XML Meta-documents – DTDs, Schema

CS 431 – February 14, 2007

Carl Lagoze – Cornell University

Namespaces

- How the web does work
  - Individually created documents linked by ambiguous references
- How the web should work
  - Global database of knowledge
- Key to doing that is to permit distributed knowledge creation and lazy integration
- Problems
  - Vocabulary collisions
  - Joins
- Namespaces
  - Build on URI notion
  - Make it possible to uniquely qualify intra-document name collisions

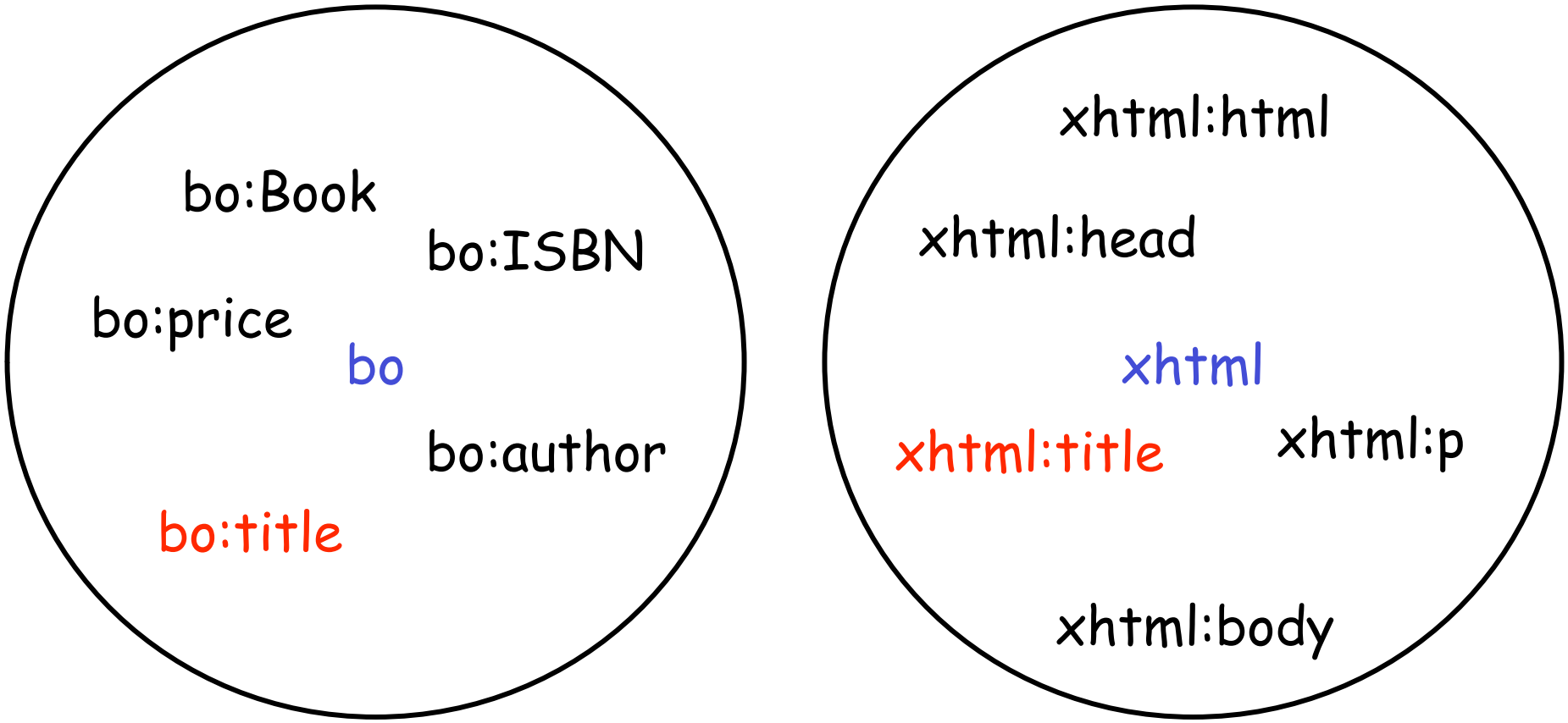
```
<?xml version="1.0" encoding="UTF-8"?>
<Book>
  <ISBN>0743204794</ISBN>
  <author>Kevin Davies</author>
  <title>Cracking the Genome</title>
  <price>20.00</price>
</Book>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p><p>My books</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p>
<p>My books
 <Book>
   <ISBN>0743204794</ISBN>
   <author>Kevin Davies</author>
   <title>Cracking the Genome</title>
   <price>20.00</price>
 </Book>
</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html>
<xhtml:head>
  <xhtml:title>My home page</xhtml:title>
</xhtml:head>
  <xhtml:body>
<xhtml:p>My hobby</xhtml:p>
<xhtml:p>My books
 <bo:Book>
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
   <bo:title>Cracking the Genome</bo:title>
   <bo:price>20.00</bo:price>
 </bo:Book>
</xhtml:p>
</xhtml:body>
</xhtml:html>
```

