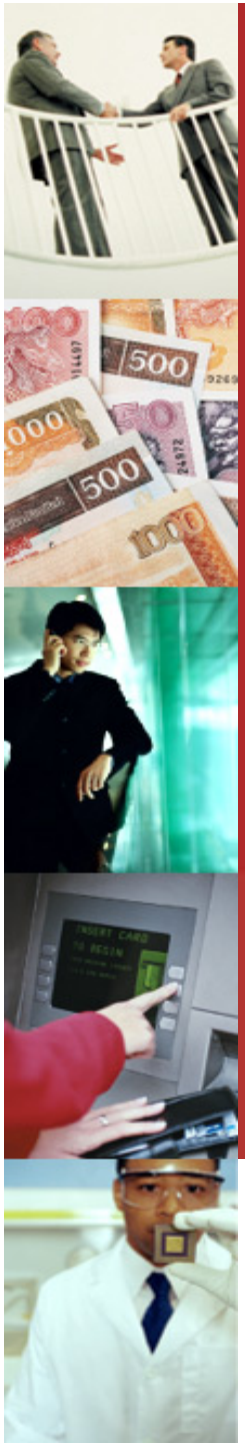


INTEGRATE BUILD PORTAL

Achieving Distributed Extensibility and Versioning XML

Dave Orchard
W3C Lead
BEA Systems

bea®

Extension & Versioning

- Perhaps the biggest goal of loose coupling is to allow distributed extension and evolution
- Provides a framework for designing languages
- Choices and Decisions Facing Designer

bea®

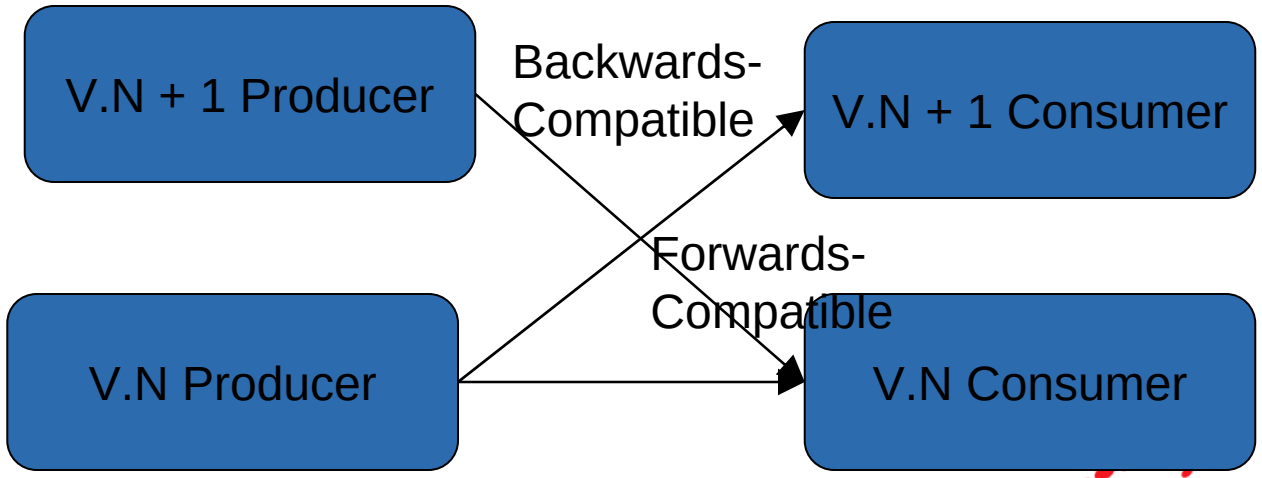
Material

- First Version:
  http://www.xml.com/pub/a/2003/12/03/versioning.html
- Second Version:
  http://www.xml.com/pub/a/2004/10/27/extend.html
- References:
  http://www.pacificspirit.com/Authoring/Compatibility
- Source for W3C TAG Finding on Extensibility and Versioning
  http://www.w3.org/2001/tag/doc/versioning
- And for the Web Architecture document
  http://www.w3.org/TR/webarch/

bea®

Compatibility

- Backwards Compatible
  - Newer software can read older versions of documents
  - ie. Update Schema and older documents validate
- Forwards Compatible
  - Older software can read newer versions of documents
- Incompatible means that software must be upgraded



