XmlSchema 2.0 Design

Data Structures and Approach

An XML schema document is, of course, an XML document. Applications *could* create and modify them by working directly with the DOM. This would be very painful and error-prone.

XmlSchema, instead, defines an object model corresponding to the elements defined by the XML Schema 1.0 specification. (It is hypothetically possible that these objects could, themselves, implement the W3C DOM API. That would be a great deal of work for very dubious return.)

The fundamental goal of the API, then, is to have an object for each schema element, and provide an API on that object to store and retrieve the attributes (and sub-elements) allowed for by the specification.

The specification imposes many validity requirements on schemas. It's desirable for the API to be able to check and enforce validity. There are, however, some limits on this. First, there is the problem of schemas 'under construction'. It would be very difficult to insist that the object graph of a schema had to correspond to a valid schema at every instance. Schema objects have to be assembled in some order, and there are intermediate states. When there is a local, clear, invariant (i.e. you can't have both a name and a ref on an element), the API imposes it.

Schemas and their relationships

A schema document can import content from other schema documents. More subtly, it can reference objects from other schemas when it 'expects' them to be present in the environment, as in the case of multiple schemas in a WSDL. Consider, for example, something like:

```
<xs:element ref="bloop:bleep"/>
```

'bloop' is a namespace prefix referring to the namespace of some schema, and 'bleep' a global element in it. The API tries, when possible, to spare the application the tasks of mapping bloop to a namespace URI, finding the XmlSchema object corresponding to that object, and looking up 'bleep' within it. XmlSchema provides this via the SchemaCollection object.