XML Schema

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring

Overview

- Introduction
  - Background
  - Simplification
  - References
- Defining a document model
  - Namespaces
  - Schema proper
  - Datatypes
    - Simple
    - Complex

Background

- There is, as yet no clear and simple explanation of the various “Schema” efforts. These include:
  - Various Structural Schema to replace DTDs
  - RDF Schema
  - Datatypes
  - Namespaces
- The RDF specification is about the clearest:

  “RDF Schemas might be contrasted with XML Schemas … an XML Schema gives specific constraints on the structure of an XML document … an RDF Schema provides information about the interpretation of the statements given in an RDF data model….RDF uses XML for its interchange encoding … XML [datatypes] should be the foundation”

Simplification

- The XML Schema efforts extend the SGML Document Type Definition (DTD) functionality
  - The DTD was designed as a means for defining the structural properties of a class of documents.
- Schema provide an alternative form for defining a documents structure. They also:
  - Allow for more precise control of the content in a document – via the datatype extensions
  - Allow for more than one definition to be applied within a document via the namespaces extension
  - Allow for simplification of parsing engines by defining schema in the form of XML documents

Definitions

- “Document instance” is a single document, in our case made up of elements that are hierarchically nested and encapsulated by begin and end tags.
- “Document Type Definition” is a description of a class of document instances that can be used to validate a given document.
- “Schema” is a description of a class of document instances, and is itself a document instance, that can be used to validate a given document.
- “Namespace” is a specification or schema. In the case of a specification, the specification defines the document class.

Complications

- Schema work has been complicated in that it is directed at a number of different goals.
- It is further complicated by the fact that there have been competing specifications
  - The Document Content Description (DCD) effort was one proposal by Textuality, Microsoft, and IBM
    - http://www.w3.org/TR/NOTE-dcd
  - Document Definition Markup Language (DDML) effort was another proposal by the Europeans
    - http://www.w3.org/TR/NOTE-ddml
- The current slides reflect the W3C specification

Selected References

- Microsoft developed an early specification for IE5.
  - Many books reference these specifications and some systems such as IE5 will probably continue to use them for a while
- This presentation looks to the W3C specification:
  - The requirements for XML Schema were set out in:
    - http://www.w3.org/TR/NOTE-xml-schema-req
  - A primer on XML Schema may be found at:
    - http://www.w3.org/TR/xmlschema-0/
  - The structural schema specification may be found at:
    - http://www.w3.org/TR/xmlschema-1/
  - The datatype specification may be found at:
    - http://www.w3.org/TR/xmlschema-2/
  - The RDF specification may be found at:
    - http://www.w3.org/TR/2000/CR-rdf-schema-20000327/

Review of XML Parsing

- What is happening when an XML parser is invoked can be confusing.
- Parsers can be validating or non-validating. All parsers check for to make sure XML documents are well formed. This does not require a DTD or schema. The document instance is simply checked to make sure it follows the syntax rules
- Validation of a document requires that the document instance have an associated DTD or schema against which it can be checked.

Schema Benefits

- Schema are more powerful than DTDs in that they:
  - Allow for inheritance (namespaces)
  - Allow modular construction
  - Provide a mechanism to avoid name collisions (namespaces)
  - Allow content control (datatypes)
  - Allow for more documentation in content description (schema)
  - Allow for simpler parsing (schema)

Namespaces

- Namespaces provide the mechanism by which:
  - Element naming collisions can be avoided – defining a scope for elements
  - Element naming can be modularized
  - Document definitions can employ multiple inheritance
- A namespace is a schema that defines a set of elements
- The attribute “xmlns” is reserved as the means by which this association is made
- Namespaces are defined in:

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

Namespace Association(1)

- A simple association between an element and a namespace would appear as shown below:

