A Compact XML Schema Syntax

Erik Wilde and Kilian Stillhard
Computer Engineering and Networks Laboratory
ETH Zürich

http://dret.net/projects/xscs/

Abstract

The new schema language defined by the W3C, XML Schema, is used in a number of applications, such as Web Services and XQuery, and will probably be used by an increasing number of users in the near future. Currently, XML Schema's data model, the "XML Schema Components", can only be represented in the rather verbose XML syntax defined in the XML Schema specification itself. We propose an alternative non-XML syntax, which is (1) much more compact than the XML syntax, (2) defined by EBNF productions, (3) re-uses well-known syntactic concepts where appropriate, and (4) is easy to implement using standard parser-generating tools. Our approach is comparable to the approach of the RELAX NG schema language, which also supports two alternative syntaxes, an XML-based one, and a more compact non-XML one.

We believe that XML Schema could be made easier to use by supporting a compact syntax. Currently, complex schemas are very hard to read due to the large amount of XML markup, and the various tools and GUIs that are on the market differ widely and in all cases support only a subset of the features of XML Schema. We believe that there should be a compact syntax, optimized for human users, which makes it easy to read and write XML Schemas, and which supports the full feature set of XML Schema. Obviously, a non-XML syntax makes it necessary to introduce new tools. However, generating parsers from EBNF productions is rather simple and well-supported by standard tools (such as yacc and JavaCC), and the other direction (i.e., generating non-XML syntax) can be implemented by using XML tools.

Our *XML Schema Compact Syntax (XSCS)* is geared towards human users, by re-using language constructs known from other application areas, such as DTDs and programming languages, and making them available for XML Schema component representation. Examples for this re-use of syntactic constructs are DTD-style content models, number ranges ("`[a,b]`" or "`(a,b]`" as in standard mathematical notation), and qualifying attributes like "`abstract`" or "`final`" known from programming languages ("`final abstract type { ... }`"). We also believe that graphical representations of complex structures such as schemas are not always suitable because some people prefer textual representations, editing might be faster when using keyboard input instead of using click-and-point operations, and graphical representations (usually) hide some information.

We fully integrate the processing of our syntax into the existing pipeline of XML-based tools by creating a parser that generates SAX events or DOM trees from the compact syntax documents. This way, we can use the existing XML Schema validation engines and XML Schema error checking facilities already implemented in validation engines like the Xerces parser. In addition, we have a serialization module to generate compact syntax documents from XML Schema DOM trees.

Our overall goal is to improve XML Schema acceptance by providing a syntax that is easier to work with than the XML syntax, and tools to process this syntax.

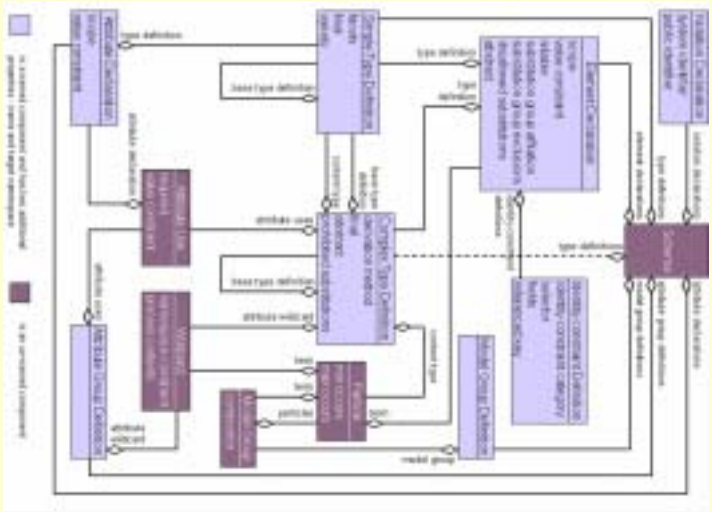
Outline

- Introduction
  - XML Schema Components
  - XML Schema Component Syntaxes
- Syntax Goals and Syntax Design
- Examples
- Prototype Implementation
- What now?
- Q&A

Introduction

- XML Schema is based on *Schema Components*
  - Schema Components are the Information Model
  - XML Schema defines an XML syntax for components
- XML Schema's XML syntax has advantages
  - uses a well-known Syntax (XML)
  - can be processed using standard XML tools
- ...and also some disadvantages
  - is very verbose (because of XML and syntax design)
  - has some less than ideal syntax constructs
- so why not define an alternate syntax?
  - an approach already taken for RELAX NG

