Web Data Management

Vera Goebel
Department of Informatics, University of Oslo
Fall 2014

Web Models
Web Search and Querying
Distributed XML Processing
Why use XML with a database?
-> XML-DBS architectures
Web Overview

- Publicly indexable web:
  - More than 25 billion static HTML pages.
  - Over 53 billion pages in dynamic web
- Deep web (hidden web)
  - Over 500 billion documents
- Most Internet users gain access to the web using search engines.
- Very dynamic
  - 23% of web pages change daily.
  - 40% of commercial pages change daily.
- Static versus dynamic web pages
- Publicly indexable web (PIW) versus hidden web (or deep web)
Properties of Web Data

• Lack of a schema
  – Data is at best “semi-structured”
  – Missing data, additional attributes, “similar” data but not identical

• Volatility
  – Changes frequently
  – May conform to one schema now, but not later

• Scale
  – Does it make sense to talk about a schema for Web?
  – How do you capture “everything”?

• Querying difficulty
  – What is the user language?
  – What are the primitives?
  – Aren’t search engines or metasearch engines sufficient?
Web Graph

• Nodes: pages; edges: hyperlinks
• Properties
  – Volatile
  – Sparse
  – Self-organizing
  – Small-world network
  – Power law network
Structure of the Web
Web Data Modeling

• Can’t depend on a strict schema to structure the data
• Data are self-descriptive
  
  ```json
  {name: {first: "Tamer", last: "Ozsu"},
   institution: "University of Waterloo",
   salary: 300000}
  ```

• Usually represented as an edge-labeled graph
  – XML can also be modeled this way
Search Engine Architecture
Web Crawling

• What is a crawler?
• Crawlers cannot crawl the whole Web. It should try to visit the “most important” pages first.
• Importance metrics:
  – Measure the importance of a Web page
  – Ranking
    • Static
    • Dynamic
• Ordering metric
  – How to choose the next page to crawl
Web Crawler Types

- Many Web pages change frequently, so the crawler has to revisit already crawled pages → incremental crawlers
- Some search engines specialize in searching pages belonging to a particular topic → focused crawlers
- Search engines use multiple crawlers sitting on different machines and running in parallel. It is important to coordinate these parallel crawlers to prevent overlapping → parallel crawlers
Indexing

• Structure index
  – Link structure

• Text index
  – Indexing the content
  – Suffix arrays, inverted index, signature files
  – Inverted index most common

• Difficulties of inverted index
  – The huge size of the Web
  – The rapid change makes it hard to maintain
  – Storage vs. performance efficiency
Web Querying

• Why Web Querying?
  – It is not always easy to express information requests using keywords.
  – Search engines do not make use of Web topology and document structure in queries.

• Early Web Query Approaches
  – Structured (Similar to DBMSs): Data model + Query Language
  – Semi-structured: e.g. Object Exchange Model (OEM)
Web Querying

• Question Answering (QA) Systems
  – Finding answers to natural language questions, e.g. *What is Computer?*
  – Analyze the question and try to guess what type of information that is required.
  – Not only locate relevant documents but also extract answers from them.
Approaches to Web Querying

• Search engines and metasearchers
  – Keyword-based
  – Category-based
• Semistructured data querying
• Special Web query languages
• Question-Answering
Semistructured Data Querying

- Basic principle: Consider Web as a collection of semistructured data and use those techniques
- Uses an edge-labeled graph model of data
- Example systems & languages:
  - Lore/Lorel
  - UnQL
  - StruQL
Searching The Hidden Web

• Publicly indexable web (PIW) vs. hidden web

• Why is Hidden Web important?
  – Size: huge amount of data
  – Data quality

• Challenges:
  – Ordinary crawlers cannot be used.
  – The data in hidden databases can only be accessed through a search interface.
  – Usually, the underlying structure of the database is unknown.
Searching The Hidden Web

• Crawling the Hidden Web
  – Submit queries to the search interface of the database
    • By analyzing the search interface, trying to fill in the fields for all possible values from a repository
    • By using agents that find search forms, learn to fill them, and retrieve the result pages
  – Analyze the returned result pages
    • Determine whether they contain results or not
    • Use templates to extract information
Searching The Hidden Web

• Metasearching
  – Database selection – Query Translation – Result Merging
  – Database selection is based on Content Summaries.
  – Content Summary Extraction:
    • RS-Ord and RS-Lrd
    • Focused Probing with Database Categorization
Distributed XML

• XML increasingly used for encoding web data → data representation language
  – Web 2.0
• Data exchange language
  – Web services
• Used to encode or annotate non-web semistructured or unstructured data
• Data repository sizes growing → use distribution
XML Overview

