

XML Schema Best Practices, Namespace, and Versioning Issues

Louis Reich
NASA/CSC
MOIMS/IPR WG Chair
Atlanta, Georgia
September 14, 2005

How Did This Activity Begin

- XML Packaging Structure and Construction Rules(XFDU)
 - Recommendation schedule needs final versions of XFDU Schema and agreement on many XML Schema Elements over the next 3-6 months
 - to allow software development,
 - interoperability testing and
 - development of Best Practices document.
- CCSDS CESH and CMC recognize similar issues at the CCSDS level
 - Resolution: 1: XML Schema Best Practices and Schema Services for XFDUs
 - Resolution 2: Share Results of Schema Study with MOIMS Area Director and other interested CESH and WG personnel
 - Naming Conventions
 - Namespace and Versioning Policies
 - XML Schema extensibility Mechanisms
 - XFDU Schema Services

I thought it would be easy XML schema had been around for four years and tools had matured. Web services had provided the killer app

- **Phase 1: Literature Study**
 - Read and compared a variety of XML Schema Best Practices documents from Space Agencies, US Government Agencies and a computer Software vender.
 - Result was very few Best Practices were unanimously approved in all the documents (we have a seven page matrix containing these rules for future analysis)
 - Also read the seminal papers on Versioning and Extending XML Schemas
 - Result was that the ideas were excellent but the key attributes of mustUnderstand and mustIgnore have not been well understood and were not present in XML Schema 1.0 or planned for XML Schema 1.1
 - Did a quick market scan for XML Component (Schemas, style sheets, dictionaries) Registry/Repository products or services.
 - Result was no obvious products that were not web service oriented. Services such as Microsoft Ebiz and OASIS Registry are no longer maintained
 - Participated by telecon in the W3C Workshop on User Experience with XML Schema.
 - Result was a clear understanding of the areas where the XML Schema had been too poorly specified or where the implementers did not understand the subtle points of the standard.
- Oh well, I was wrong, it wasn't going to be easy

- **Phase 2**
 - **Discovered several excellent 1/2 day and 1day tutorials on Best Practices for Designing XML Schemas which were given at top XML Conferences**
 - **Confirmed many of the Naming and Best Practices for developing reusable and extensible schemas.**
 - **I reviewed several of the current CCSDS recommendations that use XML Schema as a specification notation.**
 - **Though each schema parsed, each sample validated and each recommendation stated their design principals, any editor would be hard pressed to reuse any schema component from a different recommendation**
 - **Held a 1/2 day meeting with several of the NASA Editors of the XML Schema Based CCSDS Recommendations**
 - **Included GSFC engineers and system analysts who would be the users of the schemas to better understand their views**

- **Phase 3**

- **Did intensive study of the Naming and Design Rules that are derived from the UBL, OAG and CEFACT NDRs.**

- This includes the US government, various US agencies such as Navy, Environmental Protection Agency and Internal Revenue Service, several e-business consortia and other national efforts including efforts in the Netherlands and the UK.
- **Result 1; These documents are not a complete panacea though they do have many good ideas. The second version of the UBL Naming and Design Rules and the US Federal Naming and Design Rules are in committee and public review respectively**
- **Result 2: The wide acceptance seems to be a combination of e-business pressures and a desperate need for these Best practices in various E-government initiatives and overlapping membership among the organizations**

- Hold XML Editors and Experts working group meetings to recommend approaches on:
 - Consistent use of qualified vs. unqualified types, elements and attributes-1.5
 - Definition of CCSDS URL hierarchy and namespace architecture. e.g. URN for targetNamespace, URL for schemaLocation 1---Include liasoned Standard/Consortium
 - Definition of Versioning Approaches 2
 - Define various levels of XML Schema reusable components based on the UBL model
 - Look at UBL architecture as a basis for space operations information architecture
 - Is schemaLocation a resolvable URL-!!!!!!!!!!!!!!
 - Naming and Style Guides 1.5

We need to do this before any of our XML Schema based recommendations go blue and their namespaces are locked in concrete

Create a multi-area team to develop an XML schema services and information architecture

- Define an IETF RFC to obtain a CCSDS URN domain
- Look at UBL architecture as a basis for space operations information architecture
- Define various levels of XML Schema reusable components based on the UBL model
- Investigate commercial tools as a basis for CCSDS XML Schema Service(Registry and Repository)
- Develop a CCSDS NDR and Best Practices Document

- Erik Barkley
- David Berry
- Lou Reich
- Gerry Simon-
- CNES TBD
- ESA TBD

All stukees plus Felipe, John P., Leo Hartman, Paul Pechkam

- **Clarification of Goals**
 - **Goal is Guidance not Standards**
 - **Changes to Current Books are at the discretion of Current WG not this group. We can do experiments to demonstrate tradeoffs or “examples” using current areas of knowledge**
 - **Namespace and versioning - low impact**
 - **High - Guidelines and style guide**
 - **Design issues**
 - **Do one function with one construct**
 - **All element and type name are are upper camel case**
 - **All Attribute names are in lower camel case**
 - **Heritage Issues**
 - **NAV XML - Elements all Caps for consistency with ASCII**
 - **SLE Man - Use of nested choice, units attributed for documentation**
 - **XFDU - use of lower camel case for everything, use of substitution groups for extensibility**

