MSL:
A model for W3C XML Schema

Allen Brown, Microsoft
Matthew Fuchs, Commerce One
Jonathan Robie, Software AG
Philip Wadler, Avaya Labs
W3C XML Schema Formalism
(W3C working draft)

(editors)
Allen Brown, Microsoft
Matthew Fuchs, Commerce One
Jonathan Robie, Software AG
Philip Wadler, Avaya Labs
“Where a mathematical reasoning can be had, it's as great folly to make use of any other, as to grope for a thing in the dark, when you have a candle standing by you.”

— Arbuthnot
Part I

MSL by example
“Mathematicians are like Frenchmen: whatever you say to them they translate into their own language and forthwith it is entirely different.”

— Goethe
Data in XML

<bib>
  <book year="1999">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
  <book year="2002">
    <title>XML Query</title>
    <author>Fernandez</author>
    <author>Suciu</author>
  </book>
</bib>
Data in MSL

bib [ 
  book [ 
    @year [ 1999 ],
    title [ "Data on the Web" ],
    author [ "Abiteboul" ],
    author [ "Buneman" ],
    author [ "Suciu" ]
  ],
  book [ 
    @year [ 2002 ],
    title [ "XML Query" ],
    author [ "Fernandez" ],
    author [ "Suciu" ]
  ]
]
Elements in Schema

```
<element name="bib">
   <complexType>
      <sequence>
         <element name="book"
               minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>

<element name="book">
   <complexType>
      <sequence>
         <element name="title" type="xsi:string"/>
         <element name="year" type="xsi:integer"/>
         <element name="author" type="xsi:string"
               minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
   </complexType>
</element>
```
Elements in MSL

```xml
component(
    sort = element,
    name = bib,
    content =
        bib [ book* ]
)

component(
    sort = element,
    name = book,
    content =
        book [
            title [ xsi:string ],
            year [ xsi:integer ],
            author [ xsi:string ]+
        ]
)
```
Elements and types in Schema

```xml
<element name="bib" type="bibContent"/>
<complexType name="bibContent">
  <sequence>
    <element name="book"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
<element name="book" type="bookContent"/>
<complexType name="bookContent">
  <sequence>
    <element name="title" type="xsi:string"/>
    <element name="year" type="xsi:integer"/>
    <element name="author"
      minOccurs="0" maxOccurs="unbounded" type="xsi:string"/>
  </sequence>
</complexType>
```
Elements and types in MSL

component(
    sort = element,
    name = bib,
    content =
        bib [ bibContent ]
)

component(
    sort = type,
    name = bibContent,
    content =
        book* [ bookContent ]
)

component(
    sort = element
    name = book,
    content =
        book [ bookContent ]
)

component(
    sort = type,
    name = bookContent,
    content =
        title [ xsi:string ],
        year [ xsi:integer ],
        author [ xsi:string ]+
)
Derivation and abstraction in Schema

```xml
<complexType name="u" final="extension" abstract="false">
  <restriction base="t">
    <choice>
      <element name="d"/>
      <element name="e"/>
    </choice>
  </restriction>
</complexType>
```
Derivation and abstraction in MSL

component(
    sort = type,
    name = u,
    base = t,
    derivation = restriction,
    refinement = { restriction },
    abstract = false,
    content =
        d | e
    )
Part II

Syntax
“I never come across one of Laplace’s ‘Thus it plainly appears’ without feeling sure that I have hours of hard work in front of me.”

— Bowditch
**Syntax**

<table>
<thead>
<tr>
<th>Data</th>
<th>$d ::= @a[d]$</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$e[d]$</td>
<td>element</td>
</tr>
<tr>
<td></td>
<td>$d_1, d_2$</td>
<td>sequence</td>
</tr>
<tr>
<td></td>
<td>()</td>
<td>empty sequence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Groups</th>
<th>$g ::= @a[g]$</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$e[g]$</td>
<td>element</td>
</tr>
<tr>
<td></td>
<td>$g_1, g_2$</td>
<td>sequence</td>
</tr>
<tr>
<td></td>
<td>$g_1 \mid g_2$</td>
<td>choice</td>
</tr>
<tr>
<td></td>
<td>$g^*$</td>
<td>repetition</td>
</tr>
<tr>
<td></td>
<td>()</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
</tbody>
</table>
Abbreviations

