Data Interchange & XML

Henning Schulzrinne
Advanced Programming
(with material by Suhit Gupta)

Data interchange

- Unix-style files
- serialization (marshalling): convert structured data into linear stream of bytes
- for files and across networks – used to be separate
- part of RPC:
  - Sun RPC
  - Corba
  - Java
- ASN.1
- XML

Data interchange

- Work across different platforms
  - byte order, floating point, character sets, ...
  - convert to destination platform
  - common intermediate representation
- Efficient

Structured files

- Older OS had records and punch card columns
- Unix model: lines separated into columns (tabs, spaces, commas)
- Also: csv in Excel and kin
- Sometimes # for comments

Structured files

- Examples:
  - /etc/passwd

    hgs:8D6uxb.jefyxz:5815:92:Henning G. Schulzrinne:/home/hgs:/bin/tcsh
  - files for sort
  - /etc/services

```
time            37/tcp          timeserver
time            37/udp          timeserver
rlp             39/tcp          resource        # resource location
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
nameserver      42/udp          name            # IEN 116
```


XML

- IBM: SGML (structured general markup language) ➔ HTML (hypertext mark-up) ➔ XML ➔ XHTML
- idea: label content instead of presentation
- subset of SGML
- "documents", but really structured (tree) data objects
- human readable, but not necessarily terse

XML

- entities = storage units containing
  - parsed data: characters + markup
  - unparsed data
- markup = start tags, end tags, entity references, ...
- starts with document type declaration:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
```

- comments: <!– comment -->
- verbatim: `<![CDATA[<greeting>Hello, world!</greeting>]]>`

XML

- Document type definition (DTD) defines structure of XML

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE greeting [
    <!ELEMENT greeting (#PCDATA)>
    <!ATTLIST poem xml:space (default|preserve)
  'preserve'>
  ]>
<greeting>Hello, world!</greeting>
```

- Other mechanisms: XSD (later)

Tags

- Tags and attributes organized into XML name spaces
- e.g., language attribute:

```
<p xml:lang="en">The quick brown over fox...</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
<l>Habe nun, ach! Philosophie,</l>
<l>Juristerei, und Medizin</l>
<l>und leider auch Theologie</l>
<l>durchaus studiert mit heißem Bemüh'n.</l>
</sp>
```


XML – special characters and binary data

- &# and &#x introduce ISO 10646 characters, e.g., `&#x3C;` for <
- binary data is more painful:
  - base64 encoding (6 bits per character)
    - `hello` → `aGVsbG8K`
  - MIME multipart
  - external reference

XML binary: MIME

```
Content-Type: multipart/related; boundary=--xxxxxxxxxx;

--xxxxxxxxxx
Content-Type: text/xml
Content-ID: Contents

<?xml version="1.0" ?>
  <objectDef uid="?">
  <property><name>Width</name>
  <value><i4>1024</i4></value></property>
  <property><name>Height</name>
  <value><i4>1024</i4></value></property>
  <property><name>Pixels</name>
  <value><stream href=cid:Pixels /></value></property>

 --xxxxxxxxxx
Content-Type: application/binary
Content-Transfer-Encoding: Little-Endian
Content-ID: Pixels
Content-Length: 524288

....binary data here...
--xxxxxxxxxx
```


XML schema definition (XSD)

- Define semantic structure of documents, with typing (≠ DTD)

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
<shipTo country="US">
      <name>Alice Smith</name><street>123 Maple Street</street>
      <city>Mill Valley</city><state>CA</state><zip>90952</zip>
</shipTo>
<billTo country="US"> <name>Robert Smith</name> <street>8 Oak Avenue</street> <city>Old Town</city>
      <state>PA</state> <zip>95819</zip> </billTo>
<comment>Hurry, my lawn is going wild!</comment>
<Items>
<Item partNum="872-AA">
      <productName>Lawnmower</productName> <quantity>1</quantity> <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
</item><item partNum="926-AA">
      <productName>Baby Monitor</productName> <quantity>1</quantity> <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate> </Item>
</Items>
</purchaseOrder>
```

XML schema

- complex types contain other elements
- simple types contain numbers, strings, dates, ... but no subelements
- built-in simple types: string, token, byte, integer, float, double, boolean, time, dateTime, duration, anyURI, language, NMTOKEN, ...

XML schema example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <xsd:annotation> <xsd:documentation
    xml:lang="en"> Purchase order schema for Example.com. Copyright 2000 Example.com. All rights
    reserved. </xsd:documentation> </xsd:annotation>
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="PurchaseOrderType">
    <xsd:sequence> <xsd:element name="shipTo" type="USAddress"/>
            <xsd:element name="billTo" type="USAddress"/>
            <xsd:element ref="comment" minOccurs="0"/>
            <xsd:element name="items" type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
<xsd:complexType name="USAddress">
<xsd:sequence> <xsd:element name="name" type="xsd:string"/>
<xsd:element name="street" type="xsd:string"/>
<xsd:element name="city" type="xsd:string"/>
<xsd:element name="state" type="xsd:string"/>
<xsd:element name="zip" type="xsd:decimal"/>
</xsd:sequence> <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```


XML schema, cont'd.