XML – namespaces



vocabulary bo

vocabulary xhtml

But who guarantees uniqueness of prefixes?

XML – namespaces

- Give prefixes only local relevance in an instance document
- Associate local prefix with global namespace name
  - ⇒ a unique name for a namespace
  - ⇒ uniqueness is guaranteed by using a URI in domain of the party creating the namespace
  - ⇒ doesn't have any meaning, i.e. doesn't have to resolve into anything

An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

```
<?xml version="1.0" encoding="UTF-8"?>
<xhtml:html
  xmlns:xhtml="http://www.w3c.org/1999/xhtml"
  xmlns:bo="http://www.nogood.com/Book">
<xhtml:head>
  <xhtml:title>My home page</xhtml:title>
</xhtml:head>
  <xhtml:body>
<xhtml:p>My hobby</xhtml:p>
<xhtml:p>My books
 <bo:Book>
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
     ..................
 </bo:Book>
</xhtml:p>
</xhtml:body>
</xhtml:html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml"
  xmlns:bo="http://www.nogood.com/Book">
<head>
  <title>My home page</xhtml:title>
</head>
  <body>
<p>My hobby</xhtml:p>
<p>My books
 <bo:Book>
   <bo:ISBN>0743204794</bo:ISBN>
   <bo:author>Kevin Davies</bo:author>
     ……
 </bo:Book>
</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml">
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p>
<p>My books
 <bo:Book xmlns:bo="http://www.nogood.com/
Book">
    <bo:ISBN>0743204794</bo:ISBN>
    <bo:author>Kevin Davies</bo:author>
      ……
 </bo:Book>
</p>
</body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html
  xmlns="http://www.w3c.org/1999/xhtml">
<head>
  <title>My home page</title>
</head>
  <body>
<p>My hobby</p>
<p>My books
 <Book xmlns="http://www.nogood.com/Book">
   <ISBN>0743204794</bo:ISBN>
   <author>Kevin Davies</bo:author>
     ..................
 </Book>
</p>
</body>
</html>
```

What do namespace URI's point to?

- There are lots of opinions on this subject!
- The "abstraction" camp
  - A namespace URI is the id for a concept
  - It shouldn't resolve to anything
  - Example – my SSN #, it doesn't point to Carl Lagoze but to the concept of Carl Lagoze with different facets can be (ab)used in numerous concepts
- The "orthodox" camp
  - It should resolve to a schema (xml schema)
- The "liberal" camp
  - It should resolve to many things
  - RDDL (http://www.rddl.org)

- Reality: Read Wikipedia about this if you want to see how ambiguous this all is: http://en.wikipedia.org/wiki/Uniform_Resource_Identifier
- Moral: Interoperability is hard once you move beyond the basics

From well-formedness to validity

- Goal of standards is interoperability
  - Allow different communities to share data
  - Requires meta-level understanding
- Levels of XML interoperability
  - Well-formedness
    - Base-level syntax
    - Properly formed tree
  - Validity
    - Structure of tree
    - Adherence to tree constraint rules

