Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 11 - XML

XML Origin and Usages

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language, not a database language
  - Documents have tags giving extra information about sections of the document
  - For example:
    - <title> XML </title>
    - <slide> XML Origin and Usages </slide>
- Derived from SGML (Standard Generalized Markup Language)
  - standard for document description
    - enables document interchange in publishing, office, engineering, ...
  - main idea: separate form from structure
- XML is simpler to use than SGML
  - roughly 20% complexity achieves 80% functionality

XML Origin and Usages (cont.)

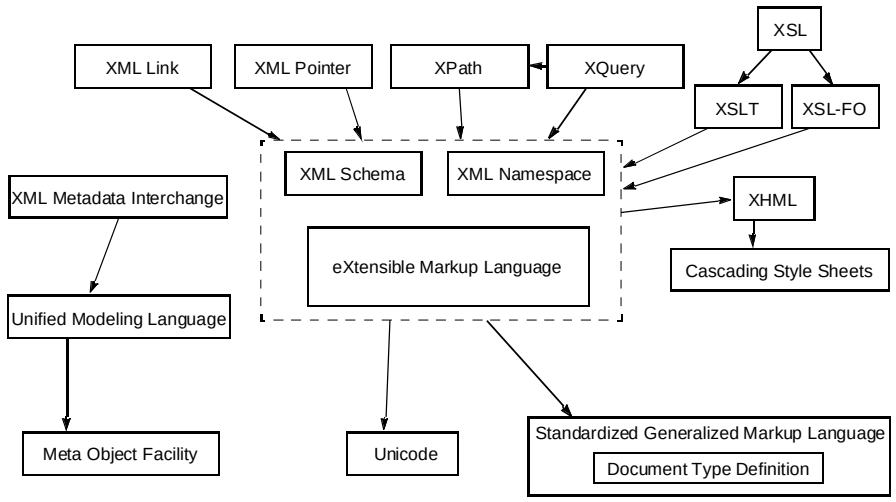
- XML documents are to some extent self-documenting
  - Tags can be used as metadata
  - Example

```
<bank>
    <account>
        <account-number> A-101     </account-number>
        <branch-name>    Downtown </branch-name>
        <balance>        500       </balance>
    </account>
    <depositor>
        <account-number> A-101     </account-number>
        <customer-name> Johnson </customer-name>
    </depositor>
</bank>
```


Forces Driving XML

- Document Processing
  - Goal: use document in various, evolving systems
  - structure – content – layout
  - grammer: markup vocabulary for mixed content
- Data Bases and Data Exchange
  - Goal: data independence
  - structured, typed data – schema-driven – integrity constraints
- Semi-structured Data and Information Integration
  - Goal: integrate autonomous data sources
  - data source schema not known in detail – schemata are dynamic
  - schema might be revealed through analysis only after data processing

XML Language Specifications



Describing XML Data: Basics - Elements

- **Tag:** label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly **nested**
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Every document must have a single top-level element
- Mixture of text with sub-elements is legal in XML
  - Example:

```
<account>
   This account is seldom used any more.
    <account-number> A-102</account-number>
    <branch-name> Perryridge</branch-name>
    <balance>400 </balance>
</account>
```

  - Useful for document markup, but discouraged for data representation

Describing XML Data: Attributes

- **Attributes:** can be used to describe elements
- Elements can have attributes

```
<account acct-type = "checking" >
        <account-number> A-102 </account-number>
        <branch-name> Perryridge </branch-name>
        <balance> 400 </balance>
</account>
```

- Attributes are specified by *name=value* pairs inside the starting tag of an element
- Attribute names must be unique within the element

```
<account  acct-type = "checking"  monthly-fee="5">
```


IDs and IDREFs

- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct
  ➔ ID attribute (value) is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document
- An attribute of type IDREFS contains a set of (0 or more) ID values. Each ID value must contain the ID value of an element in the same document
- IDs and IDREFs are untyped, unfortunately
  - Example below: The *owners* attribute of an account may contain a reference to another account, which is meaningless;
    owners attribute should ideally be constrained to refer to customer elements

XML data with ID and IDREF attributes

```
<bank-2>
    <account account-number="A-401" owners="C100 C102">
        <branch-name> Downtown </branch-name>
        <balance>500 </balance>
    </account>
        . . .
    <customer customer-id="C100" accounts="A-401">
        <customer-name>Joe</customer-name>
        <customer-street>Monroe</customer-street>
        <customer-city>Madison</customer-city>
    </customer>
    <customer customer-id="C102" accounts="A-401 A-402">
        <customer-name> Mary</customer-name>
        <customer-street> Erin</customer-street>
        <customer-city> Newark </customer-city>
    </customer>
</bank-2>
```