- 10:00-10:10 Introduction and Agenda Review
- 10:10-10:30 XML Schema Best Practices->Naming and Design Rules
 - 2001-Roger Costello starts Best Practices Web Site(xfront.com)
 - 2002-2004 - Problems with inconsistent tools and frequent use of JAVA binding tools that do not support full XML Schema cause many organizations to create internal best practice documents limiting the use of some XML Schema constructs. These could be considered “profiles” of XML Schema
 - 2005 - UN/CEFACT NDR based on UBL Published becomes core of many “best Practices/profiles”(see <http://xml.coverpages.org/ni2005-08-19-a.html>)
- 10:30 Specific Issues
 - What are the real CCSDS Requirements
 - CCSDS is not eBIZ, however the basic architecture may apply as is being attempted in US Federal NDR
 - Extensibility
 - Reuse Architecture
 - Schematron
 - Namespace issues
 - URN vs URL
 - IANA Registration Hierarchical definition

- **To optimize schema design** there must be a PLAN. A plan is based on answering questions like:
 - *Most important question:* “What’s the purpose of the XML representation?”
 - Because *Design follows function*
 - “Is reuse of the schemas and XML components (i.e., *XML elements, XML types, etc.*) important?”
 - “Is the goal to create an XML *implementation* or a set of *guidelines* used to create an XML implementation?”
 - “What is the authoritative source used to derive the XML?”
 - “Creating a stovepipe effort or a collaborative solution?”

***“There’s never enough time to do it right,
but there’s always time to do it over”***

- a flexible federal modularity model that defines the structure for creating interoperable schema and schema modules
- a clearly defined namespace scheme that ensures consistency across Agencies
- a versioning scheme that will support consistency in versioning of government schema
- a Federal canonical schema for base Data Types
- specific NDR's by government agencies or communities of practice that build on this document
- a reference to use for a mapping of different agency NDR's to each other

- **consistent, reusable XML components that may be made available for reuse such as:**
 - **Schema**
 - **Schema Modules such as reusable code lists and identifier lists**
 - **Simple and Complex Types**
 - **Elements**
 - **Attributes**
- **a set of tools to facilitate ease of development, validation, and interoperability**

- **Option 1: Change the Schema Version Attribute**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  elementFormDefault="qualified"  
  attributeFormDefault="unqualified"  
  version="1.0" >
```

- **Pros**

- Easy
- Instance documents do not change if schema only extended
- Applications could interrogate 'version' attribute and take action

- **Cons**

- Validator ignores version attribute – not enforceable

- **Option 2A: Change the schemaVersion Attribute of the Root element**

```
<xs:schema xmlns="http://www.exampleSchema"  
targetNamespace=http://www.exampleSchema ..>  
<xs:element name="Example">  
  <xs:complexType>  
    ...  
    <xs:attribute name="schemaVersion"  
type="xs:decimal" use="required" fixed="1.0"/>  
  </xs:complexType>  
</xs:element>
```

- **Option 2A: Change the schemaVersion Attribute of the Root element**
- **Approach A**
 - Defined in Schema as required attribute
 - Instance documents forced to set
 - Schema validator can enforce
- **Advantage**
 - Instances would not be valid without same version
- **Disadvantage**
 - Does not allow instance to be valid against multiple versions

- **Option 2B: Change the schemaVersion Attribute of the Root element**

```
<xs:schema ... version="1.3">  
<xs:element name="Example">  
<xs:complexType>  
...  
<xs:attribute name="schemaVersion" type="xs:decimal"  
use="required"/>  
</xs:complexType>  


---

</xs:element>  
<Example schemaVersion="1.2"  
xmlns="http://www.example"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.example MyLocation\Example.xsd">
```

- **Option 2B: Change the schemaVersion Attribute of the Root element of Instance Document**
 - **Approach B**
 - Version NOT set in root element's attribute in schema
 - Revert to Option 1 (or Similar) to set version in schema
 - Use application logic to verify version in schema vs. instance
 - **Pros**
 - Caters for compatibility across multiple versions
 - App knows about changes
 - **Cons**
 - Extra processing by application

- **Option 3: Change the Schema's targetNamespace**
 - **Approach**
 - Change targetNamespace for each new version
 - **Pros**
 - Application knows about change
 - **Cons**
 - All instance documents forced to upgrade even for minor changes

