Summer Course - Parallel XML Processing

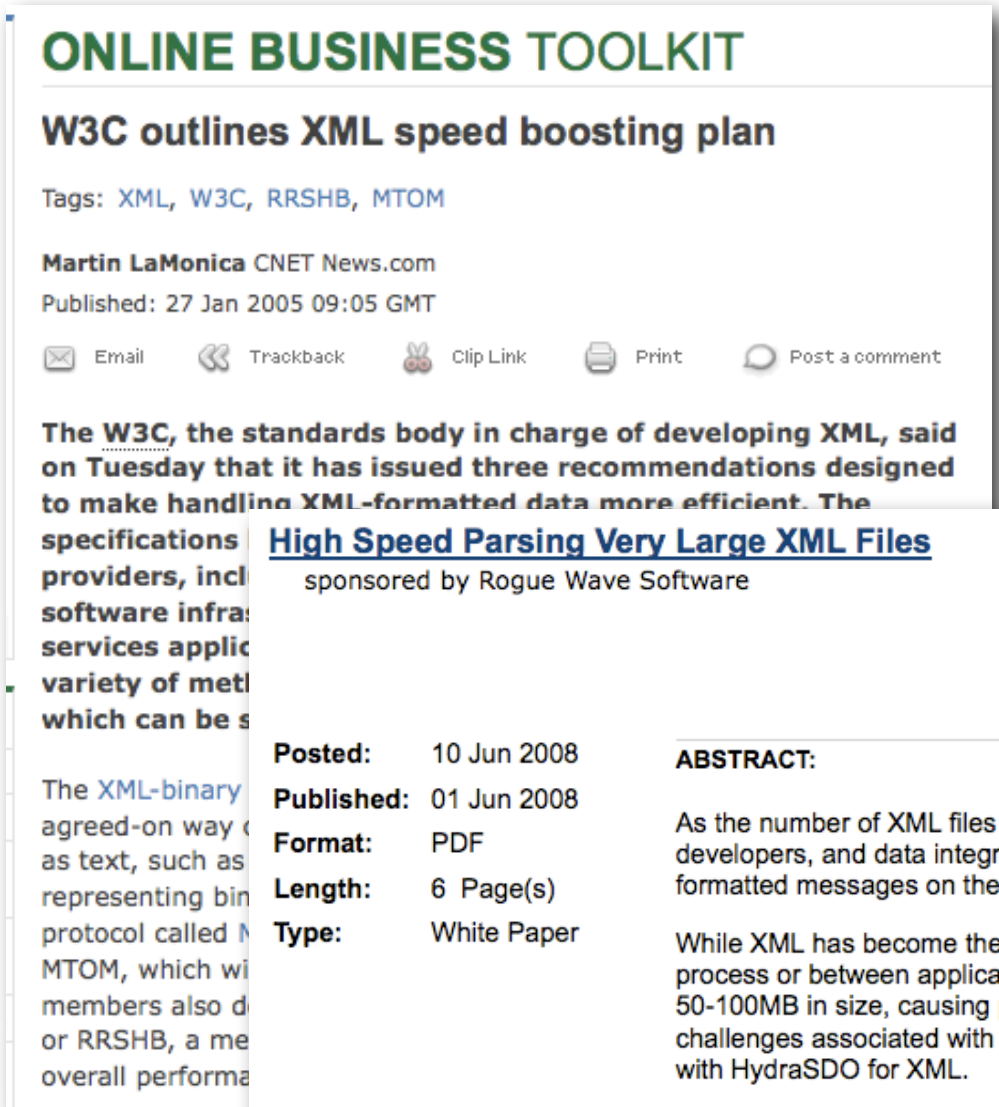
Peter Tröger

Blekinge Institute of Technology

Summer 2008

Examples partially adopted from www.w3schools.com - visit their site !

Parallel XML Processing ???



ONLINE BUSINESS TOOLKIT

W3C outlines XML speed boosting plan

Tags: XML, W3C, RRSHB, MTOM

Martin LaMonica CNET News.com

Published: 27 Jan 2005 09:05 GMT

Email Trackback Clip Link Print Post a comment

The W3C, the standards body in charge of developing XML, said on Tuesday that it has issued three recommendations designed to make handling XML-formatted data more efficient. The specifications providers, incl software infra services applic variety of met which can be s

The XML-binary agreed-on way as text, such as representing bin protocol called MTOM, which wi members also d or RRSHB, a me overall performa

High Speed Parsing Very Large XML Files

sponsored by Rogue Wave Software

ROGUE WAVE SOFTWARE

Posted:	10 Jun 2008
Published:	01 Jun 2008
Format:	PDF
Length:	6 Page(s)
Type:	White Paper

ABSTRACT:

As the number of XML files in enterprise organizations significantly increases, architects, application developers, and data integration specialists must deal simultaneously with the growing number of XML formatted messages on the network as well as their rapidly expanding file sizes.

While XML has become the 'de facto' standard for transferring data from one step to the next in a business process or between applications, there are more and more occasions when those files are easily exceeding 50-100MB in size, causing performance slow downs and memory overloads. This paper describes the challenges associated with parsing very large XML files in today's system environments and offers a solution with HydraSDO for XML.

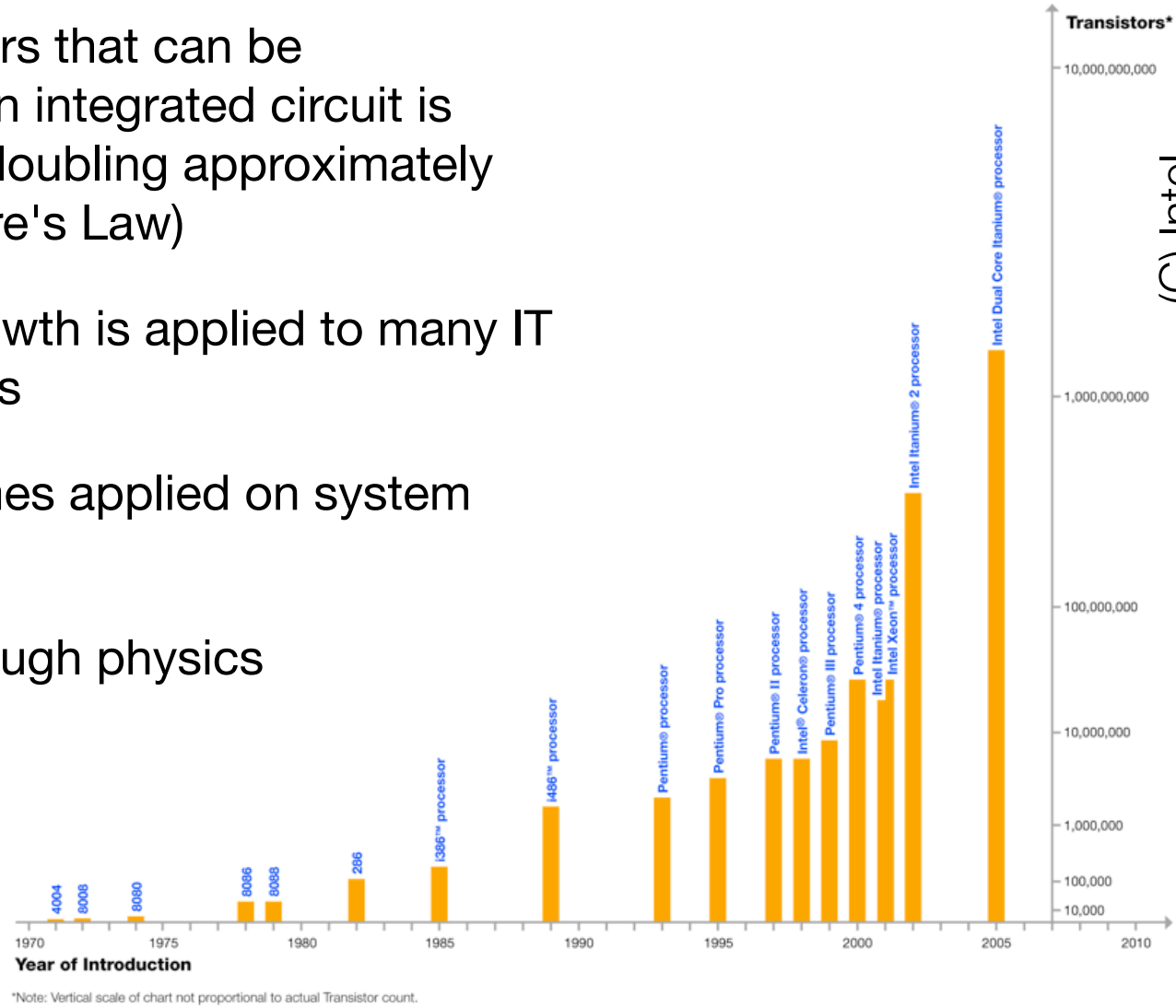
Rogue Wave introduces a new solution for parsing very large XML files: HydraSDO for XML. It is specifically

