Enabling Grids for E-sciencE

XML

Richard Hopkins
National e-Science Centre, Edinburgh
February 23 / 24 2005

Information Society

OUTLINE

- **Goals**
  - To understand the structure of an XML document
- **Outline**
  - Philosophy
  - General Aspects
  - Prolog
  - Elements
  - Namespaces
  - Concluding Remarks

A Markup Language

XML = eXtensible Markup Language

- **"Markup" means document is an intermixing of**
  - Content – the actual information to be conveyed - payload
  - Markup – information about the content - MetaData
    <date>22/10/1946</date>

    <date> … </date> is markup – says that the content is a date
  - Self-describing document
  - date is part of a markup vocabulary –
    - a collection of keywords used to identify syntax and semantics of constructs in an XML document

Extensibility

XML = eXtensible Markup Language

- **"Extensible" means the markup vocabulary is not fixed**
- **Compare with similar NON-extensible langhuage**
  - HTML (Hypertext Markup Language)
  - Fixed markup vocabulary e.g

     This is a paragraph. I like it. This is another paragraph
  - A presentation language for describing how a document should be presented for human consumption –

    This *is a paragraph. I like it.*

    This is ***another*** *paragraph*
  - For HTML the language is fixed and implicit in the fact that this is an HTML document – single-language document
- **XML requires explicit definition of the language**
- **One document can combine multiple languages**

Multi-lingual Documents

```
<businessForms:purchaseOrder>
        <date>          <USnotations:date>        10/22/2004        </..></..>
        <product>       <businessForms:barCode>123-768-252          </..></..>
        <quantity>      <metricMeasures:kilos>    17.53             </..></..>
</..>
```

- businessForms:purchaseOrder
  - This is an instance of the purchaseOrder construct within the businessForms language
- **BusinessForms (mythical)**
  - A language defining structure of business documents
  - For business interoperability
  - Doesn't prescribe the language of individual items such as dates
- Language names are actually universally unique URIs – www.DesperatelyTryingToStandardise.org/BusinessForms - see later

Multilingual Pros & Cons

```
<businessForms:purchaseOrder>
        <date>          <USnotations:date>        10/22/2004        </..></..>
        <product>       <businessForms:barCode>123-768-252          </..></..>
        <quantity>      <metricMeasures:kilos>    17.53             </..></..>
</..>
```

- **Separation of concerns – Design Factoring**
  - Design of purchase order structure and date format are independent concerns
  - Re-use of language definitions, e.g. date formats in many languages
  - Extensibility – Purchase order accommodates new product identification schemes (e.g. ISBN for book stores)
- **Of course, only works if both ends "understand" all languages used**
- **Makes things more complex –**
  - Creating and identifying the languages

Types of XML Language

- **Fundamental Standards, e.g.**
  - SOAP
    - soap-envelope:header soap-envelope:body
    - soap-envelope - the language for soap messages
    - A soap message is an XML document and its parts are identified using this vocabulary
  - Goal is a factoring that gives pick-and-mix of combinable standards
  - Associated with any WS standard will be a Schema definition of its XML language

....

- **Community conventions**
  - Perhaps, our BusinessForms language

....

- **Specific Application Language**
  - myProgram:parameter1
  - The language used in invoking particular operations of a web service

Human & Machine Oriented

How it really looks

```
<businessForms:purchaseOrder>
   <date>
      <USnotations:date>
         10/22/2004
      </USnotations:date>
   </date>
   <product>
      <businessForms:barCode>
         123-768-252
      </businessForms:barCode>
   </product>
   <quantity>
      <metricMeasures:kilos>
         17.53
     </metricMeasures:kilos>
   </quantity>
</ businessForms:purchaseOrder >
```

- **Human readable**
  - Sort of - OK with decent editor
  - Is de-buggable
  - Important for meta-data documents,
    - E.g. WSDL
- **Machine processable**

  Self description enables
  - General tools for producing and consuming XML documents
- **Verbose**
  - OK except for large data
  - Messages may have attachments not in XML