XML Schema Components



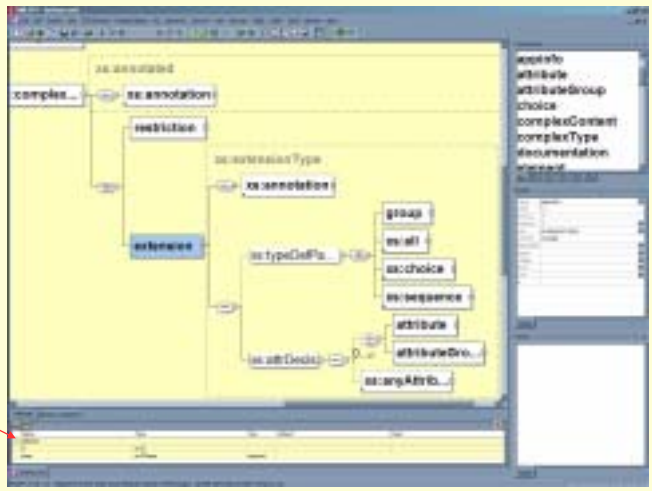
XML Schema for Schemas

```
<xs:element name="complexContent" id="complexContent">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="xs:annotated">
        <xs:choice>
          <xs:element name="restriction" type="xs:complexRestrictionType"/>
          <xs:element name="extension" type="xs:extensionType"/>
        </xs:choice>
        <xs:attribute name="mixed" type="xs:boolean"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:complexType name="extensionType">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
   <xs:attribute name="base" type="xs:QName" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

What are the attributes that are defined for the extension element?

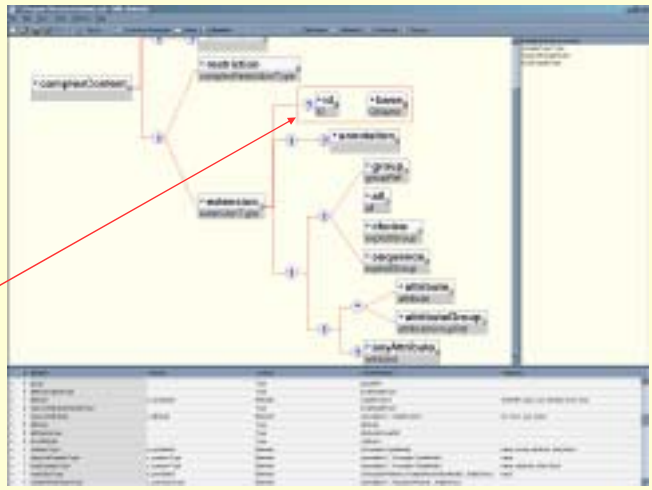
Alternate Representations (I)

What are the attributes that are defined for the extension element?



Alternate Representations (II)

What are the attributes that are defined for the extension element?



Alternate Representations (III)

- XML Schema Software often provides a GUI
  - a way of representing Schema Components
- advantages of GUIs
  - graphical overview of component structure
    - component interrelations are clearly represented
  - easier to use and manage for beginners
- disadvantages of GUIs
  - problem of dealing with XML Schema's complexity
  - dependency on a particular software
    - which may be depending on particular hardware/OS
    - which may not implement XML Schema correctly
  - not suitable as interchange format

Character-based Representation

- character-based representations
  - are easy to exchange
  - are not so easy to read
    - component interrelations are hard to find
    - component information is easy to find
  - are easy to process (by using parser building tools)
- create a new character-based representation
  - focus on maximum readability for human users
    - XML syntax is optimized for machine processing
- who will use the new syntax?
  - XML Schema power users (less noise in the syntax)
  - XML Schema users not comfortable with XML syntax

Compact Component Syntax

- *XML Schema Compact Syntax (XSCS)*
  - an alternative syntax for XML Schema
  - defined using EBNF and the XML syntax
    - easier to learn for XML Schema users
- not always a 1:1 mapping to XML syntax
  - but in most cases
  - exceptions when beneficial
    - alternative syntaxes for local element declarations
    - grouping of simple type facets
    - grouping of syntactic invariants
- XSCS is only a syntax
  - XML Schema validity constraints remain unchanged

Syntax Design Goals

- compactness
  - try to achieve optimal compact/readability ratio
- DTD-style content models
  - re-use well-known DTD content model syntax
- easier mixed content
  - avoid confusing type derivation with mixed content
- easier complex type derivation
  - avoid redunant parts of the derivation syntax
- easier facet definition
  - combine associated facets

XSCS Top-Level Components

- XML Schema top-level Components
  - these components must be named
  - appear on the top level of the XML or XSCS syntax
  - simple types, complex types, attribute definitions, element definitions, attribute groups, named model groups, notations
- represented by a consistent syntax structure
  - `options component-type name extensions { inner components };`
  - `options`: boolean or enumerated attributes in XML
  - `extensions`: strings, names or references in XML
  - `inner components`: nested components in XML

Component Options

- XML syntax defines many component options
  - boolean or enumerated attributes (finite value set)
  - setting properties for the component
- XSCS models them as atomic keywords
  - small and managable set of keywords