```
<xs:schema xmlns="http://www.exampleSchemaV1.0"  
targetNamespace="http://www.exampleSchemaV1.0"  
xmlns:xs="http://www.w3.org/2001/XMLSchema"  
elementFormDefault="qualified"  
attributeFormDefault="unqualified">
```

- **Option 3: Change the Schema's targetNamespace**
 - **Approach**
 - Change targetNamespace for each new version
 - **Pros**
 - Application knows about change
 - **Cons**
 - All instance documents forced to upgrade even for minor change

```
<xs:schema xmlns="http://www.exampleSchemaV1.0"  
targetNamespace="http://www.exampleSchemaV1.0"  
xmlns:xs="http://www.w3.org/2001/XMLSchema"  
elementFormDefault="qualified" attributeFormDefault="unqualified">
```

- **Option 4: Change the Schema's Name/Location**
 - **Approach**
 - Change schemaLocation
 - **Pros**
 - None
 - **Cons**
 - schemaLocation is not enforceable
 - Some consider it a security risk!



Roger Costello's Automated Versioning

- **Recommendation 1:** To avoid breaking namespace-aware applications with each new version of an XML Schema use the same namespace for all versions.
- **Recommendation 2:** To prevent breaking old instance documents give the new Schema version a different filename or a different URL location or both.
- **Recommendation 3:** To facilitate an application in recognizing that an element's content has changed, don't use anonymous types. Instead, use named types.
- **Recommendation 4:** If you change a type when you create a new version of a Schema then give the type a different name.



Roger Costello's Recommendation on Versioning

- **Recommendation 5:** Change the name of an element's type only if its *immediate* content has changed.
- **Recommendation 6:** Use a version attribute on the root element. If an instance document is a compound document - that is, an assembly of XML fragments - then place a version attribute on the root of each fragment.
- **Recommendation 7:** Applications should use the tag names to locate data in instance documents. Applications should be designed to anticipate that the order of tags may change.
- **Recommendation 8:** Define a system-wide protocol (e.g., fault reporting mechanism) to be used when an application is unable to process an instance document it receives from another application.

- **Create Own Datatypes..**
 - **Specify complexType**
 - Composed of other types
 - **Specify simpleType**
 - Enumerations
 - Patterns
 - etc
- **Re-use..**
 - **Derive by restriction**
 - **Derive by extension**
 - **Element substitution**
 - **etc.**

XML Schema Main Re-usable Components

- Schema
- XML Element
 - Defined in terms of one or more XML Elements, complexTypes, or simpleTypes
- ComplexType
 - Defined in terms of one or more XML Elements, complexTypes, or simpleTypes
- SimpleType¹
 - Defined as a “simple” or “base” type, i.e., a “string”, “float”, etc.
 - Can be a union of named simpleTypes
- Attribute¹
 - Must be defined as a simpleType
- Annotation (documentation)

.....
Note 1: simpleTypes and Attributes may be defined by a “user-defined type”

- Element declarations are in terms of a type definition
 - simpleType or a collection of other Elements or types using the complexType definition
- An XML Element may contain an in-line definition:


```
<element name="Count_of_Aircraft" type="xsd:integer"/>
```
- Or an XML Element can be defined by a type definition that is located "outside" itself:

```
<element name="Count_of_Aircraft"
type="SomeDB:Count_Of_Items"/>
```

This example uses a Namespace qualifier to link to a different XML Schema

**SomeDB
Schema**

```
<simpleTypeName="Count_Of_Items">
  <restriction base="xsd:integer">
    <minInclusivevalue="1"/>
    <maxInclusivevalue="999">
  </restriction>
</simpleType>
```



XML Elements and Attributes

- Debate on the distinction between them
 - One COI's Elements might another COI's Attributes
- No rules to distinguish between them
- Elements are more flexible than Attributes
 - Can be defined as complex or simple types
 - A complexType can be a collection (e.g., “sequence”, “all”., etc.) of Elements or types
 - A simpleType must be defined as an XML base type like “string”, “float” etc., or in terms of another simpleType that is an XML base type
 - Attributes
 - Must be defined as simpleTypes
 - Can't directly Attribute an Attribute

Conclusion: Since Attributes have so many restrictions, they should be used only for meta-data

XML Element and Type Definitions

- XML Elements aren't as flexible as XML type definitions. For example, assume:
 <Element name= "Count_Of_Aircraft" type="xsd:integer"/>
- Another Element *can't* re-use this definition directly:
 <Element name= "Aircraft_Quantity" type="Count_Of_Aircraft"/> is ILLEGAL
- Can re-use type definitions

<Element name= "Count_Of_Aircraft"
 type="Count"/>

<Element name= "Aircraft_Quantity"
 type="Count"/>

