Comparative Analysis of Six XML Schema Languages[1]

Dongwon Lee Wesley W. Chu

Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA
Email: {dongwon,wwc}@cs.ucla.edu

Abstract: As XML [5] is emerging as *the* data format of the internet era, there is an substantial increase of the amount of data in XML format. To better describe such XML data structures and constraints, several XML schema languages have been proposed. In this paper, we present a comparative analysis of the six noteworthy XML schema languages.

1 Introduction

As of June 2000, there are about a dozen of XML schema languages that have been proposed. Among those, in this paper, we choose six schema languages (XML DTD [5], XML Schema [9, 22, 1], XDR [17, 10, 15], SOX [8], Schematron [11], DSD [12, 13]) as representatives.

Our rationale in choosing the representatives is as follows: 1) they are backed by substantial organizations so that their chances of survival is high (e.g., XML DTD and XML Schema by W3C, XDR by Microsoft, DSD by AT&T), 2) there are publically known usages or applications (e.g., XML DTD in XML, XDR in BizTalk, SOX in xCBL), 3) the language has a unique approach distinct from XML DTD (e.g., SOX, Schematron, DSD).

First, we briefly review each schema language.

1.1 XML DTD

XML DTD (DTD in short), a subset of SGML DTD, is the *de facto* standard XML schema language of the past and present and is most likely to thrive until XML Schema finally arrives. It has limited capabilities compared to other schema languages. Its main building block consists of an *element* and an *attribute*. The real world is typically represented by the use of hierarchical element structures.

1.2 XML Schema

XML Schema is an ongoing effort of W3C to aid and eventually replace DTD in the XML world. XML Schema aims to be more expressive than DTD and more usable by a wider variety of applications. It has many novel mechanisms such as inheritance for attributes and elements, user-defined datatypes, etc.

1.3 XDR

First known as XML-Data, then later trimmed and improved to XDR (XML-Data Reduced), this language is a joint effort of Microsoft and others and is being used in Microsoft's BizTalk framework. XDR is heavily influenced by another proposal co-developed by IBM and Microsoft, DCD (Document Content Description), and thus shares many similar features.

1.4 SOX

SOX (Schema for Object-Oriented XML) is an alternative schema language for defining the syntactic structure and partial semantics of XML document types. As the name implies, it extends DTD in an object-oriented way by allowing extensible data types and inheritance among element types. The current version, 2.0, is being developed by Commerce One.

1.5 Schematron

Schematron, created by Rick Jelliffe, is quite unique from others in that it focuses on *validating* schemas using patterns instead of *defining* schemas. Its schema definition is simple enough to be defined in a single page, yet provides very powerful constraint specification via XPath [7]. The latest version is 1.4.

1.6 DSD

DSD 1.0 was co-developed by AT&T Labs and BRICS with the goals of context-dependent description of elements and attributes, flexible default insertion mechanisms, expressive power close to XSLT [6], etc. Like Schematron, DSD has a strong edge on *schema constraints*.

1.7 Other Languages

In addition, DCD [4], DDML [3], Assertion Grammars [19], RELAX [16] have been proposed.

2 Features Classification

In the following, we denote a constant value with single quotes regardless of the language specification for simplicity. Furthermore, any attribute *A* or element *E* in the language is denoted by ⟨A⟩ or ⟨E⟩. When a schema language supports a certain feature fully or partially, we denote it as `Yes` or `Partial`. Otherwise, we denote `No`. Furthermore, when there is no explicitly equivalent construct in the language, but the feature can be simulated using other constructs with reasonable complexity, we consider the feature supported by the language.

2.1 Schema

1. Syntax in XML: Using XML syntax for the schema language brings several benefits [14]: 1) users do not have to learn new proprietary syntax, 2) the schema language can be readily applicable to existing XML applications (e.g., editor, browser), 3) the schema file can be stored in a XML storage system along with XML documents, and 4) the schema language is extensible. All the schema languages except DTD are written in XML syntax:

DTD: `No` XML Schema: `Yes` XDR: `Yes`
SOX: `Yes` Schematron: `Yes` DSD: `Yes`

2. Namespace: All languages except DTD and DSD support namespace.

DTD: `No` XML Schema: `Yes` XDR: `Yes`
SOX: `Yes` Schematron: `Yes` DSD: `No`

Suppose one wants to define the `book` element by reusing the `address` element defined elsewhere (denoted as `URI`) and defining his own `title` element. This can be written in XML Schema as follows:

```
<schema xmlns:z='URI' ...>
  <element name='book'>
    <complexType>
      <element name='title' type='string'>
      <element name='address' type='z:address'>
    </complexType>
  </element>
</schema>
```

Similarly, in XDR:

```
<ElementType name='title' dt:type='string'/>
<ElementType name='book' xmlns:z='URI'>
  <element type='title'><element type='z:address'>
</ElementType>
```

SOX supports elements ⟨namespace⟩ to declare namespace and two attributes ⟨prefix⟩ and ⟨type⟩ to qualify names.

```
<namespace prefix='z' namespace='URI'/>
<elementtype name='book'>
  <model>
    <element type='title'>
    <element prefix='z' type='address'>
  </model>
</elementtype>
```

Similarly, Schematron provides an attribute ns for the element ⟨schema⟩.

3. Include & import: Sometimes, it is convenient to pull in externally defined schema fragments to the current schema. This is especially true when the schema gets larger; it becomes more desirable to have modular schema definitions for better maintainence and readability. Several schema languages support this feature. If the newly pulled-in fragments can have only the *same* target namespace as the current schema, then we refer to it as *include*. Otherwise, we refer to it as *import*. First, *Include* is supported as follows:

DTD: `No` XML Schema: `Yes` XDR: `No`
SOX: `Yes` Schematron: `No` DSD: `Yes`

In XML Schema, using ⟨include schemaLocation='URI'⟩ is conceptually equivalent to replacing the include clause with all the definitions in the `URI`. The namespace of the included fragments must be the same as that of the current schema. In SOX, a construct ⟨join⟩ allows schema definitions belonging to the same namespace to be pulled in. Similarly, ⟨include⟩ is supported in DSD as well.

