What Kind Of Thing Is It?

Roger L. Costello
November 2011

Learning a Lesson

This week I learned a valuable lesson on the difference between XML Schemas and ontologies. I think you will find it of interest.

Warning: in the following two sections I will lead you down a path and attempt to persuade you that everything is reasonable and logical. Then, in the two sections after that I will change my position 180 degrees and attempt to persuade you that what I said previously is unreasonable and illogical.

The Problem – Element Has No Information About The Kind Of Thing It Is

In this section and the next I will attempt to persuade you to connect every element in your XML Schema to a semantic identifier.

Some XML Schemas declare elements and do not associate them to anything. That is, there is no indication of what kind of thing an element is. For example, in the following XML Schema there is no indication of what kind of thing the title element is:

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

That element declaration states the name of the thing (title), the type of data that the thing can have (string), but it says nothing about what kind of thing it is.

Even when an element is declared inside another element that does not tell what kind of thing it is. For example, here the title element is declared inside a Book element:

```
<element name="Book">
  <complexType>
    <sequence>
      <element name="title" type="string" />>
    </sequence>
  </complexType>
</element>
```

That merely says that the title element will physically occur inside a Book element. But it does not answer the question, “What kind of thing is title?”

Furthermore, what kind of thing is Book?

The Solution – Categorize Your Elements

Before creating an XML vocabulary, abstractly categorize the kinds of things it will contain.

A popular way of categorizing elements is as objects and properties. Properties describe objects.

> *By associating an element with object or property, XML applications can identify what kind of thing it is.*

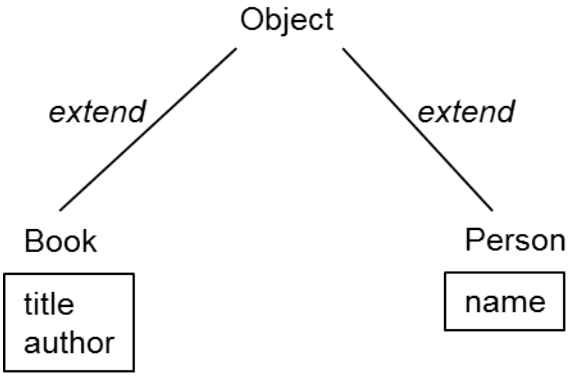
If the above title element were associated with property then XML applications would know that title is a property of some object. That's useful information!

Let's create an XML Schema for this:

```
<Book>
  <title>The Implementation of Functional Programming Languages</title>
  <author>
    <Person>
      <name>Simon L. Peyton Jones</name>
    </Person>
  </author>
</Book>
```

Book and Person are objects; title, author, and name are properties. Observe that objects contain properties and the value of a property may be either a simple type or an object. Also note the naming convention: objects begin with an uppercase letter and properties begin with a lowercase letter.

In XML Schema the mechanism for associating elements is with derive-by-extension and derive-by-restriction of complexTypes. We want to declare Book and Person to derive from Object:



Object is a complexType. Its purpose is just to name a kind-of-thing, so it has no content. Also, since it is not used in XML instance documents it is abstract. Here is how Object is defined in an XML Schema:

```
<xs:complexType name="Object" abstract="true" />
```

To associate Book with Object (that is, to express "Book is an Object") we perform derive-by-extension in Book's type:

```
<xs:element name="Book" type="BookType" />

<xs:complexType name="BookType">
  <xs:complexContent>
    <xs:extension base="Object">
      <xs:sequence>
        <xs:element ref="title" />
        <xs:element ref="author" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Likewise, to associate Person with Object (that is, to express "Person is an Object") we perform derive-by-extension in Person's type:

```
<xs:element name="Person" type="PersonType" />

<xs:complexType name="PersonType">
  <xs:complexContent>
    <xs:extension base="Object">
      <xs:sequence>
        <xs:element ref="name" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Now XML applications can follow the reference (<xs:extension base="Object">) to recognize that Book and Person are both Objects.

We proceed in a similar manner to associate elements with property. Property is a complexType. Its purpose is just to name a kind-of-thing, so it has no content. Also, since it is not used in XML instance documents it is abstract.

However, there is one key difference between Object and property. Object elements always have a complexType (*i.e.*, Object elements always have child elements). Conversely, property elements may either have simple content or complex content. For example, this property element has simple content:

<title>**The Implementation of Functional Programming Languages**</title>

Whereas this property element has an Object as its content, which is complex:

```
<author>
  <Person>
    <name>Simon L. Peyton Jones</name>
  </Person>
</author>
```

That difference requires property to be defined a little differently. Here is how property is defined in an XML Schema:

```
<xs:complexType name="property" abstract="true" mixed="true" />
```

Notice mixed="true". This is required to enable both simple and complex property elements.