Three ways of
doing anything faster [Pfister]

- Work harder
- Work smarter
- Get help

Work Harder

- „...the number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years. ...“ (Moore's Law)
  - Rule of exponential growth is applied to many IT hardware developments
  - Density rule is sometimes applied on system performance
- Meanwhile limitations through physics
  - Cooling, tunneling



(C) Intel

Getting Help by Parallelism

- Where ?
  - Inside the processor (instruction-level parallelism, multicore)
  - Through multiple processors in one machine (multiprocessing)
  - Through multiple machines (distributed computing)
- How ?
  - Data Parallelism - Same operation on disjoint data sets
  - Control / Task Parallelism - Perform different operations at the same time
    - Program must be constructed of independent portions, or contain control structures to be carried out in parallel
  - Flow Parallelism - Overlapping work on data stream (pipelines, assembly line model)

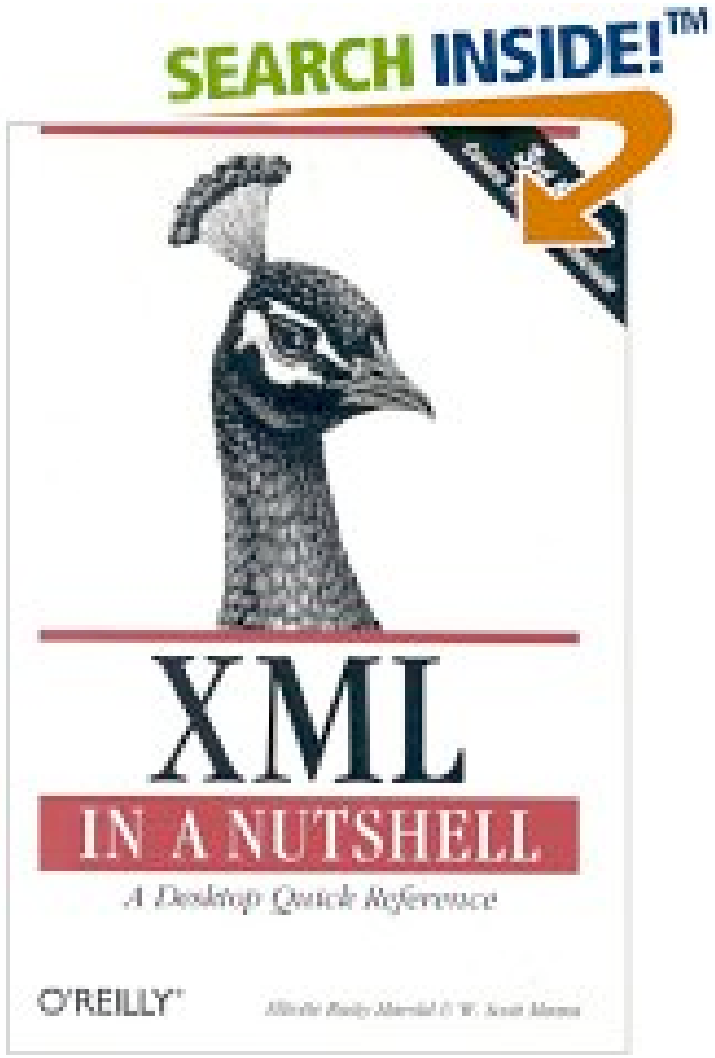
Three ways of XML processing speedup

- Faster hardware
  (XSLT processors, ...)
- More condensed XML formats
  (XOP, SOAP over TCP, ...)
- **Parallel processing**
  (multicore, SMP)

Course Agenda

- *Investigate (and improve) state-of-the-art in parallelized XML processing*
- **Today / Tomorrow**: Crash course in XML technology and vocabulary
  - First assignment: Review of one XML parallelization paper
- 3.7.2008: Mutual presentation and discussion of reviewed paper (30 min)
  - Second assignment: Choosing one paper and performing in-depth analysis, including experiment
- 28.7. / 29.7.2008: Mutual presentation and discussion of experiment results
  - Third assignment: Written report about possible improvements of paper, or about parallel XML processing in the participants own research field
- This is a summer course - relax ...

Course Book



(C) Amazon

- You might also consult
  - Annotated XML specification (http://www.xml.com/axml/testaxml.htm)
  - All original specifications (http://www.w3.org/TR)
  - Endless amount of online tutorials
    - High quality from IBM DeveloperWorks
    - http://www.w3schools.com
- Papers about parallel XML processing are provided on the course home page

Outline

- **XML Basics**
  - Markup concepts, schema definition, DTD
  - Namespaces
- Document-centric XML
  - XSL Transformations, XPath
  - XML Rendering (CSS, XSL-FO)
- Data-centric XML
  - XML Schema, XQuery
  - Programming models

XML - The Origin

- **S**tandard **G**eneralized **M**arkup **L**anguage (SGML)
  - Invented by IBM in the 70‘s, ISO standard 8879 in 1986
- Most successful SGML application - **H**yper**T**ext **M**arkup **L**anguage
- Too complicated for complete implementation - „**S**ounds **G**reat **M**aybe **L**ater“
- Februar 1998: „Lite“ version of SGML for structural markup - **XML 1.0**
  - Development started 1996
  - Second edition 2000, third edition 2004, fourth edition 2006
- XML 1.1 published on 04 February 2004, second edition 2006
  - Minor changes only, not widely implemented
  - Support for Unicode 4.0 (e.g. Amharic) in markup

SGML in 1985

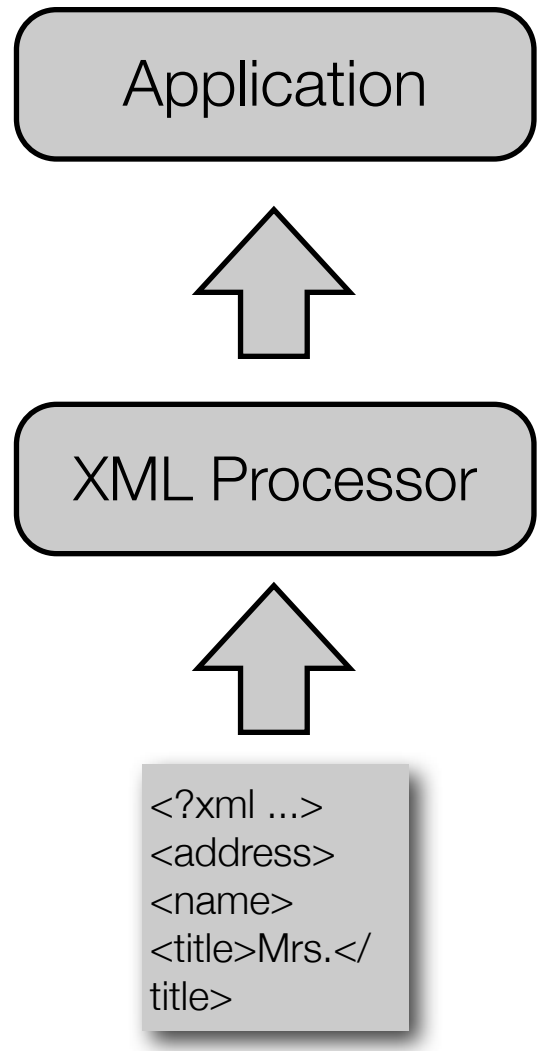
(C) Wikipedia

XML

- *eXtensible Markup Language*
- Design goals
  - Straightforwardly usable over the Internet
  - Support for a variety of applications
  - Compatible with SGML
- W3C recommendation(s) - lexical grammar and parser requirements
- Meta-markup language for text documents
  - W3C XML specification describes grammar for XML documents
  - Compliant documents are **well-formed**
- Restricted set of tags leads to one **XML application** (e.g. vector graphics)