Furthermore, *Import* is supported as follows:

DTD: `No`	XML Schema: `Yes`	XDR: `No`
SOX: `Yes`	Schematron: `No`	DSD: `No`

In XML Schema, a construct ⟨import⟩ exists. By importing multiple namespaces, XML Schema allows definitions and declarations contained in schemas under different namespaces. In SOX, special processing indicator ⟨? import ?⟩ is used to import schema that can override the default namespace declared in the current schema.

2.2 Datatype

Datatype can be categorized into two types: *simple* or *complex*. A *simple* type cannot have element content nor carry attributes while a *complex* type can. Although most schema languages support simple types separately, the support of complex type is a bit fuzzy due to the mixed definition of complex type and element type. Therefore, here, we only explicitly compare features of the simple type. The features of the complex type are interspersed through Sections 2.4 and 2.5.

1. Built-in type: This is either a primitive or derived *simple* type provided by the schema language specification. Most schema languages, except Schematron and DSD, support an array of built-in types including the plain string and XML-related types (e.g., `ID`, `NMTOKEN`). The number of such built-in types are:

DTD: `10`	XML Schema: `37`	XDR: `33`
SOX: `17`	Schematron: `0`	DSD: `0`

While DTD supports only XML-related primitive types, XML Schema supports an extensive set of 37 built-in types, covering most types being used in general programming languages. So does XDR or SOX. Since the focus of Schematron is validating XML structure, it does not provide any explicit built-in types. Similarly, DSD has no built-in types. However many types can be easily simulated through its support of regular expressions.

2. User-defined type: When schema designers consider certain types be defined as *simple* types in their schema, XML Schema, SOX, and DSD provide such a facility:

DTD: `No`	XML Schema: `Yes`	XDR: `No`
SOX: `Yes`	Schematron: `No`	DSD: `Yes`

In XML Schema, new simple types can be created by deriving from built-in or derived types via the inheritance. Details will be found in Section 2.5. In SOX, new datatypes can be defined using three *facets* ⟨enumeration⟩, ⟨scalar⟩ and ⟨varchar⟩. Although types can be simulated in Schematron, they are not treated as first-class objects as in other languages. DSD uses a construct ⟨StringTypeDef⟩ along with a rich set of operators and regular expressions to support user-defined types. For instance, a 9 digit US `zipcode` definition can be written as follows in DSD:

```
<StringTypeDef ID='zipcode'>
  <Sequence>
    <Repeat value='5'>
      <CharSet Value='0123456789'>
    </Repeat>
    <Optional>
      <String Value='-'>
      <Repeat value='4'>
        <CharSet Value='0123456789'>
      </Repeat>
    </Optional>
  </Sequence>
</StringTypeDef>
```

3. Domain constraint: Not only the type itself, but also the legal values for the type are important. Some languages support a set of constructs to limit the valid domain values for datatypes as follows:

DTD: `No`	XML Schema: `Yes`	XDR: `No`
SOX: `Partial`	Schematron: `Yes`	DSD: `Yes`

Towards this feature, XML Schema supports a multitude of facets (e.g., range, precision, length, mask) and regular expressions. SOX provides a primitive set of facets including enumeration, min or max value, maxlength, etc. However, the pattern language is not supported. Although built-in or user-defined types are not allowed in Schematron, one can simulate such types using Schematron's support of XPath. For instance, the integer type for the element `E` can be simulated as follows [*18*]:

```
<rule context='E'>
  <assert test='floor(.) = number(.)'>
    E can have only integer value.</assert>
</rule>
```

As shown in the example of the user-defined type case, DSD supports a set of pattern-related operators to constrain the legal domain for user-defined types.

4. Null: It is often preferable to differentiate among unknown, inapplicable or others by supporting the explicit ``null'' values.

DTD: `No`	XML Schema: `Yes`	XDR: `No`
SOX: `No`	Schematron: `No`	DSD: `No`

In XML Schema, there is an attribute ⟨nullable⟩ to indicate that the element content is null. In a XML instance document, the element `fullname` carries an attribute `null='true'` to represent the nullness as shown below:

```
schema  : <element name='fullname' nullable='true'/>
instance: <fullname xsi:null='true'></fullname>
```

2.3 Attribute

1. Default value: All support this feature.

DTD: `Yes` XML Schema: `Yes` XDR: `Yes`
SOX: `Yes` Schematron: `No` DSD: `Yes`

In an attribute declaration of DTD, if the declaration is neither `#REQUIRED` nor `#IMPLIED`, then the attribute value contains the declared default value.

```
<!ATTLIST list  type   (bullets|ordered) 'ordered'>
<!ATTLIST form  method CDATA #FIXED 'POST'>
```

Here, the attribute `type` of the element `list` has a default value of ``ordered'' while the attribute `method` of the element `form` has a fixed value of ``POST''. Other languages support default values similarly. The following three snippets in the order of XML Schema, XDR and SOX illustrate an attribute `nm` with a default value ``John Doe'':

```
<attribute name='nm' use='default' value='John Doe'/>

<AttributeType name='nm' dt:type='string'/>
<attribute type='fullname' default='John Doe'/>

<attrdef name='nm' datatype='string'>
  <default>John Doe</default>
</attrdef>
```

DSD provides a more sophisticated way of defining default for attributes by associating them with a boolean expression. For instance, in DSD, one can specify a default value of ``John Doe'' for male employees as follows:

```
<Default>
  <Context><Element Name='employee'>
    <Attribute Name='gender' Value='M'/>
  </Element></Context>
  <DefaultAttribute Name='nm' Value='John Doe'/>
</Default>
```

2. Choice among attributes: This feature comes in handy when schema designers want to associate multiple attributes with an element and constrain validity to one attribute at any given time.

