Information Systems

XML Essentials

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

Outline

What is XML?

- Extensible Markup Language (XML) is a globally accepted, vendor independent standard for representing text-based data.
- The organization behind XML and many other web related technologies is the World Wide Web Consortium (W3C): http://www.w3.org/

Purpose of XML

- information technology got more complicated when we moved from the mainframes and started working in a client-server model.
- This caused problems:
  - How to visually represent data that are stored on larger mainframes to remote clients:
    Computer-to-human communications of data and logic.
  - How one application sitting on one computer can access data or logic residing on an entirely different computer:
    Application-to-application communication.

Purpose of XML

Solving idea: apply markup.

- Computer-to-human communication of data and logic was solved in a large way with the advent of HTML.
- For application-to-application communication the idea was to mark up a document in a manner that enabled the document to be understood across working boundaries.
- Applying markup to a document means adding descriptive text around items contained in the document so that another application can decipher the contents of the document.
- XML uses markup to provide metadata around data points contained within the document to further define the data element.

XML

- XML was created in 1998.
- Hailed as the solution for data transfer and data representation across varying systems.

Coals of XML

Simplicity: XML documents should be strictly and simply structured.

Compatibility: XML is platform independent. It should be easy to write or update applications that make use of XML.

Legibility: XML documents should be human readable.

Why Is XML Popular?

- Easy to understand and read.
- A large number of platforms support XML and are able to manage it.
- Large set of tools available for XML data reading, writing, and manipulation.
- XML can be used across open standards that are available today.
- XML allows developers to create their own data definitions and models of representation.
- XML is simpler to use than binary formats when you want to represent complex data structures.
- etc.

Viewing and Editing XML

- XML is text. Can be read and viewed by any text editor.
- There are specific XML editors or development environments, e.g.:
  - Altova XML Spy. http://www.altova.com/.
  - XMetal. http://www.justsystems.com/emea/.
  - Microsoft XML Notepad 2007. http://www.microsoft.com/.
  - TIBCO TurboXML. http://www.tibco.com/
  - Liquid XML Studio. http://www.liquid-technologies.com/
- etc.

XML Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<folder>
  <email date='20 Aug 2003'>
    <from>robert@company.com</from>
    <to>oliver@company.com</to>
    <subject>Meeting</subject>
    Could we meet this week to discuss the
    interface problem in the NTL project?  -Rob
  </email>
</folder>
```

The structure is described by the markup (the text marked by <,>).

XML Documents

- The text of the XML document consists of
  - The text data which is being represented: character data.
  - The text of the markup (enclosed by <,>).
- The markup consists of tags (e.g. the <to>,</to> pair).
- The part of the document enclosed by a tag is an element.
- The outermost tag encloses the root element.
- An XML document must have exactly one root element and the nesting of elements must be a proper one.
- An XML document may also contain a prolog, which is text that appears before the root element.

Elements

- Elements are the primary structuring units of XML documents.
- An element is delimited by its start and end tags.
- The content of elements can be
  - element if the element contains only elements (e.g. folder in the example above),
  - character if it contains only character data (e.g. to),
  - mixed if it contains both (e.g. email),
  - empty if it contains nothing (e.g. `<x></x>`).

Elements: Children and Parents

Relationships between the elements:

- **Child** element: An element inside another one in the first nesting level.
- **Parent** element: It is the reverse of the child relationship.
- **Sibling** element: These are elements with the same parent.

```
<email date='20 Aug 2003'>
  <from>robert@company.com</from>
  <to>oliver@company.com</to>
  <subject>Meeting</subject>
</email>
```

Elements: Descendants and Ancestors

- **Descendant** element: It is an element in the transitive closure of the child relationship
- **Ancestor** element It is an element in the transitive closure of the parent relationship.

```
<email date='20 Aug 2003'>
  <from>robert@company.com</from>
  <to>oliver@company.com</to>
  <subject>Meeting</subject>
</email>
```

Empty Element Tag

- **Empty** element: An element that contains neither character data not other elements.
- Empty element tags are created by adding / to the end of start tag.
- Empty element tags do not need end tags.

