XML Schemas, Part 2

July 3rd, Lecture 6
Overview

- What you’ve learned so far
- Schema Syntax continued
- Modeling with Schemas
- Discuss the handout
What you’ve learned so far

- How to create elements and attributes
- How to ‘anonymously’ create types
- Simple ‘base’ types, such as string, integer
- Some complex types such as sequences and choices
- How to targeting namespaces
What is a simple type?

- It is a base type, such as string, integer, decimal
- It is a restriction, union, or list of those types
- We can do the above bullet point by using nested tags of course!
Restricting Simple Types

Let’s say we wanted an integer from 5 to 10

```
<simpleType name="num5_10">
  <restriction base="integer">
    <minInclusive value="5"/>
    <maxInclusive value="10"/>
  </restriction>
</simpleType>
```
Union of Simple Types

Let's say we want a type to be an integer or a token:

```xml
<complexType name="int_or_tok">
  <union memberTypes="integer
token"/>
</complexType>
```
List of Simple Types

Sometimes you just need a list of types:

```xml
<simpleType name="int_list">
  <list itemType="integer"/>
</simpleType>
```
Multiple Occurences

- How can we specify that an element should appear more than once?
- All element tags can use two tags: `minOccurs` and `maxOccurs`
- By default, `minOccurs=“1”` and `maxOccurs=“0”`
- You can also use the keyword “unbounded”
Three examples

Assume we have a tag called ‘phone’ and we want three instances where we have 0 or 1, 1 or more, and 0 or more ‘phone’ elements

```xml
<element name="phone" minOccurs="0"
<element name="phone"
    maxOccurs="unbounded">
<element name="phone" minOccurs="0"
    maxOccurs="unbounded">
```
Repeating sequences and choices

- How can we make sequences and choices repeat?
- Same thing of course! These elements too can use the minOccurs and maxOccurs attributes.
Complex Types

- Complex types are well, complex!
- They are used to extend types
- We need them because they are the only way we can add attributes
Complex Types in Detail

- They may contain `<simpleContent>` or `<complexType>`
- We’ve already seen `simpleContent`—it’s used to extend or restrict a simple type
- `complexType` also allows us to extend or restrict a `complexType` (named or anonymous)
Extending types

- The `<extension>` element, used in `simpleContent` and `complexContent`, adds attributes and/or child elements not present in the base type
- We can add 1 of the following in an extension: group, all, choice, sequence
- We can also add 0 or more attribute and attributeGroups
Groups

- Sometimes we have a set of attributes or a set of elements that we will use several times in the schema

- For these cases, we have the `<group>` and `<attributeGroup>` tags
The `<group>` element

- `<group>` can be used at the top level (as a direct child of `<schema>`)
- It can contain (all | choice | sequence) and can be named
- You can then refer to the group by name
<group name="mygroup">
  <choice>
    <element name="tag1" type="integer"/>
    <element name="tag2" type="decimal"/>
  </choice>
</group>

<element name="super-group">
  <complexType>
    <group ref="mygroup"/>
  </complexType>
</element>
attributeGroup

- We do the same thing
  <atri buteGroup>

- Create a named
  <atri buteGroup> at the top-level

- Reference it as part of a complexType in an element by the name you gave it
How can we replace ID in Schemas?

- Schemas present two powerful methods for expressing the ID concept: `<unique>` and `<key>`
- However, both require that you understand another standard, the XPath standard!
Xpath Crash Course

- All XML documents can be represented as a tree
- XPaths select a path from the root of the tree down to a leaf or subtree you’re interested in
- The root is indicated by “/” while an element and its children can simply be named, such as “phone”.
XPath Continued

How can we select a set of nodes in a schema?

<selector
    xpath="address_book/person/phone"/>

This selects all phone elements containing all person elements in the address_book tree.

<selector xpath="phone"/> is equivalent
How can we select an attribute?

- Using the field tag:

  `<field xpath="phone/@preferred"/>
  selects the preferred attribute of the phone element`

  `<field xpath="@preferred"/>
  is equivalent in this case`
We use the unique tag exactly like the ID attribute in DTDs

```xml
<element name="employee">
  <complexType>
    <attribute name="name" type="string" use="required"/>
    <attribute name="ssn" type="integer" use="required"/>
  </complexType>
  <unique>
    <selector xpath="employee"/>
    <field xpath="@ssn"/>
  </unique>
</element>
```
The `<key>` element only appears after the type of an `<element>` tag (just like unique)

The format is:

```xml
<key name="keyname">
  <selector xpath="tag"/>
  <field xpath="@field"/>
</key>
```
Keyrefs

- Once the values of the keys are calculated by its path and field selections, they are associated with the name of the key.
- Keyrefs also occur in an element after the type declaration:

```xml
<keyref refer="keyname">
  <selector xpath="tag">
    <field xpath="@field"/>
  </selector>
</keyref>
```
We all know the importance of documentation in your code.

However, if you just use <!-- comments -->, a processor could eliminate them along the way.

Hence, we have an <annotation> tag.
Annotations

- Annotations must appear immediately following your `<schema>` tag
- Annotations may contain two subtags: `<appinfo>` and `<documentation>`
- The only restriction in content in `<appinfo>` and `<documentation>` tags are that the XML markup is well formed
- The content does not affect the schema!
Modeling with Schemas

Just a few tips when modeling with Schemas:

• Take them one step at a time—add a few tags to your schema and then validate
• Declare your types instead of using ‘anonymous’ types
• You can save yourself a lot of time if you make your Schema human readable
Let's look at the handout