



# Is XML a Flexible Data Format?

**Santiago Pericas-Geertsen**

Staff Engineer

Sun Microsystems

Santiago.PericasGeertsen@sun.com



# Goal of this Talk

## What You Will Gain

Understand the difficulties in supporting schema evolution in the presence of validation. Learn about the extensibility features of XML Schema and their application to the problem of evolving schemas.

# Agenda

- Schema versioning problem
- Extensibility support in XML Schema
- Extensibility applied to schema evolution
- Conclusions
- Future directions

# Basic Notation

## Schemas and Instances

- Schemas:
  - `<xs:complexType name="Point">`
    - `<xs:sequence>`
      - `<xs:element name="x" type="xs:int"/>`
      - `<xs:element name="y" type="xs:int"/>`
    - `</xs:sequence>`
    - `</xs:complexType>`
  - Point = `x<int>,y<int>`
- Instances:
  - `<x>1</x><y>2</y>`
  - `x<1>,y<2>`
- More notation introduced along the way!

# Schema Versioning Problem

## By Example

- Two versions of the same schema type
  - V1:  
Point =  $x\langle\text{int}\rangle, y\langle\text{int}\rangle$
  - V2:  
Point =  $x\langle\text{int}\rangle, y\langle\text{int}\rangle, z\langle\text{int}\rangle$
- Can a V1 processor process the instance  $x\langle 1\rangle, y\langle 2\rangle, z\langle 1\rangle$ ?
- Can a V2 processor process the instance  $x\langle 1\rangle, y\langle 2\rangle$ ?

# Compatibility

## Definitions

- **Backward compatibility:** A version  $k$  processor can process versions  $k-1$ ,  $k-2$ , ...
  - Example: Java VM version 5 can process classes produced by version 4 compilers
- **Forward compatibility:** A version  $k$  processor can process versions  $k+1$ ,  $k+2$ , ...
  - Example: An HTML version 3.0 browser can process HTML 4.01 version pages
- **Compatibility = Forward + Backward**
  - Important for XML-based data formats

# Processing

## Definition

- What does it mean to *process* a version of a format?

Processing is defined to be

1. Validation of data format based on a schema
  2. Execution of application logic without exception conditions resulting from getting newer or older instances
- *Assumptions:*
    - Ignore unknown data rule
    - A version  $k$  processor can *only* access to schemas for versions  $k$ ,  $k-1$ ,  $k-2$ , ...



# Extensibility

**Santiago Pericas-Geertsen**

Staff Engineer  
Sun Microsystems



# Extensibility

## Definition

- Extensibility is the ability to extend without “breaking” --e.g. of a fiber
- The XML Schema specification defines mechanisms that support forms of extensibility (among them):
  - Type derivations: by extension or by restriction
  - Substitution groups
  - Wildcards
- Extensibility is the weapon of choice to tackle the schema versioning problem

# Type Derivations

## Derivation by Extension

- Derivation by extension: akin to subclassing in OO systems
- Schema:  
ElemPoint = point<Point>  
Point = x<int>,y<int>  
3DPoint = Point ++ z<int> = x<int>,y<int>,z<int>  
=> 3DPoint <++ Point>
- Instances: type annotated, no subsumption
  - *point{3DPoint}<x<1>,y<2>,z<1>>*
  - *point<x<1>,y<2>>*

# Derivation by Extension

## Example in XML Syntax

```
<xs:element name="point" type="tns:Point"/>
```

```
<xs:complexType name="Point">  
  <xs:sequence>  
    <xs:element name="x" type="xs:int"/>  
    <xs:element name="y" type="xs:int"/>  
  </xs:sequence>  
</xs:complexType>
```

```
<xs:complexType name="ThreeDPoint">  
  <xs:complexContent>  
    <xs:extension base="tns:Point">  
      <xs:sequence>  
        <xs:element name="z" type="xs:int"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>
```

# Derivation by Extension

## Example in XML Syntax

- Instance with Point

```
<point>  
  <x>1</x>  
  <y>2</y>  
</point>
```

- Instance with ThreeDPoint:

```
<point xsi:type="ThreeDPoint">  
  <x>1</x>  
  <y>2</y>  
  <z>1</z>  
</point>
```

# Type Derivations in XSD

## Derivation by Restriction

- In width: (choice restriction)

Point' = x<int>,y<int>?

1DPoint = Point' -- y<int> = x<int>

1DPoint <-- Point'

- In depth:

Point = x<int>,y<int>

PosPoint = Point -+ y<posint> = x<int>,y<posint>

PosPoint <-+ Point

- Instances: type annotated, no subsumption

# Derivation by Restriction

## Unsoundness with Subsumption in OO?

- Subsumption rule:

$d : T$  and  $T \leftarrow\# T'$  then  $d : T'$       ( $\#$  in  $\{-,+\}$ )

- Example using *update* operation:

$p : \text{PosPoint} = x\langle 1 \rangle, y\langle 2 \rangle$

$\text{log} : \text{posint} \rightarrow \text{double}$

$f(d : \text{Point}) : \text{Point} = (d.y := -1)$     // update point

$f(p);$             // typecheck using subsumption

$\text{log}(p.y);$     // typechecks since  $p : \text{PosPoint}$

$\Rightarrow \text{log}(-1)$  is a runtime error!