```
<simpleTypename="Count">
  <restriction base="xsd:integer">
    <minInclusivevalue="1"/>
    <maxInclusivevalue="999">
  </restriction>
</simpleType>
```

XML Element and Type Definitions (Cont.)

- **Re-use of the simpleType definition level rather than at the XML Element Level has consequences:**
 - Lose the knowledge that there is an equivalency between the XML Elements.
 - Also lose any of the original Element's semantic information (Annotation)
 - However, the ease of re-use makes type definitions more flexible and re-usable by far
 - Retains any semantic meaning (e.g., any additional explanations) that might be available at the type definition level

Conclusion: Don't create Elements with in-line definitions. Instead create type definition outside the XML Element to make these definitions available for re-use.

- XML's Import
- XML's Include
- XML's ref
- XML's Substitution Groups
- XML's Attribute Groups
- XML's Restriction
- XML's Extension
- XML's Model and Attribute Group Definitions
- XML's Redefine
- etc.
- Cut and Paste

- **XML “Import”**
 - **Link to other namespaces (can be another Schema). Acts like a pointer to a specific information object(s) in the linked namespace(s)**
 - **Maintains semantic information of original information**
 - **Maintains meta-data about source and versioning**
 - **In future, might be possible to have changes to the original schema roll-over to re-used components (CM issue)**
- **XML “Include”**
 - **Creates one logical Schema from multiple namespaces (can be other Schema(s))**
 - **Same advantages as “Import”**
 - **If resultant Schema is very large, might impact processing speed**

- XML's "import" or "include" (continued)

- Can re-use Global elements and type definitions
- Any elements and type definitions that exist as part of another definitions can't be re-used directly

```
<element>
```

```
  <element/>
```

```
  <element />
```

```
  <element />
```

```
</element>
```

} These elements can't be pulled out individually and be re-used

- XML's "ref"

- Can re-use Global elements

- <element ref="Element_Name" minOccurs="1"/>

- Can't change the XML Element's name

- Can't change the XML Element's composition

- Only "minOccurs", "maxOccurs", "id", and "annotation" allowed

- **XML's Substitution Groups**
 - Can modify the original XML Element: change name, restrict or extend the definition
 - Can have multiple members
 - Can't change the type (e.g., can't change integer to float)
 - Limited to Global XML Elements

- **XML's "extension base"**
 - "Adds" the original definition to the new element
 - Can't alter the original definition
 - Can't use XML elements to extend other elements

- XML's "extension base" Example:

```
<xsd:complexType name="base.type">
  <xsd:sequence>
    <xsd:element name="free_text" type="free.text" minOccurs="0"/>
    <xsd:element name="information" type="information"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="airspace.shape">
  <xsd:complexContent>
    <xsd:extension base="base.type">
      <xsd:sequence>
        <xsd:element name="track_info" nillable="true">
          <xsd:complexType>
```

...

Result: the complexType "airspace.shape" contains the two elements defined in the complexType "base.type"

Schema Capabilities for Re-use (Cont.)

- XML's "restriction base"
 - Can add restrictions to the original definition
 - In XML-MTF, most common usage of restriction was to specify legal values and ranges; for example:

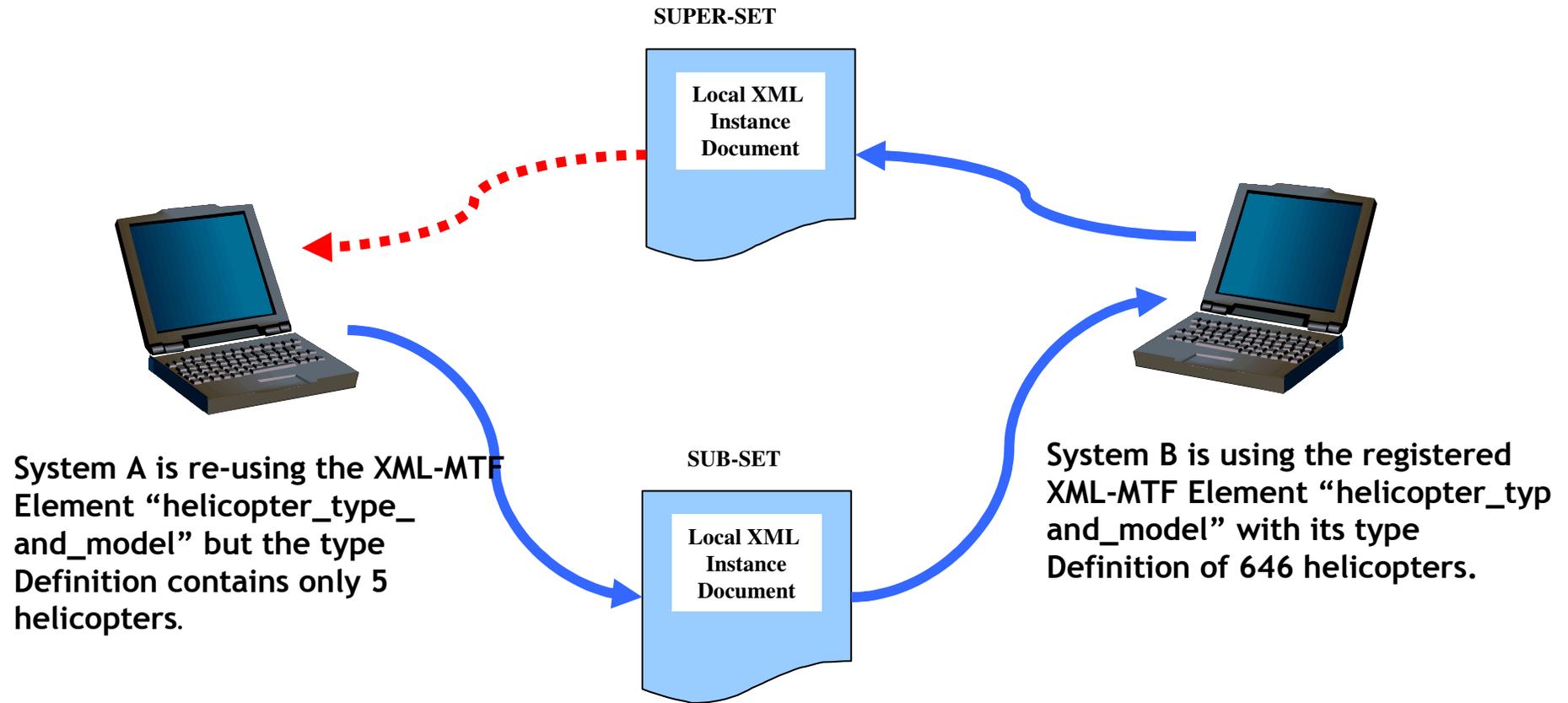
```
<xsd:simpleType name="latitude.degrees.17.1">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="00"/>  
    <xsd:maxInclusive value="90"/>  
    <xsd:pattern value="[0-9]{2}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