Philosophy Summary

- **XML goals**
  - Self-describing documents
  - Hierarchic structure
  - Enabling multiple languages
  - Human readable and reasonably clear
  - Easy to write programs that generate them
  - Easy to write programs that process them
- **For humans – easier to read than to write**
  - Leave detailed document creation to tools
  - But sometimes necessary to read them – particularly meta-data such as WSDL
  - Often need to understand how to design them
- **So rest of talk deals with some nitty gritty**

GENERAL EMENTS

- **Goals**
  - To understand the structure of anXML document
- **Outline**
  - Philosophy
  - General Aspects
  - Prolog
  - Elements
  - Namespaces
  - Concluding Remarks

Syntax

- **Syntax**
  - I will give syntax definitions of constructs
  - Mainly for your retrospective use
  - This uses notation similar to that used in the standard
    - http://www.w3.org/TR/2004/REC-xml-20040204 (Ed. 3, Feb ’04)
  - I will use some non-standard notation to make it a bit easier

Syntax

[22]	prolog ::=	XMLDecl**?** Misc* **(**doctypedecl Misc***)?**	
[23]	XMLDecl ::=	*<?xml* VersionInfo EncodingDecl**?** SDDecl S**?** *?>*	
[27]	Misc ::=	Comment **|** PI **|** S	/*syntax comment*/

- [22] — definition number – sequentially numbered in the spec.
- Prolog ::= — construct is defined to be
- XMLDecl — include anything this construct (unerlined) can be
- *<?xml* — itallic (times): exactly this (non-standard, spec. uses '>?xml')
- **(..)** — grouping (bold)
- **? * + |** — **?** optional, ***** 0 or more, **+** 1 or more, **|** alterantives
- "' …. "' — content in matching quotes – "…" or '…' (non-standard)
- … — text with some natural restrictions (non-standard)
- **…** — as … but allowing references (non-standard)
- /* … */ — a comment on the syntax

Miscellaneous items

[27] Misc ::= Comment | PI | S

- **A miscellaneous item is something outside the main structure –**
- **S Is white space –**
  - henceforth will ignore this aspect and leave it to common sense
  - there are specific rules
- **Other two are "explanatory" material**
- **Comment – for human consumption**
- **PI – Processing Instruction**
  - **For S/W consumption**
  - **Information to assist the S/W that is processing the XML**

Comments

[27]	Misc ::=	Comment | PI | S	
[15]	Comment ::=	<!-- ... -->	/* ... excludes -- */

- **A valid comment**

  <!-- This is a comment -->
- **An invalid comment**

  <!--This is -- not a comment --->
- **The "natural" restriction is –**

  you can't have -- in a comment,

  except as the --> terminator

Processing Instructions

[27]	Misc ::=	Comment | PI | S	
[16]	PI ::=	<? PITarget … ?>	/* … excludes ?> */
[16]	PITarget ::=	Name	/*not xml or XmL etc. */

- **Instructions to help the processing S/W**
- **PITarget identifies the intended S/W**

  E.g.

  <?xml-stylesheet type="text/ccs" href="greet.ccs" ?>
- **There may some S/W processing this XML to present it in human-readable form, using stylesheets to control formatting.**
- **Tells such S/W where the stylesheet is and what type it is.**
- **XML is a reserved target name – standard instructions for basic XML processing. Likewise xml, XmL, xMl etc.**

PROLOG

- **Goals**
  - To understand the structure of anXML document
- **Outline**
  - Philosophy
  - General Aspects
  - Prolog
  - Elements
  - Namespaces
  - Concluding Remarks

Document Structure

Main structure of document is

- **Prolog – like headers**
- **Element – the actual document**

[1]	document ::=	prolog element Misc*
[22]	prolog ::=	XMLDecl? Misc* (doctypedecl Misc*)?
[23]	XMLDecl ::=	*<?xml* VersionInfo EncodingDecl? SDDecl S? *?>*

```
<?xml version="1.0"  encoding="UTF-8" ?>
<!- - This is an example XML document - ->
<?xml-stylesheet  type="text/ccs"  href="greet.ccs"  ?>

<purchaseOrder> … </purchaseOrder>
```

prolog

Root element

The Prolog

[22] prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?