Here's how to declare the title element as a property with simple content:

```
<xs:element name="title" type="titleType" />

<xs:complexType name="titleType">
  <xs:simpleContent>
    <xs:restriction base="property">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:maxLength value="100" />
        </xs:restriction>
      </xs:simpleType>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

And here's how to declare the author element as a property with complex content:

```
<xs:element name="author" type="authorType" />

<xs:complexType name="authorType" mixed="true">
  <xs:complexContent>
```

```
        <xs:extension base="property">
          <xs:sequence>
            <xs:element ref="Person" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

Notice that author has mixed content. That is not what we desire. It allows this:

```
<author>
  Blah, blah, blah!
  <Person>
    <name>Simon L. Peyton Jones</name>
  </Person>
</author>
```

And we don't want that (*i.e.*, we don't want to allow the string Blah, blah, blah).

To ensure that the content of author is only the Person element, we need to either use the XML Schema 1.1 assert element or supplement XML Schema with Schematron. Here is how to use the assert element to ensure that author has only a Person element and nothing else:

```
<xs:complexType name="authorType" mixed="true">
  <xs:complexContent>
    <xs:extension base="property">
      <xs:sequence>
        <xs:element ref="Person" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
  <xs:assert test="not(normalize-space(string-join((.)/text(), "")))" />
</xs:complexType>
```

That is a kludge because mixed="true" says "*mixed content is okay*" whereas the assert says "*no it's not okay to have mixed content*."

Data Types Are Not Appropriate For Ontological Associations

In this section and the next I will attempt to persuade you that the previous two sections are wrong and you should not associate each element to a semantic identifier. Such associations should be made in an ontology not in an XML Schema.

The purpose of a "data type" in programming languages and in XML Schema is to specify a set of values. That's all. Nothing more.

The previous two sections used the data type mechanism for another purpose: to associate an element to an identifier. For example, the Book and Person elements were associated to Object, and the title, author, and name elements were associated to property.

To put it another way, the previous two sections attempted to use the data type mechanism to define a semantic or ontological relationship for the elements.

That is not a good division of labor.

Separate the definition of structure and the definition of semantics (*i.e.*, ontological relationships).

Here's what Michael Kay said about trying to categorize each element as either an Object or property:

```
I think that's a semantic categorization concerning the meaning or usage
of the element. As such it has nothing to do with the concept of 'type'
as defined in XSD, which is a categorization according to constraints on
the content of the element. This is in line with the conventional use of
the term in programming languages. I think it's probably a mistake to try and
use the concept of 'type' to represent an ontological distinction of this
nature.
```

Lesson Learned

Don't use XML Schema to define ontological relationships. Define relationships using an ontology language such as RDF Schema or OWL.

Recall in section 3 I made this assertion:

> *By associating an element with Object or property, XML applications can identify what kind of thing it is.*

Now we know that that is nonsense. XML applications should not look to an XML Schema to identify what kind of thing an element is. XML applications should look to an ontology for such information.

With XML Schemas the derive-by-extension and derive-by-restriction mechanism is just to support reusability. It is not to support the definition of ontological relationships. If you see an XML Schema with long chains of derive-by-extension and derive-by-restriction then you should suspect that the schema developer is erroneously attempting to use XML Schema to define ontological relationships.

Feedback

From Eliot Kimber:

XML schemas are nothing more than document syntax constraint specifications. There is no sense in which then can be anything more than a very weak reflection of some deeper ontology that governs the semantic objects for which the XML governed by the XSD schema is one possible serialization.

That is, ontologies describe relationships among things, schemas define syntactic constraints on XML elements. The fact that the XSD mechanism has a weak facility for defining type hierarchies does not make it a language for describing taxonomies or ontologies.

From Henry Thompson (responding to Eliot Kimber's message):

Hear hear! Confusing application domain analysis/data model design with interchange/archival document language design is a fundamental (albeit very common) mistake. Don't do that.

What Kind Of Thing Is It?

Suppose processing of XML instance documents requires answers to these questions:

- What kind of thing is Book?
- What kind of thing is Person?
- What kind of thing is title?
- What kind of thing is author?
- What kind of thing is name?

Here are the answers we expect to get:

- Book is an Object
- Person is an Object
- title is a property
- author is a property
- name is a property

In this paper I have attempted to persuade you that *what-kind-of-thing-is-it* questions are best answered with an ontology, not an XML Schema. The relationship of each element to a semantic identifier such as Object or property is readily expressed in an RDF Schema, as shown below. The below RDF Schema is read as: "A Book is a subclass of Object (*i.e.*, a Book is an Object). A Person is an Object. A title is a property. An author is a property. A name is a property."

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdfs:Class rdf:ID="Object">
  </rdfs:Class>

  <rdfs:Class rdf:ID="property">
  </rdfs:Class>

  <rdfs:Class rdf:ID="Book">
    <rdfs:subClassOf rdf:resource="#Object"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Person">
    <rdfs:subClassOf rdf:resource="#Object"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="title">
    <rdfs:subClassOf rdf:resource="#property"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="author">
    <rdfs:subClassOf rdf:resource="#property"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="name">
    <rdfs:subClassOf rdf:resource="#property"/>
  </rdfs:Class>

</rdf:RDF>
```


Acknowledgements

I acknowledge the following persons for their input to this paper: Michael Kay, Mukul Gandhi, and Simon Cox. Thank you to Ken Holman for the XPath expression in the assert element.

Thank you to Mark Lang for informing me of a couple typos. Thank you to Eliot Kimber and Henry Thompson for their feedback.