- **Cut and paste**
 - Just copy a registered XML component (element, type definition, etc.,) and paste into new Schema
- **Consequences:**
 - Lose logical link that identifies the original namespace of the information
 - Lose versioning information (if any)
 - Won't be updated when original schema is revised (if the CM process does someday supports this function)

Conclusion: The XML “Import/re-use” capabilities are preferable to the “cut and paste” method to access reusable components located in other namespaces

Impact of Restricting and Extending

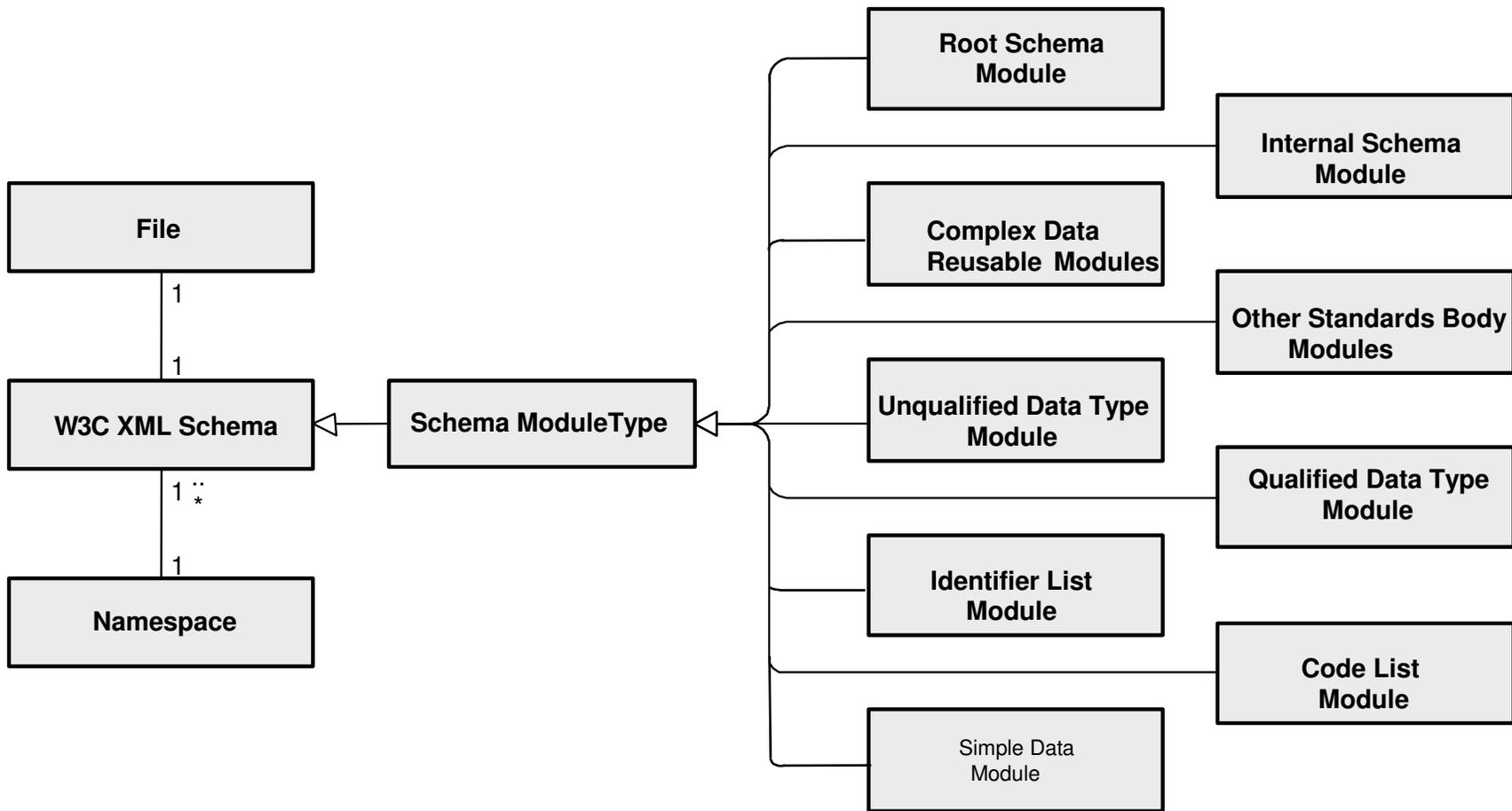
From the “Super -set” to the “Sub-set”: May be INVALID for the system using the sub-set



