XML and Databases

Lecture 1
Introduction to XML

Sebastian Maneth
NICTA and UNSW

This course will introduce you to the world of XML and to the challenges of dealing with XML in a RDMS.

Some of these challenges are:
Existing (DB) technology cannot be applied to XML data.

Similar to HTML (Berners-Lee, CERN → W3C) use your own tags.
Amount/popularity of XML data is growing steadily (faster than computing power)

HTML pages are "tiny" (couple of Kbytes)
XML documents can be huge (GBytes)

- Databases
- Relations
- Storage Management
- Index Structures
- Join/Sort Algorithms
- ...

XML and Databases

This course will introduce you to the world of XML and to the challenges of dealing with XML in a RDMS.

Some of these challenges are:
Existing (DB) technology cannot be applied to XML data.
This course will introduce you to the world of XML and to the challenges of dealing with XML in a RDMS.

Some of these challenges are:

- Existing (DB) technology cannot be applied to XML data.
- How do we store trees?
- Can we benefit from index structures?
- How to implement tree navigation?
- Additional challenges posed by W3C's XQuery proposal.
  - A notion of order
  - A complex type/schema system
  - Possibility to construct new tree nodes on the fly.

You will learn about:

- Tree structured data (XML)
- XML parsers & efficient memory representation
- Query languages for XML (XPath, XQuery, XSLT...)
- Efficient evaluation using finite-state automata
- Mapping XML to databases
- Advanced topics (query optimizations, access control, update languages...)

XML = Threat to Databases!!
XML and Databases

This course will introduce you to the world of XML and to the challenges of dealing with XML in a RDBMS.

You will learn about:
- Tree structured data (XML)
- XML parsers & efficient memory representation
- Query Languages for XML (XPath, XQuery, XSLT, ...)
- Efficient evaluation using finite-state automata
- Mapping XML to databases
- Advanced Topics (query optimizations, access control, update languages, ...)

You will NOT learn about:
- Hacking CGI scripts
- HTML
- JavaScript

You NEED to program in:
- Java or
- C++

About XML

We will talk about algorithms and programming techniques to efficiently manipulate XML data:
- Regular expressions can be used to validate XML data
- Finite state automata lie at the heart of highly efficient XPath implementations
- Tree traversals may be used to preprocess XML trees in order to support XPath evaluation, to store XML trees in databases, etc.

In the end, you should be able to digest the thick pile of related W3C standards.

(Such as, XQuery, XPointer, XLink, XHTML, XInclude, XML Base, XML Schema, ...

Course Organization

Lecture: Tuesday, 15:00 — 18:00
ChemicalSc M17 (ex AppliedSc) (K-F10-M17)
Lecturer: Sebastian Maneth
Consult: Friday, 11:00-12:00 (E508, L5)
All email to cs4317@cse.unsw.edu.au

Tutorial: Tuesday, 12:00-14:00 @ Quadrangle G040 (K-E15-G040) — before lecture
Wednesday, 12:00-14:00 @ Quadrangle G022 (K-E15-G022)
Wednesday, 14:00-16:00 @ Quadrangle G022 (K-E15-G022)
Thursday, 14:00-16:00 @ Quadrangle G040 (K-E15-G040)
Thursday, 16:00-18:00 @ Quadrangle G022 (K-E15-G022)

Tutors: ?, Kim Nguyen
All email to cs4317@cse.unsw.edu.au

Programming Assignments
5 assignments, due every other Monday. (1st is due 15th March 2nd is due 29th March...)
Per assignment: 10 points (total: 60 points) (+2 bonus points)
Final Exam:
Final exam: 40 points (must get 16/40 to pass, that is 40%)

Suggested Reading Material:
- Course slides of Marc Scholl, Uni Konstanz
  http://www-inf uni-konstanz de/dbis teaching ws0506 database xml/XMLDB.pdf
- Theory/PL oriented, book draft:
  http://lairbre is u-tokyo ac.jp/~hahosoya/xmlbook/

```
Book: None!
```

