Document Structure Description 1.0
an XML schema language

Anders Møller
Michael I. Schwartzbach
BRICS, University of Aarhus

Nils Klarlund
AT&T Labs-Research

What is XML?

- a unified notation for marked-up documents;
- essentially just labeled trees (with internal pointers).

Who invented XML?

- XML is a W3C recommandation.

Why should I care about XML?

- "XML will revolutionize the Web."

Three ways to view XML

- an alternative to HTML;
- a uniform framework for data files;
- a generalization of databases.

XML as an alternative to HTML

```
<h1>Object-Oriented Type Systems</h1>
<ul>
<li><i>Jens Palsberg</i>
<li><i>Michael I. Schwartzbach<i>
</ul>
<a href="http://www.wiley.com/Home.html">John Wiley &amp; Sons</a><br>
ISBN 0 471 94128 X<br>
1994<p>
```

```
<book isbn="0 471 94128 X">
  <title>Object-Oriented Type Systems</title>
  <author> <first>Jens</first><last>Palsberg</last> </author>
  <author> <first>Michael</first><initial>I.</initial><last>Schwartzbach</last> </author>
  <publisher> John Wiley &amp; Sons <homepage>http://www.wiley.com</homepage> </publisher>
  <year>1994</year>
  <reviews></reviews>
</book>
```

Advantages:

- a well-defined format;
- multiple layouts through stylesheets;
- more accurate search engines.

XML as a uniform framework for data files

Weather Observation Markup Format (OMF)

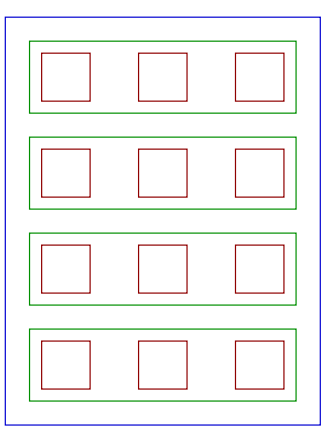
```
<Reports TStamp="888965153">
  <METAR TStamp="888965153" LatLon="36.58, -121.85" BId="724915"
         SName="KMRY, MONTEREY PENINSULA" Elev="77" Vis="80500" Ceiling="INF">
  KMRY 032245Z 29007KT 50SM SKC 15/03 A3003</METAR>
  <SPECI TStamp="888966299" LatLon="36.97, -86.42" BId="746716"
         SName="KBWG, BOWLING GREEN" Elev="167" Vis="12880" Ceiling="2400">
  KBWG 032304Z VRB05KT 8SM BKN024 OVC035 04/01 A2990 RMK AO2</SPECI>
  <METAR TStamp="888968939" LatLon="37.00, -101.9" BId="724604"
         SName="KEHA, ELKHART (AWOS)" Elev="1099">
  KEHA 032348Z AUTO 08004KT 15/M10 RMK AO1 PK WND 06 000 T01501099</METAR>
  <METAR TStamp="888967859" LatLon="36.67, -4.48" BId="84820"
         SName="LEMG, MALAGA (CIV/MIL)" Elev="7" Vis="4000" Ceiling="INF">
  LEMG 2330Z 31006KT 4000 BR FEW008 08/07 Q1027 NOSIG</METAR>
</Reports>
```

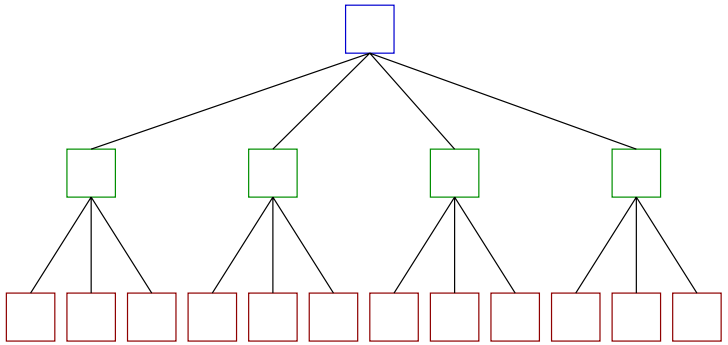
Advantages:

- no problems with tabs, newlines, or character sets;
- allows use of common XML tools;
- makes the format Web-friendly.

