Introduction to XML

Prof. Weining Zhang

What is XML

- XML: Extensible Markup Language
  - Defined by the WWW Consortium (W3C)
  - Derived from SGML (Standard Generalized Markup Language), but is simpler to implement than SGML
- XML describes document/data format
  - Self-describing (structural & semantic)
  - Simple (minimum rules)
  - Flexible (user defines tag, semi-structured)

Why Use XML

- Data interoperability
  - Textual form independent of creator apps
  - Easier to be processed other software
- Represent meta-data
  - Describing resources & data semantics
- Data exchange
  - As a common format
  - Many tools for format conversion & data processing
- Application integration
  - Components can communicate using XML-based communication protocols.

Example: XML Data

- Financial information of a bank
  <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>

Syntax of XML Data

- A few syntactic rules defined in XML 1.0
  - Document
  - Element:
    - Tags <A> … </A>
    - Element nesting (well-formed)
  - Attribute
  - DTD: document type definition
  - Processing instruction
  - Namespace
  - Comment, etc.

Element Nesting

- Elements must be properly nested
  - Proper nesting
    - <account> … <balance> … </balance> <account>
  - Improper nesting
    - <account> … <balance> … <account>
  - 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
Example of Nested Elements

```
<bank-1>
  <customer>
    <customer-name> Hayes </customer-name>
    <customer-street> Main </customer-street>
    <customer-city> Harrison </customer-city>
  </customer>
  <account>
    <account-number> A-102 </account-number>
    <branch-name> Perryridge </branch-name>
    <balance> 400 </balance>
  </account>
  <account>
    <account-number> A-101 </account-number>
  </account>
  ...
</bank-1>
```
Naming Tags

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique-name:element-name

Namespaces

- A short prefix of a tag name to make it unique

```xml
<data version="1.0" encoding="UTF-8">
  <bank Xmins:FB="http://www.FirstBank.com">
    ...<FB:branch>
      <FB:branchname>Downtown</FB:branchname>
      <FB:branchcity>Downtown</FB:branchcity>
    </FB:branch>
  </bank>
</data>
```

Document Type Definition (DTD)

- Used to specify the type of an XML document
- DTD constraints structure of XML data
  - 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 are 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

```xml
<!ELEMENT depositor (customer-name
  account-number)>
<!ELEMENT customer-name(#PCDATA)>
<!ELEMENT account-number (#PCDATA)>
```

Element Specification in DTD

- Subelement specification may have regular expressions

```xml
<!ELEMENT bank ( ( account | customer | depositor)+ )>
```

Notation:
- `|` - alternatives
- `*` - 1 or more occurrences
- `**` - 0 or more occurrences

A Bank DTD

```xml
<!DOCTYPE bank [ 
  <!ELEMENT bank ( ( account | customer | depositor)+ )>
  <!ELEMENT account (account-number branch-name balance)>
  <!ELEMENT customer (customer-name customer-street customer-city)>
  <!ELEMENT depositor (customer-name account-number)>
  <!ELEMENT account-number (#PCDATA)>
  <!ELEMENT branch-name (#PCDATA)>
  <!ELEMENT balance (#PCDATA)>
  <!ELEMENT customer-name (#PCDATA)>
  <!ELEMENT customer-street (#PCDATA)>
  <!ELEMENT customer-city (#PCDATA)>
]>```
Attribute Specification in DTD

- For each attribute, specify
  - Name
  - Type of attribute
    - CDATA
    - ID (identifier) or IDREF (ID reference) or
      IDREFS (multiple IDREFs)
      - more on this later
  - Whether
    - mandatory (#REQUIRED)
    - has a default value (value),
    - or neither (#IMPLIED)

Examples

```xml
<ATTLIST account
  acct-type CDATA "checking">
<ATTLIST customer
  customer-id ID  # REQUIRED
  accounts IDREFS # REQUIRED>
```

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
  - Thus the 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

Bank DTD with Attributes

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

XML data with ID and IDREF attributes

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

Limitations of DTDs

- No typing of text elements and attributes
  - All values are strings, no integers, reals, etc.
- Difficult to specify unordered sets of subelements
  - Order is usually irrelevant in databases
  - (A | B)* allows specification of an unordered set, but
    - Cannot ensure that each of A and B occurs only once
- IDs and IDREFs are untyped
  - 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
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
  - Element nodes have children nodes, which can be attributes or subelements
  - Text in an element is modeled as a text node child of the element
  - Children of a node are ordered according to their order in the XML document
  - Element and attribute nodes have a single parent, which is an element node

Tree Model of XML Data
- We use the terminology of nodes, children, parent, siblings, ancestor, descendant, etc., which should be interpreted in the above tree model of XML data.

Infoset: XML Information Set
- Describing information available in a well-formed XML document
- Contains information items
  - Document info item, element info item, ...
  - Info item contains properties, e.g., element info item has a namespace URL, a local name, a list of pointers to info items of sub-elements, a set of pointers to info items of attributes, ...
  - Info items & their properties are divided into core & peripheral groups
- Needed by application programs

SAX: Simple API for XML
- Event-driven
  - Implement an XML parser that generates events & provide info items to application
- Users implement event handlers, say document handler, exception handler
  - Methods to handle element start, element ends, exceptions, etc.
- Helper methods
  - Control parser behavior
  - Get info items (thru XMLReader)

DOM: Document Object Model
- An object-oriented API of XML documents.
  - Objects are tree nodes:
    - Document, Element, Attribute, etc. are modeled as objects.
  - XML data is parsed into a tree structure of nested objects (represent infoset)
  - Methods: Each object has methods to access & modify its components. Classes have methods to create instances.
  - Variety of functions are provided for traversing the DOM tree
DOM

- Java DOM API provides Node class with methods
  - `getParentNode()`, `getFirstChild()`,
  - `getNextSibling()`, `getAttribute()`,
  - `getData()` (for text node),
  - `getElementsByTagName()` …

- Also provides functions for updating DOM tree