Course slides of Marc Scholl, Uni Konstanz
http://www-inf.uni-konstanz.de/dbis/teaching/ws0506/database-xml/XMLDB.pdf

Theory / PL oriented, book draft:
http://lairbre.is.u-tokyo.ac.jp/~hahosoya/xmlbook/

About XML

- XML is the World Wide Web Consortium’s (W3C, http://www.w3.org/) Extensible Markup Language
- We hope to convince you that XML is not yet another hyped TLA, but is useful technology.
- You will become best friends with one of the most important data structures in Computing Science, the tree.
- XML is all about tree-shaped data.
- You will learn to apply a number of closely related XML standards:
  - Representing data: XML itself, DTD, XML Schema, XML dialects
  - Interfaces to connect PLs to XML: DOM, SAX
  - Languages to query/transform XML: XPath, XQuery, XSLT.
Outline - Assignments

You can freely choose to program your assignments in

→ C / C++, or
→ Java

However, your code must compile with gcc / g++, javac, as installed on CSE linux systems!

Submit code (using give) by Monday 23:59 (every other week)

Assignment 4 (harder) gets four weeks / counts double (20 Points) due date 17th May

Outline - Lectures

1. Introduction to XML, Encodings, Parsers
2. Memory Representations for XML: Space vs Access Speed
3. RDBMS Representation of XML
4. DTDs, Schemas, Regular Expressions, Ambiguity
5. Node Selecting Queries: XPath
6. Efficient XPath Evaluation
7. XPath Properties: backward axes, containment test
8. Streaming Evaluation: how much memory do you need?
9. XPath Evaluation using RDBMS
10. Properties of XPath
11. XSLT
12. XQuery
13. Wrap up, Exam Preparation, Open questions, etc

Outline

1. Three motivations for XML
   (1) religious
   (2) practical
   (3) theoretical / mathematical
2. Well-formed XML
3. Character Encodings
4. Parsers for XML
   → parsing into DOM (Document Object Model)

Outline - Assignments

1. Read XML, using DOM parser. Create document statistics
   13 days
2. SAX Parse into memory structure: Tree vs DAG
   2 weeks
3. Map XML into RDBMS
   2 weeks
4. XPath evaluation over main memory structures (+ streaming support)
   4 weeks
5. XPath into SQL Translation
   2 weeks

Outline - Lectures

1. XML Introduction

Lecture 1

XML Introduction

Religious motivation for XML:

to have one language to speak about data.
XML Motivation (historical)

→ XML is a Data Exchange Format

1974  SGML  (Charles Goldfarb at IBM Research)
1989  HTML  (Tim Berners-Lee at CERN/Geneva)
1994  Berners-Lee founds Web Consortium (W3C)
1996  XML (W3C draft, v1.0 in 1998)

http://www.w3.org/TR/REC-xml/

(2) Practical  XML = data + structure

Tom Henzinger
EPFL
Tom.Henzinger@epfl.ch

→

Helmut Seidl
TU Munich
seidl@inf.tum.de

Text file

XMLElement

"mark
it
up"

Is this a good "template"?? What about last/first name? Several affil's/email's...?
XML Documents

→ Ordinary text files (UTF-8, UTF-16, UCS-4 ...)
→ Originates from typesetting/DocProcessing community
→ Idea of labeled brackets ("mark up") for structure is not new! (already used by Chomsky in the 1960's)
→ Brackets describe a tree structure
→ Allows applications from different vendors to exchange data!
→ standardized, extremely widely accepted!

Ordinary text files (UTF-8, UTF-16, UCS-4 ...)
→ Originates from typesetting/DocProcessing community
→ Idea of labeled brackets ("mark up") for structure is not new! (already used by Chomsky in the 1960's)
→ Brackets describe a tree structure
→ Allows applications from different vendors to exchange data!
→ standardized, extremely widely accepted!