[23] XMLDecl ::= <?xml VersionInfo EncodingDecl? SDDecl ?>

```
<?xml version="1.0"  encoding="UTF-8" ?>

<!- - This is an example XML document - ->
<?xml-stylesheet  type="text/ccs"  href="greet.ccs"  ?>
```

Optional XML PI

Followed by:
- Other PIs
- Comments

- <?XML ..?> PI is optional, but should be there;
  - if so must be first
  - gives version number – must be 1.0 (for the 1.0 standard)
  - Could give the character encoding used –
    default is UTF-8, or something specifed at outer level (e.g HTTP header).
    ASCII is sub-set of UTF-8
  - Doctypedecl - To do with Document Type Declarations (DTDs) –
    - We are not using these, so ignore
  - SDDecl – standalone declaration – not clear when using schemas

ELEMENTS

- **Goals**
  - To understand the structure of anXML document
- **Outline**
  - Philosophy
  - General Aspects
  - Prolog
  - Elements
  - Namespaces
  - Concluding Remarks

Basic Element Structure

[1] document ::= prolog element Misc* /*this is root element*/

name attribute Attribute: name-value pair

```
<Invoice      customerType="trade"      dateStyle="US">
        ....
</Invoice>
```

} Start tag
} Content
} End tag

- **Primary element structure –**
  - Start Tag – <…>
    - Name of element
    - Zero or more attributes – uniquely named; order insignificant
  - Content – possibly nested elements, and other things
  - End Tag - </ … >
    - Name – MUST be same name as in matching Start Tag
- **Like HTML – but stricter – must have end tag**

Attributes

[41] Attribute ::= Name = AttValue

[10] AttValue ::= “” … “” /* excludes “” < & */

/*allows defined characters */

```
<Invoice customerType=“trade” dateStyle=“US”> …. </Invoice>
```

- **A name-value pair that is included in the start tag of an element**
- **Name is part of specific language**
- **Value may also be part of a specific language – QName – qualified name**
- **More properly the above might be**

```
< BusinessForms:Invoice
    BusinessForms:customerType =“BusinessForms:trade”
    BusinessForms:dateStyle=“USnotations:date”>
   …
</BusinessForms:Invoice>
```

- **This starts to get convoluted – necessary for designing for extensibility**

Element Tags

[39]	element ::=	STag content ETag
		| EmptyElementTag
[40]	STag ::=	< Name (Attribute)* >
[42]	ETag ::=	</ Name >
[40]	EmptyElementTag ::=	< Name (Attribute)* />

Empty Element Tag

```
<Invoice customerType="trade" dateStyle="US">
    <account accNo="17-36-2" terms="days31"/>
        ....
</Invoice>
```

Start tag

Content

End tag

Empty Element Tags –

```
<account  accNo="17-36-2"  terms="days31"/>
```

- Same as

```
<account  accNo="17-36-2"  terms="days31">
</account >
```

- Shorthand for element with no content – indicated by /> not >

Element Content

[39] element ::= STag content ETag | EmptyElementTag

[43] content ::= ...? (contentItem ...?)*

[43] contentItem ::= PI | Comment | Element | CDSect

```
<Invoice customerType="trade" dateStyle="US">
    <account accNo="17-36-2" terms="days31"/>
    <?billing Use Direct Debit>
    <!- - There now follows a list of items - ->
    <item> <date>10/22/04</date> … </item>
    <item> <date>10/24/04</date> … </item>
    The above are perishable &linebreak; Watch out!
    <item><date>10/29/04</date> … </item>
    ...
</Invoice>
```

Processing Instruction

Comment

Character Content

Elements, Non-unique names, Order is significant

Character Data Section

[43] contentItem ::= PI | Comment | element | CDSect

[18] CDSect ::= *<![CDATA[…]]>*

```
…
<item> <date>10/24/04</date> … </item>
<![CDATA[Some funny characters: <  and & ]]>
<item><date>10/29/04</date> … </item>
...
```