  <html xmlns="http://www.w3.org/TR/WD-HTML40">

- This says that the element “html”, and the allowable subelements are defined by the named attribute. (In this case, “WD-HTML40” is actually a specification and not a schema)
- The namespaces are extensible and can be combined a number of ways. As used above, the namespace applies to all children or subelements of the element “html”
- If the namespace association is “unqualified”, as the example above is, subelements of the element would appear as in previous versions of XML

Namespace Association(2)

- Formally “xmlns” is the DefaultAttName.
- The namespace specification also allows for a PrefixedAttName which is “xmlns” followed by a “:” followed by an “NCName”.
- An NCName begins with a letter or underscore and that has a few other restrictions in terms of allowed symbols in the name.
- The NCName is used as a prefix for elements from that namespace – including the one for which it is an attribute:

```
<mbs:email xmlns:mbs="http://www.pitt.edu/~spring/m_schema.xsd">
```

Modularity, Inheritance and Collision Avoidance

- Multiple namespaces may be associated with a given element and its sub elements:

```
<html:html xmlns:html="href2"
xmlns:spring="href1" xmlns:math="href3">
```

- allows the children of html to include elements from all of these namespaces

```
<html:head><<html:title>The title of the doc</html:head>
<html:body>
<math:equation>z=x+y </math:equation>
<spring:equation>some things are distinguished </spring:equation>
</html:body> </html:html>
```

- Note that there is no conflict between the “equation” elements from the two namespaces

Scope of a Namespace

- Namespaces can also be scoped within a document
- Given a top level association, subelements may be explicitly defined as belonging to another namespace:

```
<email xmlns="http://www.pitt.edu/~spring/m_schema.xsd">
<to>Joe</to>
<from>Mary</from>
<body>
<eq:eqn xmlns:eq ="http://www.pitt.edu/~spring/e_schema.xsd">
<eq:relation>some element</eq:relation></eq:eqn>
<h1>some text</h1>
</body></email>
```

Namespaces in Context

- Namespaces define both general schemas and XML specific schemas such as:
  - XML Link Language (XLL) capabilities
  - XML Style Language (XSL) capabilities
  - XSL Transformation (XSLT) capabilities
  - XML Pointer capabilites
- Parsers are based on specifications which imply specific schema. For example, XSLT parsers.
  - IE5 accepts XML documents that begin:

    <?xml version="1.0"?>

    <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">-->
  - While James Clark's XT accepts documents that begin:

    <?xml version="1.0"?>

    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

A Simple Schema

- A schema is an XML document where the top level element is “schema” and the associated namesapce is that for XML schema. Thus:

```
<schema xmlns=http://www.w3.org/2001/XMLSchema>
    <element name= “mynote”>
      <complexType >
          <sequence>
                  <element name = “To”  type = “string”/>
                  <element name = “From”  type = “string”/>
                  <element name = “Note”  type = “string”/>
          </sequence>
          <attribute name = “Date”  type = “date”/>
      </complexType>
    </element>
</schema>
```

The Schema Referenced

- Assuming the schema defined on the previous slide is located in the current directory in the file mbsnote.xsd, the following now allows validation:

```
<mynote xmlns = “mbsnote.xsd”
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="mbsnote.xsd
                http://www.pitt.edu/~spring/mynote.xsd“
    Date = “2001-05-27”>
<To>Jonathan</To>
<From>Patrick</From>
<Note>Here is a little message for you</Note>
</mynote>
```

A Schema using Type

- The “complexType” element defines structure.
- The schema namespace is qualified and the target and default namespace are the same

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns= "http://localhost/"
            targetNamespace=" http://localhost/">
  <xsd: element name= "mynote" type = "mynotetype"/>
  <xsd:complexType name= "mynotetype">
    <xsd:sequence>
      <xsd:element name = "To"  type = "xsd:string"/>
      <xsd:element name = "From"  type = "xsd:string"/>
      <xsd:element name = "Note"  type = "xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name = "Date"  type = "xsd:date"/>
  </xsd:complexType>
</xsd:schema>
```

Just for clarity

- Note the reference to my node type if the target is qualified and the schema namespace is not.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:mn= "http:///localhost/"
                targetNamespace=" http:///localhost/">
   <element name= "mynote" type = "mn:mynotetype"/>
   <complexType name= "mynotetype">
     <sequence>
                <element name = "To"  type = "string"/>
                <element name = "From"  type = "string"/>
                <element name = "Note"  type = "string"/>
     </sequence>
     <attribute name = "Date"  type = "date"/>
   </complexType>
</schema>
```