XML is Everywhere

- Document format of choice for most newly developed software
- Stock trading, games, mobile phones, physics, journalism, comics, insurance business, multimedia, travel business, telecommunication, software, ...
- **XML technologies**
  - XML Schema, DOM, SAX, XHTML, XSLT, ...
- **XML applications - special XML-based formats**
  - SVG, MathML, RDF, SOAP, ...
  - Accordingly used by specialized software
- Narrative documents vs. data-centric documents



XML as Document Format

- „Narrative documents“ - reflects SGML background
  - Used for large-scale technical documentation
- Structuring of human-readable documents
  - Chapters, sections, headings, lists, paragraphs, ...
  - Removal of all markup roughly keeps the original text semantic
  - Non-textual information is typically meta-data
- No information about rendering of structured content
  - Separate XML technologies to describe layout (CSS)
  - Description of transformation to renderable format (XSLT, DSSSL)

XML as Data Format

- Original intention of XML was a document format
- Major issue of data format portability lead to XML as data format
  - Easy generation and parsing (compared to binary formats)
  - Support for complex nested data structures
  - Independent from programming language and operating system
- Interoperability ‚glue' with XML-based network protocols
  - REST, XML-RPC, SOAP, BEEP, XMMP
- Has other implications than XML documents
  - Demand for strong typing (XML schema) and programming models

Common Misunderstandings

- HTML != XML, even though both came from SGML
  - HTML parsers accept sloppy markup, XML parsers not
  - HTML is for humans, XML for machines
  - Tags and elements are pre-defined in HTML; XML gives only syntax rules
  - XML tags describe semantic, HTML tags describe layout
- XML is not a programming language
  - Just a document, no compiler to executable code
- XML is not a network protocol
  - Protocol can send around XML documents (SOAP over HTTP)

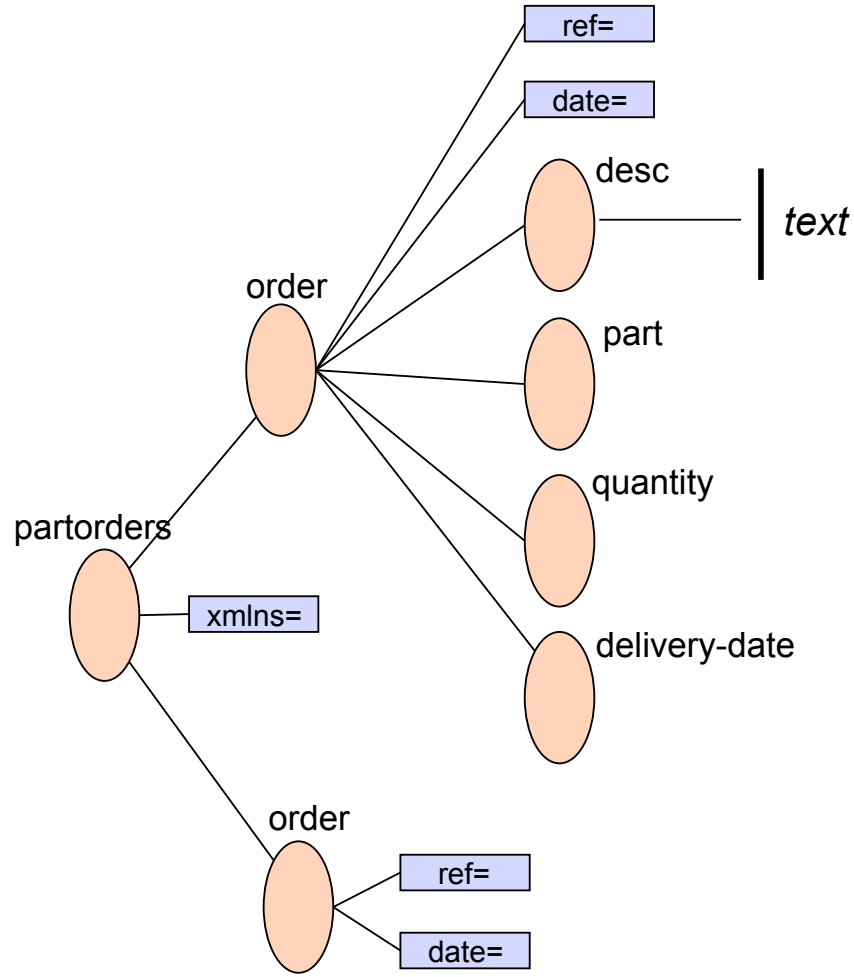
XML Markup Concepts

```
<address>
  <name>
    <title>Mrs.</title>
    <first-name>
      Mary
    </first-name>
    <last-name>
      McGoon
    </last-name>
  </name>
  <street>
    1401 Main Street
  </street>
  <city state="NC">Anytown</city>
  <postal-code>
    34829
  </postal-code>
</address>
```

- *Tag*
  - Text between angle brackets
  - Starting / ending tag
- *Element*
  - Start / end tag and *element content* in between
  - Child elements
  - Empty elements
- *Attribute*
  - Name-value pair inside the starting tag

XML Tree Structure

```
<partorders xmlns="...">
  <order date="..."
         ref="...">
    <desc> ..text..
    </desc>
    <part />
    <quantity />
    <delivery-date />
  </order>
  <order ref=".." .../>
</partorders>
```



Basic rules

- All XML documents must contain a single **root element**
- All elements (but the root element) have one parent element
- Elements can be nested, but not overlapped
- End tags are required
- Elements are case-sensitive
- Attributes must have (consistent) quoted values
- Attributes must be unique inside the element
- Recommended (but not mandatory) XML declaration on start
  - `<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>`

Basic rules

- White space
  - White space characters: tab, line-feed, carriage-return, single space
  - White space in element content is **significant**
  - White space in attributes in normalized (condensed to single one)
  - White space between elements is ignored
- EOL is normalized by XML processor
- Document must comply to defined character encoding (default UTF-8)

```
<?xml version="1.0" encoding="UTF-8"?>
<俄語>Данные</俄語>
```

XML Entities

- Macro definition for the document, as part of the DTD
- Predefined entity references for special characters
  - *<* for the less-than sign
  - *>* for the greater-than sign
  - *"* for a double-quote
  - *'* for a single quote (or apostrophe)
  - *&* for an ampersand
- Only detected and replaced in element content and attribute values

CDATA Section

```
<script>
  <![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
   {
   return 1;
   }
else
   {
   return 0;
   }
}
  ]]>
</script>
```

- Special section for unparsed character data
- Intended for human editors or too many special characters
  - XML example in DOCBOOK document
  - JavaScript code in XHTML web page

Processing Instruction

```
<?xml-stylesheet
  type="text/css" media="screen"
  title="Regular fonts"
  href="regular.css"?>

<?mso-application
progid="InfoPath.Document"?>

<list>
  <?odbc2xml q1:select * from cartoon?>
  <item>
    <?odbc2xml q1.Name?> says
    <?odbc2xml q1.CatchPhrase?>
  </item>
</list>
```

- Provide information to specific applications reading the document
- PI‘s are markup, but no elements
  - Can occur everywhere, like comments
  - Typical use for style sheet declaration
- XML declaration is not a processing instruction

Schema

- **XML application** demands description of acceptable markup
  - **Schema definition**
- Comparison of document instances with schema definition
  - Compliant documents are **valid**
  - Validity of a XML document always depends on given schema definition
  - All documents must be **well-formed**, and might be **valid**
- Many different XML schema languages
  - **Document Type Definition (DTD)** language, as part of XML 1.0
  - **XML Schema** language, separate W3C specification
  - Schematron, Hook, RELAX NG, Examplotron, ...