Ordinary text files (UTF-8, UTF-16, UCS-4 ...)
→ Originates from typesetting/DocProcessing community
→ Idea of labeled brackets ("mark up") for structure is not new! (already used by Chomsky in the 1960's)
→ Brackets describe a tree structure
→ Allows applications from different vendors to exchange data!
→ standardized, extremely widely accepted!

→ Ordinary text files (UTF-8, UTF-16, UCS-4 ...)
→ Originates from typesetting/DocProcessing community
→ Idea of labeled brackets ("mark up") for structure is not new! (already used by Chomsky in the 1960's)
→ Brackets describe a tree structure
→ Allows applications from different vendors to exchange data!
→ standardized, extremely widely accepted!

Problem: highly verbose, lots of repetitive markup, large files

Social Implications!
All sciences (biology, geography, meteorology, astrology...) have own XML "dialects" to store their data optimally.

You do this, using an XML Type definition language such as DTD or Relax NG (Oasis).

Of course, such type definition languages are SIMPLE, because you want the parsers to be efficient!

They are similar to EBNF. ➔ context-free grammar with reg. exprs in the right-hand sides.

Example DTD (Document Type Description)

```
Related ➔ (colleague | friend | family)*
colleague ➔ (name,affil*,email*)
friend ➔ (name,affil*,email*)
family ➔ (name,affil*,email*)
name ➔ (#PCDATA)
```

Element names and their content

Contra... highly verbose, lots of repetitive markup, large files
Pro... we have a standard! A Standard! A STANDARD!

→ © You never need to write a parser again! Use XML! ©
XML Documents

Example DTD (Document Type Description)

Related → (colleague | friend | family)*
colleague → (name, affil*, email*)
friend → (name, affil*, email*)
family → (name, affil*, email*)
name → (#PCDATA)

Element names and their content

Example DTD

Related → (colleague | friend | family)*
colleague → (name, affil*, email*)
friend → (name, affil*, email*)
family → (name, affil*, email*)
name → (#PCDATA)

Example DTD

Related → (colleague | friend | family)*
colleague → (name, affil*, email*)
friend → (name, affil*, email*)
family → (name, affil*, email*)
name → (#PCDATA)

Example DTD

Related → (colleague | friend | family)*
colleague → (name, affil*, email*)
friend → (name, affil*, email*)
family → (name, affil*, email*)
name → (#PCDATA)

What else: (besides element and text nodes)

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

Terminology
document is *valid wrt the DTD*
"It validates"
XML Documents

What else:
- attributes
- processing instructions
- comments
- namespaces
- entity references (two kinds)

<?php sql ("SELECT * FROM ...") ...
See 2.6 Processing Instructions

<?php sql ("SELECT * FROM ...") ...
See 2.6 Processing Instructions

<?php sql ("SELECT * FROM ...") ...
See 2.6 Processing Instructions

<?php sql ("SELECT * FROM ...") ...
See 2.6 Processing Instructions

<!-- some comment -->

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

This document was prepared on &docdate; and

<!-- the 'price' element's namespace is http://ecommerce.org/schema -->

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>

character reference
Type <key>less-than</key>
(&amp;#x3C;) to save options.
Early Markup

The term markup has been coined by the typesetting community, not by computer scientists.

With the advent of printing press, writers and editors used (often marginal) notes to instruct printers to
→ Select certain fonts
→ Let passages of text stand out
→ Indent a line of text, etc

Proofreaders use a special set of symbols, their special markup language, to identify typos, formatting glitches, and similar erroneous fragments of text.

The markup language is designed to be easily recognizable in the actual flow of text.

Early Markup

Computer scientists adopted the markup idea—originally to annotate program source code:
→ Design the markup language such that its constructs are easily recognizable by a machine.

→ Approaches
   (1) Markup is written using a special set of characters, disjoint from the set of characters that form the tokens of the program
   (2) Markup occurs in places in the source file where program code may not appear (program layout).