XML Document Schema

- Metadata and database schemas constrain what information can be stored, and the data types of stored values
- Metadata are very important for data exchange
  - Guarantees automatic and correct data interpretation
- XML documents are not required to have associated metadata/schema
  - only need to be **well-formed** (i.e., follow generic syntax rules)
- Two mechanisms for specifying XML schema
  - **Document Type Definition (DTD)**
  - **XML Schema**
- Documents may required to be **valid** w.r.t. an XML schema

Describing XML Data: DTD

- Type and structure of an XML document can be specified using a DTD
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Element Specification in DTD

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)
- Example

  <! ELEMENT depositor (customer-name account-number)>
  <! ELEMENT customer-name(#PCDATA)>
  <! ELEMENT account-number (#PCDATA)>
- Subelement specification may have regular expressions

  <!ELEMENT bank ((account | customer | depositor)+)>
  - Notation:
    - "|" - alternatives
    - "?" - 0 or 1 occurrence
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences

Example: Bank DTD

```
<!DOCTYPE bank-2[
    <!ELEMENT account (branch-name, balance)>
    <!ATTLIST account
            account-number   ID        #REQUIRED
            owners           IDREFS   #REQUIRED>
     <!ELEMENT customer(customer-name, customer-street,
                                                customer-city)>
     <!ATTLIST customer
            customer-id      ID          #REQUIRED
            accounts         IDREFS      #REQUIRED>
      ... declarations for branch, balance, customer-name,
                        customer-street and customer-city
]>
```


Describing XML Data: XML Schema

- XML Schema is closer to the general understanding of a (database) schema
- XML Schema supports
  - Typing of values
    - E.g. integer, string, etc
  - Constraints on min/max values
  - Typed references
  - User defined types
  - Specified in XML syntax (unlike DTDs)
  - Integrated with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance ..
- BUT: significantly more complicated than DTDs

XML Schema Structures

- **Datatypes (Part 2)**
  Describes Types of scalar (leaf) values
- **Structures (Part 1)**
  Describes types of complex values (attributes, elements)
  - Regular tree grammars
    repetition, optionality, choice recursion
- **Integrity constraints**
  Functional (keys) & inclusion dependencies (foreign keys)
- **Subtyping (similar to OO models)**
  Describes inheritance relationships between types
- **Supports schema reuse**

XML Schema Structures (cont.)

- Elements : tag name & simple or complex type

```
<xs:element name="sponsor" type="xsd:string"/>
<xs:element name="action" type="Action"/>
```

- Attributes : tag name & simple type

```
<xs:attribute name="date" type="xsd:date"/>
```

- Complex types

```
<xs:complexType name="Action">
    <xs:sequence>
        <xs:elemref name ="action-date"/>
        <xs:elemref name ="action-desc"/>
    </xs:sequence>
</xs:complexType>
```

XML Schema Structures (cont.)

- Sequence

```
<xs:sequence>
    <xs:element name="congress" type=xsd:string"/>
    <xs:element name="session" type=xsd:string"/>
</xs:sequence>
```

- Choice

```
<xs:choice>
    <xs:element name="author" type="PersonName"/>
    <xs:element name="editor" type="PersonName"/>
</xs:choice>
```

- Repetition

```
<xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs element name ="section" type="Section"/>
</xs:sequence>
```


Namespaces

- A single XML document may contain elements and attributes defined for and used by multiple software modules
  - Motivated by modularization considerations, for example
- Name collisions have to be avoided
- Example:
  - A **Book** XSD contains a Title element for the title of a book
  - A **Person** XSD contains a Title element for an honorary title of a person
  - A **BookOrder** XSD reference both XSDs
- Namespaces specifies how to construct universally unique names