XML Application Example

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

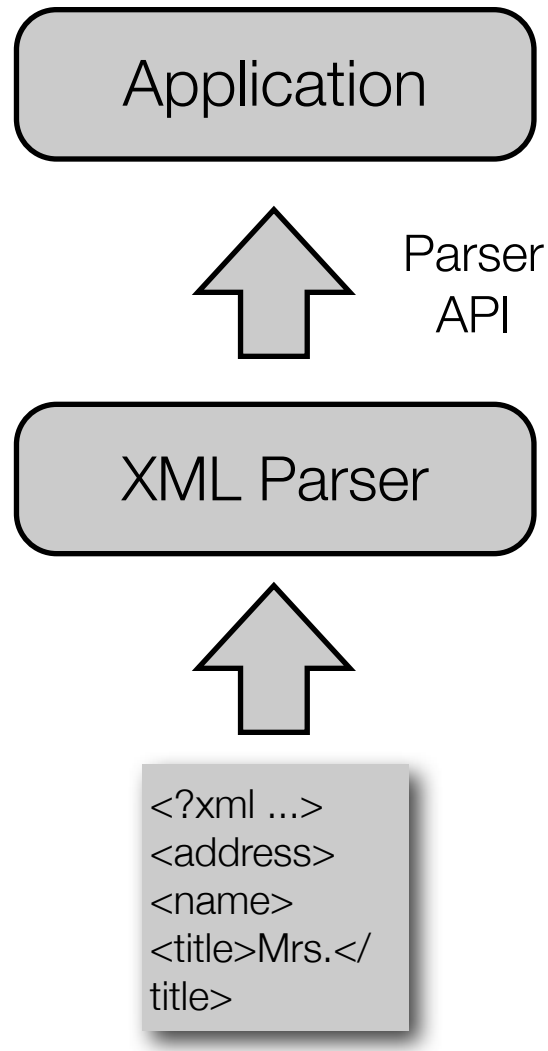
<svg width="100%" height="100%" version="1.1
     xmlns="http://www.w3.org/2000/svg">

  <defs>
    <linearGradient id="orange_red" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1"/>
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1"/>
    </linearGradient>
  </defs>

  <ellipse cx="200" cy="190" rx="85" ry="55" style="fill:url(#orange_red)"/>
</svg>
```

XML Parser

- Used by application trying to **understand** the XML document
  - Splits document in elements, attributes, and other parts
  - Hands over parts to the application
  - Always check if document is well-formed
- Possibility for **validating parser**
  - Documents contains URL to schema definition
  - On **validity error**, application decides over continuation
  - Validation is optional, and usually demands a Internet connection



XML Documents

- **Well-formed XML document**
  - Follows the rules of the XML syntax
  - Does not have an according DTD or schema definition
- **Valid XML document**
  - Follows the rules defined by XML syntax and its DTD / Schema definition
- **Invalid document**
  - Doesn‘t follow the XML syntax or its DTD / schema definition
  - Rejected by parser

Document Type Definition (DTD)

- Applications have restrictions on XML document format - **vocabulary**
  - Allowed elements and their attributes
  - Content model - elements and data allowed inside element content
  - Required attributes, default values, list of valid values for attributes
- Application using the parser decides what to do with invalid document parts
- *Document type declaration* - specification of the DTD for a document
- *Document type definition* - the DTD file containing the specification
- Everything that is not allowed is forbidden
- Based for mutual understanding of a class of XML documents

Internal / External DTD Example

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</
body>
</note>
```

```
<!DOCTYPE note SYSTEM "note.dtd">
```

- Typically external DTD, referred my multiple documents
- Parsed Character Data (PCDATA)
  - Pure text without tags and elements
- Document type declaration before root element
- Example: *www.bth.se*
- No exhaustive typing - intended for narrative documents

DTD Syntax Example

```
<!-- address.dtd -->
<!ELEMENT addressbook (address+)>
<!ELEMENT privbook (address*)>
<!ELEMENT address (name, street, city, state, postal-code)>
<!ELEMENT name (title?, first-name, (middle-initial | middle-name)?, last-name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT postal-code (#PCDATA)>
```

- `<addressbook>` contains one or more addresses, `<privbook>` zero or more
- `<address>` element contains other elements in specified order (comma)
- `<name>` element contains optional `<title>` element (question mark), followed by first name and potentially a middle initial or middle name, and the last name

DTD Examples

- DocBook-XML DTD
  - 11.000 lines
  - Definition of XML format for text processing
- Scalable Vector Graphic (SVG) DTD
  - 1.000 lines
  - Definition of XML format for vector graphics
- Most exhaustive source of DTD definitions is the W3C itself
  - XHTML, HTML, MathML, SVG, PICS, RDF, SPARQL, SMIL, XML Schema

DTD Example: Dublin Core

- Original XML hype around the automated ‚understanding' of documents
  - Dublin Core provides schema for annotation of documents
  - 15 standardized information items: title, creator, subject, ...

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF PUBLIC "-//DUBLIN CORE//DCMES DTD 2002/07/31//EN"
                         "dcmes-xml-dtd.dtd">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
         rdf:about="http://www.ilrt.bristol.ac.uk/people/cmdjb/">
    <dc:title>Dave Beckett's Home Page</dc:title>
    <dc:creator>Dave Beckett</dc:creator>
    <dc:publisher>ILRT, University of Bristol</dc:publisher>
    <dc:date>2002-07-31</dc:date>
  </rdf:Description>
</rdf:RDF>
```

Namespaces

- Documents might combine markup from different XML applications
  - XHTML document with SVG pictures and MathML formula
- Two major goals for **XML namespaces** (W3C spec.)
  - Distinguish elements with same names from different vocabularies
  - Summarize all elements and attributes of one XML application
- Every element and attribute gets a **prefix**
  - Every prefix is mapped to an URI („xml“ is a reserved prefix)
  - Elements and attributes without prefix have a default URI
  - Elements and attributes with the same URI are in the same namespace
- Separate W3C specification, created one year after XML 1.0

XML Namespaces