XML as a generalization of databases

- a relation is a tree of height 2 with an arbitrary fanout at level one and a fixed fanout at level two;
- an XML document is an arbitrary tree.

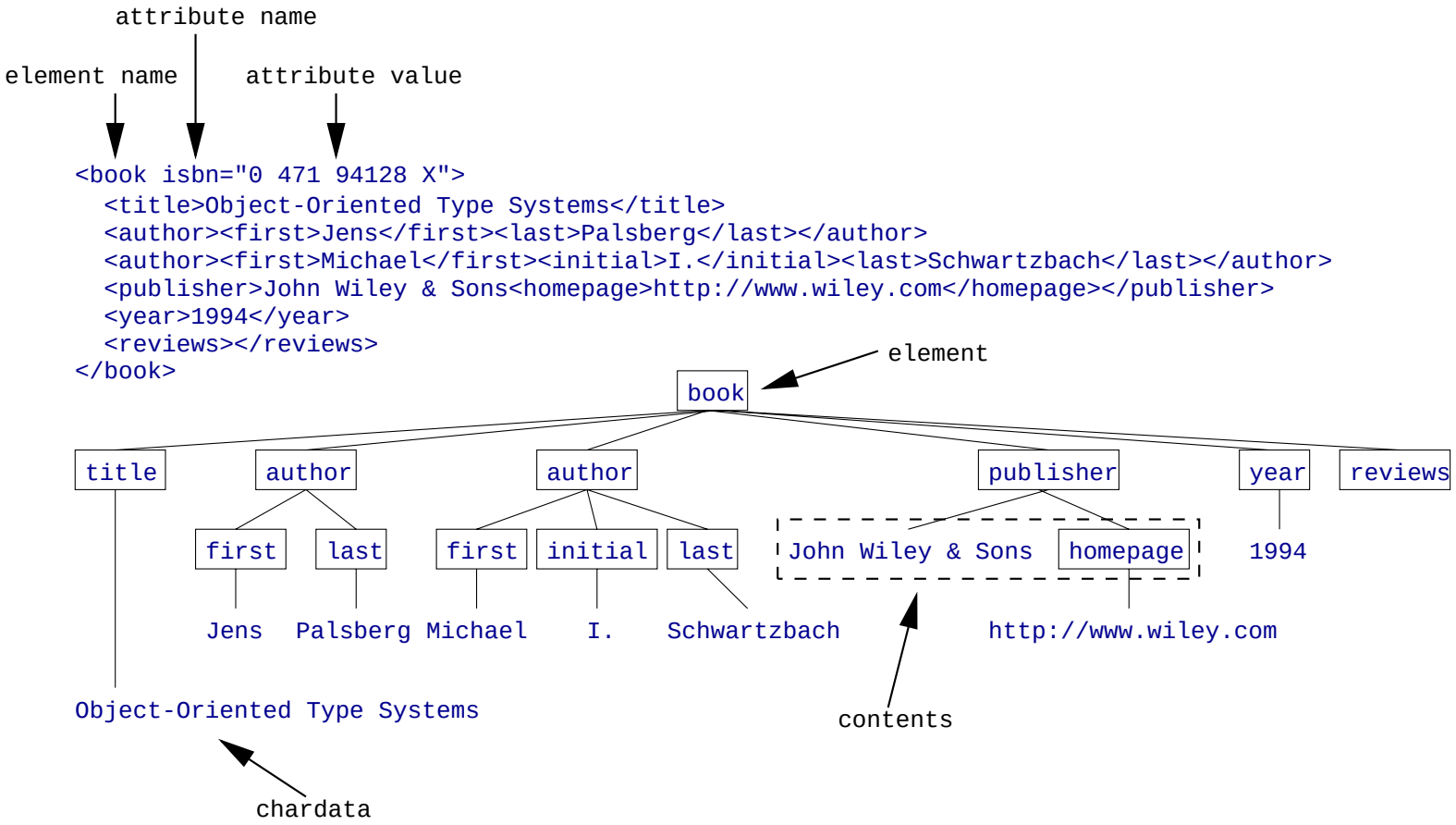




Advantages:

- more liberal schemas;
- allows queries on Web contents.

XML terminology

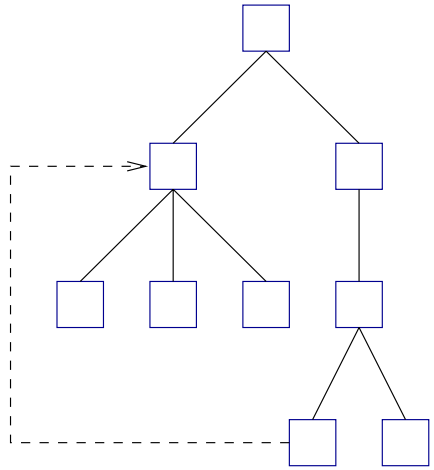


XML IDs and references

- allows pointers between elements in an XML document;
- a special attribute is an ID;
- another special attribute is an IDRef.

General requirements

- all ID values must be distinct;
- each IDRef value must equal some ID value.



XML technologies

- schemas;
- transformations;
- presentations;
- queries.

XML schemas

- define the syntax of XML documents for a particular application domain;
- several initiatives are currently in progress.

Problems with native DTD concept

- terrible syntax (not XML);
- lack of expressiveness.

Tools to expect

- validating parser;
- editor.

XML transformations

- map one XML document into another;

Possible applications

- create different views;
- translate between two application domains.

Tools to expect

- XSLT is de facto standard.

XML presentations

- create human-readable versions of XML documents.

```
<xsl:template match='/'>
 <fo:page-sequence font-family="serif">
  <fo:simple-page-master name='scrolling'/>
  <fo:queue queue-name='body'>
   <xsl:process-children/>
  </fo:queue>
 </fo:page-sequence>
</xsl:template>

<xsl:template match="title">
 <fo:block font-weight="bold">
  <xsl:process-children/>
 </fo:block>
</xsl:template>
```

Tools to expect

- XSL is de facto standard for textual layouts;
- specialized browsers for graphics, animation, etc.

XML queries

- extract and combine information from XML documents.

```
// who and in what act and scene utters the line "Et tu, ..."?

construct <quote>
            <person>$sp</>
            <act>$at</>
            <scene>$st</>
          </quote>
where     <play.act>
            <title>$at</>
            <scene>
              <title>$st</>
              <speech>
                <speaker>$sp</speaker>
                <line>"Et tu, Brute! Then fall, Caesar."</line>
              </speech>
            </scene>
          </> in "http://www.research.att.com/~mff/xmlql-demo/xml/shakespeare/j_caesar.xml"
```

