Enumeration Extension

2001-10-29

1 Overview

[Paul's note to TSC – I did not use the full template, as this probably belongs in the ExtendingSchemas doc. I used partial template to make cut and paste easier.]

1.1 Objective

HR-XML Consortium work groups are increasingly grappling with a common problem regarding the use of enumerations in schemas. The traditional use of enumerations consists of a fixed number of provided values, which are determined at the time of the schema design. The problem arises when a business process is modeled with a schema that includes enumerated lists that do not cover 100% of the foreseen cases. The main question arises: how can a work group standardize enumerated values when less than 100% of the foreseeable values are known at design time? The objective of this text is to endorse a method for standardizing enumerated values without preventing extensions to cover unknown or trading partner specific values.

1.2 Design Requirements

All extensions to approved HR-XML standards SHOULD be done consistently across the Consortium. They MUST retain the basic values of XML, such as validation, especially in the context of any Certification process. Extension methods MUST not lead to an undermining of the open standards mission.

1.3 Scope

This text focuses on Consortium-wide endorsement of XML Schema enumeration standardization and extension methods. For the purposes of this text, the problem is referred to as "incomplete enumeration lists."

2 Enabling Supported Business Processes

2.1 Possible approaches: convention or pattern extension

The two most debated approaches to standardizing incomplete enumeration lists are a union of values with a string, essentially a convention, and a union of values with a string pattern, known as pattern extension.

2.1.1 The Convention Method

The “convention” method consists of a list of known enumerated values. Consider a scenario where there is a contact method element called <VoiceNumber> (lines 01 to 09). This element models the telcom number data used for contacting some person. An attribute of this element, called “whenAvailable” (line 05) is used to model when this VoiceNumber is considered a valid contact. A schema design team may know many of the values that may occur in this attribute, i.e. “daytime”, “nighttime”, “weekends”, etc. However, they may want to allow more values than are stated at design time, while still standardizing the known values.

```
01: <xsd:element name="VoiceNumber">
02:     <xsd:complexType>
03:         <xsd:simpleContent>
04:             <xsd:restriction base="xsd:string">
05:                 <xsd:attribute name="whenAvailable" type="whenAvailableExtensionType"/>
06:             </xsd:restriction>
07:         </xsd:simpleContent>
08:     </xsd:complexType>
09: </xsd:element>
10: <!-- the known enumerations for availability -->
11: <xsd:simpleType name="whenAvailableType">
12:     <xsd:restriction base="xsd:string">
13:         <xsd:enumeration value="daytime"/>
14:         <xsd:enumeration value="nighttime"/>
15:         <xsd:enumeration value="weekends"/>
16:     </xsd:restriction>
17: </xsd:simpleType>
18: <!-- this is the unioning of the enumeration with a string -->
19: <xsd:simpleType name="whenAvailableExtensionType">
20:     <xsd:union memberTypes="whenAvailableType  xsd:string"/>
21: </xsd:simpleType>
```

In this example, the known values are stated as an enumeration (lines 11 to 17). The final step that makes this a convention is where it is unioned with a string data type (lines 19 to 21). This union means that the attribute “whenAvailable” must be either a member of the enumerated list OR a string.

Since all enumerated values given are also strings, this in effect makes the data type a string. The parser cannot know that the value “daytimmme” is a misspelling of an enumerated value and should be an error. The value is still a string, so it is considered valid. According to the parser, both of the instances below are considered valid.

```
<VoiceNumber whenAvailable="daytime">11111111</VoiceNumber>
<VoiceNumber whenAvailable="daytimmme">2222222</VoiceNumber>
```

The advantage of going to the trouble of enumerating values when the result is essentially a string, is that implementers have a set of known values it can use to pass data that need not be detailed in a Trading Partner Agreement. It standardizes values, but only as a convention.

The disadvantage is that validation of values falls on the implementation instead of the parser. The effective data type of the data is a string, and so implementations must determine if the value is valid or a misspelled enumeration for example.