XML Schema Version of Bank DTD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.banks.org"
            xmlns ="http://www.banks.org" >
<xsd:element name="bank" type="BankType"/>
<xsd:element name="account">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="account-number" type="xsd:string"/>
            <xsd:element name="branch-name"    type="xsd:string"/>
            <xsd:element name="balance"        type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>                  ..... definitions of customer and depositor ....
<xsd:complexType name="BankType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element ref="account"/>
        <xsd:element ref="customer"/>
        <xsd:element ref="depositor"/>
    </xsd:choice>
</xsd:complexType>
</xsd:schema>
```


XML Document Using Bank Schema

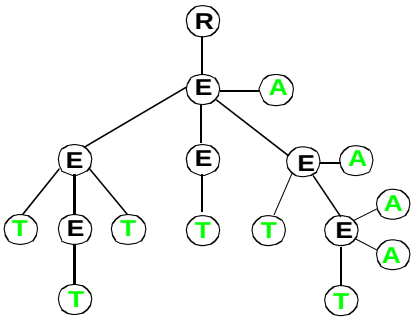
```
<bank   xmlns="http://www.banks.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.banks.org  Bank.xsd">
    <account>
        <account-number> ... </account-number>
        <branch-name> ... </branch-name>
        <balance> ... </balance>
    </account>
    ...
</bank>
```

Processing XML Data

- Querying XML data
- Translation of information from one XML schema to another
- Standard XML querying/translation languages
  - **XPath**
    - Simple language consisting of path expressions
  - **XSLT**
    - Simple language designed for translation from XML to XML and XML to HTML
  - **XQuery**
    - An XML query language with a rich set of features
    - XQuery builds on experience with existing query languages: **XPath**, **Quilt**, XQL, XML-QL, Lorel, YATL, **SQL**, **OQL**, ...

Tree Model of XML Data

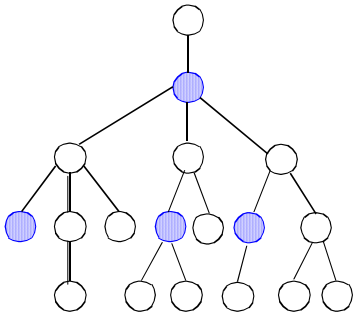
- Query and transformation languages are based on a **tree model** of XML data
- An XML document is modeled as a tree, with **nodes** corresponding to elements and attributes



- Several types of nodes:
  - Root, Element, Attribute,Text, Namespace, Comment, Processing Instruction

Processing XML Data: XPath

- XPath is used to address (select) parts of documents using path expressions
- XPath data model refers to a document as a tree of nodes
- An Xpath expression maps a node (the context node) into a set of nodes
- A path expression consists of one or more steps separated by "/"
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
  - E.g.: /bank-2/customer/customer-name evaluated on the bank-2 data returns
    - <customer-name> Joe </ customer-name>
    - < customer- name> Mary </ customer-name>
  - E.g.:/bank-2/customer/cust-name/text( ) returns the same names, but without the enclosing tags



XPath (cont.)

- The initial "/" denotes root of the document (above the top-level tag)
- In general, a step has three parts:
  - The ***axis*** (direction of movement: child, descendant, parent, ancestor, following, preceding, attribute, … - 13 axes in all -)
  - A ***node test*** (type and/or name of qualifying nodes)
  - Some ***predicates*** (refine the set of qualifying nodes)
- Path expressions are evaluated left to right
  - Each step operates on the set of instances produced by the previous step
- Selection predicates may follow any step in a path, in [ ]
  - E.g. /bank-2/account[balance > 400]
    - returns account elements with a balance value greater than 400
    - /bank-2/account[balance] returns account elements containing a balance subelement
- Attributes are accessed using "@"
  - E.g. /bank-2/account[balance > 400]/@account-number
    - returns the account numbers of those accounts with balance > 400
  - IDREF attributes are not dereferenced automatically (more on this later)

XPath (cont.)

- The following examples use XPath abbreviated notation:
  - Find the first item of every list that is under the context node
    .//list/item[1]
  - Find the "lang" attribute of the parent of the context node
    ../@lang
  - Find the last paragraph-child of the context node
    para[last()]
  - Find all warning elements that are inside instruction elements
    //instruction//warning
  - Find all elements that have an ID attribute
    //*[@ID]
  - Find names of customers who have an order with today's date
    //customer [order/date = today ( )] / name

- **XPath expressions use a notation similar to paths in a file system:**

/	means "child" or "root"
//	means "descendant"
.	means "self"
..	means "parent"
*	means "any"
@	means "attribute"

XPath (cont.): Summary

- Strengths:
  - Compact and powerful syntax for navigating a tree, but not as powerfull as a regular-expression language
  - Recognized and accepted in XML community
  - Used in XML-related applications such as XPointer

- Limitations:
  - Operates on one document (no joins)
  - No grouping or aggregation
  - No facility for generating new output structures

Transforming XML Data: XSLT

- A **stylesheet** stores formatting options for a document, usually separately from document
  - E.g. HTML style sheet may specify font colors and sizes for headings, etc.
- The **XML Stylesheet Language (XSL)** was originally designed for generating HTML from XML
- XSLT is a general-purpose transformation language
  - Can translate XML to XML, and XML to HTML
- XSLT transformations are expressed using rules called **templates**
  - Templates combine selection using XPath with construction of results

Understanding A Template

- Most templates have the following form:

