Achieving Maximum Dynamic Capability in your Schemas

Too often schemas are designed in a static, fixed, rigid fashion. Everything is hardcoded when the schema is designed. There is no variability. This is not reflective of nature. Nature constantly changes and evolves. Nothing is fixed. As a general rule of thumb: more dynamic capability = better schema

Definition of Dynamic: the ability of a schema to change at run-time (i.e., schema validation time). Contrast this with rigid, fixed, static schemas where everything is predetermined and unchanging.

Limiting Dynamic Capability

1. Hardcoding a collection of components to a namespace.

When you bind a schema to a targetNamespace you are rigidly fixing the components in that schema to a fixed semantics (in as much as a targetNamespace gives semantics to a schema).

2. Hardcoding a reference to a type to the implementation of that type.

When you specify in <import> a value for schemaLocation you are rigidly fixing the identity of the schema to implement a type.

Achieving Maximum Dynamic Capability

The key to achieving dynamic schemas is to *postpone decisions as long as possible*. Here are some ways to do that.

1. Don't hardcode a schema to a targetNamespace. That is, create no-namespace schemas. Let the application which uses the schema decide on a targetNamespace that is appropriate *for the application*. Thus we postpone binding a schema to a targetNamespace as long as possible -> until application-use-time. Also, the using-application can <redefine> components in the schema. This is making the schema dynamic/morphable. It is not fixed to one namespace (semantics). See the discussion on Zero One Or Many Namespaces for more info.
2. Don't hardcode the identity of an <import>ed schema. Example, suppose that you declare an element to have a type from another namespace, e.g.,

```
<xsd:element name="sensor" type="s:sensor_type"/>
```

Observe that sensor_type is from another namespace. Thus, this schema will need to do an <import>. Normally we see <import> elements with two attributes - namespace and schemaLocation. However, schemaLocation is actually optional. When you do specify

schemaLocation then you are rigidly fixing the identity of a schema which is to provide an implementation for sensor_type. We can make things a lot more dynamic by not specifying schemaLocation. Instead, let the instance document author identify a schema that implements sensor_type. This creates a very dynamic schema. The type of the sensor element is not fixed, static. Thus we postpone binding the type reference (type="s:sensor_type") to an implementation of the type as long as possible -> until schema-validation-time. See the discussion on Dangling Types for more info.