2.1.2 The Pattern Extension Method

The pattern extension method is similar to the convention except that the parser is able to distinguish between standard and non-standard values. Consider the same scenario as above, a <VoiceNumber> element with a “whenAvailable” attribute.

```
01: <xsd:element name="VoiceNumber">
02:     <xsd:complexType>
03:         <xsd:simpleContent>
04:             <xsd:restriction base="xsd:string">
05:                 <xsd:attribute name="whenAvailable" type="whenAvailableExtensionType"/>
06:             </xsd:restriction>
07:         </xsd:simpleContent>
08:     </xsd:complexType>
09: </xsd:element>
10: <!-- the known enumerations for availability -->
11: <xsd:simpleType name="whenAvailableType">
12:     <xsd:restriction base="xsd:string">
13:         <xsd:enumeration value="daytime"/>
14:         <xsd:enumeration value="nighttime"/>
15:         <xsd:enumeration value="weekends"/>
16:     </xsd:restriction>
17: </xsd:simpleType>
18: <!-- the pattern used for all extension enumeration values -->
19: <xsd:simpleType name="otherPatternType">
20:     <xsd:restriction base="xsd:string">
21:         <xsd:pattern value="X:\S\S*"/>
22:     </xsd:restriction>
23: </xsd:simpleType>
24: <!-- here is where we bring them together, either is it a known enumeration or
25: it is an enumeration that begins with the string "X:" -->
26: <xsd:simpleType name="whenAvailableExtensionType">
27:     <xsd:union memberTypes="whenAvailableType  otherPatternType"/>
28: </xsd:simpleType>
```

In this case, the known values for the attribute are stated the same as in the convention method (lines 11 to 17). However, a pattern is used to establish the format of any non-standard values (lines 19 to 23). This pattern states that any value conforming to this pattern must begin with the string “X:” (not including the quotes) and must consist of a string with at least one character (line 21). When the enumeration is unioned with this pattern (lines 26 to 28), the result is that the attribute must be either an enumerated value OR a string begining with “X:”.

```
<!-- valid values from the enumeration list -->
<VoiceNumber whenAvailable="daytime">11111111</VoiceNumber>
<VoiceNumber whenAvailable="nighttime">2222222</VoiceNumber>
```

```
<!-- valid values, conforming to the pattern extension-->
<VoiceNumber whenAvailable="X:everyTuesday">4444444</VoiceNumber>
<VoiceNumber whenAvailable="X:summer">5555555</VoiceNumber>

<!-- invalid values, neither enumerated values nor do they conform to the pattern -->
<VoiceNumber whenAvailable="mondaysOnly">5555555</VoiceNumber>
<VoiceNumber whenAvailable="daytimmme">5555555</VoiceNumber>
```

The advantage of this approach is that the parser can enforce the difference between standard and non-standard values using the delimiter pattern “X:”. It can also enforce the list of standard enumerations (see last instance example above).

The disadvantage is the requirement for processing the extension values, as the delimiter must be discarded.

2.2 Data and metadata: what is good XML design?

Given the two methodologies discussed, the technical temptation is to endorse the Pattern Extension method, as it utilizes the parser much more effectively. But does this violate a principle of good XML design? In designing schemas, editors often adhere to the notion that data should be separated from metadata. Further, an often-used design rule is that elements should contain data and attributes metadata. Given the Pattern Extension method, is this combining data and metadata in the attribute value?

2.3 Conclusion:

The Technical Steering Committee has determined that while the principle of separating data from metadata has significant merit, in this case, making the best use of the parser can be more important. Consequently, it endorses the Pattern Extension method of standardization of incomplete enumeration lists when most of the values are known. When only a few values are know, the Convention Method is acceptable as well as using an atomic data type such as a simple string.

2.4 Conformance note

It is important to note that the Pattern Extension method described here distinguishes between standard and non-standard values. As such, all non-standard values (those beginning with “X:”) MUST NOT be considered part of the HR-XML Consortium approved standard. Support for extended values is not required for conformance to the standard.

2.5 Limitations on use

The Pattern Extension method is a powerful tool for enabling the use of incomplete enumeration lists. However, it should not be seen as a standard practice for all enumerations. It is intended for instances where most values are known, and not where only a few are known. When only a

few values are known, and an atomic data type such as a simple string is insufficient, the Convention Method is acceptable.

Even further, it should not be used when work groups believe they know all the enumerations that should be in a schema, but want to leave the door open to “possibilities.” Excessive use of the Pattern Extension could lead to an undermining of the standard. Essentially, this design should be used sparingly.