Example: Fortran 77 fixed form source:
→ Fortran statements start in column 7 and do not exceed column 72.
→ A Fortran statement longer than 66 chars may be continued on the next line
   if a character not in (0,_) is placed in column 6 of the continuing line
→ Comment lines start with a “C” or “!” in column 1.
→ Numeric labels (DO, FORMAT statements) have to be placed in columns 1-5.
Stage 2: HTML-Style Physical Markup

```
<h1>Dilbert</h1>
<p>Panel 1</p>
1. &lt;li&gt; &lt;b&gt;Pointy-Haired Boss&lt;/b&gt; &lt;/em&gt; Speed is the key to success! &lt;/em&gt;&lt;/li&gt;
2. &lt;li&gt; &lt;/i&gt; &lt;em&gt; &lt;b&gt;Dilbert&lt;/b&gt; &lt;/em&gt; &lt;/i&gt; &lt;i&gt; &lt;em&gt;&lt;em&gt;&lt;em&gt; &lt;i&gt;&lt;em&gt;&lt;i&gt;&lt;i&gt;&lt;em&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;i&gt;&lt;}
Today, XML has many friends:

Query Languages
- XPath, XSLT, XQuery, … (mostly by W3C)

Implementations
- (Parsers, Validators, Translators)
  - SAX, Xalan, Galax, Xerces, …
    (by IBM/Apache, Microsoft, Oracle, Sun…)

Current Issues
- DB/PL support ('data binding', JBind, Castor, Zeus…)
- storage support (compression, data optimization)

XML: where is it used?

- Document formats:
  - SVG (vector images), OpenDocument Format (OpenOffice, GoogleDocs), DocBook (Text formatting), EPUB (electronic books), XHTML (web), …

- Protocol formats:
  - SOAP (Web Services), XMPP (Jabber, GoogleTalk), AJAX (custom XML messages used for interactive web sites: Facebook, Gmail, Tweet…), RSS feeds (used for blogs/news sites), …

- Custom data format:
  - Bio-informatics, Linguistics, Configuration files, Geographic data (OpenStreetMap, Google maps), Bibliographic Resources (MedLine, ADC)

XML: typical usage scenario

Regularity Tree languages (REGT).

Many characterizations:
- Regular Tree Grammars
- Tree Automata
- MSO Logic

Nice properties: Closed under intersection (union, complement) Decidable equivalence

(3) Theoretical / Mathematical

2. Well-Formed XML

From the W3C XML recommendation

http://www.w3.org/TR/REC-xml/

"A textual object is well-formed XML if,
(1) taken as a whole, it matches the production labeled document
(2) it meets all the well-formedness constraints given in this specification …"

document = start symbol of a context-free grammar ("XML grammar")

→ (1) contains the context-free properties of well-formed XML
→ (2) contains the context-dependent properties of well-formed XML

There are 10 WFCs (well-formedness constraints).
E.g.: Element Type Match "The Name in an element's end tag must match the element name in the start tag."

2. Well-Formed XML

From the W3C XML recommendation

http://www.w3.org/TR/REC-xml/

"A textual object is well-formed XML if,
(1) taken as a whole, it matches the production labeled document
(2) it meets all the well-formedness constraints given in this specification …"

document = start symbol of a context-free grammar ("XML grammar")

→ (1) contains the context-free properties of well-formed XML
→ (2) contains the context-dependent properties of well-formed XML

There are 10 WFCs (well-formedness constraints).
E.g.: Element Type Match "The Name in an element's end tag must match the element name in the start tag."
2. Well-Formed XML

Context-free grammar in EBNF

\[ \text{lhs ::= rhs} \]

\text{lhs} a non-terminal symbol (e.g., document)
\text{rhs} a string over non-terminal and terminal symbols.

Additionally (EBNF), we may use regular expressions in \text{rhs}.

Such as:

- \( r^* \) denoting \( \epsilon, r, rr, rrr, \ldots \) zero or more repetitions
- \( r^+ \) denoting \( rr^* \) one or more repetitions
- \( r^? \) denoting \( r \) | \( \epsilon \) optional \( r \)
- \( [abc] \) denoting \( a | b | c \) character class
- \( [a-zA-Z] \) denoting \( a | b | \ldots | z \) character class

XML Grammar - EBNF-style

As usual, the XML grammar can be systematically transformed into a program, an XML parser, to be used to check the syntax of XML input.

Parsing XML

1. Starting with the symbol \text{document}, the parser uses the \text{lhs ::= rhs} rules to expand symbols, constructing a parse tree.
2. Leaves of the parse tree are characters which have no further expansion.
3. The XML input is parsed successfully if it perfectly matches the parse tree's front (concatenate the parse tree's leaves from left-to-right, while removing \( \epsilon \) symbols).

Example 1
Parse tree for XML input

\(<\text{bubble speaker="phb">Um... No.</bubble}>\>

Example 2
Parse tree for XML input

\(<?xml version="1.0">\<foo/>\>

XML Grammar - EBNF-style
Character Encoding

- For a computer, a character like $X$ is nothing but an 8 (16/32) bit number whose value is *interpreted* as the character $X$, when needed.
- Problem: many such numbers $\rightarrow$ character mappings, the so called *encodings* are in use today:
  - Due to the huge amount of characters needed by the global computing community today (Latin, Hebrew, Arabic, Greek, Japanese, Chinese ...), *conflicting intersections* between encodings are common.

Example

