Enhancing Control of Schema Extensibility Using Named Wildcards

XSD 1.1 is having important changes that allow for better extensibility. However, most of these changes affect the schema that is allowing components to be added to it.

What is missing is a way for schemas specifying extensions to be able to specify where those extensions should be placed in the core schema.

For example, the following schema specifies two locations where extensions are allowed:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                   elementFormDefault="qualified"
                   xmlns="http://example.com/core.xsd"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child1" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Element2">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child2" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

An extension schema may specify an extension as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/extension.xsd"
                   elementFormDefault="qualified"
                   xmlns="http://example.com/extension.xsd"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="ExtensionElement" type="xs:int"/>

</xs:schema>
```

However, it's not clear from the extension schema which of the extension sites in the core schema are affected. Currently the site that an extension is applicable to is specified by narrative text. This has obvious disadvantages including a lack of precision in the definition and the inability of tools to be made aware of the constraints.

Hence, it is proposed that extension sites be labelled, as in the following example:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                  elementFormDefault="qualified"
                  xmlns="http://example.com/core.xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child1" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Element2">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child2" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element2"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Here socket attributes have been added to the two xs:any wildcard components, which label the extension points using NCNames.

Further it is proposed that a plugin component be defined, which is used as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/extension.xsd"
                  elementFormDefault="qualified"
                  xmlns="http://example.com/extension.xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:core="http://example.com/core.xsd">

    <xs:plugin socket="core:Element1">
        <xs:element name="ExtensionElement" type="xs:int"
                maxOccurs="unbounded"/>
    </xs:plugin>

</xs:schema>
```

The plugin component specifies that the contents of its children must be logically inserted into the location preceding the wildcard that is resolved to by the value of its socket attribute (which is a list of QNames).

Hence, after the above plugin element is applied the input core schema is logically treated as if it were:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                   elementFormDefault="qualified"
                   xmlns="http://example.com/core.xsd"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child1" type="xs:int"/>
                <xs:element name="ExtensionElement" type="xs:int"
                        maxOccurs="unbounded"/> <!--but in
                                                namespace 'extension'-->
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Element2">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child2" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element2"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Note that rather than using the ID attribute to specify the sites where components can been plugged in, a new attribute (socket) has been introduced. This is because the values of ID attributes must be unique across an entire schema, whereas it may on occasion be convenient to specify multiple plugin sites that receive the same augmented content and therefore have the same socket name. For example, you could have a core schema such as which has two extension sites with the same name:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                   elementFormDefault="qualified"
                   xmlns="http://example.com/core.xsd"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
```

```
                <xs:element name="Child1" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Element2">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child2" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

And an extension schema as:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/extension.xsd"
                  elementFormDefault="qualified"
                  xmlns="http://example.com/extension.xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:core="http://example.com/core.xsd">

    <xs:plugin socket="core:Element">
        <xs:element name="ExtensionElement" type="xs:int"
                maxOccurs="unbounded"/>
    </xs:plugin>

</xs:schema>
```

The logical equivalent of which would be:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                  elementFormDefault="qualified"
                  xmlns="http://example.com/core.xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child1" type="xs:int"/>
                <xs:element name="ExtensionElement" type="xs:int"
                        maxOccurs="unbounded"/><!--but in namespace
                                                   'extension'-->
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="Element2">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child2" type="xs:int"/>
                <xs:element name="ExtensionElement" type="xs:int"
                        maxOccurs="unbounded"/><!--but in namespace
                                                    'extension'-->
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element2"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Similarly, it may be appropriate to plug the same content into multiple differently named sites. Hence the type of the socket attribute in the plugin element is a list of QNames. As such the following extension schema can be applied to the earlier mentioned core schema with the two differently named extension sites and yield the latter mentioned augmented logical schema:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/extension.xsd"
                  elementFormDefault="qualified"
                  xmlns="http://example.com/extension.xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:core="http://example.com/core.xsd">

    <xs:plugin socket="core:Element1 core:Element2">
        <xs:element name="ExtensionElement" type="xs:int"
                maxOccurs="unbounded"/>
    </xs:plugin>

</xs:schema>
```

Attributes can be handled in the same way. For example, a schema of the following form could be specified:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                  elementFormDefault="qualified"
                  xmlns="http://example.com/core.xsd"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child1" type="xs:int"/>
            </xs:sequence>
            <xs:anyAttribute namespace="##other" processContents="lax"
                    socket="Element1"/>
        </xs:complexType>
    </xs:element>
```

```
</xs:schema>
```

with an extension schema of:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/extension.xsd"
                   elementFormDefault="qualified"
                   xmlns="http://example.com/extension.xsd"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema"
                   xmlns:core="http://example.com/core.xsd">

    <xs:plugin socket="core:Element1">
        <xs:attribute name="ExtensionAttribute" type="xs:int"/>
    </xs:plugin>

</xs:schema>
```

It is proposed that the xs:any and xs:anyAttribute extension points have separate namespaces in the same way the elements and types have different namespaces. Hence a schema of the following form would be legal:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://example.com/core.xsd"
                   elementFormDefault="qualified"
                   xmlns="http://example.com/core.xsd"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="Element1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Child1" type="xs:int"/>
                <xs:any namespace="##other" maxOccurs="unbounded"
                        processContents="lax" socket="Element1"/>
            </xs:sequence>
            <xs:anyAttribute namespace="##other" processContents="lax"
                    socket="Element1"/>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

To implement this proposal a new schema component is required as follows:

```
<plugin
    socket = list of QName
    {any attributes from non-schema namespace}
    >
    Content: (xs:annotation?, (xs:element*, xs:attribute*))
</plugin>
```

The xs:any and xs:anyAttribute schema components need to be modified with the addition of a socket attribute as follows (inserting the socket attribute at the beginning of the list of attributes for clarity purposes only):

```
<any
  socket = NCName
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  namespace = ((##any | ##other) | List of (anyURI | (##targetNamespace
| ##local)) )
  notNamespace = List of (anyURI | (##targetNamespace | ##local))
  notQName = List of QName
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</any>


<anyAttribute
  socket = NCName
  id = ID
  namespace = ((##any | ##other) | List of (anyURI | (##targetNamespace
| ##local)) )
  notNamespace = List of (anyURI | (##targetNamespace | ##local))
  notQName = List of QName
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</anyAttribute>
```

In summary this is a simple proposal that allows formal specification of how one schema extends another schema. Without such facilities such mechanisms must typically be specified using narrative text. Such a narrative approach obviously limits the accuracy of a schema set and does not allow tools to understand such constraints. This proposal fixes this problem in a simple and flexible way.