• Similarities to semistructured models
• Data is divided into pieces called **elements**
• Elements can be nested, but not overlapped
  - Nesting represents hierarchical relationships between the elements
• Elements have attributes
• Elements can also have relationships to other elements (ID-IDREF)
• Can be represented as a graph, but usually simplified as a tree
  - Root element
  - Zero or more child elements representing nested subelements
  - Document order over elements
  - Attributes are also shown as nodes
XML Preliminaries

• XML – Extensible Markup Language
  – Data model
  – API
  – Schema languages
  – Query languages
  – Transformation languages

• Web services
XML – In a Nutshell

Extensible Markup Language, World Wide Web Consortium (W3C)

- Standard of the W3C in 1998
- Documents (content)
- XML Schema or DTD (structure)
- Namespaces
- XSL, XSLT (stylesheet, transformations)
- XPath, XPointer, XLink
- XQuery

- Tagged elements with attributes
  - Extensible, self-describing
- Tag names must be unique (use namespaces)
Namespaces

• All tags must be defined in a namespace
  – Unique global names using URI references

• W3C provides certain standard namespaces (xsd, xsi, html)

• Organizations can provide domain specific namespaces

• Each document specifies the namespaces used in its schema
  – A default namespace - a standard or domain namespace
  – Its own target namespace - for elements defined in document
  – Others …

• Referencing a name (syntax)
  – Qualified names (namespace:name)  xsd:element
  – name (default namespace implied)  car
Elements

- Element declarations (see XML Schema)
  <xsd:element name="numStudents" type="xsd:positiveInteger"/>

- Element definitions
  - start tag <tagName> must be paired with an
  - end tag </tagName>
  - <emptyElement/>
    <numStudents>31</numStudents>

- Sub-elements may be repeated
  - Subject to minOccurs and maxOccurs constraints (XML Schema)
    • Both default to 1 if not specified

- References
  <xsd.element name="foo", type="xsd.string"/>
  ...
  <xsd.element ref="foo"/>
Attributes, Entities

- Attributes (of an element)
  - Name-value pairs placed in the start tags
    - name = "value"
    - Each name may only appear once (single-valued)

<elevation units="feet">5440</elevation>

- Entity
  - &mdash; an entity representing the character ‘—’
  - starts with & and ends with ;
  - Used to distinguish reserved words from content
    - &lt; represents the character ‘<‘ (also &gt; &amp; &apos; &quot;)
  - <!--ENTITY myPic SYSTEM “me.jpg” NDATA JPEGFORMAT>
<xml version="1.0"?>
<racingTeams xmlns="http://www.cart.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
  xsi:schemaLocation="http://www.cart.org team.xsd">
  <team>
    <name>"Shell"</name>
    <car number=3><make>"Helix Ford"</make>
      <engine id="47456"/></car>
    <car number=4><make>"Helix Ford"</make>...
    <driver><name>"Steve Johnson"</name>
      <points>2217</points></driver>
    <driver><name>"Paul Radisich"</name>
    <spares><engine id="102555"/>
      <engine id="102556"/></spares>
  </team>
  ...
</racingTeams>
<!-- Each document has one root element (e.g., racingTeams) whose name
     matches a schema element or <!DOCTYPE name> -->
XML in Action

1. Export: XML document written to file or stream
2. Delivery: Passed to import software
3. Import: Parse XML document, Validate using DTD or Schema (optional)
Semi-Structured Data

- Semi-structured data is data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably.
  - It generally has some structure, but does not conform to a fixed schema
  - Self-describing (e.g., in terms of XML element tags)

- Characteristics
  - Heterogeneous
  - Irregular structure
  - Large evolving schema
  - Hyperdata nodes and references

- Desirable to treat web sources like a database
  - Often XML documents are like semi-structured data
Semi-Structured Example

DreamHome (&1)
Branch (&2)
  street (&7) “22 Deer Rd”
Manager &3
Staff (&3)
  name (&8)
    fName (&17) “John”
    lName (&18) “White”
  ManagerOf &2
Staff (&4)
  name (&9) “Ann Beech”
  salary (&10) 12000
Oversees &5
Oversees &6

PropertyForRent (&5)
  street (&11) “2 Manor Rd”
  type (&12) “Flat”
  monthlyRent (&13) 375
  OverseenBy &4
PropertyForRent (&6)
  street (&14) “18 Dale Rd”
  type (&15) 1
  annualRent (&16) 7200
  OverseenBy &4
XML and Data on the Internet

• Primary purpose:
  – Sharing of data across systems
    • In particular, across the Internet
  – Standard to encode structured (and semistructured) information

• Features:
  – Text-based
    • Simultaneously human- and machine-readable format
    • Self-documenting format (structure & field names + data values)
  – Can represent most general data structures:
    • records, lists, trees
  – Platform-independent
  – Strict syntax
  – Simple, efficient, consistent parsing algorithms
  – Extensible
    • Can create your own tags, i.e., your own vocabulary
XML Data Models

