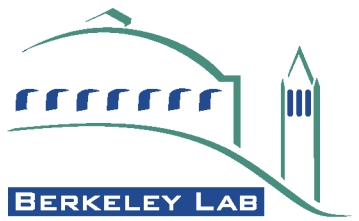
Concrete uses of XML in software development and data analysis.

Simon Patton
(L.B.N.L)



XML Overview

- **XML - eXtensible Markup Language.**
  - Derived from SGML (parent of HTML)
  - XHTML is HTML-like derivation
- **Text based meta-language.**
  - Used to defined grammar of content documents.
- **Becoming an industry standard for data description and exchange.**

IceCube Overview

- **Neutrino telescope based at the South Pole.**
- **Relatively small collaboration.**
  - Need to beg, borrow or steal as much technology as possible
- **Minimal legacy code or data from Amanda**
  - Amanda data in home-grown ASCII format
- **Core software is in Java**
  - Good selection of Java XML tools

XML Tools and their Standards

- **XML covered many useful operations.**
  - (W3C standards are *Highlighted*)
- **Parsing (*DOM* & SAX)**
- **Validation (*XML Schema* & DTD)**
- **Language Data Structures (JAXB)**
- **Transforms (*XSL/XSLT*)**
- **Searching (*XPath* & *XQuery*)**
- **Data Transfers (*SOAP*)**

Parsing

- **Example: From IceCube Engineering Data.**
  - Simplified here to be just the readout of an ATWD in a Digital Optical Module.
  - Start with complete prose description of content.
  - Create XML documents containing data
    - Example: `atwdExample.xml` (See next slide)
  - Can use Xerces to parse and read in file.

atwdExample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<daq:AtwdReadout
        xmlns:daq = "http://acme.lbl.gov/icecube/xml/daq"
        xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation = "http://acme.lbl.gov/icecube/xml/daq atwdReadout.xsd">
  <Atwd>
    <Channel number="0" bitsPerSample="8">
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    </Channel>
    <Channel number="1" bitsPerSample="8">
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0    55     0     0     0     0
    </Channel>
    <Channel number="2" bitsPerSample="8">
   67    71    72    68    73    69    71    70    77    71    72    70
   73    75    78    76    77    80    71    73    82    75    79    78
   76    81    75    85    81    86    82    86    81    84    79    80
   84   188     0     0     0     0     0     0     0     0     0     0
    </Channel>
    <Channel number="3">
  231  1010  1021   253   995  1021  1021  1021  1021   253   987  1021
  253   221  1000   253   229   982   253   219   211  1009  1021  1021
 1021   253   985  1021  1021  1021  1021  1021  1021  1021  1021  1021
  253    28    82    79    71    71    77    71    77    71    73    75
    </Channel>
  </Atwd>
</daq:AtwdReadout>
```

Parsing (2/2)

- **Two approaches:**
  - DOM - Document Object Model
    - Create hierarchical tree of XML document in memory.
    - Parsed all at once.
    - Resource hog(?)
  - SAX - Simple API for XML
    - Event based model that Requires code to keep track of context within XML document.
    - Can partially parse a document.

Validation

- **Check that an XML document follows the rules (grammar) set down in an XML Schema (or DTD).**
- **Can check items such as:**
  - Appearance of required elements.
  - Correct placement of elements in hierarchy.
  - Value of contents (and attributes) are allowed.
    - Example: `atwdReadout.xsd` (See next two slides).

atwdReadout.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://acme.lbl.gov/icecube/xml/daq"
    xmlns="http://acme.lbl.gov/icecube/xml/daq"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="AtwdReadout">
        <xs:complexType>
            <xs:sequence>
             <xs:element name="Atwd" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Channel" type="AtwdChannel" maxOccurs="4" minOccurs="4"/>
                    </xs:sequence>
                </xs:complexType>
                <xs:unique name="RequireAllChannels">
                    <xs:selector xpath="Channel"/>
                    <xs:field xpath="@number"/>
                </xs:unique>
            </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

(continued on next slide)

atwdReadout.xsd (2/2)

```
    <xs:simpleType name="SixteenBitsList">
        <xs:list itemType="xs:unsignedShort"/>
    </xs:simpleType>

    <xs:simpleType name="AtwdChannelData">
        <xs:restriction base="SixteenBitsList">
            <xs:length value="48"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="AtwdChannel">
        <xs:simpleContent>
            <xs:extension base="AtwdChannelData">
                <xs:attribute name="number">
                    <xs:simpleType>
                        <xs:restriction base="xs:nonNegativeInteger">
                            <xs:maxExclusive value="4"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute default="16" name="bitsPerSample">
                    <xs:simpleType>
                        <xs:restriction base="xs:nonNegativeInteger">
                            <xs:enumeration value="8"/>
                            <xs:enumeration value="16"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

</xs:schema>
```

Java Data Structures

- **JAXB - Java API for XML Binding.**
- **Start with an XML Schema and create java classes needed to represent it.**
- **Read and Write XML documents into and out of the classes.**
- **Still fairly new technology.**
  - Reference Implement (RI) available from SUN.
    - Example: `Unmarshall.java` (See next slide).

Unmarshall.java

```
public class Unmarshall {

    // This sample application demonstrates how to unmarshal an instance
    // document into a Java content tree and access data contained within it.