Tools to expect

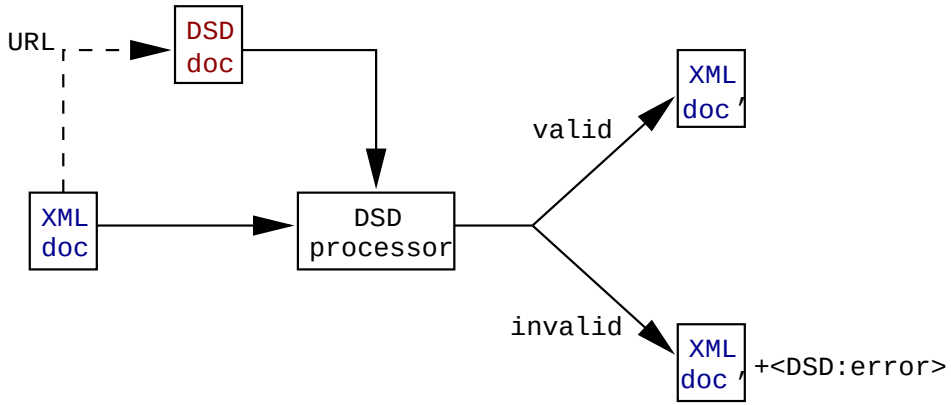
- query engines, e.g. XML-QL.

Document Structure Description 1.0

- is an XML schema language;
- is itself written in XML notation;
- is very expressive;
- is based on familiar concepts;
- is self-describable;
- allows linear-time processing of XML documents.

The DSD processor

- checks if an XML document is valid;
- generates error messages if not;
- may insert element, chardata, and attribute defaults to obtain a valid document.



DSD processing of an XML document

- a single top-down traversal of the document tree;
- each element node is assigned an elementID;
- an elementID determines a name and a constraint.

For each such element node

- verify that the names are identical;
- verify that the attributes and contents satisfy the constraint;
- as the contents is being processed, the nodes are assigned their individual elementIDs.

Element constraints

- describes which attribute names and values are legal;
- describes what contents is legal.

Syntactic categories

- attribute declarations;
- boolean expressions;
- context expressions;
- stringtype expressions;
- content descriptions.

DSD constraint expressions

constraintexp → *constraintterm**

constraintterm → *attributedecl* |
contentexp |
boolexp |
<If> *boolexp*
<Then> *constraintexp* </Then>
(<Else> *constraintexp* </Else>)?
</If>

Attribute declarations

- allow the presence of given attributes;
- may impose restrictions on their values;
- may impose restrictions on targets of references;
- each attribute present must be declared once.

DSD attribute declarations

attributedecl → <AttributeDecl Name=*AttValue attrdeclattrs*>
attrdeclcontent
</AttributeDecl>

attrdeclattrs → (Optional="*YesOrNo*")? (IDType="*IDType*")?

attrdeclcontent *stringtypeexp*? *pointsto*?

IDType (ID | IDRef | RenewID | CurrIDRef)?

pointsto <PointsTo> *boolexp* </PointsTo>

Conditional constraints

- allow subconstraints to be guarded by boolean expressions.

Example

- for an HTML `input` element, the `maxlength` attribute should only be allowed if the attribute `type` is present and has value `text` or `password`.

DSD boolean expressions

boolexp → <And> *boolexp** </And> |
<Or> *boolexp** </Or> |
<OneOf> *boolexp** </OneOf> |
<Not> *boolexp** </Not> |
<Imply> *boolexp* boolexp </Imply> |
<Equiv> *boolexp** </Equiv> |
attributedescr |
<Context> *contextexp* </Context>

attributedescr → <Attribute Name=*AttValue* Value=*AttValue*> |
<Attribute Name=*AttValue*>
stringtypeexp?
</Attribute>

Context sensitivity

- allows constraints to depend on the context of the node;
- the context is the path from the root to the node.

Examples

- HTML `input` elements are only allowed inside a `form` element;
- the `title` element is optional inside a `book` element with an `isbn` attribute.

DSD context expressions

contextexp → *contextterm**

contextterm → <SomeElements/> |
<Element (IDRef=*AttValue* | Name=*AttValue*)? >
attributedescr*
</Element>

Legality of attribute values

- is specified through standard regular expressions;
- all reasonable data types can be captured in this manner.

Example

- time and date formats;
- URI syntax;
- social security numbers;
- e-mail addresses;
- ISBN numbers.