```
<xsl:template match="emphasis">
  <i><xsl:apply-templates/></i>
</xsl:template>
```

- The whole <xsl:template> element is a **template**
- The **match pattern** determines where this template applies
  - Xpath pattern
- **Literal result element(s)** come from non-XSL namespace(s)
- XSLT elements come from the XSL namespace

XQuery

- XQuery is a general purpose query language for XML data
- Currently being standardized by the World Wide Web Consortium (W3C)
- XQuery is derived from
  - the **Quilt** ("Quilt" refers both to the origin of the language and to its use in "knitting " together heterogeneous data sources) query language, which itself borrows from
  - **XPath**: a concise language for navigating in trees
  - **XML-QL**: a powerful language for generating new structures
  - **SQL**: a database language based on a series of keyword-clauses: SELECT - FROM – WHERE
  - **OQL**: a functional language in which many kinds of expressions can be nested with full generality

XQuery Data Model

- Builds on a tree-based model, but extends it to support sequences of items
  - A value is an ordered **sequence** of zero or more **items**
    - cannot be nested – all operations on sequences automatically "flatten" sequences
      - no distinction between an item and a sequence of length 1
    - can contain heterogenous values, are ordered, can be empty
    - may contain duplicate nodes (see below)
  - An **item** is a **node** or an **atomic value**
  - **Atomic values** are typed values
    - XML Schema simple types
  - There are seven kinds of **nodes** (see tree-based model)
    - nodes have an identity
    - each node has a typed value
      - sequence of atomic values
      - type may be unknown (anySimpleType)
    - element and attribute nodes have a type annotation
      - generated by validating the node
    - document order of nodes
- Closure property
  - XQuery expressions operate on/produce instances of the XQuery Data Model

XQuery – Main Constituents

- Path expressions
  - Inherited from XPath 1.0
  - An XPath expression maps a node (the context node) into a set of nodes
- Element constructors
  - To construct an element with a known name and content, use XML-like syntax:

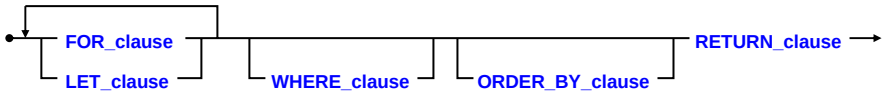
```
<book isbn = "12345">
   <title>Huckleberry Finn</title>
</book>
```

  - If the content of an element or attribute must be computed, use a nested expression enclosed in { }

```
<book isbn = "{$x}">
   {$b/title }
</book>
```

- FLWOR - Expressions

XQuery: The General Syntax Expression FLWOR



- FOR clause, LET clause generate list of tuples of bound variables (order preserving) by
  - iterating over a set of nodes (possibly specified by an XPath expression), or
  - binding a variable to the result of an expression
- WHERE clause applies a predicate to filter the tuples produced by FOR/LET
- ORDER BY clause imposes order on the surviving tuples
- RETURN clause is executed for each surviving tuple, generates ordered list of outputs
- Associations to SQL query expressions
  for ⇔ SQL from
  where ⇔ SQL where
  order by ⇔ SQL order by
  return ⇔ SQL select
  let allows temporary variables, and has no equivalent in SQL

FLWOR - Examples

- Simple FLWR expression in XQuery
  - Find all accounts with balance > 400, with each result enclosed in an <account-number> .. </account-number> tag

```
for     $x in /bank-2/account
let     $acctno := $x/@account-number
where $x/balance > 400
return <account-number> {$acctno} </account-number>
```

- Let and Where clause not really needed in this query, and selection can be done in XPath.
  - Query can be written as:

```
for       $x in /bank-2/account[balance>400]
return <account-number> {$x/@account-number}
                                    </account-number>
```


Nesting of Expressions

- Here: nesting inside the return clause
  - Example: inversion of a hierarchy

```
<book>
   <title>
   <author>
   <author>
</book>
<book>
   <title>
   <author>
   <author>
</book>
```