    public static void main( String[] args ) {
        try {

            // create a JAXBContext capable of handling classes generated into
            // the primer.po package
            JAXBContext jc = JAXBContext.newInstance( "icecube.daq" );

            // create an Unmarshaller
            Unmarshaller u = jc.createUnmarshaller();

            // unmarshal a daq instance document into a tree of Java content
            // objects composed of classes from the icecube.daq package.
            AtwdReadout atwdList =
                (AtwdReadout)u.unmarshal( new FileInputStream( "atwdSample.xml" ) );

            AtwdReadoutType.AtwdType atwd =
                    (AtwdReadoutType.AtwdType)atwdList.getAtwd().get(0);
            System.out.println("Found " + atwd.getChannel().size() +
                    " channels in the first ATWD");

        } catch( JAXBException je ) {
            je.printStackTrace();
        } catch( IOException ioe ) {
            ioe.printStackTrace();
        }
    }
}
```

Transforms

- **Example: Simple Test Framework (STF)**
  - Tests (modules) have:
    - a set of input and output parameters.
    - two methods, `init` and `execute`.
    - "setup" and "results" are XML files.
  - Definition of a test is given in XML
    - Must conform to `stfDefn.xsd`.
    - Example: `ExampleOne.xml` (See next slide).

ExampleOne.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<stf:test
    xmlns:stf="http://glacier.lbl.gov/icecube/daq/stf"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://glacier.lbl.gov/icecube/daq/stf stfDefn.xsd">

  <name>ExampleOne</name>

  <description>This is a simple Example of an STF module definition.</description>

  <version major="1" minor="0"/>

  <inputParameter>
    <name>fruit</name>
    <string default="bananas"/>
  </inputParameter>

  <inputParameter>
    <name>quantity</name>
    <unsignedInt default="1" maxValue="100" minValue="0"/>
  </inputParameter>

  <outputParameter>
    <name>fufilled</name>
    <boolean/>
  </outputParameter>

  <outputParameter>
    <name>numberRemaining</name>
    <unsignedInt/>
  </outputParameter>

</stf:test>
```

Transforms (2/3)

- XSL stylesheet can be used to create "C" header file for the test.
  - Example: `ExampleOne.h`
- "setup" and "results" documents need an XSL Schema to validate contents.
  - `stf.xsd` (See next slide).
- Each test has its own Schema, generated using another stylesheet.
  - Example: `ExampleOne.xsd`

stf.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://glacier.lbl.gov/icecube/daq/stf"
    xmlns="http://glacier.lbl.gov/icecube/daq/stf"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="ExampleOne.xsd"/>
  <xs:include schemaLocation="ExampleTwo.xsd"/>

  <xs:element name="setup">
    <xs:complexType>
      <xs:all>
        <xs:element minOccurs="0" name="ExampleOne" type="ExampleOneSetup"/>
        <xs:element minOccurs="0" name="ExampleTwo" type="ExampleTwoSetup"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <xs:element name="result">
    <xs:complexType>
      <xs:all>
        <xs:element minOccurs="0" name="ExampleOne" type="ExampleOneResult"/>
        <xs:element minOccurs="0" name="ExampleTwo" type="ExampleTwoResult"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Transforms (3/3)

- Dictionary of available tests can be made by applying a stylesheet to `stf.xsd`.
  - See `xsd2Dictionary.xsl` (See next slide).

xsd2Dictionary.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:stf="http://glacier.lbl.gov/icecube/daq/stf"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output indent="yes" method="text"/>
  <xsl:variable name="nl">
<xsl:text>
</xsl:text>
  </xsl:variable>

  <xsl:template match="/">
    <xsl:apply-templates select="xs:schema"/>
<xsl:copy-of select="$nl"/>
  </xsl:template>

  <xsl:template match="xs:schema">
#include "stf/stf.h"
  <xsl:apply-templates mode="include" select="xs:element[@name='setup']/xs:complexType/xs:all/
xs:element"/>
STF_DESCRIPTOR *stfDirectory[]={
  <xsl:apply-templates mode="descriptor" select="xs:element[@name='setup']/xs:complexType/xs:all/
xs:element"/>
   NULL
};
  </xsl:template>

  <xsl:template match="xs:element/xs:complexType/xs:all/xs:element" mode="include">
#include "stf-apps/<xsl:value-of select="@name"/>.h"
  </xsl:template>

  <xsl:template match="xs:element/xs:complexType/xs:all/xs:element" mode="descriptor">
   &amp;<xsl:value-of select="@name"/>_descriptor,
  </xsl:template>
</xsl:stylesheet>
```

Searching

- **XPath - The primary purpose is to address parts of an XML document.**
  - Can specify one, or more, elements in a document.
  - Can select on element context, attribute or content values.
- **XQuery**
  - An extension of XPath to cover XML data-bases.

Data Transfers

- **SOAP - Simple Object Access Protocol.**
  - Wraps up XML documents in an Envelope with a Header and a Body.
  - “SOAP with Attachments” allows non XML data, e.g. binary, to be transferred.
  - No examples at the moment.

Summary

- IceCube is a good experiment for investigating the use of XML in software development and data analysis.
- XML is an industry standard and there is already a large number of tools available.
- A lot of work has gone into it and its tools so we can concentrate on our own issues.
- It is a good approach from text based files and small data files.