DTD: `No` XML Schema: `No` XDR: `No`
SOX: `No` Schematron: `Yes` DSD: `Yes`

Schematron and DSD can express the requirement that exactly one of the two attributes `fn` and `gn` must be present as an attribute of `person` element as follows:

```
<rule context='person'>
  <assert test='@fn or @gn'>Or semantics</assert>
  <assert test='count(attribute::*) = 1'>
    Only one attribute</assert>
</rule>

<ElementDef ID='person'>
  <AttributeDecl Name='fn' IDType='ID'/>
  <AttributeDecl Name='gn' IDType='ID'/>
```

```
  <OneOf>
    <Attribute Name='fn'/><Attribute Name='gn'/>
  </OneOf>
</ElementDef>
```

3. Optional vs. required: In all languages, whether or not an attribute definition is required in a XML document instance can be expressed.

DTD: Yes XML Schema: Yes XDR: Yes
SOX: Yes Schematron: Yes DSD: Yes

To denote an attribute must be present, DTD uses a keyword `#REQUIRED` while XML Schema uses ⟨use='required'⟩ in the attribute declaration. Similarly, in XDR, an attribute ⟨required='yes'⟩ is used while in SOX, an element ⟨required/⟩ is used for mandatory attribute definition. Schematron can enforce this feature using a pattern ⟨assert test='@attribute-name'⟩. Like XDR, DSD supports an attribute ⟨Optional='no'⟩.

4. Domain constraint: Some languages can specify admissible values for attributes.

DTD: Partial XML Schema: Yes XDR: Partial
SOX: Partial Schematron: Yes DSD: Yes

DTD and XDR provide only the enumeration capability by which users can list all legal values for the attribute being defined. For instance, the following snippets show examples of DTD and XDR for such an enumerated attribute type, `RGB`:

```
<!ATTLIST spec RGB (red|green|blue)>
```

```
<AttributeType name='RGB' dt:type='enumeration'
  dt:values='red green blue'/>
```

In XML Schema, domain values for simple types can first be constrained using various facets and then new attributes can be defined using the simple types. SOX provides ⟨enumeration⟩ to constrain the attribute domain. In Schematron, the support for an arbitrary domain constraint rule for attribute values is possible as shown in the case of the domain constraint for datatypes in Section 2.2. In DSD, one can apply numerous operators such as ⟨Union⟩ or ⟨Repeat⟩ to the construct ⟨StringType⟩ to constrain domain values.

5. Conditional definition: Often, an attribute a_1 of an element E is relevant only when an attribute a_2 has a certain value.

DTD: No XML Schema: No XDR: No
SOX: No Schematron: Yes DSD: Yes

For instance, the following Schematron schema states that if the element `E` has the attribute `one`, then it must have the second attribute `two` as well:

```
<rule context='E'>
  <report test='(@one) or not(@one and @two)'>
    E cannot have attribute 'one' alone.</report>
</rule>
```

DSD supports this feature easily using its rich boolean operators. For instance, the following snippet states that the `salary` attribute is defined only when the `student` is a ``TA":

```
<ElementDef ID='student'>
  <If><Attribute Name='TA' Value='yes'>
    <Then><Optional>
      <AttributeDecl Name='salary'/>
    </Optional></Then>
  </If>
</ElementDef>
```

2.4 Element

1. Default value: Elements can have either *simple* or *complex* default values.

DTD: `No`	XML Schema: `Partial`	XDR: `No`
SOX: `No`	Schematron: `No`	DSD: `Yes`

In XML Schema, one can provide a string value as the default value when the element has a simple type.

```
<element name='fullname' type='string'
         default='John Doe'/>
```

DSD allows both simple and complex defaults for elements using ⟨DefaultContent⟩. For instance, one can specify that a default `address` is ``Los Angeles" and ``CA":

```
<Default>
  <Context><Element Name='address'/></Context>
  <DefaultContent>
    <city>Los Angeles</city><state>CA</state>
  </DefaultContent>
</Default>
```

2. Content model: The element content model can be 1) empty, 2) text (including datatype), 3) element, or 4) mixed (text + element).

DTD: `Yes`	XML Schema: `Yes`	XDR: `Yes`
SOX: `Partial`	Schematron: `Yes`	DSD: `Yes`

DTD supports all four content models as follows:

```
empty   : <!ELEMENT o EMPTY>
text    : <!ELEMENT p (#PCDATA) >
element : <!ELEMENT q (x?|y*|z+) >
mixed   : <!ELEMENT r (#PCDATA|x)* >
```

Similarly, XML Schema and XDR support the four content models using a construct ⟨content⟩ which supports values such as ``empty", ``textOnly", ``elementOnly" (``eltOnly" for XDR), ``mixed". Furthermore, XML Schema allows specification of a datatype for an element. SOX supports three content models using constructs ⟨empty/⟩, ⟨string/⟩ and ⟨element/⟩, respectively, but does not explicitly support the mixed content model. In Schematron, the following XPath expression can be used as a value for ⟨assert⟩ construct to specify the four content models:

```
empty   : not(*)
text    : string-length(text()) > 0
element : count(element::*) = count(*)
mixed   : by default
```

DSD also supports all four models using constructs ⟨Empty/⟩, ⟨StringType/⟩, ⟨Element/⟩ and ⟨AnyElement/⟩, respectively.