```
0xcb 0xe4 0xd3  \rightarrow  Æ  δ  Σ
iso-8859-7

0xcb 0xe4 0xd3  \rightarrow  Ý  à  Ô
iso-8859-15
```

Questions

- How can you implement the three well-formed constraints?
- When, during parsing, do you apply the checks?

Unicode

- The Unicode [http://www.unicode.org](http://www.unicode.org) initiative aims to define a new encoding that tries to embrace all character needs.
- The Unicode encoding contains characters of "all" languages of the world plus scientific, mathematical, technical, box drawing, ... symbols
- Range of the Unicode encoding: 0x0000-0x10FFFF (=16*65536)
  - Codes that fit into the first 16 bits (denoted U+0000-U+FFFF) encode the most widely used languages and their characters (Basic Multilingual Plane, BMP)
  - Codes U+0000-U+007F have been assigned to match the 7-bit ASCII encoding which is pervasive today.

Unicode Transformation Formats

Current CPUs operate most efficiently on 32-bit words (16-bit words, bytes)

Unicode thus developed Unicode Transformation Formats (UTFs) which define how a Unicode character code between U+0000 and U+10FFFF is to be mapped into a 32-bit word (16-bit word, byte).

**UTF-32**
- Simply map exactly to the corresponding 32-bit value
- For each Unicode character in UTF-32: waste of at least 11 bits!

UTF-16
- Map a Unicode character into one or two 16-bit words
  - $U+0000$ to $U+FFFF$ map exactly to the corresponding 16-bit value
  - above $U+FFFF$: subtract 0x10000 and then fill the $\Box$'s in
  - Eg. Unicode character U+012345 (0x012345 - 0x10000 = 0x02345)

UTF-16: 1101 1000 0000 1000  \rightarrow  1101 1000 0000 1000 1101 1000 0000 1000 0101
Unicode Transformation Formats

- **UTF-16** works correctly, because the character codes between
  1101 10ex xxxx xxxx and
  1101 11xx xxxx xxxx (with each x replaced by a 0)
  are left unassigned in Unicode!! (range 0xD800 – 0xDFFF is reserved)

### UTF-8

Maps a unicode character into 1, 2, 3, or 4 bytes.

<table>
<thead>
<tr>
<th>Unicode range</th>
<th>Byte sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+000000 → U+00007F</td>
<td>0xxxxx</td>
</tr>
<tr>
<td>U+000080 → U+0000FF</td>
<td>110xxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+001000 → U+10FFFF</td>
<td>1110xxxxx 10xxxxxx 10xxxxxx</td>
</tr>
</tbody>
</table>

Spare bits (□) are filled from right to left. Pad to the left with 0-bits.

E.g. U+00A9 in UTF-8 is 11000010 10101001
U+2260 in UTF-8 is 11100010 10001001 10100000

### XML and Unicode

- A conforming XML parser is required to correctly process UTF-8 and
  UTF-16 encoded documents. (The W3C XML Recommendation
  predates the UTF-32 definition)

- Documents that use a different encoding must announce so using the
  XML text declaration, e.g.,
  ```
  <?xml encoding="iso-8859-15"/>
  ```
  or
  ```
  <?xml encoding="utf-32"/>
  ```

- Otherwise, an XML parser is encouraged to _guess_ the encoding while