• XML Information Set (Infoset)
  – An XML document has an information set if it is well-formed and conforms to Namespaces standard
  – XML document’s information set consists of information items with a set of named properties
  – Usually created through parsing an XML document
  – Information set and information item similar in meaning to tree and node respectively
    • but no one-to-one mapping with nodes of DOM or the tree and nodes of the XPath data model
  – No requirements for specific interface
    • Not only tree structure, also others, e.g., event-based, query-based
  – 11 different types of information items
XML API

• DOM
  – Interface-oriented API
  – Allows for navigation in the document as if it were a tree
  – DOM parser creates a tree of objects before access is allowed

• SAX
  – Lexical, event-driven interface
  – Document is read serially; contents reported as “callbacks” to methods on a handler object
XML Schema Languages

• Purpose: to identify a *class* of XML documents
  – DTD – Document Type Definition
  – XML Schema
XML Schema

• For declaring
  – the *organization* of instance documents
  – the *datatype* of each element/attribute

• Features
  – XML-based syntax
    • Can use XML tools to generate XML Schemas
  – Integrated with namespace mechanism
    • xsd is standard default namespace for schemata
    • Included schema must have same targetNamespace
  – Rich set of built-in types
  – Elements have nested scope
  – Keys definable in terms of elements
  – Referential integrity constraints enforced with keys
  – Options for ordered (sequences) or unordered (sets)
  – Validation of documents against a schema
XML Schema Grammar

- Namespace
- Compositors
  - `<sequence>`, `<choice>`, `<all>`, `<union>`, `<list>`
- `complexType`, `simpleType`
- `complexContent`, `simpleContent`
- `restriction`, `extension`, `enumeration`
- ID, IDREF [List(IDREF) replaces IDREFS ]
- unique like a candidate key, whenever present => unique
- key minOccurs > 0, nillable = “false”
  - selector collection to which key applies (like Table)
  - field path expressions relative to selector collection
- keyref specifies foreign key constraint
- Empty elements
XML Schema Example

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.binary.org"
xmlns="http://www.binary.org">
  <xsd:element name="life">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="eat" type="xsd:string"/>
        </xsd:sequence>
        <xsd:choice>
          <xsd:element name="work" type="xsd:string"/>
          <xsd:element name="play" type="xsd:string"/>
        </xsd:choice>
        <xsd:element name="sleep" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

life ::= ( (work, eat)*, [ work | play ], sleep )*
XML

```xml
<vehicles>
  <vehicle>
    <nickname>Count Zero</nickname>
    <model>Series I, 80"</model>
    <construction>
      <start>1949-07-21</start>
      <end>1949-08-09</end>
    </construction>
  </vehicle>
</vehicles>
```

**XML Schema**

Is this data valid?

To be valid, it must meet these constraints (data business rules):

1. The vehicle must be comprised of a nickname, followed by a model, followed by a construction description.
2. Any number of vehicles can be listed in the contents of vehicles
<? xml version="1.0" ?>
<racingTeams xmlns="http://www.cart.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance"
xsi:schemaLocation="http://www.cart.org team.xsd">
<team>
  <name>Shell</name>
  <car number=3><make>"Helix Ford"</make><engine id="247456"/></car>
  <car number=4><make>"Helix Ford"</make> <engine id="102456"/></car>
  <driver><name>"Steve Johnson"</name>
    <points>2217</points></driver>
  <driver><name>"Paul Radisich"</name></driver>
  <spares>
    <engine id="102555"/>
    <engine id="102556"/>
  </spares>
</team>
...</racingTeams>

<!-- Each document has one root element (e.g., racingTeams) whose name matches a schema element or <!DOCTYPE name> -->
XML Query Languages

- XPath
- XQuery
XPath

- View XML document as a tree of nodes and references (edges)
- XPath: Sequence of steps that yields a set of nodes

/ step / step / step ....  
step = basis + [predicates]

(Like a unix path, but each step can be a query expression)

/    root node
.    current node
..   parent node
@attr access attribute "attr"
[n]   access the nth occurrence of a node at this level
last()
//   descendent or self
text()  text content of node
functions sum(), count(), ...
[...] selection condition (predicate)

- Example:

//student[status="ok" and starts_with(.//last, "E") and not (.//last=.//first)]
XQuery

- **W3C standard**
  - (XQuery 1.0) W3C recommendation Jan 2007
  - Shares common datamodel with XPath 2.0
  - Several implementations

- **Data model can handle single documents, partial fragments, and collections of documents**

- **In same query can**
  - specify what you are looking for
  - designate what its output format should look like

- **Three language "formats"**
  - surface syntax
  - alternative XML-based syntax (XQueryX)
  - formal algebraic language
XQuery Expressions