- **To make it easier to include characters which have special significance within XML – everything is taken literally except]]>**
- **Alternative is -**

```
…
<item> <date>10/24/04</date> … </item>
Some funny characters: &lt; and &amp;
<item><date>10/29/04</date> … </item>
...
```

Mixed Content

```
<Invoice customerType="trade" dateStyle="US">
    <item>  <date>10/24/04</date>  <price> 17.35 </price> … </item>
    The above are perishable &linebreak; Watch out!
    <item><date>10/29/04</date>  <price> 2173.35 </price> … </item>
</Invoice>
```

- **This is Mixed Content –**
  - Both direct character data and child elements (often excluded)
- **Generally a bad idea for web services documents**
- **Better is each content item is either**
  - Complex – all child elements
  - Simple – direct character data

```
<Invoice customerType="trade" dateStyle="US">
    <item>  <date>10/24/04</date>  <price> 17.35 </price> … </item>
    <noteLine>The above are perishable &linebreak; Watch out!</noteLine>
    <item><date>10/29/04</date>  <price>2173.35</price> … </item>
</Invoice>
```

Attribute vs Child

Pure child element approach – no attributes anywhere

```
<Invoice>
<customerType> trade </customerType>
  <dateStyle> US </dateStyle>
   <item>
      <date> 10/24/04 </date>
      <price>
           <currency>  Euro </currency>
           <amount> 17.34 </amount>
      </price>
      …
   </item>
   …
</Invoice>
```

Maximum attribute approach - use attributes wherever possible

```
<Invoice
  customerType="trade"
  dateStyle="US" >
     <item
       date="10/24/04"

        price-currency="Euro"
        price-Amount="17.34"

           …
     />
   …
</Invoice>
```

Can have unbounded number of item children

To use attribute approach for item would require defining infinite attributes item1-date item2-date …. Attribute names are unique within a tag

Not possible

Attribute vs Child

- **Use Attributes for "control" information**
  - Affects how we interpret/process the data
  - Typically a limited number of standard values – Euro, USDollar, ..
  - Often essentially "type" info
- **Use children for component data**
  - Arbitrary values within the type (any date, any integer, any general string, …)
- **Distinction is fuzzy rather than absolute**

Recommended style

```
<Invoice customerType="trade" dateStyle="US">
<item>
    <date> 10/24/04 </date>
    <price currency="Euro"> 17.34 </amount>
    …
  </item>
  …
</Invoice>
```

Notation

```
<Invoice customerType="trade" dateStyle="US">
  <item>
    <date>                          10/24/04 </>
    <price currency="Euro">         17.34 </>
    <productCode>                   17-23-57 </>
    <quantity>                      17.5 </></>
  <item>
    <date>                          10/24/04 </>
    <price currency="Euro">         17.34 </>
    <productCode>                   17-23-57 </>
    <quantity>                      17.5 </></></></>
```

- **Will use XML a lot –**
  - Schemas, WSDL; Soap messages;
- **Generally will use indentation to indicate structure and abbreviate End Tags to just </>**
- **Always have to actually put name in end tag !!!!**

NAMESPACES

- **Goals**
  - To understand the structure of anXML document
- **Outline**
  - Philosophy
  - General Aspects
  - Prolog
  - Elements
  - Namespaces
  - Concluding Remarks

Namespaces

```
<invoice>                                   <!-- INT = International -->
   <deliveryAddress>
      <UK:address> …<INT:street>…</> …<UK:county>…</> <UK:postCode>…</></>
   <billingAddress>
      <US:address> …<INT:street>…</> …<US:state>…</>    <US:zip>…</> </>
    …. …. </>
```

- **A namespace (= “language”)**
  - Does define a collection of names (vocabulary)
    - For UK : {address, county, postCode, …. }
  - Would usually have an associated syntax (e.g. Schema definition)
    - address = … county, postCode, …
    - Syntax may be available to S/W processing it
  - Implies a semantics – the (programmer writing) S/W processing a UK:address knows what it means.
  - Provides a unique prefix for disambiguating names from different originators
    - UK vs US vs INT

