

# XML Data and Process Modelling

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

## Road Map

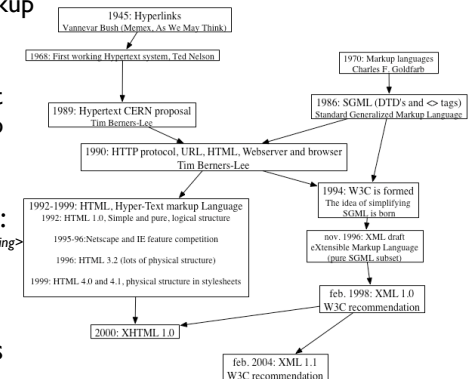
- Why XML ?
- XML data model, XML Namespaces, XMLSchema
- XML Business Process Models: WF and Com. patterns, the case of BPEL

## 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)

## 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



# Recipe Example

```
<collection>
  <description>Recipes suggested by Jane Doe</description>
  <recipe id="111">
    <title>Rhubarb Cobbler</title>
    <date>Wed, 14 Jun 95</date>
    <ingredient name="diced rhubarb" amount="2.5" unit="cup"/>
    <ingredient name="sugar" amount="2" unit="tablespoon"/>
    <ingredient name="fairly ripe banana" amount="2"/>
    <ingredient name="cinnamon" amount="0.25" unit="teaspoon"/>
    <ingredient name="nutmeg" amount="1" unit="dash"/>
    <preparation>
      <step>
        Combine all and use as cobbler, pie, or crisp.
      </step>
    </preparation>
    <comment>
      Rhubarb Cobbler made with bananas as the main sweetener.
      It was delicious.
    </comment>
    <nutrition calories="170" fat="28"
      carbohydrate="88" protein="14"/>
    <related ref="12">Garden Quiche is also yummy</related>
  </recipe>
</collection>
```

- semi-structured:  
mix of tags and unstructured text

# Ordered trees

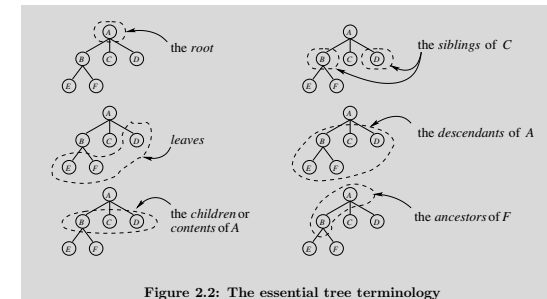


Figure 2.2: The essential tree terminology

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

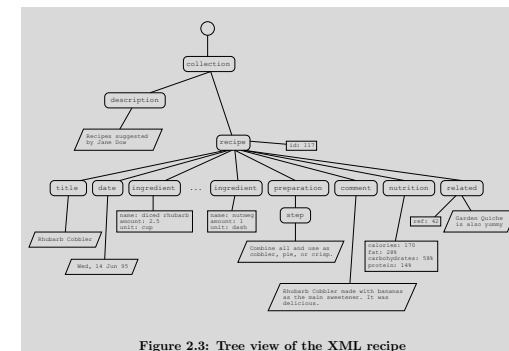


Figure 2.3: Tree view of the XML recipe

## 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

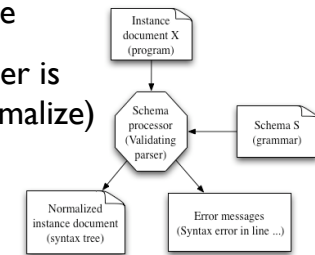
- needed for computers/programs to understand the language
- Consists of the Syntax="the set of valid sentences" and the Semantics="the meaning of the valid sentences"
- Often only syntax is specified formally...

# 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

```
<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:

```
<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)

```
<simpleType name="percentage">
  <restriction base="string">
    <pattern value="([0-9]|[1-9][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

```
<?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>
```

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

# 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)

```
<element name="order" type="n:order_type"/>
<element name="email" type="n:email_type"/>
<attribute name="id" type="unsignedInt"/>

