Describing Content Types in XML Schema

Umit Yalcinalp
Company: SAP
Dec. 10, 2005 09:48 PM

Problem:

The content of messages of web services are typically described in XML schema, but many business applications send and receive documents that use special formats as part of their message exchange. Some of these documents (or parts of a message exchanged) may or may not be expressed in XML. They may be expressed in popular formats, such as word, pdf, or even languages that are built in XML, such as rdf. This leads to a problem of how to describe the parts of XML messages that contain non-XML content without losing the capability of expressing their content accurately. This is an important problem to solve to enable applications that utilize these specific formats to process them as intended.

As you may also note, the problem of expression is not limited to web services. It is the expressive ability of XML schema in describing non-XML content within XML. There might be many different solutions for this problem, but to achieve interoperability, ideally vendors can settle down on one solution.

Here I want to talk about the well known and established trick proposed in [w3cNote] that aims at increasing the expressive ability of your schemas.

Example:

Consider an insurance service that processes new insurance applications for residential real estate. The application must contain a description of the house that needs to be insured. The description would typically include the house's dimensions, the year it was built, as well as couple of its pictures taken by the insurance agent. The pictures may be expressed in common formats, such as using gif, jpeg. Thus, the description of the house consists of its physical attributes as well as its pictures.

If we want to use XML Schema 1.0 to describe the house, we will probably come up with a definition like the the following schema. Consider the sapex namespace prefix to define the types we want for simplicity:

```
<xs:element name="house">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="sapex:AddressType"/>
            <xs:element name="yearBuilt" type="xs:integer"/>
            <xs:element name="dimensions" type="sapex:DimensionType"/>
            <xs:element name="picture" type="xs:base64Binary"
```

```
                    minOccurs="1" maxOccurs="4"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

However, you would probably notice that we lost two important aspects by describing the house in this manner:

- *Descriptive nature of the schema* when non-XML content needs to be mixed with XML content. In this representation, how do we know that picture element represents a jpeg picture or a gif picture?
- *Ability to represent the value* in its native format (thus avoiding encoding the content in base64).

The problem is that there is no appropriate data type to use to designate the pictures that are in our description, except embedding *xs:base64Binary* encoded binary data in the description to hold the pictures of the house. This means that the document that corresponds to this schema will need to adhere to the requirement that the binary content is *opaque* (because we no longer have the capability of designating what its real content is!) and we are required to represent it in an encoding which is suitable as part of an XML document.

XML Schema does not offer any solution to this problem other than the ability to designate part of the schema with binary data types, such as *xs:base64Binary*(or *xs:hexBinary*) elements. In this regard, the schema that corresponds to the message of our sample application is to designate them with binary data types where the values of these datatypes need to be encoded in *xs:base64Binary* for each of the picture. Further, the data types that most applications will be interested in are probably defined elsewhere, such as IANA tokens and replicating this information within XML schema hierarchies would be a non-acceptable solution since the IANA tokens are extensible and the content types are domain specific.

This problem becomes more evident when documents that need to utilize such mixed content need to be exchanged in the world of web services. Typically, non-XML data is sent via attachments but XML content of the document becomes part of the payload of the message, i.e. SOAP body. At this point, there are two choices to consider. Either the binary encoding as suggested by the schema is used which results in imploding the size of the document by including the whole content as part of the SOAP body or a means to represent the attachment retaining its intended content has to be found without using encoding. Efficient transmission of content is vital to web services, thus the solution should take into account the preservation of binary data while retaining the logical definition of the content. This approach would require you to make up the logical description of the house from its physical description and jpeg files and use a solution that is geared towards expressing the attachments appropriately in your description.

Let's look at the description problem first, because without the correct description one can not design efficient encoding or serialization approaches anyway.

Using the Global Attribute Trick:

Unfortunately, XML Schema does not offer a built in solution for this problem, but luckily it offers us the basis for a trick that is widely used:

Trick: Define global XML Schema attributes and use them as annotations to designate metadata for your content.

This solution should be of interest to anyone who wants to utilize a richer data type model within XML Schema.

Using this principle, w3c WSD and XMLP working groups published a joint note that uses this specific trick to designate specific metadata markup as annotations [w3cNote]. The note defines two global attributes, *xmime:expectedContentTypes* and *xmime:contentType*.

- The *xmime:expectedContentTypes*, is used to annotate a binary element/type to list possible content types in the schema itself. By using this attribute, the content is expected to be one of the items that are listed in the list, including wildcards.
- The *xmime:contentType* is used in an instance document and indicates what the actual content would be. It is most useful in conjunction with the *xmime:expectedContentTypes* markup.

In combination, the annotations indicate design time (in schema) constraint and document (in instance) specific content type in conjunction with binary elements that would otherwise be opaque. This definition allows tools to be able to interpret and provide additional data binding capabilities that are geared towards processing the specific content.

Example Reworked to use the Metadata attributes:

If our previous insurance application would be able to process both jpeg and gif pictures, it may be expressed with the following schema:

```
<xs:element name="house">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type="sapex:AddressType"/>
            <xs:element name="yearBuilt" type="xs:integer"/>
            <xs:element name="dimensions" type="sapex:DimensionType"/>
            <xs:element name="picture" type="xs:base64Binary"
                xmime:expectedContentTypes="image/jpeg, image/gif"
                minOccurs="1" maxOccurs="4"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Note that the description of the picture element describes that its content may be either jpeg or gif, but does not specify what the actual content type is unless there is a single value, such as "image/jpeg" in the expectedContentTypes attribute.