Namespace Names

- **To get uniqueness of namespace name, use a URI**
  - **UK:**postCode is really
    www.UKstandards.org/Web/XMLForms:postCode
    (mythical)
  - The URI <u>might</u> be a real URL, for accessing the syntax definition, documentation, ….
  - But it may be just an identifier within the internet domain owned by the namespace owner

Namespace Names

- **To get uniqueness of namespace name, use a URI**
  - **UK:**postCode is really
    www.UKstandards.org/Web/XMLForms:postCode

- **But www.UKstandards.org/Web/XML/Forms:postCode is**
  - Tediously long to use throughout the document
  - Outwith XML name syntax
    - Namespaces are not part of XML
    - A supplementary standard http://www.w3.org/TR/REC-xml-names
      A W3C recommendation
- **In an XML document**
  - declare a namespace prefix, as an attribute of an element
    - xmlns:**UK=**"www.UKstandards.org/Web/XML/Forms"
  - then use that for names in that namespace - UK:postCode
    - **UK:**post code is called a **QName** (qualified name)

Namespace Prefix Declarations

```
<BF:invoice … xlmns:BF="www/1"   xlmns:UK="www/2"   xmlns="www/3">
   <BF:deliveryAddress>
      <UK:address> …<street>…</> …<UK:county>…</> <UK:postCode>…</></>
   <BF:billingAddress xlmns:US="www. …" >

      <US:address > …<street>…</> …<US:state>…</>    <US:zip>…</> </>
   …. …. </BF:invoice>
```

- **Namespace declaration occurs as an attribute of an element**
  - i.e. within a start tag
- **Scope is from beginning of that start tag to matching end tag**
  - Excluding scope of nested re-declarations of same prefix
- **Can declare a default namespace**
  - xlmns="www/3" – this is the name space for all un-qualified names in the scope of this declaration, eg. Street
  - But not for attributes – if no prefix, no namespace

Overriding namespace declarations

```
<Document xmln:s1="www.1 xmlns="www/2" >
   < thing >  …                          <s1:thing> …      </></>
   < thing xmlns:s1="www/1" > …          <s1:thing >       </></>
   < thing >  …                          <s1:thing> …      </></> </>
```

- **xmlns:s1="www/1" Re-defines explicit namespace**
- **is bad idea –Unnecessary Confusion**

```
<Document  xmlns="www/me" >
   < thing >  …                                          <thing> …</></>
         <!- - following is presentation material in xhtml –
                                        default names space changed - ->
   < thing   xmlns="www/xhtml" > … <thing > </></>
    < thing >  …                                         <thing> …</></> <
```

- xmlns="www/xhtml" **Re-defines default namespace - reasonable**
- **Note if no default declared, then un-prefixed name has no namespace!**

NAMESPACES

- **Goals**
  - To understand the structure of anXML document
- **Outline**
  - Philosophy
  - General Aspects
  - Prolog
  - Elements
  - Namespaces
  - Concluding Remarks

Well-formed and Valid

- **Well-formed means it confoms to the XML syntax, e.g.**
  - Start and end tags nest properly with matching names
- **Valid means it conforms to the syntax defined by the namespaces used**
  - Can't check this without a definition of that syntax –
    - Normally a Schema
    - DTD (document Type Definitions) – deprecated
    - Others type dfinition system
      - *– some more sophisticated than Schemas*

Final Comments

- **A specialisation of SGML – a very general document markup language – any XML document is a an SGML document**
- **This is XML 1.0 Defined by WG3 – a recommendation**
  - http://www.w3.org/TR/2004/REC-xml-20040204 (Ed. 3, Feb '04)
- **Specification of the standard has a lot to do with DTDs which we have been ignoring – assume using Schemas instead**
- **A generalisation of HTML**
  - But not an actual extension.
  - An HTML document is not an XML document
  - There is a XML specialisation XHTML which gives HTML functionality
- **Definitions are now in terms of Infosets – an abstraction of XML with XML being the standard representation**

The End

THE END