3. Ordered sequence: The order among sub-elements is consequential.

DTD: `Yes` XML Schema: `Yes` XDR: `Yes`
SOX: `Yes` Schematron: `Yes` DSD: `Yes`

In DTD, sub-elements listed with an operator ``,'' obey the order among them. Likewise, in XML Schema, the order needs to be preserved unless otherwise specified. Otherwise, one can explicitly specify that the order is sequential using a grouping construct ⟨sequence⟩. In XDR, the ⟨order='seq'⟩ attribute specifies that sub-elements are required to appear in a sequential order. SOX supports ⟨sequence⟩ content models as well. For instance, in SOX, the following states that the `person` element must have the sub-element `fn` followed by the sub-element `ln`:

```
<elementtype name='person'>
  <model><sequence>
    <element name='fn'/><element name='ln'/>
  </sequence></model>
</elementtype>
```

The same schema can be written in Schematron as follows:

```
<rule context='person'>
  <assert test='(*[position()=1] = fn)
     and (*[position()=2] = ln)'>
    fn must be followed by ln.</assert>
</rule>
```

The ordered sequence in DSD is expressed in a similar fashion by the construct ⟨Sequence⟩ in an element content definition.

```
<ElementDef ID='person'>
  <Sequence>
    <Element Name='fn'/><Element Name='ln'/>
  </Sequence>
</ElementDef>
```

4. Unordered sequence: The order among sub-elements is inconsequential.

DTD: `No` XML Schema: `Yes` XDR: `Yes`
SOX: `No` Schematron: `Yes` DSD: `Yes`

Unlike SGML which offers an operator & to create an unordered sequence, DTD does not offer an explicit operator for unordered sequence. Instead, one needs to encode all the possible combinations of the sub-elements. For instance, to express an unordered sequence of sub-element (a & b & c) of SGML in DTD, one has to write ((a,b,c)|(a,c,b)|(b,a,c)|(b,c,a)|(c,a,b)|(c,b,a)) or somewhat incorrectly (a|b|c)* [*20*]. Using a grouping construct ⟨all⟩ in XML Schema, one can specify the unordered sequence. In XDR, the ⟨order='many'⟩ attribute specifies that sub-elements can appear in any order. In Schematron, if one does not specify any patterns, then it takes the unordered sequence by default.

In DSD, a single content expression describes a set of allowed sequences of string data and elements. Several content expressions describe all *mergings* of sequences, one from each expression. Thus, by cleverly using this feature, one can capture "floating elements", i.e., mixes of ordered and unordered contents. This feature in DSD is more expressive than the simple ordered or unordered content model.

5. Choice among elements: Only one sub-element among candidates is allowed.

DTD: `Yes` XML Schema: `Yes` XDR: `Yes`
SOX: `Yes` Schematron: `Yes` DSD: `Yes`

DTD uses an operator ``|'' to denote choice among elements. Using a grouping construct ⟨choice⟩ in XML Schema, one can specify that only one of the sub-elements in the group must appear. In XDR, the ⟨order='one'⟩ attribute specifies that only one sub-element can be used. SOX supports the ⟨choice⟩ content model for an element. Schematron can express its choice among elements using rules similar to the case of choice among attributes in Section 2.3. In DSD, the construct ⟨OneOf⟩ is supported as follows:

```
<ElementDef ID='person'>
  <OneOf>
    <Element Name='fn'/><Element Name='gn'/>
  </OneOf>
</ElementDef>
```

6. Min & Max occurrence: In this scheme, the language can support if minimum occurrence is *k* and maximum occurrence is *l*.

DTD: `Partial` XML Schema: `Yes` XDR: `Yes`
SOX: `Yes` Schematron: `Yes` DSD: `Partial`

In DTD, the occurrences of elements can be only primitively controlled by the three kleene operators: 1) ``?'' for 0 or 1, 2) ``*'' for 0 or many and 3) ``+'' for 1 or many. In XML Schema, an element declaration carries minOccurs='*k*' and maxOccurs='*l*'. In XDR, ⟨minOccurs⟩ and ⟨maxOccurs⟩ attributes specify how many times an element can appear within another element. In SOX, an element definition carries ⟨occurs⟩ attribute that indicates the number of repetitions of the instanced element. It can take 1) the three kleene operators (i.e., ?, *, +), 2) a value of the form ``*k*,*l*'', or 3) ``*k*,*''. In Schematron, this can be written as ⟨assert test='count(E)>=k'⟩ and ⟨assert test='count(E)<=l'⟩. In DSD, the occurrences of elements can be specified as ⟨Optional⟩, ⟨ZeroOrMore⟩, ⟨OneOrMore⟩, and ⟨Union⟩, but cannot be specified with respect to the exact minimum and maximum numbers.

7. Open model: An open content model enables additional elements or attributes to be present within an element without having to declare each and every element. This provides an extensibility mechanism.

DTD: No XML Schema: No XDR: Yes
SOX: No Schematron: Yes DSD: No

In XDR, the model is open by default. One has to specify a closed model explicitly with ⟨model='closed'⟩. Suppose, for instance, one has the following person element definition (the city and state elements are defined elsewhere):

```
<ElementType name='address' model='closed'>
  <element type='city'/><element type='state'/>
</ElementType>
<ElementType name='person'>
  <element type='address'/>
</ElementType>
```

This definition states that the address element can have only two sub-elements city and state while the person element can have a sub-element address and possibly others since it is an ``open'' content model. Thus, the following XML document instance is valid although the unknown element name is added to the person element.

```
<person>
  <address>
    <city>Los Angeles</city><state>CA</state>
  </address>
  <name>John Doe</name>
</person>
```

In Schematron, the content model is open by default. The closed model also can be expressed using a count() function in XPath. For instance, the following schema states that the person element is closed (when the name and address are all the sub-elements of the person):

```
<rule context='person'>
  <assert test='count(name|address) = count(*)'>
    There is an extra element.</assert>
</rule>
```

In languages that support ``any'' element concept, since any well-formed XML fragment is allowed for the any element, the open model can be simulated in some sense. However, since this requires the ``any'' element be defined in the schema beforehand, it is less flexible than the explicit open model.

8. Conditional definition: Often, elements are allowed only in certain situations.

DTD: No XML Schema: No XDR: No
SOX: No Schematron: Yes DSD: Yes

For instance, the following Schematron schema states that, in HTML, the element input can appear only if it is inside the element form.

```
<rule context='E'>
  <report test='not(parent::form) and input'>
    Element input cannot appear.</report>
</rule>
```

DSD supports this feature using its boolean operators. The usage is similar to the case of the conditional definition for attributes.