  reading the very first bytes of the input XML document:

<table>
<thead>
<tr>
<th>Head of doc (bytes)</th>
<th>Encoding guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 0x3C 0x00 0x3F</td>
<td>UTF-16 (little Endian)</td>
</tr>
<tr>
<td>0x3C 0x00 0x3F 0x00</td>
<td>UTF-16 (big Endian)</td>
</tr>
<tr>
<td>0x3C 0x3F 0x78 0x6D</td>
<td>UTF-8 (or ASCII)</td>
</tr>
</tbody>
</table>

  Notice: < = U+003C, ? = U+003F, x = U+0078, m = U+006D

### Questions

- What does “guess the encoding” mean? Under which circumstances
does the parser know it has determined the correct encoding?

- Are there cases when it can NOT determine the correct encoding?

- What about efficiency of the UTFs? For different texts, compare the
  space requirement in UTF-8/16 and UTF-32 against each other.
Which characters do you find above 0xFFFF in Unicode? Can you imagine a scenario where UTF-32 is faster
than UTF-8/16?
The XML Processing Model

- On the physical side, XML defines nothing but a flat text format, i.e., it defines a set of (e.g., UTF-8/16) character sequences being well-formed XML.
- Applications that want to analyze and transform XML data in any meaningful way will find processing flat character sequences hard and inefficient!
- The nesting of XML elements and attributes, however, defines a logical tree-like structure.

![Diagram of XML elements and attributes]

The XML Processing Model

- Virtually all XML applications operate on the logical tree view which is provided to them by an XML processor (i.e., "parse & store").
- XML processors are widely available (e.g., Apache's Xerces).

How is the XML processor supposed to communicate the XML tree structure to the application?

- For many PL's there are "data binding" tools. Gives very flexible way to get PL view of the XML tree structure.

But first, let's see what the standard says...

### XML Information Set - http://www.w3.org/TR/xml-infoset

<table>
<thead>
<tr>
<th>Node type</th>
<th>Property</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC</td>
<td>children: DOC→ELEM</td>
<td>root element</td>
</tr>
<tr>
<td></td>
<td>base-uri: DOC→STRING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>version: DOC→STRING</td>
<td></td>
</tr>
<tr>
<td>ELEM</td>
<td>localname: ELEM→STRING</td>
<td>[..]=sequence type</td>
</tr>
<tr>
<td></td>
<td>children: ELEM→[NODE]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>attributes: ELEM→[ATTR]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parent: ELEM→NODE</td>
<td></td>
</tr>
<tr>
<td>ATTR</td>
<td>localname: ATTR→STRING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value: ATTR→STRING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>owner: ATTR→ELEM</td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>code: CHAR→UNICODE</td>
<td>a single character</td>
</tr>
<tr>
<td></td>
<td>parent: CHAR→ELEM</td>
<td></td>
</tr>
</tbody>
</table>
Information set of a sample document