A Schema with Occurrence(1)

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
            xmlns= “http://localhosrt/”
            targetNamespace=" http://localhos/”>
  <xsd: element name= “mydoc” type = “mydoctype”/>
  <xsd:complexType name= “mydoctype”>
    <xsd:sequence>
            <xsd:element name = “FrontMatter”  type = “FMT”/>
            <xsd:element name = “Body”  type = “BODYT”/>
            <xsd:element name = “EndMatter”  type = “EMT”/>
    </xsd:sequence>
    <xsd:attribute name = “Editor”  type = “xsd:string”/>
    <xsd:attribute name = “Status”  type = “xsd:string”/>
    <xsd:attribute name = “ISBN”  type = “xsd:string”/>
  </xsd:complexType>
```

A Schema with Occurrence(2)

```
<xsd:complexType name= “FMT”>
  <xsd:sequence>
            <xsd:element name = “Title”  type = “xsd:string”/>
            <xsd:element name = “Author”  type = “xsd:string”
                        maxOccurs=“unbounded”/>
            <xsd:element name = “Pubdate”  type = “xsd:string”/>
            <xsd:element name = “Acknowledge”  type = “xsd:string”
                        minOccurs=“0”/>
            <xsd:element name = “Dedication”  type = “xsd:string”
                        minOccurs=“0”/>
            <xsd:element name = “Preface”  type = “xsd:string”
                        minOccurs=“0”/>
  </xsd:sequence>
</xsd:complexType>
```

A Schema with Occurrence(3)

```
<xsd:complexType name= “BODYT”>
  <xsd:choice>
        <xsd:element name = “Part”  type = “PARTT”
                minOccurs=“1” maxOccurs=“10”/>
        <xsd:element name = “Chapter”  type = “CHAPTERT”
                minOccurs=“1” maxOccurs=“unbounded”/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name= “PARTT”>
  <xsd:element name = “Chapter”  type = “CHAPTERT”
        minOccurs=“1” maxOccurs=“unbounded”/>
</xsd:complexType>
....
</xsd:schema>
```

Choices, Sequences, and Sets

- The example above shows how minOccurs and maxOccurs can be used to control the elements in an instance
- Sequence and choice elements can also be used and nested in a variety of ways.
- Schema also allows, with some restrictions, and “all” element that says all the elements named must appear, but they may appear in any order. The restrictions are:
  - It must occur at the top level of the schema
  - The occurrence indicators can only be “0” or “1”
  - The all group may not be nested in a sequence or choice

Global Declarations & References

- Normally, the form for a schema is:
  - Schema
    - ElementTop (referencing TypeX)
    - TypeX (defining subelements of ElementTop)
- Other elements can be defined at the top level:
  - Schema
    - ElementTop (referencing TypeX)
    - ElementA
    - TypeX
      - ElementC (referencing ElementA)
- Global elements must be typed. They may not use minOccurs, maxOccurs or the “ref” attributes.

Example of a Global Element and Reference

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
            xmlns= “http://localhost/”
            targetNamespace=" http://localhost/”>
  <xsd:element name= “mydoc” type = “mydoctype”/>
  <xsd:element name = “comment” type = “xsd:string”/>
  <xsd:complexType name= “mydoctype”>
    <xsd:sequence>
            <xsd:element ref=“comment” minOccurs=“0”/>
            <xsd:element name = “FrontMatter”  type = “FMT”/>
            <xsd:element name = “Body” type =“BODYT”/>
            <xsd:element name = “EndMatter”  type = “EMT”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Anonymous Types(1)

- In our first example, we used “anonymous types”

```
<element name= “mynote”>
    <complexType >
        <sequence>
            <element name = “To”  type = “string”/>
            <element name = “From”  type = “string”/>
            <element name = “Note”  type = “string”/>
        </sequence>
        <attribute name = “Date”  type = “date”/>
    </complexType>
</element>
```

Anonymous Types(2)

- Note that simple types may not have attributes.
- To create a “simple type”* that allows attributes, derive it from a simple type by extension.