2.5 Inheritance

As in object-oriented inheritance, inheritance is done by *extending* or *restricting* the base type. In this section, we divide the target of the inheritance into *simple* and *complex* types. When some languages support inheritance toward *attribute* and *element* instead, we treat them as the simple and complex type inheritance, respectively.

1. Simple type by extension: In this scheme, new simple types may be created by deriving from other simple types with more relaxed domain constraint. The set of legal values of the new type is a *superset* of that of the base type. No languages support this feature.

DTD: `No` XML Schema: `No` XDR: `No`
SOX: `No` Schematron: `No` DSD: `No`

2. Simple type by restriction: The set of legal values of the new type is a *subset* of that of the base type.

DTD: `No` XML Schema: `Yes` XDR: `No`
SOX: `Yes` Schematron: `No` DSD: `No`

In XML Schema, inheritance among simple types are allowed as shown in the following example, where a 9 digit US `zipcode` is created from the base type `string`:

```
<simpleType name='zipcode' base='string'>
  <pattern value='[0-9]{5}(-[0-9]{4})?'/>
</simpleType>
```

By constraining the domain values using the pattern expression, the legal values for the `zipcode` have been restricted from the `string` type. In SOX, new datatypes may be *refined* from built-in or derived types. For instance, the new datatype `RGB` allows only three values from the `color` type.

```
<datatype name='RGB'>
  <enumeration datatype='color'>
    <option>Red</option>
    <option>Green</option>
    <option>Blue</option>
  </enumeration>
</datatype>
```

3. Complex type by extension:

DTD: `No` XML Schema: `Yes` XDR: `No`
SOX: `Yes` Schematron: `No` DSD: `No`

XML Schema supports type inheritance using constructs ⟨base⟩ and ⟨derivedBy='extension'⟩. Newly added elements are always appended at the end. In SOX, ⟨extends type='basetype'⟩ is supported, where *appending* new elements and attributes are allowed. Given the `person` element defined elsewhere, the following example illustrates how the new element `new-person` inherits the content model of the `person` element and has an additional element `address` and attribute `email`.

```
<elementtype name='new-person'>
  <extends type='person'>
    <append>
      <element name='address' type='addr'/>
    </append>
    <attdef name='email' datatype='string'>
  </extends>
</elementtype>
```

In DSD, any definition can be *redefined* using the ⟨RenewID⟩ and ⟨CurrIDRef⟩ constructs. However, once the new type is defined, the original type is no longer accessible. Therefore, this feature is for *renewing* rather than *deriving*.

4. Complex type by restriction:

DTD: `No`	XML Schema: `Yes`	XDR: `No`
SOX: `No`	Schematron: `No`	DSD: `No`

In XML Schema, it is possible to derive new types by restricting the content models of existing types. The values represented by the new type are a subset of the values represented by the base type. For instance, the following schema shows the newly defined element `E` whose type is `ResItemType` which is required to have at least one `item` sub-element as a new restriction.

```
<complexType name='ItemType'>
  <element name='item' minOccurs='0'>
</complexType>
<complexType name='ResItemType'
  base='ItemType' derivedBy='restriction'>
  <element name='item' minOccurs='1'>
</complexType>
<element name='E' type='ResItemType'>
```

2.6 Being unique or key

1. Uniqueness for attribute: All languages support this feature.

DTD: `Yes`	XML Schema: `Yes`	XDR: `Yes`
SOX: `Yes`	Schematron: `Yes`	DSD: `Yes`

DTD, XDR, SOX and DSD use `ID` type for an attribute to ensure uniqueness while XML Schema uses ⟨unique⟩ where the scope and target object of the uniqueness are specified by ⟨selector⟩ and ⟨field⟩ constructs, respectively. Since Schematron does not have an explicit construct equivalent to `ID` in DTD, uniqueness for an attribute must be simulated using pattern ``count()=1".

2. Uniqueness for non-attribute: Schema languages like XML Schema, Schematron, or DSD specify uniqueness not only for attributes but also for arbitrary elements or even composite objects (attribute + element) in a portion of the document or the whole document.

DTD: No XML Schema: Yes XDR: Partial
SOX: No Schematron: Yes DSD: No

This feature can be easily expressed in XML Schema using the same construct ⟨unique⟩. For instance, the following schema ensures there exists a unique phone element under addr sub-elements of the person element.

```
<unique>
  <selector>person/addr</selector>
  <field>phone</field>
</unique>
```

In XDR, elements support the ID attribute type as if they are attributes albeit this is not implemented yet in Internet Explorer 5.

```
<ElementType name='phone' dt:type='ID'/>
```

However, XDR cannot support uniqueness of composite objects. In Schematron, the same constraint can be written as follows:

```
<rule context='person/addr'>
  <assert test='count(phone) = 1'>
    phone is not unique.</assert>
</rule>
```

3. Key for attribute: In databases, being a key requires being unique as well as not being null. A similar concept is defined in XML Schema.

DTD: No XML Schema: Yes XDR: No
SOX: No Schematron: Yes DSD: No

Using almost identical syntax as ⟨unique⟩, a construct ⟨key⟩ can specify an attribute as a key in XML Schema. In Schematron, this feature can be simulated as follows:

```
<rule context='person'>
  <assert test='@ssn and count(@ssn) = 1'>
    Is ssn unique?</assert>
  <assert test='string-length(@ssn) > 0'>
    Is ssn not empty?</assert>
</rule>
```

4. Key for non-attribute: XML Schema allows specification of arbitrary elements or composite objects as key.

DTD: No XML Schema: Yes XDR: No
SOX: No Schematron: Yes DSD: No

For instance, the following schema in XML Schema defines the combination of an employee's department code (element) and employee's name (attribute) as a key.

```
<key name='ekey'>
```

```
  <selector>employee</selector>
  <field>dept/code</field><field>@name</field>
</key>
```

Schematron supports this feature similarly using patterns.