```
<html xmlns="http://www.w3.org/HTML/1998/html4"
      xmlns:xdc="http://www.xml.com/books">
 <head><title>Book Review</title></head>
 <body>
  <xdc:bookreview>
   <xdc:title>XML: A Primer</xdc:title>
   <table>
    <tr align="center">
     <td>Author</td><td>Price</td>
     <td>Pages</td><td>Date</td></tr>
    <tr align="left">
     <td><xdc:author>Simon St. Laurent</xdc:author></td>
     <td><xdc:price>31.98</xdc:price></td>
     <td><xdc:pages>352</xdc:pages></td>
     <td><xdc:date>1998/01</xdc:date></td>
    </tr>
   </table>
  </xdc:bookreview>
 </body>
</html>
```

XML Namespaces Rules

- Namespace binding is valid for element itself and it‘s element content
- Default namespace
  - Applies when element / attribute has no namespace prefix
- URL‘s as namespace were chosen to re-use concept of domain names
  - No need for valid content, but typical source of confusion
  - This is ok: `<html xmlns:pt=“mailto:peter@troeger.eu“>`
- Standardized syntax: `[prefix]:[local part]`
  - Complete construct forms a **qualified name (QName)**
- In practice, people tend to overlook the shortcut nature of the prefix

Namespace Facts

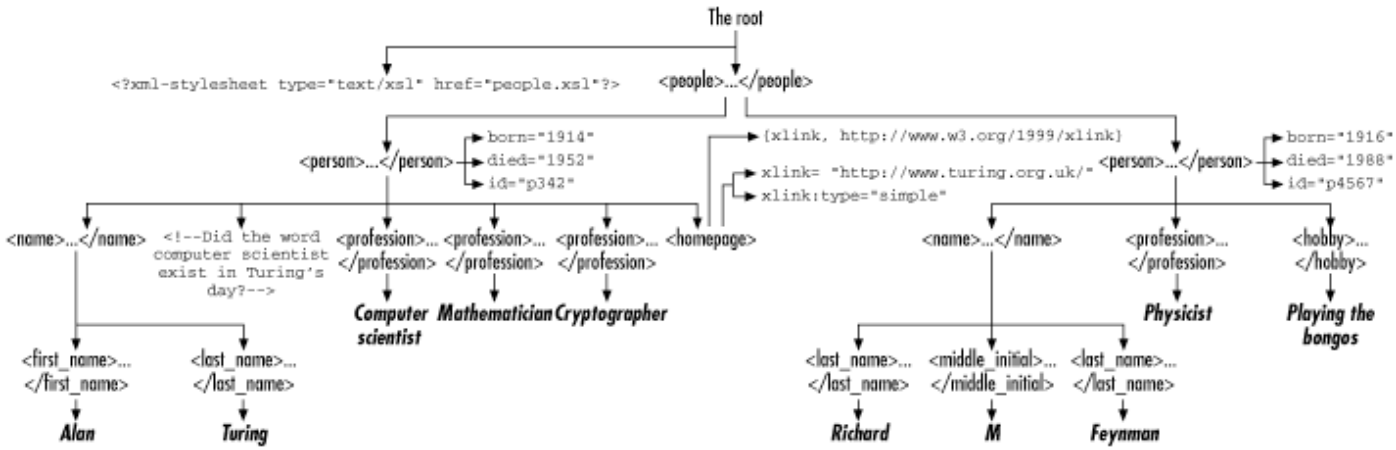
- Separation of XML specification and XML Namespaces specification
  - Non-namespace parser can still handle the documents
- Aware parser performs additional checks
  - Every prefix must be mapped to a URI
  - Configurable feature for most parsers
- DTDs and namespaces are independent
  - DTD definitions must exactly reflect chosen prefixes
  - Some tricks to make prefixes in DTDs still changeable (check course book)

Outline

- XML Basics
  - Markup concepts, schema definition, DTD
  - Namespaces
- **Document-centric XML**
  - XSL Transformations, XPath
  - XML Rendering (CSS, XSL-FO)
- Data-centric XML
  - XML Schema, XQuery
  - Programming models

XPath

- Non-XML language, used for identifying parts of XML documents
- Originally developed for XSLT
- Basis for other XML standards - XSLT, XForm, DOM, XML Schema, ...
- 7 types of nodes in a XML document
  - root nodes, element nodes, attribute nodes, text nodes, comment nodes, processing instruction nodes, name space nodes



(C) Harold et. al

XPath Location Path

- **Location path** for a set of nodes in a document
  - Build from sequential **location steps**
  - Every location step is processed relative to a **context node**
- Example: „/“ location step selects the root node of a document
- Example: Single element name as location step selects all child nodes
- Location steps can also address attributes („@attr“) or other node types
- Different wildcards for matching multiple nodes
- Location steps in a location path are separated by „/“
  - Either absolute or relative location path
- „//“ for ancestor, „..“ for parent element, „.“ for context node

XPath Examples

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year><price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year><price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">XQuery</title>
    <author>James McGovern</author>
    <year>2003</year><price>49.99</price>
  </book>
</bookstore>
```

- Select all nodes: *”/bookstore/book”*
- Select the first book node: *„/bookstore/book[0]”*
- Select the prices: *„/bookstore/book/price/text()”*
- Select expensive books: *”/bookstore/book[price>35]/price”*

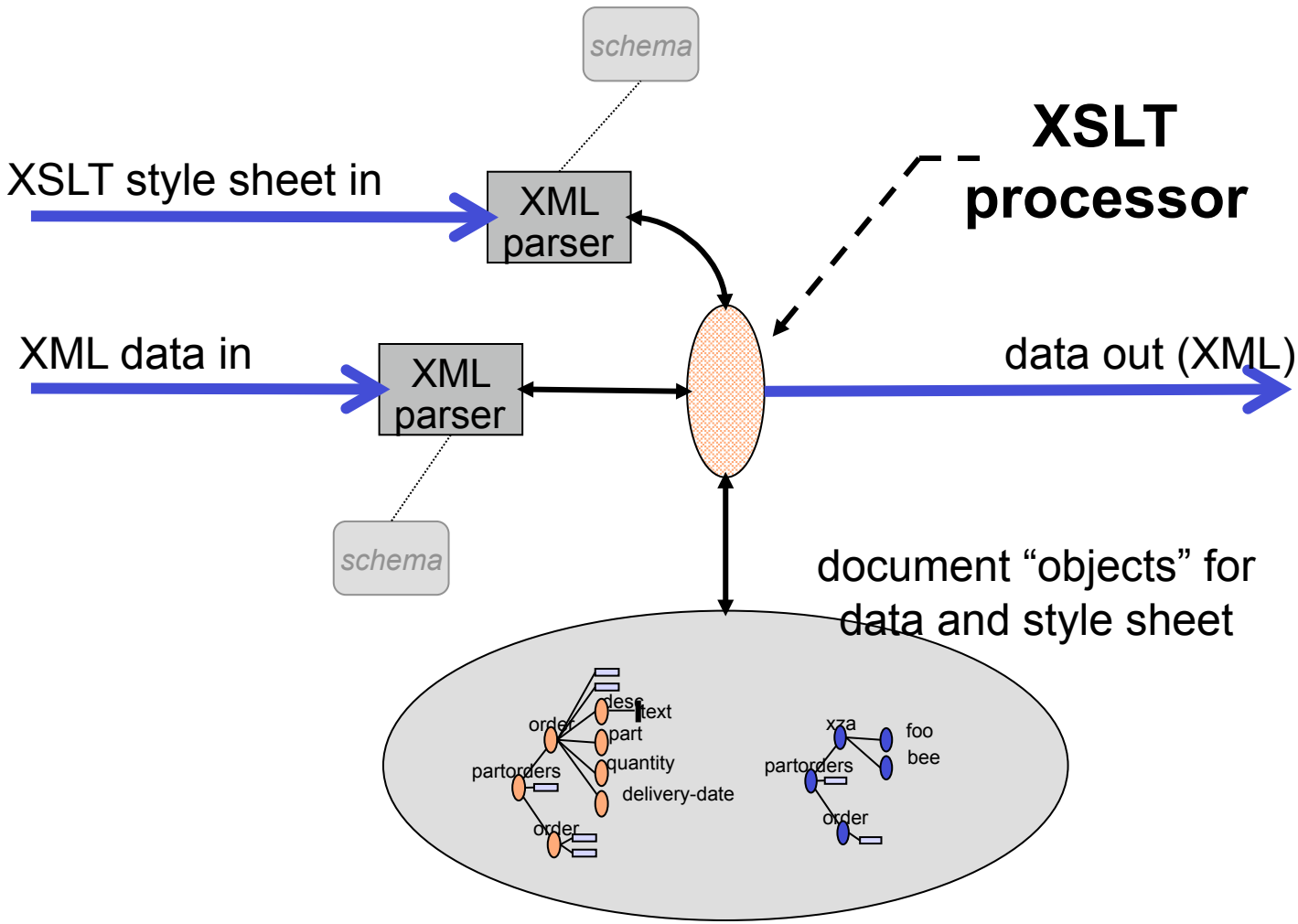
More XPath Features

- Number functions
- Boolean functions
- String functions
- Functions for handling of node sets
- See Wikipedia article for good explanation ...