```
<complexType final="union" abstract="true" ...

final-union abstract complexType ...
```

Component Extensions

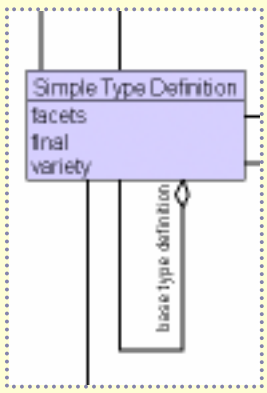
- component properties with string values
  - used with a variety of properties

XML Syntax	XSCS
<extension base="name">	extends name
<restriction base="name">	restricts name
substitutionGroup="name"	substitutes name
fixed="value"	= "value"
default="value"	<= "value"

Simple Type Restriction

```
<xs:simpleType name="short">
 <xs:restriction base="xs:int">
  <xs:minInclusive value="-32768"/>
  <xs:maxInclusive value="32767"/>
 </xs:restriction>
</xs:simpleType>
```

```
simpleType short
   { xs:int { [-32768,32767] } }
```



Facet Syntax

- may be preceded by a `fixed` keyword
- some facets are identified by keyword
  - *length, whitespace, totalDigits, fractionsDigits
  - `length = 15` or `length = [,255]`
- "range" facets use interval notation
  - different delimiters for open and closed bounds
    - () denotes open intervals (`*Exclusive` facets)
    - [] denotes closed intervals (`*Inclusive` facets)
  - delimiters may be mixed: `(0,2048]`
- patterns use a perl-like syntax (`/yes|no/`)
- enumerations use lists of strings

Complex Type Extension

```
<xs:complexType name="extensionType">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
   <xs:attribute name="base" type="xs:QName" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

```
complexType extensionType extends xs:annotated {
 ( @xs:typeDefParticle?, @xs:attrDecls );
 required attribute base { xs:QName };
};
```

Model Groups

- goal: DTD-like syntax for model groups
- XML Schema extends DTD content models
  1. re-introduces the SGML & connector (`all` group)
  2. more flexible occurrences (numeric min and max)
  3. local element definitions
  4. more flexible element wildcards
- re-use and extend XML content model syntax
  1. revive the SGML & connector
  2. use `[n,m]` syntax for non-DTD occurrences
  3. allow inline or out-of-line local element definitions
  4. extend the `ANY` syntax with namespaces
- result: intuitive model group syntax

Local vs. Global Elements

- local and global element definitions
  - local elements are defined inside a type definition
  - global elements are defined by top-level elements
- XSCS mirrors local and global definitions
  - global elements in model groups are referenced
  - local elements in model groups are defined
  - and introduces an alternative local definition
    - separating the content model from the elements
    - using the element name as local reference
- when converting XSCS to XML syntax
  - a name is a local definition if there is a definition
  - otherwise it is converted into a reference

Local Element Declarations (I)

```
<element name="page">
 <complexType>
  <sequence>
   <element name="head" type="string"/>
   <element name="section" type="string" maxOccurs="4"/>
   <element name="foot" type="string"/>
  </sequence>
 </complexType>
</element>
```


Local Element Declarations (II)

```
element page {
 ( head { string },
   section { string } [,4],
   foot { string } ) }
```

```
element page {
 ( head, section [,4], foot)
 element head { string }
 element section { string }
 element foot { string } }
```

XSCS Results

- XSCS version of the Schema for Schemas
  - stripped annotations and XML comments
  - removed all empty lines
  - whitespace characters do not count
  - used "reasonable" line wrapping

	Lines			Characters		
	XML	XSCS	Reduction	XML	XSCS	Reduction
datatypes.xsd	502	126	75%	14030	4520	68%
structures.xsd	936	315	66%	23820	9238	61%

Prototype Implementation

- XSCS syntax is defined in EBNF productions
  - syntax is not a LL(1) grammar
  - many parser generators support LL(k) grammars
- implementation is based on DOM
  - there is no standard component API
    - top-level component order should be preserved
  - Java implementation (using Apache's Xerces)
- XSCS generation
  - XSCS serializer working on the DOM
  - could be improved with some conversion options
- XSCS parsing
  - based on JavaCC parser generator tool

What now?

- XSCS 1.0 is specified and implemented
  - Java-based implementation
  - existing conversion tools for testing
- improving XSCS based on feed-back
  - fine-tuning the syntax
- based on feed-back about its usefulness
  - keep it as a personal tool for compact XML Schema
  - or establish some kind of standardization process
- XSCS support in XML Schema processors
  - enables much better error reporting
  - could lower the entry barrier for XML Schema users

Thank You! — Q&A

http://dret.net/projects/xscs/