5. Foreign key for attribute: *Foreign key* states if 1) who is a referencing key and 2) who is being referenced by the referencing key.

DTD: `Partial`	XML Schema: `Yes`	XDR: `Partial`
SOX: `Partial`	Schematron: `Yes`	DSD: `Yes`

Like `ID` type, DTD, XDR, SOX and DSD use `IDREF` type for a referencing attribute. XML Schema uses ⟨keyref⟩. In addition to this, XML Schema and DSD support a method to specify whom the foreign key actually points to using constructs ⟨refer⟩ and ⟨PointsTo⟩, respectively. Furthermore, DSD even allows association of arbitrary boolean expressions with the ⟨PointsTo⟩ construct. Using this, for instance, one can specify ``an attribute *A* points to either attribute *B* in an element E_1 or *C* in element E_2" in DSD. In Schematron, this feature can be expressed using patterns. For instance, the following schema states that `dno` attribute of `employee` element should reference the unique identifier of `dept` element.

```
<rule context = 'employee[@dno]'>
  <assert test='(name(id(@dno)) = 'dept')'>
    Error occurred.</assert>
</rule>
```

6. Foreign key for non-attribute:

DTD: `No`	XML Schema: `Yes`	XDR: `No`
SOX: `No`	Schematron: `No`	DSD: `Yes`

Similar to specifying uniqueness for non-attributes, XML Schema can specify foreign keys for arbitrary elements or composite objects using the same ⟨keyref⟩ construct.

```
<keyref refer='ekey'>
  <selector>project</selector>
  <field>emp-dept</field><field>@ename</field>
</keyref>
```

Similarly, the following DSD example illustrates that an attribute `book-ref` is referencing an element `book`.

```
<AttributeDecl ID='book-ref' IDType='IDRef'>
  <PointsTo>
    <Context><Element Name='book'/></Centext>
  </PointsTo>
</AttributeDecl>
```

2.7 Miscellaneous Features

1. Dynamic constraint: In Schematron, one can selectively turn on and off the constraints using ⟨phase⟩ construct so that only part of the schema constraints can be dynamically evaluated at any given time.

DTD: `No` XML Schema: `No` XDR: `No`
SOX: `No` Schematron: `Yes` DSD: `No`

2. Version: Sometimes it is desirable to allow several different attribute or element definitions with the same name. That is, several *versions* of an attribute or element coexist.

DTD: `No` XML Schema: `No` XDR: `No`
SOX: `No` Schematron: `No` DSD: `Yes`

XML Schema has a construct ⟨version⟩ for schema definition, but the current specification does not define any further semantics for that; it is simply provided as a convenience. DSD utilizes both ``Name" as well as ``ID" attributes for element definition so that the attributes with same names are legal as long as their IDs are different. Furthermore, by using the ⟨RenewID⟩ and ⟨CurrIDRef⟩, any definition can be renewed, making a new version of the definition. For instance, the following schema illustrates the redefinition of the DSD constraint `book-constraints`:

```
<ConstraintDef ID='book-constraints'/>
<ConstraintDef RenewID='book-constraints'>
  <Constraint CurrIDRef='book-constraints'/>
  ... modification ...
</ConstraintDef>
```

3. Documentation: At minimum, all languages support commenting on schema fragments using a construct `<-- comment -->`. However, here we consider documentation features beyond commenting such as: 1) textual description to explain a schema fragment for human readers, 2) embedded documentation for application programs, or 3) error or hint messages to aid schema validation and debugging.

DTD: `No` XML Schema: `Yes` XDR: `No`
SOX: `Yes` Schematron: `Yes` DSD: `Yes`

XML Schema provides ⟨documentation⟩ and ⟨appinfo⟩ elements to support description for both human readers as well as application programs. SOX provides the ⟨intro⟩ element to provide an introduction to the schema as a whole and ⟨explain⟩ element to provide a hook for including documentation within a schema fragment. However, there is no support for automatic debugging message or application programs. In Schematron, by using the assertion semantics provided by constructs ⟨assert⟩ and ⟨report⟩, detailed documentation for validating XML structures can be provided. DSD supports three keywords: ⟨Label⟩, ⟨BriefDoc⟩ and ⟨Doc⟩. Using these, it is straightforward to implement, for instance, a debugging system.

4. Embedded HTML: Due to HTML's popularity, it is often convenient to be able to embed HTML fragments inside XML documents.

DTD: `No` XML Schema: `Yes` XDR: `No`

SOX: `Yes` Schematron: `Partial` DSD: `Yes`

Using ⟨any⟩ element, XML Schema allows specification that any well-formed XML is permissible in a type's content model. Hence, well-formed HTML code can be easily embedded in XML document. SOX provides similar feature using ⟨explain⟩ element. Schematron allows a few HTML tags (e.g., `<p>`, `<emph>`), but not general ones. In DSD, one can use the documentation facility to embed HTML.

5. Self-describability: The following languages provide a meta schema (i.e., representing the schema specification using the schema itself being defined). The meta schema is useful in bootstrapping the implementation of the language.

DTD: `No` XML Schema: `Partial` XDR: `No`
SOX: `No` Schematron: `Partial` DSD: `Yes`

While XML Schema and Schematron provide meta schemas that capture only the syntactic requirements, DSD provides a meta schema that captures both the syntactic and semantic requirements.