<complexType name="order_type" mixed="true">
  <choice>
    <element name="address">
      <sequence>
        <element name="street" type="string"/>
        <element name="number" type="positiveInteger"/>
        <element name="floor" type="string"/>
      </sequence>
    </element>
    <sequence>
      <element ref="n:email"
        minOccurs="0" maxOccurs="unbounded"
        default="no email provided"/>
      <element name="phone" type="n:phone_type"/>
    </sequence>
  </choice>
  <attribute ref="n:id" use="required"/>
  <attribute name="priority" default="normal">
  <simpleType>
    <restriction base="string">
      <enumeration value="normal"/>
      <enumeration value="high"/>
    </restriction>
  </simpleType>
  </attribute>
</complexType>
```

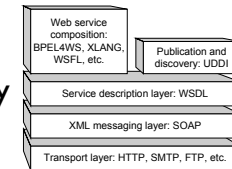
# XML Summary

- 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

# Business Process Modeling (design and execution)

- Web services composition across organizational boundaries
- Technology push
- Need to improve efficiency
- BPEL4WS v1.1 Schema:

<http://xml.coverpages.org/BPEL4WSv1.1-Schema-xsd.html>



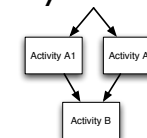
# BPEL overview

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

# WF Patterns in BPEL

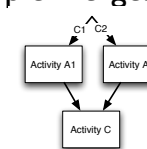
- Sequencing, Parallel Split & Synchronisation:

```
<sequence>
  <flow>
    <!-- activityA1 -->
    <!-- activityA2 -->
  </flow>
  <!-- activity B -->
</sequence>
```



- Exclusive Choice and Simple Merge:

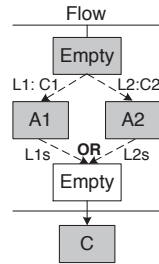
```
<sequence>
  <switch>
    <case condition="C1">
      <!-- activityA1 -->
    </case>
    <case condition="C2">
      <!-- activityA2 -->
    </case>
  </switch>
  <!-- activity C -->
</sequence>
```



(Example of condition: `getVariableData('status')='rejected'` )

- Multi-Choice and Synchronizing Merge: (using the WSFL features)

Figure 2



Listing 5

```

1 <flow>
2 <links>
3 <link name="L1"/>
4 <link name="L2"/>
5 <link name="L1s"/>
6 <link name="L2s"/>
7 </links>
8 <empty>
9 <source linkName="L1"
10 <transitionCondition="C1"/>
11 <source linkName="L2"
12 <transitionCondition="C2"/>
13 </empty>
14 activityA1
15 <target linkName="L1"/>
16 <source linkName="L1s"/>
17 activityA2
18 <target linkName="L2"/>
19 <source linkName="L2s"/>
20 activityC
21 <joinCondition="L1s OR L2s"
22 <target linkName="L1s"/>
23 <target linkName="L2s"/> ...
24 </flow>
  
```

- Cycles (using while loops)

```

<while condition="C">
  <sequence>
    <!-- activityA -->
    <!-- update condition data -->
  </sequence>
</while>
  
```

- More detailed example: (multiple instances)

```

<variable name="i" type="xsd:integer"/>
...
<assign><copy><from expression="0"/><to variable="i"/></copy>
</assign>
<while condition="bpws:getVariableData(i) !=5">
  <sequence>
    <invoke .../>
    <assign>
      <copy><from expression="bpws:getVariableData(i)+1"/>
        <to variable="i"/>
    </copy>
    </assign>
  </sequence>
</while>
  
```

- Multiple Instances with Synchronisation

Listing 6

```

1 <processA>
2 <while cond="C1">
3 <invoke processB ... >
4 </invoke>
5 </while>
6 </process>
  
```

Listing 7

```

1 <processB>
2 <receive processA ...
3 <createInstance="yes">
4 </receive>
5 </process>
  
```

Listing 8

```

1 moreInstances:=True
2 i:=0
3 <while moreInstances OR i>0>
4 <pick>
5 <onMessage StartNewActivityA>
6 <invoke activityA
7 i:=i+1
8 </onMessage>
9 <onMessage ActivityAFinished>
10 i:=i-1
11 </onMessage>
12 <onMessage NoMoreInstances>
13 <moreInstances:=False
14 </onMessage>
15 </pick>
16 </while>
  
```

- Deferred Choice: (using pick)

```

<pick name="messageortimeout">
  <onMessage partnerLink=".." ...>
    <!-- activityA -->
  </onMessage>
  <onAlarm for="PT10D">
    <!-- activityB -->
  </onAlarm>
</pick>
  
```

- Cancel Activity: Using fault and compensation handlers

- Cancel Case: <terminate/> action

## 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)

```
Listing 11
1 <process name="processA">
2   <sequence>
3     ...
4     <invoke partner="processB" ...
5       inputContainer="Request"
6       outputContainer="Response">
7     </invoke>
8     ...
9   </sequence>
10 </process>

Listing 12
1 <process name="processB"> ...
2   <sequence>
3     <receive partner="processA" ...
4       container="Request">
5     </receive>
6     ...
7     <reply partner="processA" ...
8       container="Response">
9     </reply>
10   </sequence>
11 </process>
```

- 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!!