Language Designer's Choices

- Can Designer extend language in a compatible way?
  - Forwards-compatible requires substitution rules
    - Transform newer extension into older instance
- Designer can always do incompatible change
- Can 3rd party extend language in compatible way?
- Can 3rd party extend language in incompatible way?
- Is language stand-alone or extension

bea®

Design decisions

1. Schema language
   - Can extension schemas be written?
2. Given schema language, constructs for extensibility
   - Schema's compatible extension is the wildcard
3. Substitution mechanism: needed for compat change
   - Must Ignore Unknown:
     Any unknown component is ignored
   - Others: XSLT Fallback model, …
4. Component version identification
   - Namespaces, version #s
5. Indicating incompatible changes
   - New version, mandatory extensions

bea®

Decision: Extensibility

- Schema Wildcard
- <xs:any processContents="" namespace="">
- attributes:
  - processContents for schema validation
    - Lax allows WSDL to control validation
  - namespace for which namespaces are allowed
- Schema does not have a “default” extensibility model
  - sometimes called open content model
- Extensibility Rule: Allow new elements or attributes

Decision: Substitution mechanism

- Substitution Rule: Provide substitution mechanism
- This is critical for compatible changes
  - Sender can put new information in without breaking receiver
- Candidate: Must Ignore Unknown
  - Receivers must ignore content they don't understand
- example, receiver must ignore ns2:middle

```
<ns:name xmlns:ns="myns" xmlns:ns2="extns">
        <ns:first>Dave</ns:first>
        <ns:last>Orchard</ns:last>
        <ns2:middle>Bryce</ns2:middle>
</ns:name>
```

Extension vs Version

- Extension: Additional content by others
- Version: Additional/changed content by designer
- Historically version identification is by “x.y” format
  - x changes are major and usually incompatible
  - y changes are minor and often compatible
- Namespaces allow designers to control versions
- Modularize their languages
- And 3rd parties to make additions
  - Compatible vs incompatible
- Blurry distinction:
  - Language designer references another language?
  - Extension author makes incompatible change

bea®

Component Version Identification factors

- Allow compatible evolution
  - Typically means retaining namespace name
- Complete schema for extensions/versions
- Generic tools usage
  - Version #s may preclude usage of

bea®

Decision: Component version identification

1. All components in new namespace(s) per version

-
```
<ns:name>
    <ns:first>Dave</ns:first>
    <ns:last>Orchard</ns:last>
</ns:name>
```
```
<ns2:name>
    <ns2:first>Dave</ns2:first>
    <ns2:last>Orchard</ns2:last>
    <ns2:middle>Bryce</ns2:middle>
</ns2:name>
```

2. All new components in new namespace for each compatible version

-
```
<ns:name>
    <ns:first>Dave</ns:first>
    <ns:last>Orchard</ns:last>
    <ns2:middle>Bryce</ns2:middle>
</ns:name>
```

3. All new components in existing or new namespace(s) for each compatible version

-
```
<ns:name>
    <ns:first>Dave</ns:first>
    <ns:last>Orchard</ns:last>
    <ns:middle>Bryce</ns:middle>
</ns:name>
```

4. All new components in existing or new namespace for each version + version identifier

-
```
<ns:name version="1.0">
    <ns:first>Dave</ns:first>
    <ns:last>Orchard</ns:last>
</ns:name>
```
```
<ns:name version="2.0">
    <ns:first>Dave</ns:first>
    <ns:last>Orchard</ns:last>
    <ns:middle>Bryce</ns:middle>
</ns:name>
```

bea®

First Solution: CVI Strategy #3

- <s:complexType name="NameType">
  <s:sequence>
  <s:element name="first" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name="last" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##any"
  minOccurs="0" maxOccurs="unbounded" />
  </s:sequence>
  <s:anyAttribute/>
  </s:complexType>

- <ns:name>
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>

  </ns:name>

- <ns:name>
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>
  <ns:middle>Bryce</ns:middle>

  </ns:name>

First Solution Problem

- Problem: Can't create new Schema for the extension
- Illegal:
- <s:complexType name="NameType">
  <s:sequence>
  <s:element name="first" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name="last" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name="middle" type="s:string" minOccurs="0" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##any"
  minOccurs="0" maxOccurs="unbounded" />
  </s:sequence>
  <s:anyAttribute/>
  </s:complexType>
- In fact, many if not most schemas have optional content at end in V1

bea®

Deterministic Content Model

- XML DTDs and XML Schema require “deterministic content models”.
- “the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the XML processor cannot know which b in the model is being matched without looking ahead to see which element follows the b.”
- Optional element followed by <any targetNamespace=“ns”> are NOT allowed
- <s:element name=“middle" type="s:string" minOccurs=“0" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##any“ minOccurs="0"
  maxOccurs="unbounded" />
- <s:element ref=“ns2:middle” minOccurs=“0" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##other” minOccurs="0"
  maxOccurs="unbounded" />

Second Solution: CVI Strategy #2

- <s:complexType name="NameType">
  <s:sequence>
  <s:element name="first" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name="last" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##other"
  minOccurs="0" maxOccurs="unbounded" />
  </s:sequence>
  <s:anyAttribute/>
  </s:complexType>

- <ns:name>
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>
  <ns2:middle>Orchard</ns2:middle>
  </ns:name>