```
<element name= “myinteger”>
  <complexType >
    <simpleContent>
        <extension  base = “integer”>
            <attribute name = “coin”  type = “string”/>
        </extension>
    </simpleContent>
  </complexType>
</element>
```

- This now allows:

```
<myinteger coin = “dime”>27</myinteger>
```

---

* The derived type is by definition complex.

Mixed Content Models

- To this point, all elements have been made up of:
  - elements only
  - data only
- An element that contains both is said to have a mixed content model.
- Schema allow mixed content via a complex type.
- Unlike XML 1.0 which allowed subelements and data to be randomly intermingled, the schema specification says the order of the subelements must be as specified by the complex type. An example is shown on the next slide

Mixed Content Declaration

```
<xsd:element name = “example”>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = “Title” type = string/>
      <xsd:element name = “Explan”>
        <xsd:complexType mixed=“true”>
          <xsd:sequence>
              <xsd:element name = “emph”  type = “xsd:string”/>
              <xsd:element name = “warning”  type = “xsd:string”/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:attribute name = “Num” type = “xsd:integer”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Mixed Content Example

- Given the declaration on the previous slide, the following instance is now valid:

```
<example Num = “23”>
<Title>Here is another example</Title>
<Explan>
Here is some <emph> mixed content</emph> and a
<warning>caution about being careful when using
and declaring these.</warning>
</Explan>
</example>
```

Compound Schema(1)

- A schema may include references to other schema

```
<element name= “mynote”>
  <complexType >
    <sequence>
      <element name = “to” type=“string”/>
      <element name = “from” type=“string”/>
      <element name = “htmlbody”>
        <complexType >
          <sequence>
                <any namespace =http://www.w3.org/1999/xhtml
                                minOccurs=“1” maxOccurs=“unbounded”
                                processingContents=“skip”/>
          </sequence>
        </complexType>
      </element>
    </sequence>
    <attribute name = “Date”  type = “date”/>
  </complexType>
</element>
```

Compound Schema(2)

- A schema may also explicitly include other schema using the include element

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:mn= "http://localhost/"
        targetNamespace=" http://localhost/">
<include schemaLocation=
        http://someloaction/schemas/definitions.xsd/>

   <element name= "mynote" type = "mn:mynotetype"/>
   ........
</schema>
```

The Schema Annotation Element

- The schema provides for an annotation element that allows human consumable and machine consumable information to be provided:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:mn= "http:///localhost/"
                targetNamespace=" http://localhost/">
<annotation>
<documentation>This belongs to me</documentation>
<appinfo>here is some processing instruction</appinfo>
</annotation>
    ……..
</schema>
```

Content Data Control under DTD's

- XML (SGML) was designed to structure content and excluded control of data apart from syntax checking. An element could be checked for other structures, but the structure of the text could not be controlled.
- Element content was defined as:
  - #PCDATA –text searched for elements & entities
  - #RDATA – text searched for entities but not elements
  - #CDATA – text not searched
  - #NDATA – non-character data

Attribute Value Control under DTDs

- Attribute values could be somewhat more finely controlled, most notably as enumerated values, but the control was still generally limited to very broad classes
- Some of the built in datatypes for XML for attributes included
  - ID
  - IDREF
  - NMTOKEN
  - NOTATION

DataTypes under Schema

- If XML was to be used for e-commerce, and as a wrapper for data interchange, more control was needed
- XML schema are designed to allow detailed control of element content and attribute values.
- All of the DTD controls (PCDATA , RDATA, and CDATA are replaced with the string type
- Schemas have a total of 44 built in primitive(19) and derived(25) data types versus the 10 under DTD's (most of which were for attribute values
- Schemas also provide powerful tools for defining additional user defined derived data types