```
<xsd:complexType name="Items">
    <xsd:sequence>
     <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
     <xsd:complexType>
          <xsd:sequence>
               <xsd:element name="productName" type="xsd:string"/>
               <xsd:element name="quantity">
                    <xsd:simpleType>
                         <xsd:restriction base="xsd:positiveInteger">
                         <xsd:maxExclusive value="100"/>
                    </xsd:restriction>
                    </xsd:simpleType>
               </xsd:element>
               <xsd:element name="USPrice" type="xsd:decimal"/>
               <xsd:element ref="comment" minOccurs="0"/>
               <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
    </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```


XML – values

- <xsd:simpleType name="MyInteger" base="xsd:integer">
  <xsd:minInclusive value="1"/>
  <xsd:maxInclusive value="99"/>
  </xsd:simpleType>

- <xsd:simpleType name="Sku" base="xsd:string">
  <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:simpleType>

XML – minOccurs/maxOccurs

Adding Attributes to the Inline Type Definition

```
<xsd:element name="Item" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType base="xsd:positiveInteger">
              <xsd:maxExclusive value="100"/>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="price" type="xsd:decimal"/>
          <xsd:element ref="comment" minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
          <xsd:attribute name="partNum" type="Sku"/>
          <xsd:attribute name="weight" type="xsd:decimal"/>
          <xsd:attribute name="shipBy">
            <xsd:simpleType base="string">
              <xsd:enumeration value="air"/>
              <xsd:enumeration value="land"/>
              <xsd:enumeration value="any"/>
            </xsd:simpleType>
          </xsd:attribute>
        </xsd:complexType>
</xsd:element>
```


XML – max/min

- You can specify max and min values
- By combining and nesting the various groups provided by XML Schema, and by setting the values of minOccurs and maxOccurs, it is possible to represent any content model expressible with an XML.

XML – Attribute groups

Adding Attributes Using an Attribute Group

```
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
   <xsd:element name="productName" type="xsd:string"/>
   <xsd:element name="quantity">
    <xsd:simpleType base="xsd:positiveInteger">
.
.
.
.
<xsd:attributeGroup name="ItemDelivery">
 <xsd:attribute name="partNum" type="Sku"/>
 <xsd:attribute name="weight"  type="xsd:decimal"/>
 <xsd:attribute name="shipBy">
  <xsd:simpleType base="xsd:string">
    <xsd:enumeration value="air"/>
    <xsd:enumeration value="land"/>
    <xsd:enumeration value="any"/>
  </xsd:simpleType>
 </xsd:attribute>
</xsd:attributeGroup>
```


XML attribute groups

- Using an attribute group in this way can improve the readability of schema, and facilitates updating schema because an attribute group can be defined and edited in one place and referenced in multiple definitions and declarations.
- These characteristics of attribute groups make them similar to parameter entities in XML

XML – Choice and Sequence

Nested Choice and Sequence Groups

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:choice>
    <xsd:group   ref="shipAndBill" />
    <xsd:element name="singleAddress" type="Address" />
  </xsd:choice>
  <xsd:element   ref="comment"    minOccurs="0" />
  <xsd:element   name="items"     type="Items" />
  <xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>

