Comparing and Diffing XML Schemas

Priscilla Walmsley, Datypic, Inc.

Balisage

August 13, 2015

Course Outline

Introduction

The project

- Build a tool to identify the differences between two schemas.
- For each difference, determine whether it is backward compatible.
  - Or, determine whether one schema is a "subset" of another schema.
- Show the results in as simple/concise a report as possible.

OH in the trenches

Schema developers on internal project teams:

“The changes were really minor, I think all I did was add element `z`.”	“Don't worry. They shouldn't affect your code.”	“The tool was acting kind of strange when I saved the schemas. Hopefully it didn't break anything.”

Designer of XML-based standard:

“Let's hand-compile a list of all the changes we made in version 3.1.”	“When we redesigned the schemas from Russian Doll to Venetian Blind, I hope we didn't change what's valid.”	“All our changes in version 3.1 *should* be backward compatible with 3.0. We were pretty careful.”

Implementer of XML-based standard:

“Hmmm, I wonder what exactly changed in version 3.1. These release notes are kind of vague.”	“To be conformant to this standard I have to use a strict subset. Guess I'll start deleting things from the schema and hope for the best.”	“Wish I had a list of all the extensions I added to this standard, so I could communicate it to the Java developers.”

Simplifying assumptions

- XSD 1.0 only
  - Schemas must be a valid, coherent set.
  - No support (yet) for certain XSD features, for example:
    - redefine/override
    - chameleon includes
    - identity constraints
  - No attempt to parse regexes (xs:pattern) or XPaths (identity constraints, assertions) to determine backward compatibility.
- No attempt to compare more than two things at a time.
- We only care about what is valid in an instance.
  - Documentation, schema constructs and design patterns are ignored.
- Diffs only need to be shown at one level.
  - A change to a leaf element affects all its ancestors, but it only needs to be shown as a diff once.

Note to self

- The diffing doesn't have to be perfect!
  - Dramatic changes are unlikely between versions.
  - Cover the most common ~75% of cases.
  - The worst case scenario is to show a complete deletion and insertion of the content model.
- It doesn't always have to be able to determine backward compatibility.
  - "I don't know" is an acceptable answer.

"Data-oriented" example: NIEM

- National Information Exchange Model
- Large (10,000-element) model used mostly by US federal, state and local government
- Characteristics
  - Highly structured content models
    - No mixed content.
    - Heavy use of type inheritance and substitution groups.
    - All elements and types are global.
    - The outermost model group is always `sequence`.
    - Many, many namespaces.
  - Strict rules for what XSD constructs are allowed/disallowed.
  - Role of an XML Schema Diff tool:
    - Ensure that your NIEM subset is indeed a subset of NIEM.
    - Ensure that your "constraint" schema allows a subset of your base schema.
    - Track changes to your NIEM-based extension schemas over time.
    - Find out what's new in a new version of NIEM.

"Document-oriented" example: JATS

- Journal Article Tag Suite
- Characteristics
  - Narrative content
    - Lots of mixed content.
    - Mostly global elements, local types.
    - Slightly more complex content models than NIEM.
    - No namespace (except for incorporating MathML, etc. namespaces).
  - Three tags sets with different levels of strictness (Archiving, Publishing Authoring)
  - Role of an XML Schema Diff tool:
    - Find out what's new in a new version of JATS.
    - Create a restriction of JATS and ensure that it allows a strict subset.
    - Analyze the differences between the three tag sets.
  - Examples in this presentation show the diffs between JATS 1.0 and JATS 1.1d3 MathML2.

The approach

- Use XSLT. (What else?)
- Use a pipeline (in Ant) to break it down into several simpler steps.
- Build it "iteratively". Which means:
    1. Write some code.
    2. Run it.
    3. Bang head against wall.
    4. Reread note to self.
    5. Goto 1.
- Current status: basic working code, much untested code, long list of future enhancements.