# Substitution Groups

## Description

- Orthogonal views:
  - Akin to type derivation but for elements
  - A form of unbounded element choice for elements of related types

- Schema:

Point = x<int>,y<int>

3DPoint = Point ++ z<int> = x<int>,y<int>,z<int>

ElemPoint = point<Point>

Elem3DPoint = 3dpoint<3DPoint> ➡ ElemPoint

ElemLine = line<ElemPoint,ElemPoint>

- Instance:

*line<3dpoint<...>,3dpoint<...>>*

# Substitution Groups

## Example in XML Syntax

```
<xs:complexType name="Point">  
  <xs:sequence>  
    ...  
  </xs:sequence>  
</xs:complexType>
```

```
<xs:complexType name="ThreeDPoint">  
  <xs:complexContent>  
    ...  
  </xs:complexContent>  
</xs:complexType>
```

```
<xs:element name="point"  
            type="tns:Point"/>
```

```
<xs:element name="threedpoint"  
            type="tns:ThreeDPoint"  
            substitutionGroup="tns:point"/>
```

# Wildcards

## Description

- Also known as *open content*
- Define schema *holes* that can be filled with arbitrary data:
  - Elements: using `xs:any`
  - Element content: using `xs:anyType`
  - Attributes: using `xs:anyAttribute`
- Can be parameterized over namespace kinds:
  - `#target`: target namespace only
  - `#other`: any namespace *excluding* target
  - `#any`: any namespace *including* target

# Target Namespace

## Description

- It is the namespace associated with a schema document
- Can be required for all elements or just top-level elements
- Can be used to connect document instances with schema definitions

- Example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:tns="http://www.sun.com/point"
            targetNamespace="http://www.sun.com/point">
```

# Wildcards

## Defining Extensible Point Types

- Schema:

Point = x<int>,y<int>,#target\*

- Instance:

x<1>,y<2> : Point

x<1>,y<2>,z<1> : Point

x<1>,y<2>,z<1>,color<'red'> : Point

x<1>,y<2>,x<3>,y<6> : Point // Line??

- In XML syntax, #target\* is written as:

