XML Data and Process Modelling

Fall 2005
Thomas Hildebrandt
Programming, Logic and Semantics Group, ITU

Why XML?

- Models used for communication and interoperability by humans and programs
- “inter-business and inter-application”
- XML the de facto standard for tailor-made specifications of (semi) structured data collections shared and exchanged between humans and programs
- support for internationalisation (unicode) and platform independence (text based)

Road Map

- Why XML?
- XML data model, XML Namespaces, XMLSchema
- XML Business Process Models:
  WF and Com. patterns, the case of BPEL

XML very briefly

- XML: eXtensible Markup Language
- a merge of Hypertext languages and Markup Languages (SGML)
- text + markup tags: <greeting style="big">hello world</greeting>
- a specific XML language constrains the use of tags
- support for internationalisation (unicode) and platform independence (text based)
Recipe Example

- semi-structured: mix of tags and unstructured text

Ordered trees

The underlying data model of XML is that of ordered, node labelled trees

Node types

- text nodes
- element nodes
- attribute nodes
- comment nodes
- processing instruction nodes
- root nodes

XML recipe collection as tree

Figure 2.1: Tree view of the XML recipe
Textual representation

- properly nested, case sensitive tags
- attributes in-lined in start element
- predefined entities written as Unicode character references (e.g. `< as &lt;)
- comments as <!-- .... -->
- usually begins with an XML declaration
  <xml version="1.0" encoding="UTF-8">...

Applications of XML

- data-oriented (e.g. replacing relational DB)
- document-oriented (e.g. XHTML)
- protocols and programming languages (e.g. XMLSchema, XSLT, WSDL, SOAP, BPEL4WS)
- hybrids (e.g. the RecipeML language)

XML Namespaces

- The XML namespaces mechanism allows one to combine XML languages
  <example xmlns:foo="http://www.itu.dk/people/hilde/myfoospace"
           xmlns="http://www.itu.dk/people/hilde/mydefspace">
    <foo:fooelement></foo:fooelement>
  </example>

Formal Language Specification

- needed for computers/programs to understand the language
- Consists of the Syntax="the set of valid sentences" and the Semantics="the meaning of the valid sentences"
- Often only syntax is specified formally...
Syntax

- Natural languages and programming languages are usually specified by a grammar (e.g. in Backus-Naur Form)
- Examples: the 40-year old program Eliza, PASCAL, Algol 60 programming language
- Data collections are usually described by so-called schemas (e.g. databases and XML)

Schema languages

- A Schema/grammar can be described in a Schema/grammar language
- A schema processor/parser is used to validate (and normalize) a document

Why Schemas/grammars?

- Humans write documents that programs can parse and give “meaningful” error messages to invalid docs
- Why use standard schema language: Standard parsers, program generation, ...
- Schema language should be expressive, efficient and comprehensive

XMLSchema

- Plenty of XML Schema languages, but DTDs and XMLSchema by far the most common
- XMLSchema is the most expressive of the two (and is itself an XML language)
- both uses regular expressions as subgrammar
Regular expressions

- Build from symbols, ?, sequencing, | and *
- Very effective (linear time, constant space)
- Very comprehensive
- Limited expressiveness: Correspond to finite automata, that is, finite memory and thus can not express that a begin tag should be matched by an end tag

XMLSchema overview

- Namespace declarations
- Simple type definitions
- Complex type definitions
- Element declarations
- Attribute declarations