Steps

1. Canonicalize
2. Flatten
3. Simplify
4. Sort
5. Find Diffs
6. Determine Backward Compatibility
7. Report

Part 1: Simplification

Step 1: Canonicalize

Purpose: Make schema documents more similar to make comparison easier.

- fill in default values for attributes
- normalize attribute values
- eliminate anything not relevant to validation (documentation, schema-only features like `final`)

Output is valid XSD schema documents with the same meaning.

```
<xs:element name="foo" minOccurs="01" maxOccurs=" 12" id="foo" dty:doc="foo"/>
```

```
<xs:element name="foo" minOccurs="1" maxOccurs="12" nillable="false" type="xs:anyType"
 form="qualified"/>
```

Gory details:

- fill in default values (canonical representation) for all attributes.
- normalize space in attribute values (except where it could be significant, e.g. `pattern` or `enumeration`).
- normalize attribute values based on type, e.g. `@minOccurs` is an integer, so use canonical representation of integer.
- move schema document-level defaults (e.g. `@elementFormDefault`, `@blockDefault`) down to individual components.
- eliminate anything not relevant to validation (`@id`, `@version`, `@final`, `annotation`, non-native attributes).
- add generic types to declarations that have none.
- `@mixed` goes on `complexType`, not `complexContent`.
- all `complexType`s have `simpleContent` or `complexContent`.
- convert `length` into `minLength` and `maxLength`.

Step 2: Flatten

Purpose: Normalize out all the XSD constructs to get down to the essential properties and content model.

- resolve all complex type extensions and restrictions
- resolve all groups and attribute groups
- turn substitution groups into choices
- resolve all simple type restrictions (merging facets), back to primitive type

Output is a single "schema document" that is not valid XSD.

```
<xs:sequence minOccurs="1" maxOccurs="1">
   <xs:element name-ns="ns1" name="a"/>
   <xs:element name-ns="ns1" name="b"/>
   <xs:element name-ns="ns2" name="c"/>
   <xs:element name-ns="ns2" name="d"/>
</xs:sequence>
```

```
<xs:simpleType>
  <union xmlns="">
      <atomic primitive="string">
         <xs:pattern value="-?([0-9]+|[0-9]*\.[0-9]+)*(em|ex|px|in|cm|mm|pt|pc|%)?"/>
      </atomic>
      <atomic primitive="string">
         <xs:enumeration value="medium"/>
         <xs:enumeration value="thick"/>
         <xs:enumeration value="thin"/>
      </atomic>
   </union>
</xs:simpleType>
```

Step 3: Simplify

Purpose: further simplify the content models by eliminating unnecessary model groups (sequence, choice, all)

```
<xs:sequence>
 <xs:sequence>
  <xs:element name="a"/>
  <xs:element name="b"/>
 </xs:sequence>
 <xs:sequence minOccurs="0">
  <xs:element name="c" minOccurs="0"/>
  <xs:element name="d" minOccurs="0"/>
 </xs:sequence>
 <xs:choice minOccurs="0"
            maxOccurs="unbounded">
  <xs:element name="e"/>
 </xs:choice>
 <xs:choice maxOccurs="2">
  <xs:element name="f" maxOccurs="3"/>
 </xs:choice>
 <xs:choice>
  <xs:element name="g"/>
   <xs:choice>
    <xs:element name="h"/>
    <xs:element name="i"/>
   </xs:choice>
 </xs:choice>
 <xs:sequence/>
 <xs:sequence maxOccurs="0">
  <xs:element name="j"/>
  <xs:element name="k"/>
 </xs:sequence>
 <xs:element name="l" abstract="true"
             minOccurs="0"/>
</xs:sequence>
```

```
<xs:sequence>

 <xs:element name="a"/>
 <xs:element name="b"/>


 <xs:element name="c" minOccurs="0"/>
 <xs:element name="d" minOccurs="0"/>



 <xs:element name="e" minOccurs="0"
                      maxOccurs="unbounded"/>

 <xs:element name="f" maxOccurs="6"/>

 <xs:choice>
  <xs:element name="g"/>

  <xs:element name="h"/>
  <xs:element name="i"/>
 </xs:choice>









</xs:sequence>
```

