Representing functional views in schema

A Gregory, M Hebing, O Hopt, D Smith, A Wackerow

Minneapolis Sprint 2015

Introduction

The first approach for representing functional views in XML schema was to have a root level element for each view in its own namespace. This will cause the issue, that two users, making use of different views, could not validate documents exchanged between them, even if they should be compatible. Also, the users would have to choose from a large number of possible root elements first, before they could start exploring DDI.

To meet this issue, we explored two alternatives. One is based on substitutions, the other on inclusion of XML instances.

Substitution Approach

The general idea is, to have a general root element that is defined to contain instances of an overall view element, which will then contain all DDI elements designed in the model. The entire model library will be produced in one single (version based) namespace.

```
<xs:schema>
  <xs:element name="DDI" type="DDIType"/>
  <xs:complexType name="DDIType">
    <xs:sequence>
      <xs:element ref="View" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="View" type="ViewType"/>
  <xs:complexType name="ViewType">
    <xs:sequence>
      <xs:element ref="Concept" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Variable" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Universe" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Concept" type="ConceptType"/>
  <xs:complexType name="ConceptType">
    <xs:sequence>
      <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1"/>
```

```
      <xs:element name="Description" type="xs:string" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
...
</xs:schema>
```

A functional view is now defined in a new XSD file but within the same namespace using xs:override, to narrow down the overall view:

```
<xs:schema>
  <xs:override schemaLocation="DDI-library.xsd">
       <xs:complexType name="FunctionalViewType">
       <xs:complexContent>
       <xs:restriction base="ViewType">
       <xs:sequence>
              <xs:element ref="Concept" minOccurs="1" maxOccurs="unbounded"/>
              <xs:element ref="Variable" minOccurs="1"
maxOccurs="unbounded"/>
       </xs:sequence>
       </xs:restriction>
       </xs:complexContent>
       </xs:complexType>
  </xs:override>
</xs:schema>
```

We tried to instantiate validating documents for both views that can be processed with the same XSL transformation addressing the content nodes without using wildcards for the view. It is predicted, that the instance validating to the functional view will also validate against the library schema.

Inclusion Approach

The idea is to separate content of a XML instance and validation of it. Validation against multiple XML Schemas makes sense. These can include XML Schema for a specific functional views, for the global view, and for multiple versions of these XML Schemas.
The separation can be realized in a way that the content is in one file, and a wrapper - which enables the validation - is in another file. The wrapper, which is specific to the selected XML Schema, imports the content file.
There are two mechanisms available to import a XML file into another: XML entities and XInclude. XML entities work with all XML processors. XInclude works only with a limited number of XML

processors. Furthermore, there is the limitation that XInclude cannot really be combined with validation according to a XML Schema.Therefore only the XML entity is explored in detail.
The advantage of this approach is that the content of a functional view can be validated against any XML Schema i.e. the XML Schema of a specific functional view, the global view or multiple versions of both.
The disadvantage is that it requires two files and a specific step - the inclusion - to validate the content. The user has to obbey the related documentation.

Code Examples

Content of the functional view Codebook

```
<ddi:CodebookViewContent xmlns:ddi="ddi">
  <ddi:stuff>bla</ddi:stuff>
</ddi:CodebookViewContent>
```

Wrapper for validation of the Codebook content against the XML Schema of the functional view Codebook

```
<!DOCTYPE CodebookViewWrapper [
      <!ENTITY CodebookViewContent SYSTEM "CodebookViewContent.xml">
]>
<view:CodebookViewWrapper xmlns:view="Codebook" xmlns:ddi="ddi"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="Codebook CodebookViewWrapper.xsd">
&CodebookViewContent;
</view:CodebookViewWrapper>
```

Wrapper for validation of the Codebook content against the XML Schema of the global functional view

```
<!DOCTYPE GlobalViewWrapper [
      <!ENTITY CodebookViewContent SYSTEM "CodebookViewContent.xml">
]>
<view:GlobalViewWrapper xmlns:view="Global" xmlns:ddi="ddi"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="Global GlobalViewWrapper.xsd">
&CodebookViewContent;
</view:GlobalViewWrapper>
```

Conclusion and open questions

- We recommend to represent functional views in schema using the substitution approach.
- Technical testing has still to be done, also the implementation within the binding process.
- We discussed, how the used view could be documented within an instance. A first Idea was a fixed value, mandatory "name" attribute for the View element with the value overridden in the definition of the functional view.
- We also discussed that—with the same mechanism—restrictions to content elements could be handled.