• Path expressions
  – XPath syntax + dereference operation (->) + range predicate
  – Path expression returns an ordered list of nodes
  – Start at document(string), /, or //
• Element constructors
• FLWR expressions
  – FOR
  – LET
  – WHERE
  – RETURN
• Expressions involving operators and functions
• Conditional expressions
• Quantified expressions
• List constructors
• Expressions that test or modify data types
XQuery Expressions

• Path expressions
  – XPath syntax + dereference operation (->) + range predicate
  – Path expression returns an ordered list of nodes
  – Start at document(string), /, or //
• Element constructors
• FLWOR expressions
  – FOR
  – LET
  – WHERE
  – ORDER BY
  – RETURN
• Expressions involving operators and functions
• Conditional expressions
• Quantified expressions
• List constructors
• Expressions that test or modify datatypes
FLWOR
(pronounced "flower")

FOR var IN expr
LET var IN expr
WHERE expr
ORDER BY expr
RETURN expr

FOR and LET clauses generate a list of tuples of bound expressions, preserving document order;
WHERE clause applies a predicate, eliminating some of the tuples;
ORDER BY clause rearranges the sequence of surviving tuples;
RETURN clause is executed for each resulting tuple, generating an ordered list of outputs.

- FOR iterates through a sequence of individual nodes out of the selected collection, in order, one at a time;
- LET binds a variable to the set of nodes in the selected collection;
- FOR or LET clauses are considered nested; later clauses may reference variables bound in previous clauses.
FLWR (2)

• WHERE clause is optional
  – Predicates using bound variables

• RETURN generates the output (constructor expression for output)
  – node | [ordered] forest of nodes | primitiveValue

  – The constructor is executed once for each tuple of bindings from FOR or LET clauses that satisfies the WHERE clause.
  – It preserves ordering where ordering exists
  – The RETURN expression often contains Element constructors
    • With references to bound variables and nested subelements
  – A list may be constructed by enclosing zero or more expressions in square brackets, separated by commas
XQuery Example

XML document:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v2006 (http://www.altova.com) -->
<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="WEB">
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
<author>Per Bothner</author>
<author>Kurt Cagle</author>
<author>James Linn</author>
<author>Vaidyanathan Nagarajan</author>
<year>2003</year>
<price>49.99</price>
</book>

<book category="WEB">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>

</bookstore>
```

XQuery:

```xquery
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Result:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```
XQuery Examples

Path expression example:
document("zoo.xml")/chapter[title = "Frogs"]//figref/@refid->fig/caption

FLWR expression example:
FOR $p$ IN distinct(document("bib.xml")//publisher)
  LET $a :=$ avg(document("bib.xml")
     /book[publisher = $p$/price)
  RETURN
  <publisher>
    <name> $p/text() </name>
    <avgprice> $a </avgprice>
  </publisher>

Note: result(/) = result(.)+result(/)
Probably here / and // give the same result

Note: this “publisher” name is in the local scope of document("bib.xml")
from the context of the path expression

And this “publisher” name must match the name of an element in the default namespace 😊
XML Transformation Languages

• XSLT – Extensible Stylesheet Language Transformations
  – Language for transforming an XML document into another XML document, or to HTML, or another text-based format
  – Rule-based

• CSS – Cascading Style Sheets
  – For describing the presentation (colors, fonts, layout,...) of a document written in a markup language
  – Allows for separation of document content (written in e.g. XML) from document presentation (written in CSS)
  – Non-XML notation
Example XSLT

XSLT stylesheet

```xml
```

Source XML file

```xml
<?xml version="1.0"?>
<PEOPLE>
  <PERSON>
    <NAME>Mark Wilson</NAME>
    <ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
    <TEL>(+612) 12345</TEL>
    <FAX>(+612) 12345</FAX>
    <EMAIL>Mark.Wilson@example.com</EMAIL>
  </PERSON>
  ...
</PEOPLE>
```

Resulting HTML output

```html
<TABLE BORDER="2"><TR><TD>Name</TD><TD>Address</TD><TD>Tel</TD><TD>Fax</TD><TD>Email</TD></TR><TR><TD>Mark Wilson</TD><TD>911 Somewhere Circle, Canberra, Australia</TD><TD>(+612) 12345</TD><TD>(+612) 12345</TD><TD>Mark.Wilson@example.com</TD></TR>```