```
<xs:any minOccurs="0"  
      maxOccurs="unbounded"  
      namespace="##targetNamespace"/>
```



# Schema Evolution

**Santiago Pericas-Geertsen**

Staff Engineer  
Sun Microsystems



# Evolving Schemas

## Type Derivation

- Scenario:  $V_k$  defines type Point;  $V_{k+1}$  derives 3DPoint from Point
- Backward Compatibility:
  - Validation and execution are possible
- Forward Compatibility:
  - Validation impossible without new schema
  - Execution *may* be possible

	Validation	Execution
Backward	☺	☺
Forward	☹	☺

# Evolving Schemas

## Substitution Groups

- Scenario:  $V_k$  defines element point;  $V_{k+1}$  defines 3dpoint in same substitution group
- Backward Compatibility:
  - Validation and execution are possible
- Forward Compatibility:
  - Validation impossible without new schema
  - Execution *may or may not* be possible

	Validation	Execution
Backward	☺	☺
Forward	☹	☹/☹

# Evolving Schemas

## Wildcards

- Scenario:
  - $V_k$  defines  
Point =  $x\langle\text{int}\rangle,y\langle\text{int}\rangle,\#\text{target}^*$
  - $V_{k+1}$  re-defines  
Point =  $x\langle\text{int}\rangle,y\langle\text{int}\rangle,z\langle\text{int}\rangle,\#\text{target}^*$
- Is this enough for forward and backward compatibility?
  - Not backward compatible: a  $V_{k+1}$  processor cannot validate  
 $x\langle 1\rangle,y\langle 2\rangle$
  - New components must be optional! I.e., in  $V_{k+1}$   
Point =  $x\langle\text{int}\rangle,y\langle\text{int}\rangle,z\langle\text{int}\rangle?,\#\text{target}^*$

# Evolving Schemas

## Wildcards

- Success?
  - $V_{k+1}$  re-defines  
Point =  $x<int>,y<int>,z<int>?,\#target^*$
- This is forward and backward compatible but ... not a *legal* XML Schema ☹
  - It violates the so-called *Unique Particle Attribution* (UPA) rule
- UPA not precisely defined in the spec
  - Intuition: given the instance  
 $x<1>,y<2>,z<1>$   
it matches either  
 $x<int>,y<int>,z<int>,\epsilon$  or  $x<int>,y<int>,\epsilon,\#target$

# Evolving Schemas

## Wildcards

- Back to the drawing board ...
  - $V_k$  defines  
Point =  $x\langle\text{int}\rangle, y\langle\text{int}\rangle, \text{ext}\langle\#\text{target}^*\rangle?$
  - $V_{k+1}$  re-defines  
Point =  $x\langle\text{int}\rangle, y\langle\text{int}\rangle, \text{ext}\langle z\langle\text{int}\rangle, \text{ext}\langle\#\text{target}^*\rangle? \rangle?$
  - $V_{k+2}$  re-defines  
Point =  $x\langle\text{int}\rangle, y\langle\text{int}\rangle,$   
 $\text{ext}\langle z\langle\text{int}\rangle, \text{ext}\langle\text{color}\langle\text{string}\rangle, \text{ext}\langle\#\text{target}^*\rangle? \rangle? \rangle?$
- Instances:
  - $V_k$ :  $x\langle 1 \rangle, y\langle 2 \rangle$
  - $V_{k+1}$ :  $x\langle 1 \rangle, y\langle 2 \rangle, \text{ext}\langle z\langle 1 \rangle \rangle$
  - $V_{k+2}$ :  $x\langle 1 \rangle, y\langle 2 \rangle, \text{ext}\langle z\langle 1 \rangle, \text{ext}\langle\text{color}\langle\text{'red'}\rangle \rangle \rangle$

# Evolving Schemas

## Wildcards

- Forward compatibility requires extension points in first version
  - V1 must include extension  
Point = x<int>,y<int>,ext<#target\*>?
- Nesting unnecessary obscures instances after a few versions
  - Other proposals avoid nesting, but not all of them work as advertised!
- It is also possible to use namespaces other than target for extensions in order to avoid UPA

# Evolving Schemas

## Summary

- Forward compatibility is hard using type derivations or substitution groups
  - These features work better in a *closed world*
- Wildcards can be used for compatibility, but solutions add complexity to the system
  - Extra elements and/or additional structure
  - Do not work with existing systems not designed with versioning in mind
- Many of the problems go away if validation is turned off!



# Conclusions & Future Work

**Santiago Pericas-Geertsen**

Staff Engineer  
Sun Microsystems



# Other Considerations

## Not Covered

- Unknown data cannot be ignored
  - must understand attributes are used to prevent older processors to *ignore* important data
  - Example: newer schema requires credit card security code
- Version information and/or schema location is part of the instances
  - Old processors can access newer schemas

# Conclusions

## What Have We Learned?

- Most of XML's flexibility is lost with validation
  - XML is flexible, *valid XML* is not
  - Validation makes no use of meta-data for flexibility
- Data-oriented applications of XML *may* require a weaker form of validation
- An instance validates the type Point *if and only if* it is a sequence of an x element followed by a y element
  - Only one direction may suffice
- The use of *weak* or *strong* validation can be a system's designer decision

# Conclusions

## Weaker Validation

- Intuition:
  - An instance is of type Point *if* it contains an x and a y component:  
 $x\langle 1 \rangle, color\langle 'red' \rangle, y\langle 2 \rangle : \text{Point}$   
 $x\langle 1 \rangle, z\langle 1 \rangle, y\langle 2 \rangle, color\langle 'red' \rangle : \text{Point}$   
 $\dots, x\langle \text{int} \rangle, \dots, y\langle \text{int} \rangle, \dots <: \text{Point}$
- Could be defined for each type constructor:
  - $e_1\langle T_1 \rangle, \dots, e_n\langle T_n \rangle <: e_{s_1}\langle T_{s_1} \rangle, \dots, e_{s_m}\langle T_{s_m} \rangle$   
 $m \leq n$  and  $s_i \in [1..n]$  and  $s_i < s_{i+1}$  with  $i \in [1..m)$
  - $e_1\langle T_1 \rangle \& \dots \& e_{n+m}\langle T_{n+m} \rangle <: e_1\langle T_1 \rangle \& \dots \& e_n\langle T_n \rangle$
  - ... ?

# Future Work

## Where Are We Heading?

- XML Schema WG at W3C is looking at UPA and wildcards: *weak* wildcards
  - Point = `x<int>,y<int>,z<int>?,#target*` will be legal
- Alternative forms of validation?
  - Need for a precise definition of weak validation
    - Complexity of XML Schema makes this hard
- Schema evolution is a big problem today, but will be even bigger tomorrow!

# References

## More Information?

- Extensibility, XML Vocabularies, and XML Schema  
<http://www.xml.com/pub/a/2004/10/27/extend.html>
- Designing Extensible, Versionable Formats  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexxml/html/xml07212004.asp>
- Versioning XML Languages  
<http://www.w3.org/2001/tag/doc/versioning>
- Extending and Versioning XML Languages with XML Schema  
<http://www.pacificspirit.com/Authoring/Compatibility/ExtendingAndVersioningXMLLanguages.html>



# Thank you

**Santiago Pericas-Geertsen**

Staff Engineer

Sun Microsystems

[Santiago.PericasGeertsen@sun.com](mailto:Santiago.PericasGeertsen@sun.com)