Tree constraint languages

- Document Type Definitions (DTDs)
- XML Schema
- Schematron
- RELAX NG

DTD – Document Type Definition

- Artifact of XML's roots in SGML
- Defines validity XML document
- Problems with XML DTD's:
  - DTD's are not extensible: Can import declarations but can not inherit or refine those declarations.
  - A document must be valid according to 1 DTD: prevents building on elements from different DTDs
  - Limited support of namespaces
  - Poor data typing: DTDs are mainly about "text". No provision for numeric data types, dates, times, strings conforming to regular expressions, URI's, …
  - DTD's are defined in non-XML syntax => Can not use XML tools!
  -

XML Schema

- W3C Recommendation
  - http://www.w3.org/XML/Schema#dev
- Very complex standard
  - Fortunately there is a primer
  - http://www.w3.org/TR/xmlschema-0/
  - Some really good online materials: http://www.w3schools.com/schema/

Interoperability & Extensibility

- XML schema are building blocks to interoperability between multiple data sources
  - Enforces shared markup
    - E.g., a <person> must have a <firstname> and <lastname>
  - Enforces shared types
    - E.g. <person age="18"> - age must be a number between 0 and 120
- XML schema are building blocks for extensibility
  - Reuse
  - Type derivation

Expressed in XML

- All tags are in the http://www.w3.org/2001/XMLSchema namespace
- Can be manipulated by standard XML tools

Simple Schema Example

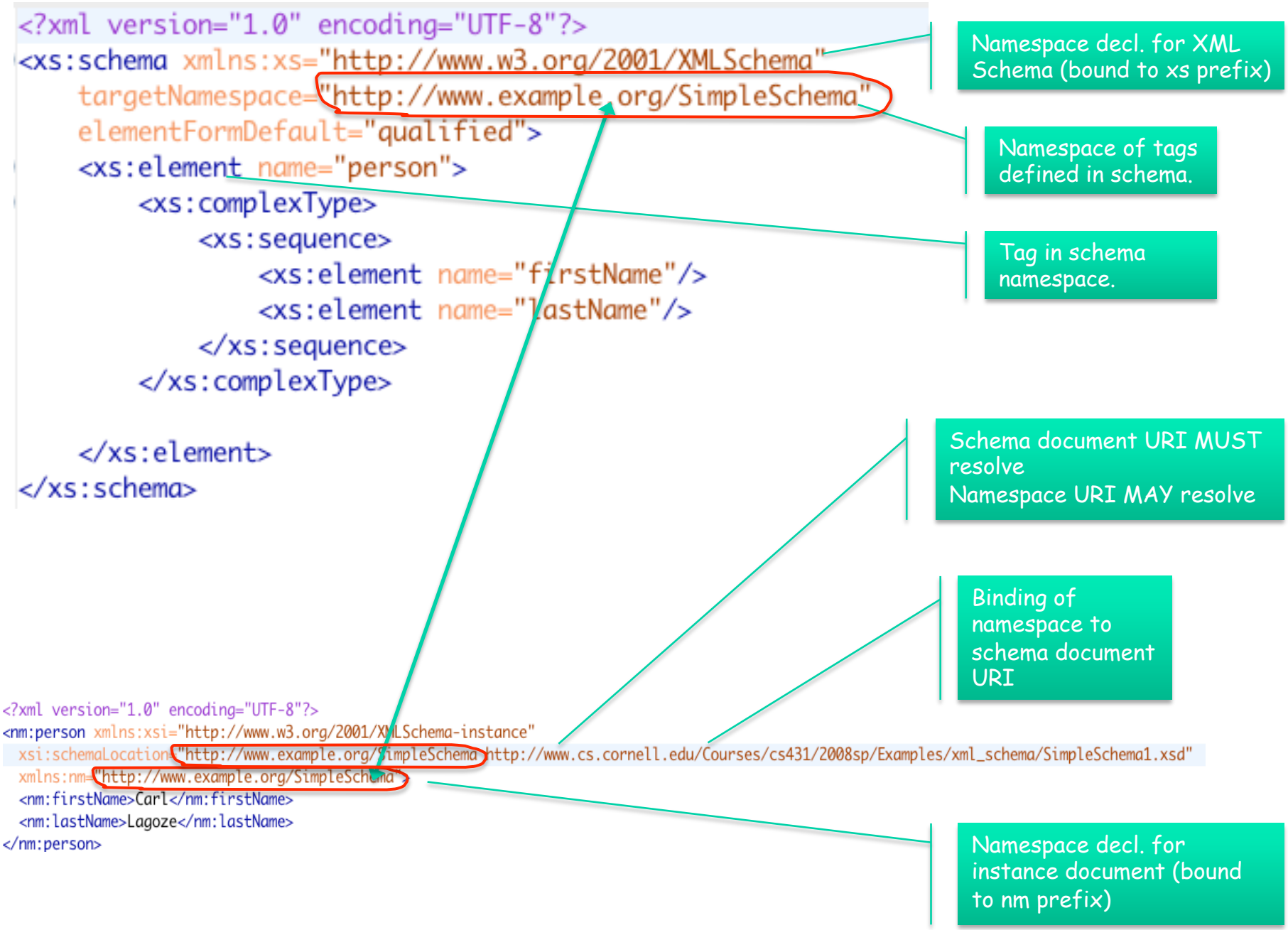
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/SimpleSchema"
    elementFormDefault="qualified">
    <xs:element name="person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="firstName"/>
                <xs:element name="lastName"/>
            </xs:sequence>
        </xs:complexType>