Gory details:

- Some can be eliminated outright:
  - maxOccurs="0" (elements, wildcards or groups)
  - Any empty model group
  - An abstract element
- Some can be eliminated and their children processed:
  - Any model group with cardinality 1..1 that has only one child
  - Any model group with cardinality 1..1 whose parent is the same kind of model group
  - A `sequence` or `all` with cardinality 0..1 whose parent is the same kind of model group and whose children are all optional
- Some can be eliminated and their cardinalities merged with their children:
  - Any group with one child (minOccurs and maxOccurs are products of group and child)
- Some min/maxOccurs can be "canonicalized":
  - Sequence group that is optional and all children are optional - minOccurs can be 1
  - Choice, all group that is optional and at least one child is optional - minOccurs can be 1
  - Particle within a choice that is maxOccurs="unbounded" doesn't also need to be repeating
- Some can be merged
  - Element referenced twice in the same group (consecutively in the case of sequence)
  - Wildcards that appear in the same group (consecutively in the case of sequence)

Step 4: Sort

Purpose: make it easier to compare components line by line.

- Sort all things where order doesn't matter:
  - values in the namespace attribute of wildcards
  - global components in the schema
  - facets on a simple type
  - particles within a choice or all group

(In retrospect, only the last one was relevant to code further in the pipeline.)

```
<xs:choice>
   <xs:element name="foo"/>
   <xs:any namespace="ns2 ns3 ns1"/>
   <xs:element name="bar"/>
</xs:choice>
```

```
<xs:choice>
   <xs:element name="bar"/>
   <xs:element name="foo"/>
   <xs:any namespace="ns1 ns2 ns3"/>
</xs:choice>
```

Part 2: Diffing

Step 5: Diff

Purpose: Determine (at a detailed level) what changes were made.

- Top-level unit of diffing is a global element or attribute declaration.
  - Named types and local elements/attributes are only compared in the context of a particular global declaration.
- Easy enough to find all the deleted and inserted global declarations.
- For matching declarations (same qualified name before and after):
  1. Compare basic attributes (nillable, fixed, etc.)
  2. Compare basic attributes of the associated types (mixed, block, etc.)
  3. Compare content
     - Simple content is fairly straightforward comparison of primitive types and facets
     - Complex content - argh. Reread note to self.
  4. Compare attributes

Diffing complex content

Given two lists of particles:

- If list1[1] is comparable* to list2[1], compare them:
  - If they are identical, don't even bother to show them in the output
  - If they have some differences, show them as a "modification" (side-by-side insertion and deletion)
- Otherwise, if list1[1] is comparable to something anywhere in list2, consider list2[1] an insertion.
- Otherwise, if list2[1] is comparable to something anywhere in list1, consider list1[1] a deletion.
- Otherwise, it is a separate deletion AND an insertion (not shown side by side like a modification is).

*definition on next slide

`<xs:element name="a"/>`
`<xs:element name="b"/>`
`<xs:element name="c"/>`
	`<xs:element name="a"/>`
	`<xs:element name="c"/>`
	`<xs:element name="d"/>`

...

- element b

...

+ element d

Defining what is "comparable"

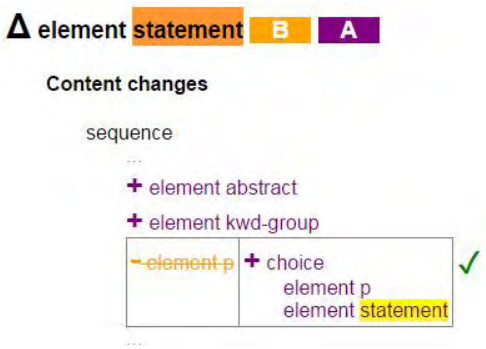
The following particles are "comparable":

- Two element declarations with the same qualified name.
- Two element wildcards.
- Two groups of the same kind (sequence, choice or all) that have at least one descendant element declaration in common.
- Two groups of different kinds that have ALL element declarations in common.
- An element and a group that has that element as a descendant.