```html
..
```
Xpointer

• Xpointer: an Xpath expression within a URI +
  points (position within a document), or
  ranges (start-point, end-point)

  – xpointer(id('list37')/item)
  – xpointer(id('list37')/item[1]/range-to(following-sibling::item[2]))

• http://someURL#xpointer(XPath expression) (entities, escapes)

  – doc.xml#xpointer(string-range(/P,"a little hat ^^"))

• Nested elements from different namespaces (same local name)

  – xmlns(x=http://example.com/foo) xmlns(y=http://example.org/bar)
    xpointer(/x:a/y:a)
XML Metadata Interchange

- XMI is an XML language to interchange the objects of software systems:
  - UML/MOF\(^1\) designs
  - Computer software classes and interfaces
  - Databases and database schemas
  - DTDs and XML schemas
- Added to the list of OMG adopted technologies in February, 1999 as XMI 1.0
- Most recent minor revision is XMI 1.1 (November 2000)

---

\(^1\) MOF = Meta-Object Facility, describing data about data, i.e., the classes and associations structure of the data
XMI: Sharing Objects

- XML - Sharing Data
- XMI - Sharing Objects
  - Creates custom
    - XML Schemas (DTDs)
    - XML documents from class definitions.

![Diagram showing data exchange between applications and key elements related to XML and Unicode.]
## XMI Object Mapping

<table>
<thead>
<tr>
<th>MOF Concept</th>
<th>XML Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Class</td>
<td>XML element, odd levels</td>
</tr>
<tr>
<td>• Object typed attribute</td>
<td>XML element, even levels</td>
</tr>
<tr>
<td>• Containment</td>
<td>XML element, even levels</td>
</tr>
<tr>
<td>• Reference</td>
<td>XML attribute IDREF</td>
</tr>
<tr>
<td>• Basic attribute (int, String)</td>
<td>XML attribute CDATA</td>
</tr>
</tbody>
</table>
XMI Mapping Example

Class `Car` {  // Class
  String `color`;  // Field, String
  Person `customer`;  // Field, Reference
  Engine `standardEngine`;  // Field, Object
  Engine `optionalEngine`;  // Field, Object
}

`<Car color="Blue" customer="Joe">`

  `<Car.standardEngine>`
  `<Engine ..... />`
  `</Car.standardEngine>`

  `<Car.optionalEngine>`
  `<Engine ..... />`
  `</Car.optionalEngine>`

`</Car>`

XMI Instance

- XML element, odd levels
- XML element, even levels
- XML attribute IDREF
- XML attribute CDATA
Validation in XMI

MOF model (XMI Document) → Generated XML Schema

- **Generate**
- **Semantic Validation**
- **Optional Syntactic Validation**

Objects (XMI Document) → XML Parser

- **Well-formed**
XML Schema: Specifies the Properties for a Class of Resources

```
<xsd:complexType name="Book">
    <xsd:sequence>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
        <xsd:element name="Date" type="xsd:year"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

"For the class of Book resources, we identify five properties - Title, Author, Date, ISBN, and Publisher"
"For a specific instance of a Book resource, here are the values for the properties.

Use schemaLocation to identify the companion document (i.e., the schema) which defines the Book class of resources."
Metadata Interchange using XML Schema

- XML Schema Strategy - two documents are used to provide metadata:
  - a schema document specifies the properties (metadata) for a class of resources (objects);
  - an instance document provides specific values for the properties of each object instance being transferred.

- Describing a schema for an object model using XML Schema requires another XML Schema that describes the object model’s metamodel!

- XMI uses an XML Schema for the UML metamodel 😊
Web Services

- Collection of protocols and standards that allows making processing requests to remote systems using a common, non-proprietary language and using common transport protocols
Internet Application Architecture: Today

Client Tier

Data Sources

Gateways

Other Data Sources

OLE/DB Data source

Middle-Tier Application

Physical Middle Tier

Application messages

Remote messages

Browser

HTTP

Browser

HTTP

WEB/APP Server

Data Integration, Storage, Query, Management

ORDBMS

authoring tools, etc.

authoring tools, etc.
Internet Applications

- Entertainment
  - Games, Music, Films, Multi-person chat
- Public information
  - Maps, Tax return helper
- Advertisement
  - Interactive catalogues for products and services
- Medicine
  - Diagnosis, Consultation, Remote surgery
- Education
  - Learning-on-demand (for a degree), virtual museums, tour remote spaces
- Engineering
  - Collaborative design, remote parallel simulation services
- Publishing
  - Submit, Review, Proof-editing (text and graphics)
- Tele-communication
  - Conferencing
- ...
Web Services
Architecture Principles

• Message orientation
  – Using only messages to communicate between services

• Protocol composability
  – Flexible infrastructure protocol building blocks

• Autonomous services
  – Endpoints can be built, deployed, managed, versioned, and secured independently

• Managed transparency
  – Controlling which aspects of an endpoint are visible to external services

• Protocol-based integration
  – Restricting cross-application coupling to wire artifacts only
Web Services
Core Specifications

• **SOAP**: Message envelope format
  – XML-based

• **WSDL**: Service interface description language
  – XML-based
  – Description of service and bindings to protocols
  – To generate server and client code and for configuration

• **UDDI**: Protocol for publishing and discovering metadata about Web services
  – To enable applications to find Web services at design time or runtime

• **WS-Security**: For secure message exchange
  – How to use XML_Encryption and XML_Signature

• **WS-ReliableExchange**: Protocol for reliable messaging between two Web services
Why Use XML with a Database?