In order to designate the actual content type, the document instance would need to express the content type, if desired by using the *xmime:contentType* attribute. A

modified version to include this attribute would make our schema to look like this. Note that for simplicity, the xmime annotation is moved to the type definition instead of the element declaration as it also allows localizing the constraints better.

```
<xs:element name="house">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="address" type=">sapex:AddressType"/>
            <xs:element name="yearBuilt" type="xs:integer"/>
            <xs:element name="dimensions" type="sapex:DimensionType"/>
            <xs:element name="picture" type="sapex:PictureType"
                    minOccurs="1" maxOccurs="4"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="PictureType"
    xmime:expectedContentTypes="image/jpeg, image/gif">
    <xs:simpleContent>
        <xs:restriction base="xmime:base64Binary" >
           <xs:attribute ref="xmime:contentType" use="required" />
        </xs:restriction>
    </xs:simpleContent>
</xs:complexType>
```

You will also note that there is a predefined *xmime:base64Binary* type already defined for convenience in [w3cNote]. This type contains the contentType attribute. Alternatively, we could have retained the annotation at the element declaration and used the predefined type. The choice is simply yours, however you should pay attention to the specific rules in tools that provide language binding, such as JAXB 2.0 which may provide more guidance as where you should use the annotations.

Here is an instance document that may use the contentType attribute to indicate the actual content.

```
<sapex:house>
    <sapex:address>
       <sapex:street>...</sapex:street>...
      </sapex:address>
    <sapex:yearbuilt>1960</sapex:yearbuilt>
    <sapex:dimensions>....</sapex:dimensions>
    <sapex:picture
        xmime:contentType="image/jpeg"> adXADW...</sapex:picture>
</sapex:house>
```

Note that we included the attribute as a required attribute in order for the documents to indicate the actual content in the document. However, the usage of the attribute is not required especially if there may be a single expected content type value. As a matter of fact, as we talk about this below the attributes may be used as hints for recent data binding solutions.

You may already note that this solution may also be used to embed XML content within XML content where there may be different charsets or constraints that govern different portions of the XML content. Since the attributes contain a valid content type definition, it may be more specific and indicate parameters or character sets. As a matter of fact, the

definition of these attributes allow a wide variety of parameters as well. For full production rules and recommendations, please refer to the actual note [w3cNote].

Content Types as Hints for DataBinding:

As I indicated above, data binding solutions may be able to use these hints to provide better data binding. Java programmers already may be using the benefits of this approach.

[JAXB 2.0] utilizes some of the possible values of *xmime:expectedContentTypes* for providing a better binding for Java programs and links the existing data types in Java to some of the known content type values.

Where all binary data may be mapped by using *javax.activation.DataHandler* class, JAXB 2.0 provides more specific databinding by utilizing the content types "image/jpeg", "image/gif", "application/xml", "text/xml" attribute values for databinding when they are specified as expected content type values on binary elements. The corresponding Java types are *java.awt.Image*for images (jpeg and gif) and *javax.xml.transform.Source* for xml content respectively. This mapping solves the problem of mapping Schema types to Java types.

In order to generate the desired marker in XML, JAXB 2.0 introduces a Java annotation *@XmlMimeType* to indicate known expected content Type in the schema. This Java annotation is used to generate the appropriate Schema markup from a Java program for binary data.

Conclusion:

Utility of media types in describing content within an XML document may be achieved by using the XML Schema annotations and this solution is now being utilized by data binding solutions, such as JAXB 2.0. I happened to be one of the editors of the note and wanted to illustrate how these attributes may solve the data description problem. In this weblog, I only talked about the description part of the problem, however the other problem is avoid encoding of binary data altogether. This is subject to another weblog.

References:

[w3cNote] Describing Media Content of Binary Data in XML, Anish Karmarkar and Umit Yalcinalp (editors), http://www.w3.org/TR/2005/NOTE-xml-media-types-20050504/

[JAXB 2.0] Java API for XML Data Binding 2.0
http://jcp.org/aboutJava/communityprocess/edr/jsr222/

Umit Yalcinalp is an architect in NetWeaver Web Services Standards team.