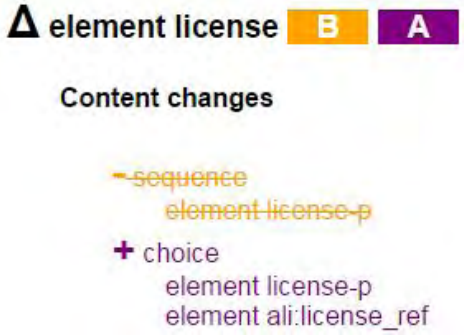
Why bother with the concept of "comparable"?

```
<xs:sequence minOccurs="1" maxOccurs="1">
 <xs:element ref="label" minOccurs="0"
                         maxOccurs="1"/>
 <xs:element ref="title" minOccurs="0"
                         maxOccurs="1"/>




 <xs:element ref="p" minOccurs="1"
                     maxOccurs="unbounded"/>
 </xs:sequence>
```

```
<xsd:sequence minOccurs="1" maxOccurs="1">
 <xs:element ref="label" minOccurs="0"
                         maxOccurs="1"/>
 <xs:element ref="title" minOccurs="0"
                         maxOccurs="1"/>
 <xs:element ref="abstract" minOccurs="0"
                        maxOccurs="unbounded"/>
 <xs:element ref="kwd-group" minOccurs="0"
                         maxOccurs="unbounded"/>
 <xsd:choice minOccurs="1"
             maxOccurs="unbounded">
  <xs:element ref="p" minOccurs="1"
                      maxOccurs="1"/>
  <xs:element ref="statement" minOccurs="1"
                              maxOccurs="1"/>
 </xsd:choice>
</xsd:sequence>
```



Where my definition breaks down:



Step 6: Determine backward compatibility

Four possible outcomes:

1. # "`true`": there is no impact on validation, and no significant differences in the PSVI
2. ! "`true-val`": there is no impact on validation, but the PSVI might be different
   - change simple type from `decimal` to `float`
   - change simple type from `anySimpleType` to `string`
   - change the order of member types in a union type from `integer string` to `string integer`
   - add a default or fixed value
   - change `whiteSpace` facet
3. **X** "`false`": validation is impacted in some or all cases
4. "": unknown; cannot be determined by the current code

"Subset" is the other side of the same coin?

- If Schema A is backward compatible with Schema B, then Schema B is a strict subset of Schema A. True?

Backward compatible changes: Examples

Declaration Properties

- Changing `nillable` from false to true
- Changing `abstract` from true to false

Type Properties

- Changing `mixed` from false to true
- Changing `block` from `extension restriction` to just `restriction`

Complex Content

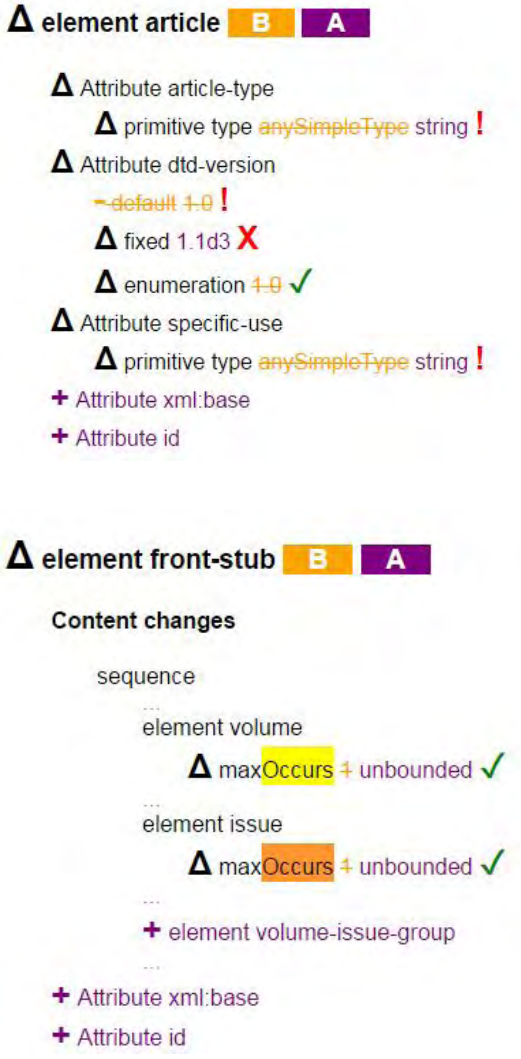
- Making occurrence constraints less restrictive.
- Replacing a sequence with a choice (with all the same declarations).
- Replacing an element with a choice that contains that element.
- Replacing an element with a wildcard.