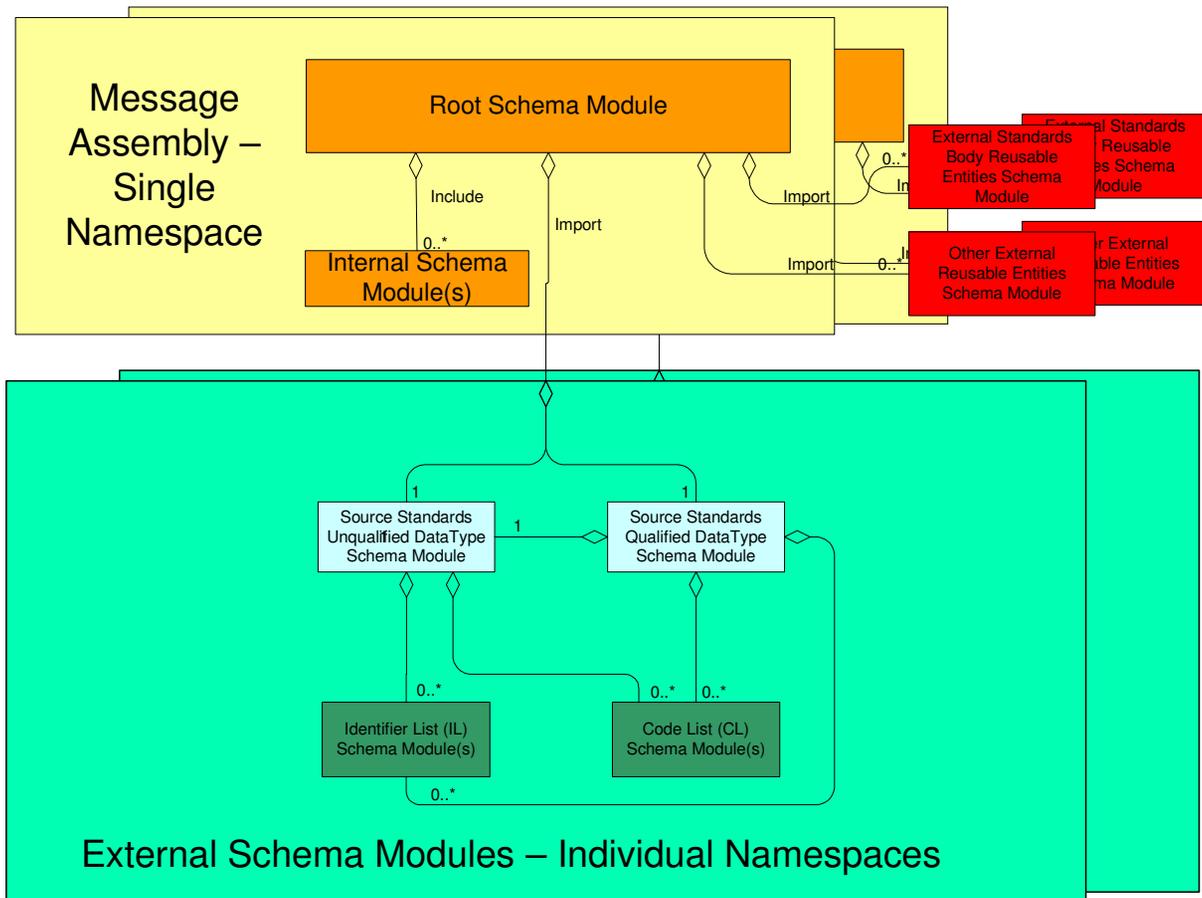
From the “Sub-set” to the “Super-set”: Will be valid for both end systems