Features	DTD 1.0	XML Schema 1.0	XDR 1.0	SOX 2.0	Schematron 1.4	DSD 1.0
Schema						
syntax in XML	No	Yes	Yes	Yes	Yes	Yes
namespace	No	Yes	Yes	Yes	Yes	No
include	No	Yes	No	Yes	No	Yes
import	No	Yes	No	Yes	No	No
Datatype						
built-in type	10	37	33	17	0	0
user-defined type	No	Yes	No	Yes	No	Yes
domain constraint	No	Yes	No	Partial	Yes	Yes
null	No	Yes	No	No	No	No
Attribute						
default value	Yes	Yes	Yes	Yes	No	Yes
choice	No	No	No	No	Yes	Yes
optional vs. required	Yes	Yes	Yes	Yes	Yes	Yes
domain constraint	Partial	Yes	Partial	Partial	Yes	Yes
conditional definition	No	No	No	No	Yes	Yes
Element						
default value	No	Partial	No	No	No	Yes
content model	Yes	Yes	Yes	Partial	Yes	Yes
ordered sequence	Yes	Yes	Yes	Yes	Yes	Yes
unordered sequence	No	Yes	Yes	No	Yes	Yes
choice	Yes	Yes	Yes	Yes	Yes	Yes
mim & max occurrence	Partial	Yes	Yes	Yes	Yes	Partial

open model	No	No	Yes	No	Yes	No
conditional definition	No	No	No	No	Yes	Yes
Inheritance						
simple type by extension	No	No	No	No	No	No
simple type by restriction	No	Yes	No	Yes	No	No
complex type by extension	No	Yes	No	Yes	No	No
complex type by restriction	No	Yes	No	No	No	No
Being unique or key						
uniqueness for attribute	Yes	Yes	Yes	Yes	Yes	Yes
uniqueness for non-attribute	No	Yes	Partial	No	Yes	No
key for attribute	No	Yes	No	No	Yes	No
key for non-attribute	No	Yes	No	No	Yes	No
foreign key for attribute	Partial	Yes	Partial	Partial	Yes	Yes
foreign key for non-attribute	No	Yes	No	No	No	Yes
Miscellaneous						
dynamic constraint	No	No	No	No	Yes	No
version	No	No	No	No	No	Yes
documentation	No	Yes	No	Yes	Yes	Yes
embedded HTML	No	Yes	No	Yes	Partial	Yes
self-describability	No	Partial	No	No	Partial	Yes

Table 1: Summary of the feature comparisons.

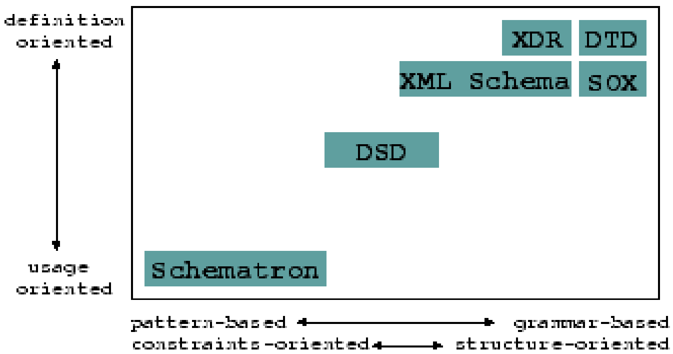
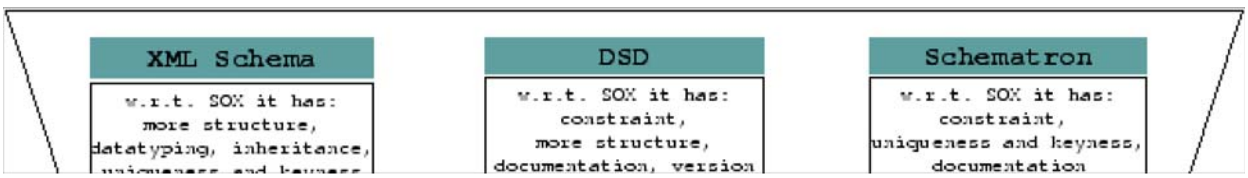


Figure 1: XML schema languages classification.



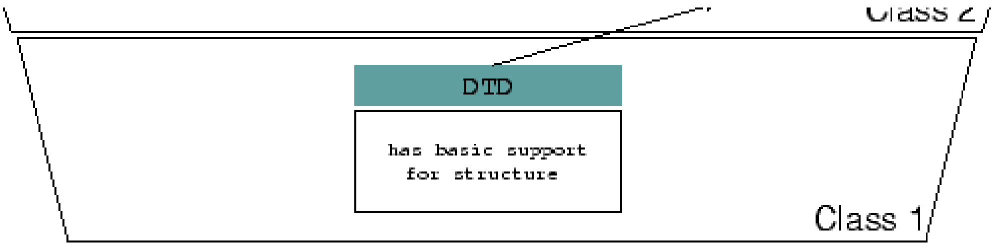


Figure 2: Classification of the expressive power of the six languages

3 Conclusion

Our comparative review of the features is summarized in Table 1.

From an ``ease of use'' point of view, DTD is arguably the easiest schema language to learn despite its use of proprietary syntax. Since the new additions to XDR and SOX are relatively manageable, we think the migration curve from DTD to these languages is not steep. Although the language specification of Schematron is very simple, it exhibits much power. However, this requires users to learn yet another language XPath. Due to the extensive set of features supported by XML Schema and DSD, we expect them to be more difficult to learn than others. Since DSD uses explicit operators for regular expressions (e.g., ⟨Repeat⟩, ⟨OneOf⟩), DSD schema tends to be more verbose than XML Schema or Schematron schema.

From a ``language'' point of view, the six reviewed XML schema languages can be roughly divided into two camps based on factors such as grammar-based vs. pattern-based, definition-oriented vs. validation-oriented, structure-oriented vs. constraints-oriented, etc. The classification is summarized in Figure 1. Based on our study, DTD, XML Schema, XDR and SOX belong to the grammar-based language group while Schematron belongs to the pattern-based language group. DSD stands in-between, supporting both features together. The grammar-based language group especially has an advantage in XML querying since knowing the structure and definition of the schema helps users write more optimized queries and detect errors in the queries more easily. On the other hand, the pattern-based language group is naturally superior with respect to the expressiveness of constraints in the application.