XSL Transformations (XSLT)

- eXtensible Stylesheet Language (XSL)
  - HTML tags are predefined, so rendering is clear - not with XML !
  - XML-based formulation of so-called *style sheets*
  - Contains *XSL Transformations (XSLT)* and *XSL Formatting Objects (XSL-FO)*
- XSLT is a XML application
  - Define rules to transform *source tree* to *result tree*
  - An XSLT document is a *style sheet,* containing a *transformation template*
    - Contains *template rules*
    - Navigation in the source tree by *XPath*
- XSLT processor in browser (IE6), application server (Cocoon) or library (Xalan)

XSLT Concept



(C) Ian Graham

XSLT Concepts

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=...>
</xsl:stylesheet>
```

- Style sheet is a set of `<xsl:template>` elements
  - Rules tell the XSLT processor what to do with markup of the source tree
    - Empty style sheet just copies the content
  - `match` attribute - determines pattern for matching input
- Style sheet must output well-formed markup
  - Otherwise, style sheet itself is not well-formed

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="person">
      <p>Eine Person</p>
   </xsl:template>
</xsl:stylesheet>
```

XSLT Concepts

- `<xsl:value-of>`
  - Extract value of XML element and add it to the output stream
    - Element value: Text without tags and with resolved entity references
- `<xsl:for-each>`
  - Select every element of a specified node set
  - Support for filtering and sorting
- Support for different kinds of conditions (if, choose, ...)
- `<xsl:apply-templates>`
  - Define processing order for source tree (based on XPath)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
   <html><body>
    <h2>My CD Collection</h2>
    <table border="1">
     <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
     </tr>
     <xsl:for-each select="catalog/cd">
      <tr>
       <td><xsl:value-of select="title"/></td>
       <td><xsl:value-of select="artist"/></td>
      </tr>
     </xsl:for-each>
    </table>
   </body></html>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cat.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
```

Integrated Template Rules

- 7 types of nodes in a XML document
  - root nodes, element nodes, attribute nodes, text nodes, comment nodes, processing instruction nodes, name space nodes
- Default XSLT rule for every node type
  - Text and attribute nodes are copied
  - For element and root nodes, template rules are applied to all child nodes
  - Comment and processing instruction nodes are ignored
  - Name space nodes are ignored, but all necessary name space declaration are written to the output

XSLT Output Configuration

- `<xsl:output>`
  - *method*="xml" | "html" | "text" | qname
  - Default method is *html* if root node has an element child, the local name of first such element is "html" (any case), and the namespace is null
  - Otherwise, default method is *xml*
- *version* = <version of the output method>
  - XML: default is 1.0, HTML: default is 4.0
- *encoding*=<name of output encoding>
- *media-type*=<MIME type> (text output: default is text/plain)
- ...

More concepts in XSLT

- XSLT is a turing-complete (programming) language
- Numbering and sorting of output elements
- Conditional processing
- Iterations
- Extension elements and functions
- Import of other style sheets
- Utilization of complex XPath expressions

Cascading Stylesheets (CSS)

- Non-XML language for describing presentation of XML documents
- CSS is defined in different levels
  - Level 1 - only used for HTML formatting
  - Level 2 - XML and HTML can be formatted in a similar manner
- Trivial syntax
  - Every line denotes an element and the styles to be applied
  - Over 100 style properties, most with useful default values
- Included in HTML / XML page as processing instruction
- Some simple selectors: * (all non-covered elements), attributes, ID‘s, ...
- Exhaustive output formatting by XSL Formatting Objects (XSL-FO)

CSS Example

```
<?xml-stylesheet href="style.css" type="text/css"?>
<card>
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.com</email>
  <phone>(202) 555-1414</phone>
  <logo url="widget.gif"/>
</card>
```

```
card  { background-color: #cccccc; border: none;}
name  { display: block; font-size: 20pt; margin-left: 0;}
title { display: block; margin-left: 20pt;}
email { display: block; margin-left: 20pt;}
phone { display: block; margin-left: 20pt;}
```

Outline

- XML Basics
  - Markup concepts, schema definition, DTD
  - Namespaces
- Document-centric XML
  - XSL Transformations, XPath
  - XML Rendering (CSS, XSL-FO)
- **Data-centric XML**
  - XML Schema, XQuery
  - Programming models

XML Schema

- Alternative to DTD definition - description of document structure and data
  - XML schema definition („XSD“) has an XML syntax
    - Can be processed as any other XML
    - Example: XSLT for transformation of XML schema to web form
  - Improved data type support
    - Integers, floats, datetime, decimal, string, URL, sets, ...
    - Simple and complex data types, inheritance
  - More expressive power
    - String length, regular expressions, support for XML namespaces
- XML described by XML schema definition is called **instance document**

XML Schema Example - Namespace Support

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.troeger.eu">
   ...
</xs:schema>
```

```
<bookstore
    type="science fiction"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.troeger.eu bookstore.xsd" >
  <book>
    <author>...</author>
    <title>...</title>
    <publisher>...</publisher>
    <isbn>...</isbn>
    <price>...</price>
  </book>
  <book>
    ...
  </book>
</bookstore>
```

XML Schema Example - Documentation

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:annotation>
     <xs:documentation xml:lang="en">...</xs:documentation>
   </xs:annotation>
   <xs:element name="bookstore">
     <xs:complexType>
     ...
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<bookstore type="science fiction">
  <book>
    <author>...</author>
    <title>...</title>
    <publisher>...</publisher>
    <isbn>...</isbn>
    <price>...</price>
  </book>
  <book>
    ...
  </book>
</bookstore>
```

- Child elements of `<xs:schema>` with the type `<xs:element>` determine possible root nodes
- Possibility for in-build documentation by `<xs:annotation>` element

XML Schema Example - Matryoshka Design

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="bookstore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          ...
        </xs:element>
      </xs:sequence>
      <xs:attribute name="type" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<bookstore type="science fiction">
  <book>
    <author>...</author>
    <title>...</title>
    <publisher>...</publisher>
    <isbn>...</isbn>
    <price>...</price>
  </book>
  <book>
    ...
  </book>
</bookstore>
```

- One `<bookstore>` element contains multiple books and an type attribute

XML Schema Example - Matryoshka Design

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="bookstore">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="book" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="author" type="xs:string"/>
       <xs:element name="title" type="xs:string"/>
       <xs:element name="publisher" type="xs:string"/>
       <xs:element name="isbn" type="xs:string"/>
       <xs:element name="price" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  <xs:attribute name="type"
      type="xs:string"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

```
<bookstore type="science fiction">
  <book>
    <author>...</author>
    <title>...</title>
    <publisher>...</publisher>
    <isbn>...</isbn>
    <price>...</price>
  </book>
  <book>... </book>
</bookstore>
```

XML Schema - Element Definition

- **Simple element**, contains no more elements or attributes
  - `<xs:element name="xxx" type="yyy"/>`
  - Content can have type: `anyURI, base64Binary, boolean, byte, integer, language,dateTime, duration, string, ...`
  - May have default value or fixed value
- **Complex element** can have attributes and child elements
  - Complex type definition directly bound to element definition
    - `<xs:element name="xxx"><xs:complexType>...`
  - Complex type definition referred by type attribute in element definition
    - `<xs:element name="xxx" type="ctype"/>...`
      `<xs:complexType name="ctype">`

XML Schema - Attributes

- Attribute is part of a complex type
  - `<xs:attribute name="xxx" type="yyy"/>`
  - May have default value or fixed value
  - By default optional, can be mandatory
  - `<xs:attribute name="lang" type="xs:string" use="required"/>`

XML Schema - Restrictions

- Define acceptable values for element content or attributes

```
<xs:element name="age">
 <xs:simpleType>
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="0"/>
   <xs:maxInclusive value="120"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