- **XML provides many capabilities that support re-use**
 - Many depend on using global definitions
 - Each re-use method has pros and cons
 - Elements are not as flexible to re-use as type definitions
- **Other re-use methodologies (cut and paste) will be used**
 - May have an impact on interoperability
- **CM of the re-used Schema components is going to be an issue**
- **Thinking re-use should impact your Schema design**

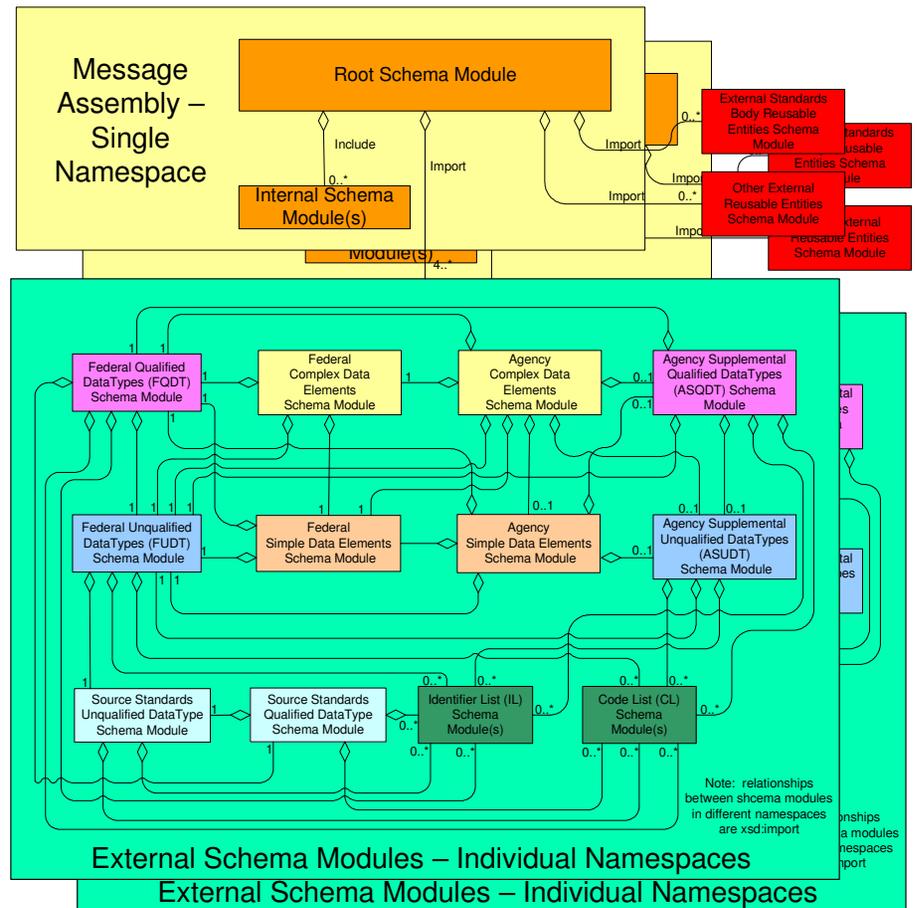
Modularity Model – Reuse Approach



- UBL and other standards bodies are converging on UN/CEFACT Data Type schema modules
- UBL and other standards bodies are converging on single approach to code lists



- Government organizations and initiatives can leverage standards body and federal schema
- Organizations can create organizationally unique
 - Qualified and unqualified Data Types
 - Complex Data Element reusables
 - Simple data element reusables
 - Code Lists
 - Identifier Lists



- **What About the Following Scenarios..**
 - **Element Dependencies..**
 - If value of <minimum> greater than value of <maximum>
 - **Attribute Dependencies..**
 - If value of attribute travel mode is water value of <Transportation> is either ship or boat
 - If value of attribute travel mode is air value of <Transportation> is plane
 - If value of attribute travel mode is land value of <Transportation> is either car or bicycle
 - **Cross-document Dependencies..**
 - Compare value of doc1 <PaymentReceived> with doc2 <PaymentDue>

- **Possible Approaches..**
 - **Option 1: Supplement with another schema language**
 - E.g. Schematron
 - **Option 2: Write code using conventional programming language**
 - **Option 3: Use XSLT**