From a ``database'' point of view, no single language suffices the needs completely. The SQL DDL allows specification of not only a set of relations and attributes, but also information about the domain of values associated with each attribute, integrity constraints, indices for each relation, security, etc [*21*]. While XML Schema fulfills the support for a variety of built-in domain types, it could not express, for instance, an arbitrary SQL `CHECK` or `ASSERT` clause. Furthermore, although Schematron or DSD can express such integrity constraints, they have no support of physical indices for boosting performance. Since a substantial amount of web documents are generated from an underlying database by the user's request, it is important to be able to handle such data-centric features as SQL DDL do. We feel this is one of the areas where database researchers can contribute.

From an ``expressive power'' point of view, the six languages can be organized into the following three classes as depicted in Figure 2.

- **Class 1**: DTD has the weakest expressive power. Its support of schema structure is minimal and it severely lacks the support for schema datatype and constraint.
- **Class 2**: XDR and SOX belongs to the middle tier. Their support for schema datatype is not enough

(e.g., lack of null and user-defined type) although schema structure can be supported rather sufficiently. Like DTD, however, they mostly fail to support constraint specification to express the semantics of the schema.

- **Class 3**: XML Schema, Schematron and DSD have the strongest expressive power. Whereas XML Schema supports features for schema datatype and structure fully, Schematron provides a very flexible pattern language that can describe the detailed semantics of the schema. DSD tries to support common features supported by XML Schema (e.g, structure) and Schematron (e.g., constraint) along with some additional features.

In our study, we have found that the support of constraints in the schema language (e.g., Schematron, DSD) is a very attractive feature. However, at the same time, ignoring the schema definition aspect completely like Schematron raises some concern as a general purpose schema language. Although XML Schema identifies many commonly recurring schema constraints and incorporates them into the language specification, we still feel XML Schema is too rigid in that sense. It would be interesting to see if the support of constraints will be added to XML Schema in the future.

Acknowledgment

The authors wish to thank Rick Jelliffe (ASCC) for answering questions regarding Schematron and Michael I. Schwartzbach (BRICS) for his helpful comments on DSD during the writing of this paper.

References

[1]
P. V. Biron, A. Malhotra (ed.) ``XML Schema Part 2: Datatypes'', *W3C*, April 2000. (http://www.w3.org/TR/xmlschema-2)

[2]
A. Bonifati, S. Ceri, ``Comparative Analysis of Five XML Query Languages'', *ACM SIGMOD Record*, 29(1), 2000.

[3]
R. Bourret, J. Cowan, I. Macherius, S. St. Laurent (ed.), ``Document Definition Markup Language (DDML) Specification, Version 1.0'', January 1999. (http://www.w3.org/TR/NOTE-ddml)

[4]
T. Bray, C. Frankston, A. Malhotra (ed.), ``Document Content Description for XML'', July 1998. (http://www.w3.org/TR/NOTE-dcd)

[5]
T. Bray, J. Paoli, C. M. Sperberg-McQueen (ed.), ``Extensible Markup Language (XML) 1.0'', *W3C*, February 1998. (http://www.w3.org/TR/REC-xml)

[6]
J. Clark (ed.) ``XML Transformations (XSLT) Version 1.0'', *W3C*, November 1999. (http://www.w3.org/TR/xslt)

[7]
J. Clark, S. DeRose (ed.) ``XML Path Language (XPath) Version 1.0'', *W3C*, November 1999. (http://www.w3.org/TR/xpath)

[8]

A. Davidson, M. Fuchs, M. Hedin, et al., ``Schema for Object-Oriented XML 2.0", *W3C*, July 1999. (http://www.w3.org/TR/NOTE-SOX)

[9]

D. C. Fallside (ed.), ``XML Schema Part 0: Primer", *W3C*, April 2000. (http://www.w3.org/TR/xmlschema-0)

[10]

C. Frankston, H. S. Thompson, ``XML-Data reduced", *Internet Document*, July 1998. (http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm)

[11]

R. Jelliffe, ``Schematron", *Internet Document*, May 2000. (http://www.ascc.net/xml/resource/schematron/)

[12]

N. Klarlund, A. Moller, M. I. Schwatzbach, ``Document Structure Description 1.0", *Internet Document*, 1999. (http://www.brics.dk/DSD/)

[13]

N. Klarlund, A. Moller, M. I. Schwatzbach, ``DSD: A Schema Language for XML", *Proc. 3rd ACM Workshop on Formal Methods in Software Practice*, 2000.

[14]

S. St. Laurent ``Describing Your Data: DTDs and XML Schemas", *XML.com*, December 1999. (http://www.xml.com/pub/1999/12/dtd/)

[15]

A. Layman, E. Jung, et al., ``XML-Data", *W3C*, January 1998. (http://www.w3.org/TR/1998/NOTE-XML-data)

[16]

M. Makoto, ``RELAX (REgular LAnguage description for XML), *Internet Document*, April 2000. (http://www.xml.gr.jp/relax/)

[17]

Microsoft, ``XML Schema Developer's Guide", *Internet Document*, May 2000. (http://msdn.microsoft.com/xml/XMLGuide/schema-overview.asp)

[18]

N. Miloslav, ``Schematron Tutorial", *Internet Document*, May 2000. (http://www.zvon.org/HTMLonly/SchematronTutorial/General/contents.html)

[19]

D. Raggett, ``Assertion Grammars", *Internet Document*, May 1999. (http://www.w3.org/People/Raggett/dtdgen/Docs/)

[20]

A. Sahuguet ``Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask", *Proc. 3rd Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, 2000

[21]

A. Silberschatz, H. F. Korth, S. Sudarshan, ``Database System Concepts (3rd Edition)", *McGraw-Hill*

Co., 1997.

[22]
H. S. Thopmson, D. Beech, M. Maloney, N. Mendelsohn (ed.) ``XML Schema Part 1: Structures'', *W3C*, April 2000. (http://www.w3.org/TR/xmlschema-1)

1
The title and structure of this paper imitate those of [*2*] in the hope of being a sequel.

This document was translated from L^AT_EX by HEVEA.