<xsd:group name="shipAndBill">
  <xsd:sequence>
    <xsd:element name="shipTo" type="Address" />
    <xsd:element name="billTo" type="Address" />
  </xsd:sequence>
</xsd:group>
```


XML – Choice and Sequence

- A choice group element allows only one of its children to appear in an instance.
- Un-named groups of elements can be constrained so that only one of the elements may appear in an instance.
- Alternatively, they can also be defined, and along with elements in named groups, they can be constrained to appear in the same order (sequence) as they are declared.

XML – DOM

- "The XML Document Object Model (DOM) is a programming interface for XML documents. It defines the way an XML document can be accessed and manipulated."
- DOM data structures used with XML are essentially trees.
- http://www.w3schools.com/dom/dom_intro.asp

Node properties

Name	Description
attributes	Returns a NamedNodeMap containing all attributes for this node
childNodes	Returns a NodeList containing all the child nodes for this node
firstChild	Returns the first child node for this node
lastChild	Returns the last child node for this node
nextSibling	Returns the next sibling node. Two nodes are siblings if they have the same parent node
nodeName	Returns the nodeName, depending on the type
nodeType	Returns the nodeType as a number
nodeValue	Returns, or sets, the value of this node, depending on the type
ownerDocument	Returns the root node of the document
parentNode	Returns the parent node for this node
previousSibling	Returns the previous sibling node. Two nodes are siblings if they have the same parent node

Node methods

Name	Description
appendChild(newChild)	Appends the node newChild at the end of the child nodes for this node
cloneNode(boolean)	Returns an exact clone of this node. If the boolean value is set to true, the cloned node contains all the child nodes as well
hasChildNodes()	Returns true if this node has any child nodes
insertBefore(newNode, refNode)	Inserts a new node, newNode, before the existing node, refNode
removeChild(nodeName)	Removes the specified node, nodeName
replaceChild(newNode, oldNode)	Replaces the oldNode, with the newNode

XML – DOM

- DOM validate XML
  - http://www.w3schools.com/dom/dom_validate.asp
- Some DOM resources
  - http://www.w3schools.com/dom/dom_resources.asp

Programming with XML

- Don't want to write a new parser for each application
- Two APIs for C++ and Java:
  - SAX – events as parsed
  - DOM – object model (build tree & query)
- Both implemented in Xerces (Apache)
- Also, more specific implementations for XML RPC

Sample Code

```
<html>
<body>
<script type="text/vbscript">

txt="<h1>Traversing the node tree</h1>"
document.write(txt)

set xmlDoc=CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")

for each x in xmlDoc.documentElement.childNodes
  document.write("<b>" & x.nodename & "</b>")
  document.write(": ")
  document.write(x.text)
  document.write("<br>")
next

</script>
</body>
</html>
```


Sample code

```
public class OtherParser implements Parser {

   private Document doc;

   public OtherParser(Document arg) {
      doc = arg;
   }

   public SourceTuple parseDoc() {
      String protocol=null;
      String url=null;
      int size=-1;
      String type=null;
      long created=-1;
      long last_mod=-1;
      String src=null;
      String opt[] = null;
      Element root;

      root = doc.getRootElement();
      try {
         created = root.getAttribute("createDate").getLongValue();
      } catch (Exception e) {}
      protocol = root.getChild("Protocol").getText();
      url = root.getChild("Name").getText();
      type = root.getChild("Type").getText();
      try {
         size = Integer.parseInt(root.getChild("Size").getText());
         last_mod = Long.parseLong(root.getChild("LastModified").getText());
      } catch (Exception e) {}
                                        System.err.println(url);
                                        return new SourceTuple(-1, protocol, url, size, type, created, last_mod, -
                                        1, opt);

                                           }
                                        }
```


SAX Java example - main

```
import java.io.FileReader;
import org.xml.sax.XMLReader;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
public class MySAXApp extends DefaultHandler {
   public static void main (String args[]) throws Exception {
    XMLReader xr = XMLReaderFactory.createXMLReader();
    MySAXApp handler = new MySAXApp();  xr.setContentHandler(handler);
    xr.setErrorHandler(handler);
    // Parse each file provided on the command line.
    for (int i = 0; i < args.length; i++) {
      FileReader r = new FileReader(args[i]);
      xr.parse(new InputSource(r));
    }
   }
   public MySAXApp () {
    super();
   }
}
```

Java SAX example - handlers

```
public void startDocument () {
   System.out.println("Start document");
}
public void endDocument () {
   System.out.println("End document");
}
public void startElement (String uri, String name, String qName,
   Attributes atts) {
   if ("".equals (uri))
     System.out.println("Start element: " + qName);
   else System.out.println("Start element: {" + uri + "}" + name);
}
public void endElement (String uri, String name, String qName) {
   if ("".equals (uri))
     System.out.println("End element: " + qName);
   else System.out.println("End element: {" + uri + "}" + name);
}
```


Java SAX example - characters

