above the referencing has no advantage since the elements are used only once. Now we can reduce our definition by taking the references out.

```
<?xml version ="1.0"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema">
  <element name="addressbook">
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element name="person">
          <complexType>
            <sequence>
              <choice>
                <element name="nickname" type="string"/>
                <element name="fullname">
                  <complexType>
                    <sequence>
                      <element name="givenname" type="string"/>
                      <element name="middlename" type="string"
minOccurs="0" maxOccurs="1"/>
                      <element name="familyname" type="string"/>
                    </sequence>
                  </complexType>
                </element>
              </choice>
              <element name="email" type="string" minOccurs="1"
maxOccurs="unbounded"/>
              <element name="phonenumber" type="string" minOccurs="1"
maxOccurs="unbounded"/>
            </sequence>
            <attribute name="category" use="default" value="friend">
              <simpleType base="ENUMERATION">
                <enumeration value="friend"/>
                <enumeration value="work"/>
              </simpleType>
            </attribute>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

3.3.2 Attributes

Attributes are defined as `attribute` elements. In our example

```
<attribute name="category" use="default" value="friend">
  <simpleType base="ENUMERATION">
    <enumeration value="friend"/>
    <enumeration value="work"/>
  </simpleType>
</attribute>
```

declares an attribute name `category` which default value is `friend` and possible values are `friend` and `work`. There are also means of declaring an optional attribute and fix its value.

3.3.3 Built-in and User Definable Simple Types

Simple types are either built-in types or user definable types. These can be used in attributes as well as elements. In our example a user definable simple type is declared by `simpleType` element for an attribute:

```
<attribute name="category" use="default" value="friend">
  <simpleType base="ENUMERATION">
    <enumeration value="friend"/>
    <enumeration value="work"/>
  </simpleType>
</attribute>
```

Simple types can also be declared by inheritance where a built-in type is used as a base and then some constraint is applied to it. A simple example of defining a number those minimum is 10000 and maximum 99999:

```
<simpleType name="myInteger">
  <restriction base="integer">
    <minInclusive value="10000"/>
    <maxInclusive value="99999"/>
  </restriction>
</simpleType>
```

Some examples of built-in types are: boolean, string, integer, positiveInteger, negativeInteger, decimal, double, date, time, year, century.

3.3.4 Complex Types

Complex types typically contain a set of element declarations, element references, and attribute declarations. They can be used for defining a reoccurring complex structure that is often referenced. Example:

```
<element name="PostDelivery" >
  <complexType>
    <element name="sender"      type="Address" />
    <element name="reciever"    type="Address" />
  </complexType>
</element>

<complexType name="Address" >
  <sequence>
    <element name="name"      type="string" />
    <element name="street"    type="string" />
    <element name="city"      type="string" />
    <element name="postcode"  type="string" />
```

```
  </sequence>
  <attribute name="customerCode" type="integer"
</complexType>
```

3.3.5 User Definable Datatypes and Regular Expressions

Let's set some new constraints to our address book example. We want to define that `email` is a string where @ character is surrounded by other characters and that `phonenumber` consist of at least six digits. For this we can use regular expressions and define new datatypes:

```
<simpleType name="emailAddress">
  <restriction base="string">
    <pattern value=".+@.+"/>
  </restriction>
</simpleType>

<simpleType name="phonenumber">
  <restriction base="string">
    <pattern value="[0-9]{6,}"/>
  </restriction>
</simpleType>
```

Then we can simply use these new types using `type` attribute:

```
<element name="phonenumber" type="phonenumber"/>
```

3.3.6 Other

The main features in XML Schema were shortly introduced with a simple example above. Still there is a lot more ways of expressing the document instances model. XML Schema has 36 different element names which all are structure or datatype related. Apart from that there are 31 different attributes which are used in the elements. Schemas are not only a comprehensive modelling tool but also a rich language where the same kind of results may be achieved in different ways.

3.4 XML Namespaces and Schemas

The purpose of namespaces is to guarantee unique names for elements and attributes that may be declared in various separate document models like DTDs and schemas. A namespace is a collection of names that are identified by URI reference. XML Namespace technology is used in various cases with XML Schema. The most important are declaring target namespaces and dividing Schemas in multiple documents.

3.4.1 Target Namespaces

A schema can be viewed as a collection of type definitions and element declarations whose names belong to a particular namespace called a target namespace. All the schemas use names XML Schema namespace, which is often called "schema for schemas". These include the keywords like `simpleType`, `element` or `integer`. Let's put our definition for an email address to a schema and create a namespace for it:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
        targetNamespace="http://www.example.org/Email">

  <simpleType name="email">
    <restriction base="string">
      <pattern value=".+@.+"/>
    </restriction>
  </simpleType>
</schema>
```

Now the target namespace is declared using `targetNamespace` attribute and an instance document may reference it using the namespace technology.

