XML Schemas

Lecture 5
July 1, 2003
Overview

- What are Schemas and why do we want them?
- Syntax of Schemas
- Exercise
Why Schemas?

- Using Schemas, you will get:
  - Simple and complex types
  - Inheritance and derivation of types
  - Constraints on the number of elements
  - A document structure definition that is namespace aware
Schema Basics

- Let’s look at an example for a simple document:

```xml
<?xml version="1.0"?>
<book>Mostly Harmless</book>
```
Schema Example

```xml
<?xml version="1.0"?>
<xs:schema
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="book" type="xs:string"/>
</xs:schema>
```
Now let’s modify the xml document to use the schema:

```xml
<?xml version="1.0”>
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="book-schema.xsd”>
    Mostly Harmless
</book>
```
Notes about Example

- Every Schema document has a single root xs:schema element.
- Elements declared by top-level elements in the schema (ie, children of xs:schema element) are considered ‘global’
- Any ‘global’ element may be the root of a document.
Schema Syntax

- Simple elements are declared using `xs:element`

```xml
<xs:element name="book"
  type="xs:string"/>
```

- Simple types cannot contain other elements
Types for Simple Types

- **anyURI**: A Uniform Resource Identifier
- **base64Binary**: Base64 content-encoded binary data
- **boolean**: May contain true or false, 1 or 0
- **byte**: A signed byte quantity $\geq -128$ and $\leq 127$
- **dateTime**: An absolute date and time value combination
Types continued...

- **duration**: A relative amount of time, expressed in units of years, months, days, hours, etc...
- **ID, IDREF, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS** (same as DTD)
- **integer**: any positive or negative counting number
- **language**: may contain the same values as xml:lang attribute from the XML 1.0 recommendation
More types...

- **Name**: an XML name
- **normalizedString**: String with newline, tab, and carriage-return normalized to spaces
- **string**: unicode string
- **token**: Same as normalizedString with multiple spaces collapsed and trailing spaces removed
What about attributes?

- For example, we want to do:
  `<book language="en-GB">Mostly Harmless</book>`

- There is an xs:attribute element in schemas, but you have to use a complex type
Attributes

- To declare a complex type, use the following syntax:
  \[<\text{xs:complexType}>\]

- Now declare that the element has simple content:
  \[<\text{xs:complexType}>\]
  \[<\text{xs:simpleContent}>\]
Extending types

- Now we use `xs:extension` to extend the simple type of `xs:string`

```xml
<xs:element name="book">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="language" type="xs:language" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```
Targetting Namespaces

- Since you’ve just learned the benefit of namespaces, how can we associate a schema with a namespace?

- Instead of using the xsi:noNamespaceSchemaLocation attribute, use xsi:schemaLocation
Example

```xml
<book
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.cs.uoregon.edu/~chrisg/book
book-schema.xsd" language="en-GB">
   Mostly Harmless</book>
```

- Note that schemaLocation has two tokens, a namespace URI and location of the actual schema.
Example, part 2

- You also need to add a targetNamespace attribute to your schema document:

```xml
<xs:schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.cs.uoregon.edu/~chrisg/book">
```
Example, part 3 (finally!)

- We’ll still get an error if we validate the document! We need to declare the namespace for this document:

```xml
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns="http://www.cs.uoregon.edu/~chrisg/book">
  Mostly Harmless
</book>
```
Complex Types

- Only complex types can contain elements that are nested and have attributes.
- Only elements can have complex types

<x:element ...
  <xs:complexType>
Complex Structures

- Sequences are declared as:
  `<xs:sequence>
   … elements…
  </xs:sequence>`

- Choices are declared as:
  `<xs:choice>
   … elements…
  </xs:choice>`
Structure continued...

- The xs:all element:

  `<xs:all>
    ... elements...
  </xs:all>

- states that all elements must appear at least once, though in any order
Exercise

- Again, let’s take the checkbook example from last time, and design a schema for it