XML Schema and Validation Approaches

September 18, 2007

Shahram Orandi
Image Group

What’s on the menu…

- Brief look at origin of markup languages
- XML validation approaches and origins
- Benefit / pitfall comparison

A Brief Look at History

Everything that has happened so far…

- First…there was GML (~1960s)
- Then came SGML…(~1980s)
- Then came XML (~1990s)
  - Initial Standard Included Basic Validation (DTD)
- Then came XML Schema (2001)
  - Offered Better Validation

Markup Languages

- A traditional text data stream may look like this:

```
John Doe 65000
```

(14 bytes total)

- This same data stream when marked up can look like this:

```
<employeename>John Doe</employeename>
<salary>65000</salary>
```

(60 bytes total)

- Cost is higher, but benefits are many

Validation

Validation

- How can we make sure salary is valid?

```
<salary>65000</salary>        ✓

<salary>$65000</salary>       ?

<salary>65000.00</salary>     ?

<salary>65k</salary>          ?
```

Popular Validation Options in the Early Days

- Standard: XML DTD (Document Type Definition), part of the XML 1.0 spec.
- Proprietary: Write your own code or COTS

DTD (Document Type Definition)

- DTD is part of the XML spec, but limited:

```
...
<!ELEMENT employee (name,salary)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>

...
Where #PCDATA = parsed character data
                          (string)
```

- Basically checks if something is there or not.

Code it Yourself

- Write your own validation code

```
If IsCurrency(sSalary$) and
    val(sSalary$)> 0 and
    val(sSalary$)< l_MaxSalary then
    return True
else
    return False
endif
```

- It takes lots of code to validate data

Then Came XML Schema

- Ratified a few years after 1.0 spec
- Both XML Schema and DTD allow: Element nesting, attribute types/defaults, element occurrence constraints.
- XML Schemas adds much more: User defined types, namespaces, better data constraints, etc.

Salary Validation Revisited

- Lets tighten up the rules with XML Schema:

```
<xs:attribute name="salary" type="xs:integer">
    <xs:annotation>
            <xs:documentation>Specifies a
            salary.</xs:documentation>
    </xs:annotation>
</xs:attribute>
```

How strict do you want to be?

- What if someone sends over the wire “65000”? Or “65000.01”?
- We could loosen rules a little:

```
<xs:attribute name="salary" type="xs:decimal">
```

  Allows “65000” or “65000.01”

- Or relax things completely…

```
<xs:attribute name="salary" type="xs:string">
```

  Allows “65000”, “65000.00”, “$65000” or “65k”… but everything else may come through as well…

Validation Challenges

Validation Challenges: Off-Spec Data

- Would ideally be relaxed enough to allow valid-but-off-spec transactions that otherwise would be rejected with strict validation:

  65000 ok! 65000.00 ok! $65000 ok!

- Too lax and you may allow ambiguous or incorrect transactions through as well:

  You might let “-65000.%” through

Validation Challenges: Mapping Asymmetry

	Conventional(Legacy)	XML Standard
1		

(Easiest Case, One to One Mapping… Life is good!)

2		

(XML side is superset of legacy, will you accept legacy transaction?)

3		

(Legacy is a superset, will you keep extra info? reject transaction?)

Looking at some options

XML + No Validation: Not going to happen.

- What it is: Hope all data coming down the wire was constructed properly, cross fingers.
- Benefits:
  - Not much… maybe some development time savings?
- Pitfalls:
  - Format errors, missing/ambiguous data, disasters of grand scale.

Looking at some options (cont'd)

XML + Custom Code Validation

- What it is: Build your own validation into business logic to verify data
- Benefits:
  - Flexibility, genetic diversity
- Pitfalls:
  - Redundant work, genetic diversity, as rules change you need to keep up, lots of effort (code)

Looking at some options (cont'd)

XML + DTD

- What it is: A liberal contract on data format and structure
- Benefits:
  - Simple, standard, centralized
- Pitfalls:
  - Simple (limited)... Much of higher level validation has to be implemented in redundant code

Looking at some options (cont'd)

XML + XML Schema

- What it is: A contract (liberal or strict) on data format and structure
- Benefits:
  - Comprehensive, centralized, saves code
- Pitfalls:
  - Going too strict can cut certain parties out, may lock everyone in… (continued on next slide)

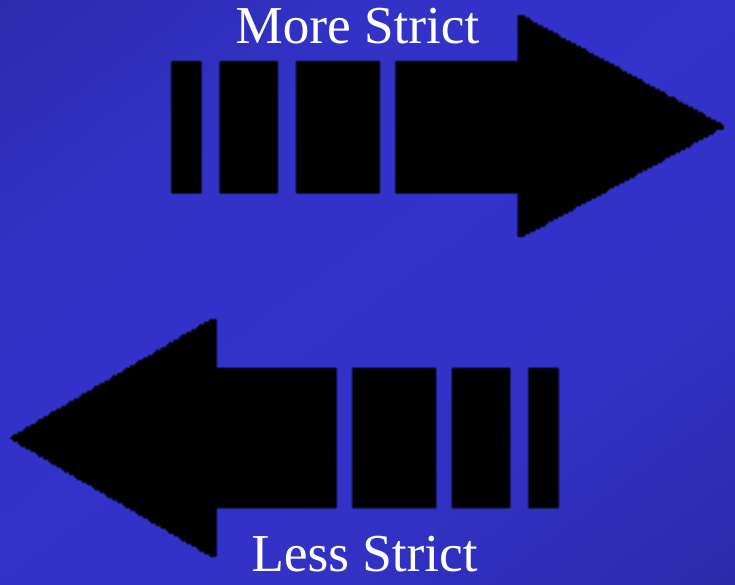
Lax vs. Strict

Benefits:

- Allows off-spec transactions through.
- Provides some tolerance for slight changes due to improvements in technology or precision.

Pitfalls:

- May allow incorrect or ambiguous data through.
- May muddy the database as more and more off-spec data is enrolled.
- Puts greater burden on individual implementations for higher-level error checking.



Benefits:

- Ensures consistency in data, facilitates inter-op.
- Reduces additional validation workload from core application.

Pitfalls:

- Greater chance of rejecting transactions (some of which may be off-spec but valid)
- Any changes to underlying data due to improvements in technology will require a new (updated) schema.

Partings thoughts…

- Prepare to be open minded on validation approach after an XML data standard has been agreed to.
- Try to think about what we can and can't live with early in the process of defining strictness.
- There are some lessons learned by other enterprises in going to XML (HL7) that may be helpful to examine.
- Genetic diversity in the user population can be a strength not a weakness, but can push limits of inter-op. Try to build in some flexibility.

Q&A / Contact Info

Shahram Orandi
NIST Image Group
sorandi@nist.gov