    </xs:element>
</xs:schema>
```

Document with Schema Association

```
<?xml version="1.0" encoding="UTF-8"?>
<nm:person xmlns:nm="http://www.example.org/SimpleSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/SimpleSchema SimpleSchema1.xsd ">
  <nm:firstName>Carl</nm:firstName>
  <nm:lastName>Lagoze</nm:lastName>
</nm:person>
```

- Note!!
  - Multiple namespaces
  - Namespace vs. schemaLocation


Namespace decl. for XML Schema (bound to xs prefix)
Namespace of tags defined in schema.
Tag in schema namespace.
Schema document URI MUST resolve
Namespace URI MAY resolve
Binding of namespace to schema document URI
Namespace decl. for instance document (bound to nm prefix)

Complex Types (contain sub-tree)

- Define the structure of the sub-tree within the element

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Controls on complex types

- sequence – specific order
- all – any order
- choice – only one

- cardinality – minOccurs, maxOccurs

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Simple Types (no sub-tree within)

- Define an element to have a simple value
  - Constrain value to a specific data type
  - Set of data types in xs namespace
- Syntax
  - <xs:element name="xxx" type="yyy"/>
- Examples
  - <xs:element name="lastname" type="xs:string"/>
  - <xs:element name="age" type="xs:number"/>
  - <xs:element name="age" type="xs:date"/>

Facets for simple values

- Restrictions on values within type context
  - E.g. range for an integer value, controlled set for string
- Examples

```
<xs:element name="age">

<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="16"/>
    <xs:maxInclusive value="34"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

```
<xs:element name="car">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Mercedes"/>
    <xs:enumeration value="Volvo"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

String types and patterns

```
<xs:element name="initials">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Mixed Content

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<letter>
Dear Mr.<name>John Smith</name>.
Your order <orderid>1032</orderid>
will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Declaring attributes

- Define type
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time
- Define optional or required

```
<xs:attribute name="lang" type="xs:string" use="optional"/>
```

Use of attributes

- Always a complex type

```
<xs:element name="shoesize" type="shoetype"/>

<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Another Example

- Memo Schema
  - http://www.cs.cornell.edu/courses/CS431/2008sp/examples/xml_schema/memo.xsd
- Instance Document
  - http://www.cs.cornell.edu/courses/CS431/2008sp/examples/xml_schema/memo.xml

Extending a complex type

- Add values to sequence

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Type Reuse

```
<xs:import namespace="http://carl.org/stuff1"
schemaLocation="http://www.cs.cornell.edu/Courses/cs502/2002SP/demos/xmlschema/address.xsd"/>
```

Type Reuse Example

- Address schema
  - http://www.cs.cornell.edu/courses/CS431/2008sp/examples/xml_schema/address.xsd
- Person schema
  - http://www.cs.cornell.edu/courses/CS431/2008sp/examples/xml_schema/person.xsd
- Instance document
  - http://www.cs.cornell.edu/courses/CS431/2008sp/examples/xml_schema/person.xml