DSD stringtype expressions

stringtypeexp $\rightarrow$

<Sequence> *stringtypeexp** </Sequence> |

<Optional> *stringtypeexp* </Optional> |

<ZeroOrMore> *stringtypeexp* </ZeroOrMore> |

<OneOrMore> *stringtypeexp* </OneOrMore> |

<Union> *stringtypeexp** </Union> |

<Intersection> *stringtypeexp** </Intersection> |

<Complement> *stringtypeexp* </Complement> |

<Empty/> |

<String Value=*AttValue*/> |

<CharSet Value=*AttValue*/> |

<CharRange Start="*Char*" End="*Char*" /> |

<AnyChar/>

Sequential contents

- is specified through standard regular expressions.

Example

- a `book` contains one or more `author`, a `publisher`, a `year`, and optionally some `reviews`;
- the corresponding regular expression is:

```
author+ publisher year reviews?
```

DSD content descriptions

```
contentexp  →  <Sequence> contentexp* </Sequence> |
               <Optional> contentexp </Optional> |
               <ZeroOrMore> contentexp </ZeroOrMore> |
               <OneOrMore> contentexp </OneOrMore> |
               <Union> contentexp* </Union> |
               <AnyElement/> |
               <Empty/> |
               <If> boolexp
                 <Then> contentexp </Then>
                 (<Else> contentexp </Else>)?
               </If> |
               stringtype |
               elementdescr
```

Unordered contents

- is a problem for regular expressions;
- may cause a combinatorial explosion.

Example

- an `author` node must contain in sequence `first`, optionally `initial`, and then `last`; anywhere in between, a `homepage` is allowed;
- the corresponding regular expression is:

```
f(i?)l | hf(i?)l | fh(i?)l | f(i?)hl | f(i?)lh
```

Solution

- multiple descriptions and projected contents.

Multiple content descriptions

- are checked one at a time;
- only look at the contents projected onto the elements that they each mention;
- each consume the content nodes that they recognize;
- at the end, each content node must be consumed once.

Example

- the two descriptions `f(i?)l` and `h` in combination solve the earlier problem.

Content descriptions are eager

- `A*` consumes as many `A`-elements as possible;
- `x | y | z` chooses the first summand that succeeds;
- full backtracking is not performed.

Algebraic rules are no longer valid

- `A*A` $\not\equiv$ `A+`;
- `AA | AAB` $\not\equiv$ `AA(B?)`;
- this is not a problem in ordinary use.

DSD defaults

- allow the processor to insert missing attributes, elements, or chardata;
- defaultability must be explicitly declared;
- insertion is guided by the constraint checking;
- available defaults are guarded by context expressions;
- similar to XSL, but independent of formatting models;
- often allows briefer XML documents;
- cascading defaults can be specified in the application XML document.

DSD definitions

- allow construction to be identified with IDs;
- these definitions can be invoked through IDRefs.

```
<ConstraintDef ID="select_constraint">
  <AttributeDecl Name="class" Optional="yes"/>
  <OneOrMore>
    <Element IDRef="option_element"/>
  </OneOrMore>
  <Element Name="help" Defaultable="yes">
    <Content IDRef="audio_content"/>
  </Element>
  <Element Name="prompt" Defaultable="yes">
    <Content IDRef="audio_content"/>
  </Element>
</ContentDef>

<ElementDef ID="select">
  <Constraint IDRef="select_constraint"/>
</ElementDef>
```

DSD redefinitions

- allow previous definitions to be extended or updated;
- `RenewID` gives a new meaning to a definition;
- `IDRef` refers to the newest definition;
- `CurrIDRef` refers to the textually closest definition.

```
<?include URI= "http://www.brics.dk/DSD/examples/1.0/phone_with_constraints.dsd"?>

<ConstraintDef RenewID="select_constraint">
  <Constraint CurrIDRef="select_constraint"/>
  <AttributeDecl Name="interaction_class" Optional="yes"/>
  <AttributeDecl Name="ATT:speech-engine" Optional="yes"/>
  <AttributeDecl Name="ATT:speech-engine-timeout" Optional="yes"/>
</ContentDef>
```