Student Records

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:s="http://www.brics.dk/ixwt/students"
         targetNamespace="http://www.brics.dk/ixwt/students">
  <element name="student" type="s:StudentType"/>
  <attribute name="id" type="string"/>
  <attribute name="score" type="s:Score"/>
  <simpleType name="Score">
    <restriction base="integer">
      <minInclusive value="0"/>
      <maxInclusive value="100"/>
    </restriction>
  </simpleType>
  <complexType name="StudentType">
    <attribute ref="s:id" use="required"/>
    <attribute ref="s:score" use="required"/>
  </complexType>
</schema>
```

Definitions and declarations populate the target namespace

Simple type examples

- The student record example contains both primitive types and derived types:
  ```xml
  <attribute name="id" type="string"/>
  <attribute name="score" type="s:Score"/>
  <simpleType name="Score">
    <restriction base="integer">
      <minInclusive value="0"/>
      <maxInclusive value="100"/>
    </restriction>
  </simpleType>
  ```
Derived types

• Patterns (regular expressions)
  \n  <simpleType name="percentage">
    <restriction base="string">
      <pattern value="(0|9)|1-\d|9(0|9)|100%"/>
    </restriction>
  </simpleType>

• Enumeration
  <simpleType name="passnonpass">
    <restriction base="string">
      <enumeration value="passed"/>
      <enumeration value="not passed"/>
    </restriction>
  </simpleType>

• Union
  <simpleType name="mark">
    <union>
      <simpleType>
        <restriction base="passnonpass"/>
      </simpleType>
      <simpleType>
        <restriction base="percentage"/>
      </simpleType>
    </union>
  </simpleType>

XMLSchema declarations

• associates a Schema to an XML document
  • may appear at sub-elements

<?xml version="1.0"?>
<student xmlns="http://www.brics.dk/ixwt/students"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.brics.dk/ixwt/students
  http://www.itu.dk/courses/IXMR/student.xsd">
  id="hilde071072"
  score="97"
</student>

Complex Types

• Complex types allow to specify elements and attributes as content of elements
  Example with only attributes:
  <element name="student" type="s:StudentType"/>
  <complexType name="StudentType">
    <attribute ref="s:id" use="required"/>
    <attribute ref="s:score" use="required"/>
  </complexType>

• elements as content may be specified by a regular expression:
  ○ Element reference <element ref="t:..." cardinality />
  ○ Concatenation <sequence> ... </sequence>
  ○ Union <choice> ... </choice>
  and cardinalities: minOccurs, maxOccurs = "0","1","..", "unbounded"
  (default cardinalities are 1)
XML Summary

- de facto standard for tailor-made semi-structured data collections shared on the internet
- use namespaces to combine XML languages
- use XMLSchema to specify syntax formally, expressive - tends to be incomprehensible

BPEL overview

- Process = PartnerLinks + variables + flowchart of activities
- Primitive activities: invoke, receive, reply, wait, assign, throw, terminate, empty
- Structured activities: sequence, flow, while, switch, pick, scope

Business Process Modeling (design and execution)

- Web services composition across organizational boundaries
- Technology push
- Need to improve efficiency

WF Patterns in BPEL

- Sequencing, Parallel Split & Synchronisation:
  ```xml
  <sequence>
    <flow>
      <!-- activity A -->
      <!-- activity B -->
    </flow>
  </sequence>
  ```

- Exclusive Choice and Simple Merge:
  ```xml
  <sequence>
    <switch>
      <case condition="C1"> <!-- activity A -->
      <case condition="C2"> <!-- activity A -->
      <case condition="C3"> <!-- activity C -->
    </switch>
  </sequence>
  ```

(Example of condition: getVariableData('status')='rejected')
**Multi-Choice and Synchronizing Merge:**
(Using the WSFL features)

Figure 2

Flow

A1  A2

L1  C  L2

List 6

```xml
<processA>
  <while cond='oC1'>
    <invoke processB ...>
  ... ssSerializable attribute is set to "yes" and thereby guaranteeing concurrency control on shared containers.
```

**More detailed example:** (multiple instances)

Listing 5

```xml
<sequence>
  <flow>
    <link name='L1'/>
    <link name='L2'/>
    <link name='L3'/>
  </flow>
</sequence>
```

Listing 6

```xml
<processA>
  <while cond='C1'>
    <while cond='C2'>
      <source linkName='L1'/>  
      <transitionCondition=v'C1'/>
      <source linkName='L2'/>
      <transitionCondition=v'C2'/>
      <assign><copy><from expression='0'/><to variable='i'/></copy>
      <variable name='i' type='xsd:integer'/>
    </while>
  </while>
</processA>
```

Listing 7

```xml
<processB>
</processB>
```

Listing 8

```xml
<i/>  
<while cond='C'/>  
<while cond='C'/>  
</while>
```

**Deferred Choice:** (using pick)

```xml
<pick name='messagePortTimeout'>
  <onMessage partnerLink='..'/>
</pick>
```

**Cancel Activity:**
Using fault and compensation handlers

**Cancel Case:** <terminate/> action

**Cycles** (using while loops)

```xml
<while condition='C'>
  <sequence>
    <!-- activity -->
    <onMessage partnerLink='..'/>
    <!-- update condition data -->
  </sequence>
</while>
```
Patterns not directly supported in BPEL

- WP8: Multi-merge
- WP17: Interleaved Parallel Routing
- WP18: Milestone

Communication Patterns

- Synchronous Communication:
  Request/reply and One-way (empty reply)

- Asynchronous Communication:
  Message passing: no outputContainer
  - No direct support for publish/subscribe nor broadcast

BPEL conclusions

- Anchored in the web services world
- Mix of flow/graph and control structures
- Expressive, perhaps too expressive
- Formalisation could provide precise semantics
- Not the only proposal (BPML, XPDL, WSCL,..)

Course Evaluation

- Please spend the time... we do listen to your comments!!