- This is the most common extensibility model

CVI Strategy #2 Problem

- Problem: Can’t create new Schema for this
- Illegal:
- <s:complexType name=“NameType">
  <s:sequence>
  <s:element name=“first" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name=“last" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element ref=“ns2:middle" minOccurs=“0" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##other“
  minOccurs="0" maxOccurs="unbounded" />
  </s:sequence>
  <s:anyAttribute/>
  </s:complexType>

Extension schema choices

1. extension is required - incompatible
2. extension is optional, then:
   1. Lose the extensibility point
      - This means only 1 new version
   2. Do not add the extension into the schema
      - Validate any extensions found if you can
      - Can't validate the "right" extensions are in the "right" place
        - <name><first/><areacode/>….. <phone><last/>

bea®

CVI Strategy #3: Schema design #2

- There are complex ways to write the schema
- Can write a compatible Schema
  - Extensions & versions
- #1: Extension element for same namespace
  - This element has only a wildcard
  - This does the “swap” extensibility for element content trick
- #2: Dare Obasanjo’s Sentry technique

bea®

Extension element

- <s:complexType name=“name">
  <s:sequence>
  <s:element name=“first" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name=“last" type="s:string" minOccurs="1" maxOccurs="1"/>
  <s:element name="Extension" type=“ns:ExtensionType"
  minOccurs="0" maxOccurs="1"/>
  <s:any processContents="lax" namespace="##other“
  minOccurs="0" maxOccurs="unbounded" />
  </s:sequence>
  <s:anyAttribute/>
  </s:complexType>
  <s:complexType name="ExtensionType">
  <s:sequence>
  <s:any processContents="lax" namespace="##targetnamespace“
  minOccurs="0"maxOccurs="unbounded" />
  </s:sequence>
  <s:anyAttribute/>
  </s:complexType>

Sample

- <ns:name>
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>
  <ns:extension>
  <ns:middle>Bryce</ns:middle>
  </ns:extension>

  </ns:name>
- Another revision
- <ns:name>
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>
  <ns:extension>
  <ns:middle>Bryce</ns:middle>

  <ns:extension>
  <ns:prefix>Mr.</ns:prefix>
  </ns:extension>

  </ns:extension>

  </ns:name>

Schema V2

```
<s:complexType name=“name">
   <s:sequence>
      <s:element name=“first" type="s:string" minOccurs="1" maxOccurs="1"/>
      <s:element name=“last" type="s:string" minOccurs="1" maxOccurs="1"/>
      <s:element name="Extension" type=“ns:MiddleExtensionType"
                  minOccurs="0" maxOccurs="1"/>
      <s:any processContents="lax" namespace="##other“
                  minOccurs="0"  maxOccurs="unbounded" />
    </s:sequence>
    <s:anyAttribute/>
   </s:complexType>
   <s:complexType name="MiddleExtensionType">
    <s:sequence>
      <s:element name=“middle" type="s:string" minOccurs=“0" maxOccurs="1"/>
      <s:element name="Extension" type=“ns:ExtensionType"
                  minOccurs="0" maxOccurs="1"/>
      <s:any processContents="lax" namespace="##other“
                  minOccurs="0"  maxOccurs="unbounded" />
    </s:sequence>
    <s:anyAttribute/>
   </s:complexType>
   <s:complexType name="ExtensionType">
```

#5: Incompatible Extensions

- When adding required functionality
- Extension authors don't "own" namespace
- Designer provides mechanism to indicate extension is required
- "Must Understand" rule is a good one
  - Software must fault if extension with mU isn't understood
- Overrides the Must Ignore rule
- Example: add required ns2:middle as SOAP header
- <soap:header>
  <ns2:middle xmlns:ns2="extns" soap:mustUnderstand="1">Bryce</ns2:middle>
  </soap:header>
  <soap:body>
  <ns:name xmlns:ns="myns">
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>
  </ns:name> …
- Another solution: provide a mustUnderstand model
- <ns:name xmlns:ns="myns" xmlns:ns2="extns">
  <ns:first>Dave</ns:first>
  <ns:last>Orchard</ns:last>
  <ns2:middle ns:mustUnderstand="1">Orchard</ns2:middle>
  </ns:name>
- Complexities: scope of mU, partial processing

bea®

Versioning activities

- W3C TAG work
  - Web architecture doc, finding
- XML Schema 1.1(2.0?)
  - 1.0 doesn't make extensibility easy
    - explicit wildcards and determinism constraints
    - No "must ignore unknowns", "must understand"
  - 1.1 may relax determinism for wildcards
- RelaxNG, OWL/RDF have open content model

bea®

Summary

- Choices
  - Versions
  - Extensions
    - Optional, mandatory
- Decisions
  - Schema language
  - Schema design
  - Substitution model
  - Component version identification strategy

bea®

Thank you, Questions?

bea®