```
<forecast date="Thu, May 16">
  <condition>sunny</condition>
  <temperature unit="Celsius">23</temperature>
</forecast>
```

**Questions**

1. A NODE type can be one of DOC, ELEM, ATTR, or CHAR. In the two places of the property functions where NODE appears, which of the four types may actually appear there?

   For instance, is this allowed?

   ```
   localname(e1) = "forecast"
   children(e1) = [e2,e3]
   attributes(e1) = [a1]
   parent(e1) = d
   localname(a1) = "date"
   value(a1) = "Thu, May 16"
   owner(a1) = e1
   localname(e2) = "condition"
   children(e2) = [c1,c2,c3,c4,c5]
   attributes(e2) = []
   parent(e2) = e1
   ```

   What about WHITESPACE? Where in an XML document does it matter, and where not?

   Where in the Infoset Example (previous slide) are the returns and indentations of the document? (did we do a mistake? If so, what is the correct Infoset?)

**3. Parsers for XML**

Two different approaches:

1. Parser stores document into a fixed (standard) data structure (e.g., an Infoset compliant data structure, such as DOM)

   ```
   parser.parse("foo.xml");
   doc = parser.getDocument();
   ```

2. Parser triggers "events". Does not store!

   User has to write own code on how to store / process the events triggered by the parser.

**DOM – Document Object Model**

→ Language and platform-independent view of XML

→ DOM APIs exist for many PLs (Java, C++, C, Perl, Python, ...)

DOM relies on two main concepts

1. The XML processor constructs the **complete XML document tree** (in-memory)

2. The XML application issues DOM library calls to **explore and manipulate** the XML tree, or to **generate** new XML trees.

**Advantages**

- easy to use
- once in memory, no tricky issues with XML syntax anymore
- all DOM trees serialize to well-formed XML (even after arbitrary updates)

**Disadvantage**

Uses LOTS of memory!
Character strings (DOM type DOMString) are defined to be encoded using UTF-16 (e.g., Java DOM represents type DOMString using its String type).

Some methods

<table>
<thead>
<tr>
<th>DOM type</th>
<th>Method</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>nodeValue</td>
<td>nodeValue</td>
</tr>
<tr>
<td></td>
<td>nodeValue</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parentNode</td>
<td>Parent node</td>
</tr>
<tr>
<td></td>
<td>firstChild</td>
<td>Leftmost child</td>
</tr>
<tr>
<td></td>
<td>nextSibling</td>
<td>Returns NULL for root element or last child or attributes</td>
</tr>
<tr>
<td></td>
<td>childNodes</td>
<td>NodeList</td>
</tr>
<tr>
<td></td>
<td>attributes</td>
<td>NamedNodeMap</td>
</tr>
<tr>
<td></td>
<td>replaceChild</td>
<td>Node</td>
</tr>
<tr>
<td>Document</td>
<td>createElement</td>
<td>Element</td>
</tr>
<tr>
<td></td>
<td>createComment</td>
<td>Comment</td>
</tr>
<tr>
<td></td>
<td>getElementsByTagName</td>
<td>List of all element nodes in document order</td>
</tr>
</tbody>
</table>

DOM Level 1 (Core)

Creating an element/attribute using createElement/setAttribute does not wire the new node with the XML tree structure yet.

DOM type NodeList makes up for the lack of collection data types in most programming languages.

DOM type NamedNodeMap represents an association table (nodes may be accessed by name).

Example:

```
bubble.getProperties().getNamedItem("speaker")
```

DOM Level 1 (Core)

E.g. Find all occurrences of Dogbert speaking (attribute speaker of element bubble) of dom.bcc 1

```
// Face off  
void digbort(DOMDocument d) {
  DOMNodeList bubbles;
  DOMNode bubble, speaker;
  DOMNamedNodeMap attrs;
  for (unsigned i = 0; i < bubbles.getLength (); i++) {
    bubble = bubbles.item (i);
    attrs = bubble.getProperties ()
    if (attrs != 0) {
      if (attrs.getNamedItem("speaker") != 0) {
        compareString (DOMString("Dogbert"), attrs); // cost << "Found Dogbert speaking:" << endl;
      }
    }
  }
}
```

Questions

Given an XML file of, say, 50K, how large is its DOM representation in main memory?

How much larger, in the worst case, is a DOM representation with respect to the size of the XML document?

(difficult!)

How could we decrease the memory need of DOM, while preserving its functionality?
"Even trigger" Parsers for XML:

→ Build your own XML data structure and fill it up as the parser triggers input "events".