```
FOR $a IN distinct-values(//author)
ORDER BY $a/name
RETURN
    <author>
        <name> { $a/text() } </name>
        { FOR $b IN //book[author = $a]
            RETURN $b/title }
    </author>
```

```
<author>
   <name>
   <title>
   <title>
</author>
<author>
   <name>
   <title>
   <title>
</author>
```

XQuery: Joins

- Joins are specified in a manner very similar to SQL

```
for   $a  in  /bank/account,
      $c  in  /bank/customer,
      $d  in  /bank/depositor
where $a/account-number = $d/account-number
   and $c/customer-name = $d/customer-name
return <cust-acct>{ $c $a }</cust-acct>
```

- The same query can be expressed with the selections specified as XPath selections:

```
for   $a in /bank/account
      $c in /bank/customer
      $d in /bank/depositor[
                  account-number =$a/account-number and
                  customer-name = $c/customer-name]
return <cust-acct>{ $c $a }</cust-acct>
```


XQuery - Status

- Some Recent Enhancements
  - Complete Specification of XQuery Functions and Operators
  - Joint XQuery/XPath data model
  - Type checking model
    - static vs. dynamic type checking as an option
    - with/without schema information
  - A lot of problems fixed
  - Current status: working draft under public review
    - fairly close to becoming a w3c recommendation
- Ongoing and Future Work
  - Full-text support
  - Insert, Update, Delete
  - View definitions, DDL
  - Host language bindings, APIs
    - JSR 225: XQuery API for JavaTM (XQJ)
    - problem to overcome: tradtional XML processing API is based on well-defined documents

SQL/XML

- Subproject of SQL standard
  - Part 14 “XML-related Specifications” of upcoming SQL 2003
- Goal: standardization of interaction/integration of SQL and XML
  - how to represent SQL data (tables, results, ...) in XML (and vice versa)
  - how to map SQL metadata (information schema) to XML schema (and vice versa)
  - ...
- Potential areas of use
  - "present" SQL data as XML
  - integration of XML data into SQL data bases
  - use XML for SQL data interchange
  - XML views over relational
    - possible foundation for XQuery

SQL/XML Features

- SQL/XML includes the following:
  - XML data type
    - Enables storage and retrieval of XML documents as typed values
  - Host language bindings for values of XML type
  - XML “publishing functions”
  - Mapping SQL Tables to XML Documents
  - Mapping SQL identifiers to XML Names and vice versa
  - Mapping SQL data types to XML Schema data types
  - Mapping SQL data values to XML

XML Publishing Functions - Example

```
SELECT   XMLELEMENT ( NAME "Department",
                      XMLATTRIBUTES ( e.dept AS "name" ),
                      XMLAGG (XMLELEMENT (NAME "emp", e.lname))
                    ) AS "dept_list",
         COUNT(*) AS "dept_count"
FROM     employees e
GROUP BY dept ;

==>
```

dept_list	dept_count
<Department name="Accounting"> <emp>Yates</emp> <emp>Smith</emp> </Department>	2
<Department name="Shipping"> <emp>Oppenheimer</emp> <emp>Martin</emp> </Department>	2

Application Programming with XML

- Application needs to work with XML data/document
  - **Parsing** XML to extract relevant information
  - Produce XML
    - Write character data
    - Build internal XML document representation and **Serialize** it
  - Simple API for XML (SAX)
    - "Push" parsing (event-based parsing)
    - Parser sends notifications to application about the type of document pieces it encounters
    - Notifications are sent in "reading order" as they appear in the document
    - Preferred for large documents (high memory efficiency)
  - Document Object Model (DOM)
    - "One-step" parsing
    - Generates in-memory representation of the document (parse tree)
    - DOM specifies the types of parse tree objects, their properties and operations
      - Independent of programming language (uses IDL)
    - Bindings available to specific programming languages (e.g., Java)

XML Advantages for Integration

- Integrates data and meta-data (tags)
  - Self-describing
- XMLSchema, Namespaces
  - Defining valid document structure
  - Integrating heterogenous terminology and structures
- XML can be validated against schema (xsd, dtd) outside the application
- Many technologies exist for processing, transforming, querying XML documents
  - DOM, SAX, XSLT, XPath, XQuery
- XML processing can help handle schema heterogenity, schema evolution
  - Focus on known element tags, attributes, namespaces ...
  - Powerful filter and transformation capabilities
- XML is independent of platforms, middleware, databases, applications ...

XML Support for DBMS: Direction