• XML as transport format
  – to/from database
• XML for semistructured data
  – schema to be stored not known at design time
• XML document archiving
  – retaining complete XML documents

The application domain has an impact on the choice of XDBMS
XML as Transport Format

- When
  - publishing as XML data stored in a DB, or storing data arriving as XML documents
    - between an existing DB and applications
    - between two DBs
- Why XML
  - platform independence

Publishing relational data as XML:

Storing XML data in an RDB:
XML for Semistructured Data

• When
  – application domains where
    • all data formats cannot be predicted in advance +
    • large volumes of data

• Why
  – why XML: extensibility
  – why DB: need to store, manage, and query data
XML Document Archiving

• When
  – retaining XML documents

• Why
  – processing
  – contracting/legislation
  – granularity: XML document or document fragment is the logical unit
Classes of XML Documents

• Data-centric XML document:
  – Fairly regular structure
  – Fine-grained data

• Document-centric XML document:
  – Irregular structure
  – Larger grained data

The XML documents being data-centric or document-centric has an impact on the choice of XDBMS
Data-Centric XML Document

• Characteristics:
  – Discrete pieces of data
  – Fairly regular structure
  – Fine-grained data
    • smallest independent unit of data is at the level of a PCDATA-only element or an attribute
  – Designed for machine consumption
  – Not important to the application or DB that the data is for the time being stored in an XML document
Example

<SalesOrder Number="123">
  <OrderDate>2003-07-28</OrderDate>
  <CustomerNumber>456</CustomerNumber>
  <Item Number="1">
    <PartNumber>XY-47</PartNumber>
    <Quantity>14</Quantity>
    <Price>16.80</Price>
  </Item>
  <Item Number="2">
    <PartNumber>B-987</PartNumber>
    <Quantity>6</Quantity>
    <Price>2.34</Price>
  </Item>
</SalesOrder>
Document-Centric XML Document

- Characteristics:
  - Mixed content
  - Irregular structure
  - Larger grained data
    - Smallest independent unit of data at the level of an element with mixed content or the entire document
  - Designed for human consumption
  - The document order (order of siblings etc.) is significant
Example

<Product>
<Intro>
The <ProductName>Turkey Wrench</ProductName> from <Developer>Full Fabrication Labs, Inc.</Developer> is <Summary>like a monkey wrench, but not as big.</Summary>
</Intro>
<Description>
<Para>You can:</Para>
<List>
<Item><Link URL="Order.html">Order your own turkey wrench</Link></Item>
<Item><Link URL="Wrenches.htm">Read more about wrenches</Link></Item>
<Item><Link URL="Catalog.zip">Download the catalog</Link></Item>
</List>
<Para>The turkey wrench costs <b>just $19.99</b> and, if you order now, comes with a <b>hand-crafted shrimp hammer</b> as a bonus gift.</Para>
</Description>
</Product>
The turkey wrench costs just $19.99 and, if you order now, comes with a hand-crafted shrimp-hammer as a bonus gift.

Full Fabrication Labs, Inc. like a monkey wrench, but not as big. You can: Order your own turkey wrench, Read more about wrenches, Download the catalog.
XML Document Content Types

• **Element content**
  – Content consists of child elements only

• **PCDATA-only content**
  – Content is a single piece of text

• **Mixed content**
  – Children are a mixture of text-only and elements
The <ProductName>Turkey Wrench</ProductName> from <Developer>Full Fabrication Labs, Inc.</Developer> is <Summary>like a monkey wrench, but not as big.</Summary>

You can:

- <Link URL="Order.html">Order your own turkey wrench</Link>
- <Link URL="Wrenches.htm">Read more about wrenches</Link>
- <Link URL="Catalog.zip">Download the catalog</Link>

The turkey wrench costs <b>just $19.99</b> and, if you order now, comes with a <b>hand-crafted shrimp hammer</b> as a bonus gift.
Order Can Matter!

You can: and, if you order now, comes with a <b>just $19.99</b> The turkey wrench costs <b>hand-crafted shrimp hammer</b> as a bonus gift.
Hybrid XML Documents

• Otherwise document-centric documents, but
  – with parts that contain fine-grained, regularly structured data

• Otherwise data-centric documents, but
  – with parts that contain large-grained, irregularly structured data
XDBMS - How to Achieve XML + DB Technology?

• XML + XML tools used as a ”DBMS”
  – Managing without a traditional DBMS

• XML-enabled DBMS
  – Data from XML documents are stored in a traditional database
  – The DBMS is augmented with special-purpose sw for managing XML data storage and retrieval

• Native XML DBMS (NXDBMS)
  – DBMS designed especially for storing XML documents
XML as a DBMS, Pros and Cons

- Idea: XML documents are a natural format for storing data
  Use XML documents + XML tools + specialized sw as a "DBMS"

+ Hierarchical model
  + Rich data structure
+ Unicode
  + Portability
+ Text format
  + Readable by humans
+ Schema language
  + XML Schema, DTD
+ Query languages
  + XPath, XQuery
+ APIs
  + SAX, DOM, JDOM

- Inefficient storage format
  - Verbose
  - Need to parse slow data access
- Lacks
  - indices
  - transactions
  - multiuser access
  - security
  - logging
  - referential integrity
  - ...