```
public void characters (char ch[], int start, int
  length) {
  System.out.print("Characters: \"");
  for (int i = start; i < start + length; i++) {
    switch (ch[i]) {
      case '\\': System.out.print("\\\\"); break;
      case '"': System.out.print("\\\""); break;
      case '\n': System.out.print("\\n"); break;
      case '\r': System.out.print("\\r"); break;
      case '\t': System.out.print("\\t"); break;
      default: System.out.print(ch[i]); break;
    }
  }
  System.out.print("\"\n");
}
```


SAX for Java – output

```
<?xml version="1.0"?>
<poem xmlns="http://www.megginson.com/ns/exp/poetry">
    <title>Roses are Red</title>
    <l>Roses are red,</l>
    <l>Violets are blue;</l>
</poem>
```

java -Dorg.xml.sax.driver=*com.example.xml.SAXDriver* MySAXApp roses.xml

```
Start document
Start element: {http://www.megginson.com/ns/exp/poetry}poem Characters: "\n"
Start element: {http://www.megginson.com/ns/exp/poetry}title
Characters: "Roses are Red"
End element: {http://www.megginson.com/ns/exp/poetry}title
Characters: "\n"
Start element: {http://www.megginson.com/ns/exp/poetry}l
Characters: "Roses are red,"
...
End element: {http://www.megginson.com/ns/exp/poetry}poem
End document
```


Sax for C++ -- handler example

```
void SAXPrintHandlers::startElement(const XMLCh* const name,
   AttributeList&  attributes)
{
   // The name has to be representable without any escapes
   fFormatter << XMLFormatter::NoEscapes << chOpenAngle <<
   name;
   unsigned int len = attributes.getLength();
   for (unsigned int index = 0; index < len; index++) {
        fFormatter << XMLFormatter::NoEscapes
                   << chSpace << attributes.getName(index)
                   << chEqual << chDoubleQuote
                   << XMLFormatter::AttrEscapes
                   << attributes.getValue(index)
                   << XMLFormatter::NoEscapes
                   << chDoubleQuote;
    }
    fFormatter << chCloseAngle;
}
```


DOM counting example (C++)

```
DOM_Document doc = parser->getDocument();
unsigned int elementCount =
  doc.getElementsByTagName("*").getLength();
// Print out stats collected and time taken.
cout << xmlFile << ": " << duration << " ms ("
     << elementCount << " elems)." << endl;

// delete the parser
delete parser;
// And call the termination method
XMLPlatformUtils::Terminate();
```


SOAP

- RPC mechanism:
  - XML + schema for request, response
  - HTTP and other transports

SOAP example

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
<env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
        env:actor=http://www.w3.org/2001/12/soap-envelope/actor/next
        env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
      <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
      </m:reservation>
      <n:passenger xmlns:n=http://mycompany.example.com/employees
  env:actor=http://www.w3.org/2001/12/soap-envelope/actor/next
    env:mustUnderstand="true">
    <n:name>John Q. Public</n:name>
    </n:passenger>
</env:Header>
    <env:Body>
          <p:itinerary
        xmlns:p="http://travelcompany.example.org/reservation/travel">
            <p:airportChoices> JFK LGA EWR </p:airportChoices>
        </p:itinerary>
    </env:Body>
</env:Envelope>
```


XML examples

- XHTML for hypertext markup
- MathML for mathematics in web pages
  - **$x^2 + 4x + 4 = 0$**
  - `<apply> <plus/> <apply> <power/> <ci>x</ci> <cn>2</cn> </apply> <apply> <times/> <cn>4</cn> <ci>x</ci> </apply> <cn>4</cn> </apply>`
- SVG for line graphics
- VoiceXML for voice browsers
- RDF for describing resources

XML: pros & cons

- ☺ Rich set of related languages:
  - XSL for transformation
  - XML Query for queries on XML documents
  - XSD for structure definition
- ☺ Lots of tools:
  - parser for C/C++, Java: Xerces
  - Tcl, Python, etc.
- ☺ Can be generated easily with text editors and printf
- ☺ Buzzword compliant
- ☹ Not space efficient
- ☹ Not well-suited for binary data
- ☹ Untyped data except with XSD