```
<xs:element name="car" type="carType"/>
 <xs:simpleType name="carType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Golf"/>
   <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="letter">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:minLength value="5"/>
   <xs:maxLength value="8"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema - Complex Type Indicators

- Order indicators: `all, choice, sequence`
  - `<all>`: Child elements can occur in any order and only once
  - `<choice>`: Either one or another child element can occur
  - `<sequence>`: Child elements must occur in specific order
- Occurrence indicators: `maxOccurs, minOccurs`
  - Default value 1 for all other indicators

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema - Extensibility

- `<any>` element specifies content not further declared by this schema
  - Allows placeholder semantic
  - Additional markup is then declared by another schema file

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type=
   <xs:element name="lastname" type="
   <xs:any minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<xs:element name="children">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="childname" type="xs:string"
   maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

XML Schema - Extensibility

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns="http://www.troeger.eu"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:SchemaLocation="http://www.troeger.eu family.xsd
                             http://www.bth.se children.xsd">
  <person>
     <firstname>Hege</firstname>
     <lastname>Refsnes</lastname>
     <children>
        <childname>Cecilie</childname>
     </children>
  </person>
  <person>
     <firstname>Stale</firstname>
     <lastname>Refsnes</lastname>
  </person>
</persons>
```

XML Schema - There is a lot more ...

- Complex types with mixed content
- Data type details
- Element substitution
- Schema validation
  - Input is document, document schema, and schema of XML schema
  - Resulting documents are called „schema-valid“
  - XSD validator tools online, use tools for editing
- XSD-based XML applications: WSDL, XML Digital Signatures, OWL, ...

Other Schema Languages

Everybody who actually touches the technology has known the truth for years, and it's time to stop sweeping it under the rug. W3C XML Schemas (XSD) suck. They are hard to read, hard to write, hard to understand, have interoperability problems, and are unable to describe lots of things you want to do all the time in XML.

...

It's a pity; when XSD came out people thought that since it came from the W3C, same as XML, it must be the way to go, and it got baked into a bunch of other technology before anyone really had a chance to think it over. So now lots of people say "Well, yeah, it sucks, but we're stuck with it." Wrong! The time has come to declare it a worthy but failed experiment, tear down the shaky towers with XSD in their foundation, and start using RELAX NG for all significant XML work.

Tim Bray, 2006/11/28

Relax NG

```
<addressBook>
  <card>
    <name>John Smith</name>
    <email>js@example.com</email>
  </card>
  <card>
    <name>Fred Bloggs</name>
    <email>fb@example.net</email>
  </card>
</addressBook>
```

```
<element name="addressBook"
         xmlns="http://relaxng.org/ns/structure/1.0">
   <zeroOrMore>
      <element name="card">
        <element name="name"><text/></element>
        <element name="email"><text/></element>
        <optional>
           <element name="note"><text/></element>
        </optional>
      </element>
   </zeroOrMore>
</element>
```

XQuery

- Designed as SQL counterpart - a query language for XML documents
  - Relies on XPath, similar concept of node types
  - Supported by all major database engines
  - W3C recommendation since 2007
- Select nodes from a XML file
  - **Functions** for data extraction
  - **Path expressions** for navigating through the document
  - **Predicates** for limiting the returned result set
- All XQuery constructs are expressions

XQuery Expressions

- Path expressions
  - `doc("books.xml")/bookstore/book[price<30]`
- FLOWR („For, Let, Where, Order by, Return“) expressions
  - ```
    for $x in doc("books.xml")/bookstore/book
    where $x/price>30
    order by $x/title
    return $x/title
    ```

```
<ul>
  {
    for $x in doc("books.xml")/bookstore/book/
    title
    order by $x
    return <li>{data($x)}</li>
  }
</ul>
```
```

XQuery - Complex Example

```
declare variable $firstName  as
xs:string external;
<videos featuring="{$firstName}">
{
   let $doc := .
   for $v in $doc//video,
      $a in $doc//actors/actor
   where ends-with($a, $firstName)
      and $v/actorRef = $a/@id
   order by $v/year
   return
      <video year="{$v/year}">
         {$v/title}
      </video>
}
</videos>
```

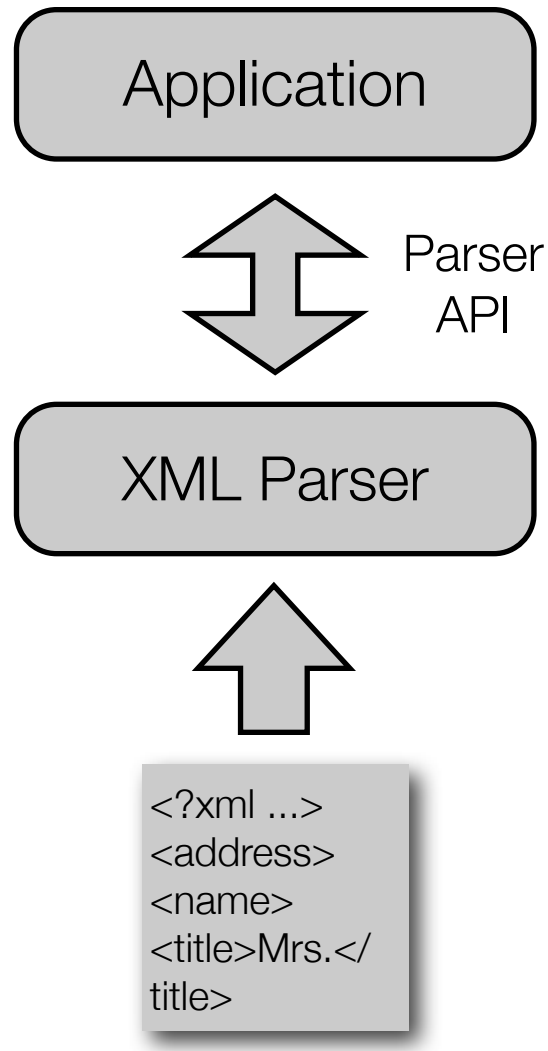
See http://www.stylusstudio.com/xquery_primer.html

XQuery Features

- Conditionals - Formulate if-then-else expressions
- Comparisons - General comparisons (=, !=, <, <=, ...) and value comparisons
- Adding new elements and new attributes to output
- Over 100 build-in functions
  - String values, numeric values, date / time handling, QName manipulation, sequence manipulation
  - Support for own function definitions (again in XQuery vocabulary)
- Originally intended as query language for XML data sets, meanwhile also possible replacement for XSLT
  - XQuery is better in data handling, while XSLT is better for narrative docs

Programming Models

- XML parser is used by application trying to **understand** the XML document
  - Splits document in elements, attributes, and other parts
  - Hands over parts to the application
  - Checks if document is well-formed / valid
- Application demands API to the parser
  - Reflects the programming model for XML processing on the application side
  - Must be standardized in order to get portable applications / exchangeable parsers



XML Processing Models

- XML parser moves from document start to document end
- **Event-oriented XML processing**
  - Parser notifies the application of processing events
    - Reading of start tag, content, end tag, processing instructions, ...
  - Advantage of efficiency and speed
    - Application can process incoming data while it is read by parser
  - Application needs to implement a state machine
  - API‘s in practice: *Simple API for XML (SAX)* and *Expat*

XML Processing Models

- **Tree-based XML processing**
  - XML documents are naturally described as tree structures (e.g. XPath)
  - After complete parsing, application can access tree representation of the XML document
    - Application don‘t have to deal with incomplete content
    - Easy changes to the in-memory tree structure
    - Full-fledged navigation capabilities (e.g. XPath)
    - Possible high memory footprint
  - API‘s in practice: Document Object Model (DOM), XOM

XML Processing Models

- **Pull-based XML processing**
  - Similarities to event-based processing
  - Application demands content from the parser
    - Easier application code
    - No need for complicated parse state consideration
    - Still speed and efficiency advantage as with event-based approach
  - API‘s in practice: Streaming API for XML (StAX), .NET XMLReader class
- Combined approaches
  - Generation of complete trees out of parsing events

Document Object Model (DOM)

- Set of W3C recommendations
  - Representation of hierarchical documents in memory
  - Original specification is independent from programming language (IDL)
  - Supports reading and manipulation of XML / HTML / CSS
- Set of abstract interfaces, realized by implementations
  - Interfaces are separated in modules, applications can support only parts
    - Example: XML parser don‘t need to implement HTML-specific parts
- Different versions defined as **levels**
  - Other XML specifications extended DOM for their specific needs (e.g. SVG)

DOM Kernel Concepts

- Starting point is the generic *Node* interface
  - Support for localisation of parent nodes, child nodes, and ancestor nodes
  - Traverse document tree without knowledge of node types
  - Different node attributes: *nodeType, nodeValue, namespaceURI, ...*
- Specific interfaces for different node types, all derived from *Node*
  - *Comment, Document, Element, Entity, ProcessingInstruction, Text, ...*
  - *Node* object is never created directly by the parser or the application
- Several interfaces to ease up work with DOM
  - *NodeList* - ordered access to content of a node
  - *NamedNodeMap* - Accessing unordered set of elements (e.g. attributes)