The book example: books.dsd

```
<?dsd URI="http://www.brics.dk/DSD/dsd.dsd"?>

<DSD IDRef="books" DSDVersion="1.0">

<ElementDef ID="books">
   <ZeroOrMore><Element IDRef="book"/></ZeroOrMore>
</ElementDef>

<ElementDef ID="book">
  <AttributeDecl Name="isbn" Optional="yes">
     <StringType IDRef="isbn"/>
  </AttributeDecl>
  <Sequence>
    <If><Attribute Name="isbn"/>
        <Then><Optional><Element IDRef="title"/></Optional></Then>
        <Else><Element IDRef="title"/></Else>
    </If>
    <OneOrMore><Element IDRef="author"/></OneOrMore>
    <Element IDRef="publisher"/>
    <Element Name="year"><StringType IDRef="digits"/></Element>
    <Optional><Element IDRef="reviews"/></Optional>
  </Sequence>
</ElementDef>

<ElementDef ID="title" Defaultable="yes">
   <StringType IDRef="simple"/>
</ElementDef>
```

The book example: books.dsd

```
<ElementDef ID="author">
  <Sequence>
     <Element Name="first"><StringType IDRef="simple"/></Element>
     <Optional>
       <Element Name="initial"><StringType IDRef="simple"/></Element>
     </Optional>
     <Element Name="last"><StringType IDRef="simple"/></Element>
  </Sequence>
  <Optional><Element IDRef="homepage"/></Optional>
</ElementDef>

<ElementDef ID="publisher">
  <StringType IDRef="simple"/>
  <Optional><Element IDRef="homepage"/></Optional>
</ElementDef>

<ElementDef ID="homepage">
  <StringType IDRef="url"/>
</ElementDef>

<ElementDef ID="reviews">
   <ZeroOrMore>
      <Element Name="review"><StringType IDRef="url"/></Element>
   </ZeroOrMore>
</ElementDef>
```

The book example: books.dsd

```
<StringTypeDef ID="simple">
   <OneOrMore>
      <Union>
         <CharRange Start="a" End="z"/>
         <CharRange Start="A" End="Z"/>
         <CharSet Value="._- &amp;amp;"/>
      </Union>
   </OneOrMore>
</StringTypeDef>

<StringTypeDef ID="digits">
  <ZeroOrMore><CharRange Start="0" End="9"/></ZeroOrMore>
</StringTypeDef>

<StringTypeDef ID="url">
  <Sequence>
    <String Value="http://">
    <ZeroOrMore><AnyChar/></ZeroOrMore>
  </Sequence>
</StringTypeDef>
```

The book example: books.dsd

```
<StringTypeDef ID="isbn">
  <Sequence>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <StringType IDRef="digitspace"/>
     <CharSet Value="0123456789X"/>
  </Sequence>
</StringTypeDef>

<StringTypeDef ID="digitspace">
  <Sequence>
    <CharSet Value="0123456789"/>
    <Optional><String Value=" "/></Optional>
  </Sequence>
</StringTypeDef>

<Default>
  <Context><Element Name="book"/></Context>
  <DefaultContent><title>Untitled</title></DefaultContent>
</Default>

</DSD>
```