3.4.2 Dividing Schema in Multiple Documents

A schema may be divided in several documents. Let's assume the email address type is defined as in previous chapter in file `email.xsd`. Now we create a simple schema for a list of email addresses that uses the address type already defined in another document:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:addr="http://www.example.org/Email"
        targetNamespace="http://www.example.org/Email">

  <include schemaLocation="http://www.example.org/schemas/email.xsd"/>

  <element name="emailAddressList">
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element name="email" type="addr:email"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

In the above, the `include` element imported the contents of the `email.xsd` to the schema from address `http://www.example.org/schemas/email.xsd`. The reference `addr` to namespace `http://www.example.org/Email` was used when defining the `type` of element `email`. This is needed for implying that the type `email` is not part of the default namespace but a part of the namespace we created in the previous chapter. If `include` is used the all of the schemas must have the same target namespace. If schemas have different namespaces the importing may be performed using `import` element.

3.5 Processing an Instance Document against a Schema

An instance document may be processed against a schema to verify whether the instance conforms the rules specified in the schema. This is performed by a schema processor. The processing actually does two things. It checks instance against the schema rules to see if the instance is *schema-valid*. It may also add supplementary information to the instance like types and default values.

The author of an instance document may provide the location of the schema by using XML Schema instance namespace and `schemaLocation` attribute. Usually a schema has a target namespace. The `schemaLocation` attribute has two values. The first is the

namespace of the schema and the second the locations of the schema document. If the schema does not have a target namespace attribute `noNamespaceSchemaLocation` is used.

In the following example we have a schema describing a list of email addresses:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
        targetNamespace="http://www.example.org/Email">

  <element name="emailAddressList">
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element name="email">
          <simpleType>
            <restriction base="string">
              <pattern value=".+@.+"/>
            </restriction>
          </simpleType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

The following XML document points to the schema above thus is valid against it:

```
<?xml version="1.0"?>
<emailAddressList
    xmlns="http://www.example.org/Email"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"

    xsi:schemaLocation="http://www.example.org/Email"
                       "http://www.example.org/schemas/email.xsd">

  <email>joe.dalton@example.com</email>
  <email>jack.dalton@example.com</email>
  <email>william.dalton@example.com</email>
  <email>awerell.dalton@example.com</email>
  <email>lucky.luke@example.com</email>
</emailAddressList>
```

Note however the schema processor is not obliged to use a schema that is provided in the instance document. In fact the processor is free to use other schemas obtained by any suitable means, or to use no schema at all. For example, an HTML editor may have a built-in HTML schema.

4 CONCLUSIONS

XML Schema is a powerful language for describing the vocabulary of a XML document. The language has a number of features that can be used for declaring almost any kind of XML data structures and datatypes. While the advantaged features require a good knowledge of the XML technologies like namespaces and the language itself, the basics are quite easy to learn and deploy.

The XML Schema technology is already an important part of XML. The need for

sharing domain specific vocabulary and the range of applications sharing XML data itself requires a common description language. Common vocabulary helps communities to have an unique agreement on the subject in question. Applications exchanging XML documents do not have to worry about receiving or sending invalid data. Schema-validity works as a “spell checker” for users creating XML documents. Because of these it is foreseen that the XML Schema will have an even more essential role in the future.

REFERENCES

[1] Goossens, M. 2000. XML, a new start for the Web. Academic Training, May 2000. URL: *http://goossens.home.cern.ch/goossens/xml2000.pdf*

[2] Fallside, D. C. (Editor), 2000, XML Schema Part 0: Primer, W3C Candidate Recommendation, October 2000. URL: *http://www.w3.org/TR/xmlschema-0/*

[3] Beech, D. Maloney, M. Mendelsohn N. Thompson, H. S. (Editors), 2000, XML Schema Part 1: Structures, W3C Candidate Recommendation, October 2000. URL: *http://www.w3.org/TR/xmlschema-1/*

[4] Paul V. Biron, P. Malhotra, A. (Editors), 2000, XML Schema Part 2: Datatypes, W3C Candidate Recommendation, October 2000.URL: *http://www.w3.org/TR/xmlschema-2/*