Idea: XML documents are a natural format for storing data
Use XML documents + XML tools + specialized sw as a "DBMS"
XML as a DBMS, Conclusion

• Yes/maybe
  – small, single-user "databases" (configuration files, small contact lists,...)

• In general: No
  – must write a lot of code already found in modern DBMSs
  – most production environments require a real DBMS
XML-Enabled DBMSs

• For data-centric XML documents
• When XML is used as a data transport format
• How
  – Store by shredding
  – Publish by composing
Properties of XML-Enabled DBMSs

• DB schema models primarily the data in the XML document
  – Data model is ”traditional”
    • relational/object-relational/...
  – No XML ”visible” in DB

• Keeps existing data and applications intact
  – adding XML functionality is adding and configuring data transfer (data mapping) sw
Properties of XML-Enabled DBMSs (cont.)

- **XML mapping software**
  - integrated into the DB engine or external to it

- **Lossy modelling**
  - XML document as such is discarded after shredding
  - In general, XML mapping sw can handle only a subclass of XML documents
    - retains information that is important to the DB model:
      - data itself
      - hierarchical relationships (parent, child, siblings)
      - **not** entity references, CDATA sections, comments, processing instructions, DTD, maybe not ordering of siblings,...
  - Cannot reconstruct XML document in full detail
XML to DB Model Mapping

• XML mapping characteristics
  – Many-to-many
    • An XML document can be mapped to several DB schemas
    • Data of a DB schema can be mapped to several XML documents
  – Design time
    • Runtime (automatic) mapping has to make too many assumptions about DB schema

• Mapping techniques:
  – Bidirectional
    • From XML documents to the DB schema, and vice versa
  – Unidirectional
    • Using a query language to construct XML documents from a DB
XML to Relational DB
Bidirectional Mapping Techniques

• Table-based
  – Restrictions on XML document structure

• Object-relational
  – XML document is viewed as a set of objects
Table-Based Mapping

• Restrictions on XML document structure
  – document structure must reflect RDB table - row - column structure
  – Straightforward transformation between XML schema and DB schema
Table-Based XML Document

```xml
<Database>
  <SalesOrders>
    <SalesOrder>
      <Number>123</Number>
      <OrderDate>2003-07-28</OrderDate>
      <CustomerNumber>456</CustomerNumber>
    </SalesOrder>
    <SalesOrders>
      <Items>
        <Item>
          <Number>1</Number>
          <PartNumber>XY-47</PartNumber>
          <Quantity>14</Quantity>
          <Price>16.80</Price>
        </Item>
        <Item>
          <Number>2</Number>
          <PartNumber>B-987</PartNumber>
          <Quantity>6</Quantity>
          <Price>2.34</Price>
        </Item>
      </Items>
    </SalesOrders>
  </SalesOrders>
</Database>
```
Resulting Table-Based XML and DB Schemas

DB schema (tables):

SalesOrders (Number, OrderDate, CustomerNumber)
Items (SONumber, Number, PartNumber, Quantity, Price)

XML Schema:

<!ELEMENT Database (SalesOrders, Items)>
<!ELEMENT SalesOrders (SalesOrder*)>
<!ELEMENT SalesOrder (Number, OrderDate, CustomerNumber)>
<!ELEMENT Number (#PCDATA)>
<!ELEMENT OrderDate (#PCDATA)>
<!ELEMENT CustomerNumber (#PCDATA)>
<!ELEMENT Items (Item*)>
<!ELEMENT Item (SONumber, Number, PartNumber, Quantity, Price)>
<!ELEMENT PartNumber (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
Object-Relational Mapping

- XML document is viewed as a set of objects
- Use techniques from O-R to perform mapping
  - objects map to tables
  - object properties map to columns
  - inter-object relationships map to primary key/foreign key relationships
Object-Relational Mapping
Object-Relational Mapping

<SalesOrder Number="123">
    <OrderDate>2003-07-28</OrderDate>
    <CustomerNumber>456</CustomerNumber>
    <Item Number="1">
        <PartNumber>XY-47</PartNumber>
        <Quantity>14</Quantity>
        <Price>16.80</Price>
    </Item>
    <Item Number="2">
        <PartNumber>B-987</PartNumber>
        <Quantity>6</Quantity>
        <Price>2.34</Price>
    </Item>
</SalesOrder>
Object-Relational Mapping