PrimitiveDataTypes

- String “Hello”
- Boolean {true, false}
- Decimal 9.4
- Float 13.78E1
- double 13.78E1
- duration P2Y2M4Detc
- dateTime CCYY-MM-DDhh-mm-ss
- Time hh:mm:ss.sss
- Date CCYY-MM-DD
- gYearMonth CCYY-MM
- gYear CCYY
- gMonthDay MM-DD
- gDay DD
- gMonth MM
- hexBinary 00A8
- base64Binary dPm6
- anyURI http://xyz.com
- Qname a qualified namespace name
- NOTATION a NOTATION from XML

Derived DataTypes

- normalizedString A string with tabs, line feeds, and carriage returns converted to spaces
- Token A normalized string with consecutive spaces reduced to one
- Language one of the established language codes
- IDREFS Attributes only, same as previously defined
- ENTITIES Attributes only, same as previously defined
- NMTOKEN Attributes only, same as previously defined
- NMTOKENS Attributes only, same as previously defined
- Name an XMl name
- NCName a namespace name
- ID Attributes only, same as previously defined
- IDREF Attributes only, same as previously defined
- ENTITY Attributes only, same as previously defined

Derived DataTypes

- Integer decimal with no fractional part
- nonPositiveInteger 0 to negative infinity
- negativeInteger –1 to negative infinit
- Long the integers from–9223372036854775808 to 9223372036854775807
- int the integers from–2147483648 to 2147483647
- Short the integers from -32768 to 32767
- Byte the integers from –128 to 127
- nonNegativeInteger integers greater than or equal to 0
- unsignedLong non negative integer less than 18446744073709551615
- unsignedInt unsignedLong less than 4294967295
- unsignedShort unsignedInt less than 65535
- unsignedByte unsignedShort less than 255
- positiveInteger nonNegativeInteger gretaer than or equal to 1

User Derived Data Types

- While the built-in types, both primitive and derived provide a significant increase in data control, schema allow further refinement by defining various restrictions.
- Restrictions are placed on facets which vary for the various datatypes.
- In addition, the derived types may be defined as lists, unions, or atomic constructions.
- Additional information about the datatype and its ability to be further extended or refined may also be included.

Constraining Facets

- Examples of Constraining facets for a couple primitive types would include the following:

Type	Facets
string	length, minLength, maxLength, pattern, enumeration, whiteSpace
boolean	pattern, whiteSpace
decimal	totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive

DataType Restrictions

- Simple datatype restricted by range

```
<element name = "pubyear">
    <simpleType>
      <restriction base = “positiveinteger">
                <minInclusive>1000</minInclusive>
                <maxInclusive>3000</maxInclusive>
      </restriction>
    </simpleType>
</element>
```

- Simple datatype restricted by enumeration

```
<element name = “status">
    <simpleType>
      <restriction base = “string">
                <enumeration value = “draft”>
                <enumeration value = “final”>
      </restriction>
    </simpleType>
</element>
```

DataType Restrictions

- Named datatype restricted by length and pattern

```
<xsd:simpleType name="TelephoneNumber">
  <xsd:restriction base="xsd:string">
    <xsd:length value="8"/>
    <xsd:pattern value="\d{3}-\d{4}"/>
  </xsd:restriction>
</xsd:simpleType>
```

The Schema in Overview

- The schema element serves as a logical container for:
  - **type definitions**: A set of named simple and complex type definitions.
  - **attribute declarations**: A set of named (top-level) attribute declarations.
  - **element declarations**: A set of named (top-level) element declarations.
  - **attribute group definitions**: A set of named attribute group definitions.
  - **model group definitions**: A set of named model group definitions.
  - **notation declarations**: A set of notation declarations.
  - **annotations**: A set of annotations.