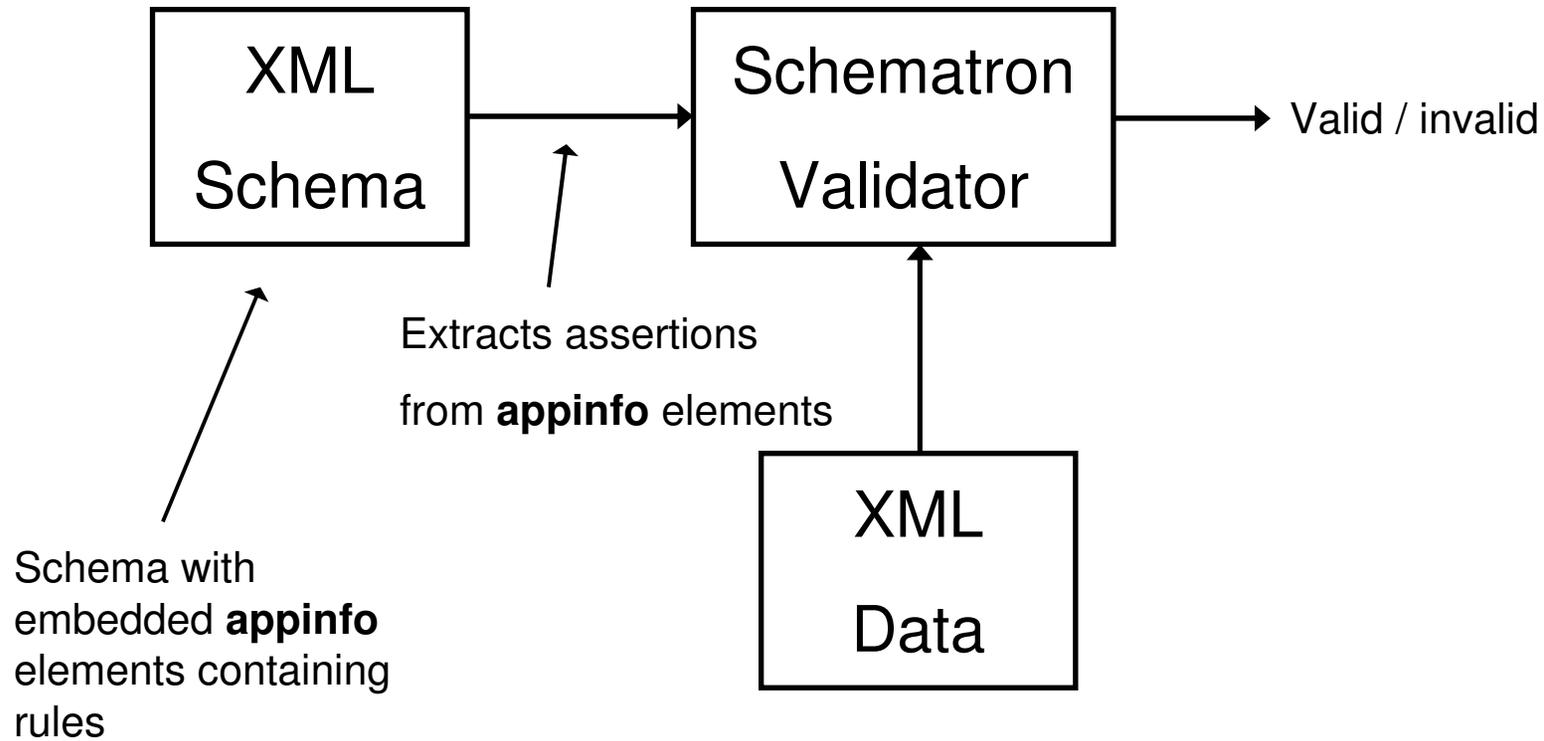
- **Using Schematron..**
 - **Language for making assertions about patterns**
 - **Relies heavily on XPath expressions**
 - **Requires use of XML Schema `xsd:appinfo` element**
 - **Provides machine readable comment to programs**

XPath Expression

```
<xsd:annotation>  
  <xsd:appinfo>  
    <assert test="A &gt; B" >A greater than B</assert>  
  </xsd:appinfo>  
</xsd:annotation>
```

↑
Emitted if assertion is false

- Processing Model



- appinfo elements extracted by schematron
- **Schematron defines assert/report elements**
 - “test” attribute is XPath expression

```
<xs:annotation>
  <xs:appinfo>
    <sch:pattern id="onLoanTests">
      <sch:rule context="bk:book">
        <sch:report test="@on-loan and not(@return-date)">
          Every book that is on loan must have a return date
        </sch:report>
      </sch:rule>
    </sch:pattern>
  </xs:appinfo>
</xs:annotation>
```

- **Schematron.Net**
 - **.Net Implementation – Validator class**

```
using System;  
using System.Xml;  
using NMatrix.Schematron;
```

```
Validator validator = new Validator();  
validator.AddSchema("books.xsd");  
validator.Validate(new  
XmlTextReader("books.xml"));
```

```
} catch (Exception e) {  
    Console.WriteLine(e);  
}
```

```
}
```

```
}
```



- Using XSLT