\[ g? = g \mid () \] \quad \text{optional}

\[ g^+ = g, g^* \] \quad \text{one or more}

\[ g\{m+1,n+1\} = g, g\{m,n\} \] \quad \text{counting}

\[ g\{0,n+1\} = (g, g\{0,n\})? \]

\[ g\{0,0\} = () \]

\[ g\{m+1,\infty\} = g, g\{m,\ast\} \]

\[ g\{0,\infty\} = g^* \]
Part III

Inference rules
Modus ponens  Frege, 1879

Gentzen, 1934

\[
\frac{\vdash B \to A \quad \vdash B}{\vdash A} \quad (\to\text{-I})
\]
Frege's *Begriffsschrift*, 1879
Inference rules

\[
\frac{d \in g}{e[d] \in e[g]} \quad \text{(element)}
\]

\[
\frac{d_1 \in g_1 \quad d_2 \in g_2}{d_1, d_2 \in g_1, g_2} \quad \text{(sequence)}
\]

\[
\frac{}{() \in ()} \quad \text{(empty)}
\]

\[
\frac{d \in g_1}{d \in g_1 \mid g_2} \quad \text{(choice 1)}
\]

\[
\frac{d \in g_2}{d \in g_1 \mid g_2} \quad \text{(choice 2)}
\]

\[
\frac{d_1 \in g \quad d_2 \in g^*}{d_1, d_2 \in g^*} \quad \text{(repeat 1)}
\]

\[
\frac{}{(\cdot) \in g^*} \quad \text{(repeat 2)}
\]
How typing works: element and sequence

"Data on the Web" ∈ String

ten["Data on the Web"] ∈ title[String]

1999 ∈ Integer

year[1999] ∈ year[Integer]

ten["Data on the Web"],year[1999] ∈ title[String],year[Integer]

book[title["Data on the Web"],year[1999]] ∈ book[title[String],year[Integer]]
How typing works: repetition

```
"B" ∈ String

"A" ∈ String

auth["B"] ∈ auth[String]

auth["A"] ∈ auth[String]

(auth["B"],()) ∈ auth[String]*

auth["A"],(auth["B"],()) ∈ auth[String]*

auth["A"],(auth["B"],()) = (auth["A"],auth["B"]),()

auth["A"],auth["B"]
```
Part IV

Derivation by restriction
YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.

DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?

GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.

Copyright © 2001 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited
“Besides it is an error to believe that rigor in the proof is the enemy of simplicity. On the contrary we find it confirmed by numerous examples that the rigorous method is at the same time the simpler and the more easily comprehended. The very effort for rigor forces us to find out simpler methods of proof.”

— Hilbert
Derivation by restriction

We write $g < \text{res} g'$ if the instances of group $g$ are a subset of the instance of group $g'$. That is, $g < \text{res} g'$ if for every document $d$ such that $d \in g$ it is also the case that $d \in g'$. 
Derivation by restriction

We write \( g < \text{res} \ g' \) if the instances of group \( g \) are a subset of the instance of group \( g' \). That is, \( g < \text{res} \ g' \) if for every document \( d \) such that \( d \in g \) it is also the case that \( d \in g' \).

\[
\forall d. \ d \in g \Rightarrow d \in g' \\
g < \text{res} \ g'
\]  
(restriction)
Part V

Conclusions
What’s in MSL

- Model groups and validity.
- Derivation by extension and restriction.
- Interleaving (all groups).
- Attributes.
- Normalized names.
What’s not in MSL

- Identity constraints.
- The mapping from XML Schema syntax into components.
- Skip and lax wildcard validation.
- The unambiguity restriction on content models.
- The sibling element constraint.
- The xsi:nil attribute.
- A check that abstract components are not instantiated.
- Support for form and form default.
- Support for final, block, use, and value.
- The Post Schema Validation Infoset.
- Atomic datatypes.
“Much intellectual mediocrity can be and actually is concealed by some technique sufficiently recondite to discourage outside criticism.”

— George Sarton
“Never express yourself more clearly than you are able to think.”

— Niels Bohr