Simple Content

- Making facets less restrictive.
- Changing an atomic or union type into a list or union of that atomic type

Attributes

- Adding optional attributes.
- Making attributes' types or attributes less restrictive.



Step 7: Report

Previous steps generate a diffs.xml file that can be turned into a report.

```
   <global-decl type="element" chgtype="Changed" name="statement">
      <content>
         <sequence name="">
            <skip/>
            <skip/>
            <change type="ContentModel" chgtype="Added" bc="true">
               <element name="abstract"/>
            </change>
            <change type="ContentModel" chgtype="Added" bc="true">
               <element name="kwd-group"/>
            </change>
            <change type="ContentModel" chgtype="Changed" bc="true">
               <change type="ContentModel" chgtype="Deleted">
                  <element name="p"/>
               </change>
               <change type="ContentModel" chgtype="Added">
                  <choice>
                     <element name="p"/>
                     <element name="statement"/>
                  </choice>
               </change>
            </change>
            <skip/>
         </sequence>
      </content>
      <change type="Attribute" name="content-type" chgtype="Changed">
         <content>
            <change type="Property" name="primitive type" before-value="anySimpleType" after-
value="string" chgtype="Changed" bc="true-val"/>
         </content>
      </change>
      <change type="Attribute" name="specific-use" chgtype="Changed">
         <content>
            <change type="Property" name="primitive type" before-value="anySimpleType" after-
value="string" chgtype="Changed" bc="true-val"/>
         </content>
      </change>
      <change type="Attribute" name="xml:base" chgtype="Added"/>
   </global-decl>
```

Reporting approach

One HTML document with all diffs for a schema.

Default approach (versioning focus):

- Identical parts of content model are omitted for brevity.
- Diffs are shown like change markup, strikethrough and +/-/#- imply "before" and "after".
- Backward compatibility icons are shown.
- Example (JATS 1.0 vs. JATS 1.1d3): http://www.datypic.com/events/diffs.html

Alternate view:

- Entire content model is shown for clarity.
- Diffs are shown with colors (blue/red) to take away versioning implication.
- Backward compatibility icons are not shown.
- Example (JATS 1.0 vs. JATS 1.1d3): http://www.datypic.com/events/diffs2.html

Integrated with Schema Central: JATS 1.0, JATS 1.1d3.

Conclusions

Conclusions

- The diffing is not perfect but works well enough for my purposes.
- The backward compatibility of content models needs to be reworked, to be evaluated on the type as a whole rather than individual "compares".
- Flattening has pros and cons:
  - Pro: Less complex, requires less knowledge of XSD structure.
  - Con: Repetitive, e.g. if a change is made to a base type and shows up in 15 different places.

Future directions

- Change algorithm for determining backward compatibility of content models.
- Rethink diffs.xml structure/vocabulary.
- Improve readability of report.
- Fix known bugs, unsupported corner cases.
- Support other schema languages: similar concepts apply.
- Support more XSD features, or at least ignore them more gracefully.
- Improve efficiency.

Thank you

Questions? Let's discuss.

...and you can contact me at `pwalmsley@datypic.com` to ask questions or request the code.