SalesOrder
   Number - 123
   OrderDate = 2003-07-28
   CustomerNumber = 456
   Items = [pointers to Item objects]

Item
   Number = 1
   PartNumber = "XY-47"
   Quantity = 14
   Price = 16.80

Item
   Number = 2
   PartNumber = "B-987"
   Quantity = 6
   Price = 2.34

SalesOrder

<table>
<thead>
<tr>
<th>Number</th>
<th>Date</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>2003-07-28</td>
<td>456</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Item

<table>
<thead>
<tr>
<th>SONumber</th>
<th>Number</th>
<th>PartNumber</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>1</td>
<td>&quot;XY-47&quot;</td>
<td>14</td>
<td>16.80</td>
</tr>
<tr>
<td>123</td>
<td>2</td>
<td>&quot;B-987&quot;</td>
<td>6</td>
<td>2.34</td>
</tr>
</tbody>
</table>
XML to Relational DB

Unidirectional Mapping Techniques

• Mapping by querying
  – SQL/XML
    • extending SQL with constructs for creating XML fragments
  – XQuery
    • for querying XML documents
    • requires a pre-mapping of the relational DB to one or more virtual XML documents
SQL/XML

• Extensions to SQL for creating XML documents and document fragments from relational data
  – XML data type
    • Valid XML data types: NULL, XML document, element content
  – Scalar functions for creating XML
    • XMLELEMENT, XMLATTRIBUTES, XMLFOREST, XMLCONCAT,...
Example
SQL/XML Mapping

**Schema:** customers(id, name, address)

**SQL/XML statement:**

```sql
SELECT XML2CLOB(
  XMLELEMENT(
    NAME Customer,
    XMLATTRIBUTES(customers.id AS ID),
    XMLELEMENT(NAME Name,
                customers.name)
  )
) AS CustomerXML
FROM customers
```

**Result:** For each row in customers, the following value in the CustomerXML column:

```xml
<Customer ID="customer id">
  <Name>customer name</Name>
</Customer>
```
Example
XQuery Mapping

Schema: customers(id, name, address)

Pre-mapped XML document:

```xml
<customers>
  <Customer>
    <id>customer id</id>
    <name>customer name</name>
    <address>cust. address</address>
  </Customer>
  <Customer>
    ...
  </Customer>
</customers>
```

XQuery statement:

```xml
FOR $c IN document("http://cust.xml")//Customer
RETURN
  <Customer ID="{ $c/id }">
    <Name>{ $c/name }</Name>
  </Customer>
```

97
NXDBMS: Native XML DBMS

- For document-centric XML documents
  - need to be able to retain XML documents as-is
- For semi-structured data
  - when other DB models are inappropriate, e.g. since they require changing the DB schema when the XML schema changes/evolves
NXDBMS

• Specialized for storing XML data
  – Stores all components of the XML model intact

• XML document is the logical unit
  – (Cf. the logical unit in an RDBMS, which is a row)

• **Schema-independent NXDBMS:**
  – Can store any XML document regardless of its schema
Properties of NXDBMS

• Data model:
  – XML document is the fundamental (logical) unit of storage
  – The NXDBMS defines a data model for XML documents
    • As a minimum, elements, attributes, text (PCDATA), document order
    • Ex.: XPath data model, XML Infoset, models implied by DOM and the events in SAX 1.0
  – In addition, the NXDBMS may support traditional DB models
Properties of NXDBMS (cont.)

- Implementation of own choice
  - as CLOBs
  - as fixed set of tables
  - as DOM trees in OODBs
  - as indexed set of hash tables

- Physical storage of own choice

- NXDBMS abstracts away details; application developer can build applications using XML technologies
Properties of NXDBMS (cont.)

• Retrieval speed
  – Can utilize physical storage if reflects XML document structure
    • Performance depends heavily on retrieval patterns coinciding with XML document structure

• Data export format
  – XML (for almost all NXDBMSs)
  – If application needs other format, must parse
    • cumbersome for local applications
    • OK for distributed applications that use XML as data transport format

• Schema-independent NXDBMSs
  – Flexibility
  – Easy application development
  – Risk of low data integrity
Properties of NXDBMS (cont.)

• Query languages
  – XPath
    • Extended to query across collections of documents
    • Not designed as a database query language
  – XQuery
    • Still under development

• Indices
  – What kind of indices are needed?
  – Along which of XML’s axes?
    • child, descendant, parent, ancestor, following-sibling, preceding-sibling, following, preceding, attribute, namespace

• Updates
  – Currently hardly supported:
    • Retrieve complete document, change, return to database
Conclusions

- Very large amounts of XML data and documents in the Internet -> Management?
- DBS only good for well-structured data
- XML & DBS for semi-structured data
  - 3 kinds of XDBMSs
  - Performance problems:
    XML workload <-> DBS workload
  - Multi-layer hierarchical architectures