```
public class ElementPrinting {
  public static void main(String args[]) {
    try
    {
      DOMParser parser = new DOMParser();
      parser.parse(args[0]);
      DocumentImpl document = (DocumentImpl)parser.getDocument();
      Node root = document.getLastChild();
      NodeIteratorImpl iterator =
        (NodeIteratorImpl)document.createNodeIterator(root,
            NodeFilter.SHOW_ELEMENT, (NodeFilter)allelements, true);
      printElements(iterator);
    }
    catch (Exception e) {
      e.printStackTrace();
  }}

  public static void printElements(NodeIteratorImpl iter) {
    Node n;
    while ((n = iter.nextNode()) != null) {
      System.out.println(n.getNodeName());
}}}

class AllElements implements NodeFilter {
  public short acceptNode (Node n) {
    if (n.getNodeType() == Node.ELEMENT_NODE)
      return FILTER_ACCEPT;
    return FILTER_SKIP;
  }}
```

```
public class XmlCreateAndPrint {
    public XmlCreateAndPrint() {
        try {
            DocumentBuilderFactory dbfac=DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = dbfac.newDocumentBuilder();
            Document doc = docBuilder.newDocument();

            //create the root element and add it to the document
            Element root = doc.createElement("root");
            doc.appendChild(root);

            //create child element, add an attribute, and add to root
            Element child = doc.createElement("child");
            child.setAttribute("name", "value");
            root.appendChild(child);

            //add a text element to the child
            Text text = doc.createTextNode("Hello World");
            child.appendChild(text);

            //Output the in-memory XML tree by transformation to tree
            TransformerFactory transfac = TransformerFactory.newInstance();
            Transformer trans = transfac.newTransformer();
            StringWriter sw = new StringWriter();
            StreamResult result = new StreamResult(sw);
            DOMSource source = new DOMSource(doc);
            trans.transform(source, result);
            String xmlString = sw.toString();
            System.out.println("Here's the xml:\n\n" + xmlString);

        } catch (Exception e) {System.out.println(e);}
}}
```

DOM Example (Python)

```
import sys
from xml.dom import minidom, Node

def showNode(node):
    if node.nodeType == Node.ELEMENT_NODE:
        print 'Element name: %s' % node.nodeName
        for (name, value) in node.attributes.items():
            print '    Attr -- Name: %s  Value: %s' % (name, value)
        if node.attributes.get('ID') is not None:
            print '    ID: %s' % node.attributes.get('ID').value

def main():
    doc = minidom.parse(sys.argv[1])
    node = doc.documentElement
    showNode(node)
    for child in node.childNodes:
        showNode(child)

if __name__ == '__main__':
    main()
```

Simple API for XML (SAX)

- Originally developed as Java interface, but ported to other OO languages (C++, Python, Perl, Eiffel)
  - `org.xml.sax` package
- Since 2004 SAX2 available (namespace support)
- `XMLReader` interface abstracts access to the parser
  - Properties and features to configure all aspects of the parser
- `ContentHandler` interface is implemented by the application
  - `XMLReader` is configured with an instance of the implementation
  - Functions are called by the parser when the event occurs
- Support for filter implementations

ContentHandler interface

```
package org.xml.sax;
public interface ContentHandler
{
    public void setDocumentLocator (Locator locator);
    public void startDocument () throws SAXException;
    public void endDocument() throws SAXException;
    public void startPrefixMapping (String prefix, String uri)
       throws SAXException;
    public void endPrefixMapping (String prefix) throws SAXException;
    public void startElement (String uri, String localName,
       String qName, Attributes atts)throws SAXException;
    public void endElement (String uri, String localName,String qName)
       throws SAXException;
    public void characters (char ch[], int start, int length)
       throws SAXException;
    public void ignorableWhitespace (char ch[], int start, int length)
       throws SAXException;
    public void processingInstruction (String target, String data)
       throws SAXException;
    public void skippedEntity (String name) throws SAXException;
}
```

Registering the Content Handler

```
XMLReader parser =
org.xml.sax.helpers.XMLReaderFactory.createXMLReader();

// Create a new instance and register it with the parser
ContentHandler contentHandler = new MyHandlerImplementation();
parser.setContentHandler(contentHandler);
```

Python Example

```
from xml.sax import make_parser
from xml.sax.handler import ContentHandler

class DocHandler(ContentHandler):
    def startDocument(self):
        self.Storage = ""
    def endDocument(self):
        # Print approximate number of words
        print len(string.split(self.Storage))
    def characters(self,content):
        self.Storage = self.Storage + content

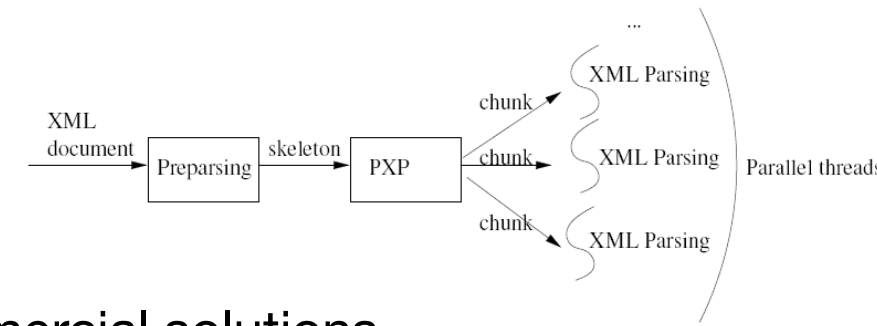
parser = make_parser()
parser.setContentHandler(DocHandler())
parser.parse(open("foo.xml"))
```

There is more ...

- XML information set
- Unicode in XML
  - `xml:lang` definition according to ISO-639
  - Sub-codes according to ISO-3166
- XPointer, XLinks, XInclude
- All the XML applications
- Fancy parser interfaces and schema languages
- ...

Parallelization ?

- Different technologies provide different parallelization possibilities
  - Parallelized XML parsing
    - 6 papers - most obvious (and most interesting ?) approach
    - PXP project (University of Indiana)
  - Parallelized XPath evaluation - 2 papers
    - No standalone solution
  - Parallelized XSLT processor
    - 1 paper, but unknown number of commercial solutions
  - Parallelized XQuery evaluation - 5 papers, pure database topic
  - Parallelized style sheet - based rendering ???



Other Unsolved Questions

- Degree of multi-threading in common open source XML processors
  - Apache Xerces, Gnome libxml2, Expat, Sun Java XML support (XML parser, DOM, SAX, XML Schema)
  - Apache Xindice (XML database)
  - Apache Xalan, Sablotron (XSLT processor, XPath implementation)
  - Apache Batik (SVG toolkit)
- Getting performance measurements
  - Sun‘s XMLTest, XMLBench project