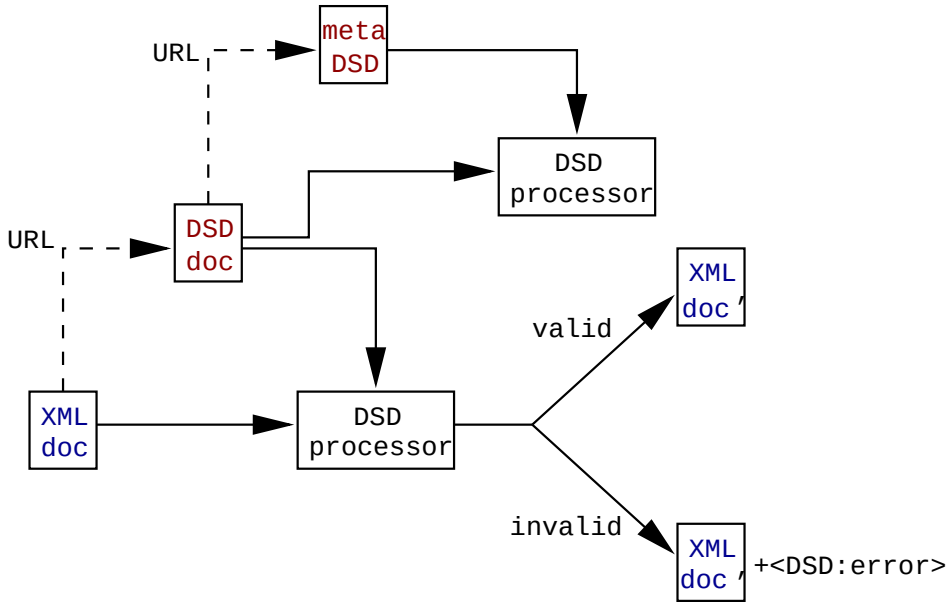
The book example: books.xml

```
<?dsd URI="http://www.brics.dk/DSD/Examples/books.dsd"?>

<books>
  <book isbn="0 471 94128 X">
    <title>Object-Oriented Type Systems</title>
    <author> <first>Jens</first><last>Palsberg</last> </author>
    <author>
       <first>Michael</first>
       <initial>I.</initial>
       <homepage>http://www.brics.dk/~mis/</homepage>
       <last>Schwartzbach</last>
    </author>
    <publisher>John Wiley &amp;amp; Sons <homepage>http://www.wiley.com</homepage> </publisher>
    <year>1994</year>
    <reviews>
      <review>
         http://www.amazon.com/exec/obidos/ISBN%3D047194128X/highcounthostbooA/002-2012864-0937861
      </review>
      <review>
         http://www.informika.ru/eng/accu/bookcase/books14.html#11167
      </review>
    </reviews>
  </book>
</books>
```

DSD is self-describable

- a DSD specification is itself an XML document;
- legal DSD specifications are described by a particular DSD, the *meta-DSD*;
- this captures *all* static requirements.



DSD implementation

- is available in open source distribution;
- experimental prototype;
- guarantees linear-time processing of XML documents (the constant depends on the given DSD);
- self-application of 450 line meta-DSD takes $\frac{1}{2}$ sec.

Related efforts

- XML Schema (W3C working group);
- DCD, Sox, DDML.

Unique features of DSD

- conditional constraints;
- element IDs as non-terminals;
- constraints on reference targets;
- context-based default insertion;
- simple redefinitions;
- linear-time implementation.

DSD project homepage: `http://www.brics.dk/DSD/`

DSD

Document Structure Description

A *Document Structure Description (DSD)* is a specification of a class of XML documents together with a default mechanism and documentation. The DSD notation is defined in the DSD specification. DSD is being developed at AT&T Labs Research and at BRICS, University of Aarhus.

DSD provides an alternative to XML DTDs. It adds expressive power, increases readibility, and contains support for default attributes and contents. Furthermore, it guarantees linear time processing in the size of the application document.

A prototype DSD processor has been implemented. It is freely available for experimentation and further development.

The DSD language has been designed by Nils Klarlund, Anders Møller, and Michael I. Schwartzbach.

Available resources:

- **DSD 1.0 Specification**
- **DSD description of DSD**
- **Prototype implementation, including all source code**
- **DSD–to–HTML XSL style sheet**
- **Example DSDs and application documents**
- **Presentations of DSD**