```
<empty_element_tag />
```

Naming Conventions

Names for elements can be chosen according to the following rules.

- Names are taken case sensitive.
- Names cannot contain spaces.
- Names starting with "xml" (in any case combination) are reserved for standardization.
- Names can only start with letters or with the '_ ', ':' characters.
- They can contain alphanumeric characters and the '_ ', '-', ':', '.' characters.

Attributes

- Attributes are name='value' pairs, listed in the start-tags of elements.

  ```
  <email date='20 Aug 2003'> ...  </email>
  ```

- The naming rules of elements apply also for attributes.
- Elements can contain zero or more attributes.
- The names of the attributes must be unique within a start-tag.
- Attributes cannot appear in end-tags.
- Attribute values must be enclosed in single or double quotes.

Elements vs Attributes

- Attributes can be resolved into elements and elements with character content can be put into attributes.
- ```
  <email date='21 Aug 2003'
    from='oliver@company.com'
    to='rob@company.com'
    cc='amy@company.com'>
    <subject>Re:  Meeting</subject>
    ...
  </email>
  ```
- ```
  <email>
    <date>21 Aug 2003</date>
    <from>oliver@company.com</from>
    <to>rob@company.com</to>
    ...
  </email>
  ```

Elements vs Attributes

- How do I know whether to use elements or attributes?
- No good answer to this question.
- The argument concerning usefulness of attributes in ongoing.

Brief Summary of the Section

- XML: a simple markup language
- easy to construct and easy to read.
- The means to store data in XML documents: elements and attributes.
- Elements: tags containing character data, other elements, or both.
- Attributes: name-value pairs placed within element start-tags.
- Element and attribute names are case sensitive and follow certain rules.

Well-Formed XML

- An XML document must obey a few simple rules to be syntactically correct, or well-formed.
- If you know HTML, many of these rules will be familiar to you.
- However, not all well-formed HTML documents are well-formed XML documents.

Start-Tags and End-Tags

- In XML, every element must have a start-tag and an end-tag.
- Elements such as HTML's <br> can not exist in XML (but <br/> can).
- A well-formed fragment consisting of start-tag, some data, and end-tag:

```
<text>Some text</text>
```

- This is not well-formed, because it lacks an end-tag:

```
<linebreak>
```

Overlapping Tags

- XML elements can not overlap.
- This rule does not exist in HTML. There is is legal, e.g., to have `<i>` tags carrying through multiple `<p>` tags.
- Well-formed example of nested tags:

```
<para>
  This <ital>element</ital> is
  <bold>well-formed</bold>.
</para>
```

- This example in not well-formed:

```
<para>
  This <ital>element is
  <bold>not</ital>well-formed/<bold>.
</para>
```

Root Element

- Every XML document must have exactly one root element.
- In XML, the root element can be any legal element name, whereas in HTML, it must be `<html>`.
- Well-formed XML document:

```
<root>
  <data>text</data>
  <data>more text</data>
</root>
```

- This in not well-formed:

```
  <data>text</data>
  <data>more text</data>
```

Attributes

- XML attribute values must be enclosed in either single or double quotation marks.
- XML attributes must be unique within a particular element.
- Well-formed:

```
<element id="2" type="47">
```

- This in not well-formed:

```
<element id=2 type=47>
<element type="46" type="47">
```

Entity References

- Special characters have to be substituted with the corresponding entity references.

Character	Entity reference
<	&lt;
>	&gt;
"	&quot;
’	&apos;
&	&amp;

Summary of the Section

- XML document must be well-formed to be usable.
- The rules for well-formed XML are simple and intuitive.
- Three basic statements:
  - Every start-tag needs the corresponding end-tag, and tags can not overlap.
  - Encapsulate attribute values in either single or double quotation marks.
  - Watch out for reserved characters and replace them with proper entity reference.

Other XML Syntax

- XML declaration
- Processing instructions
- Comments

XML declaration

- Use to identify a document as an XML document.
- Usually appears on the first line of an XML document.
- Not strictly required.
- A typical XML declaration:

```
<?xml version="1.0" encoding="utf-16"
standalone="yes"?>
```

XML declaration

- The version attribute is mandatory.
- The encoding attribute specifies how the document text is encoded.
- Standard encodings: UFT-8 (ASCII) or UFT-16 (Unicode)
- The `standalone` attribute indicates whether the document depends upon an external DTD or is an independent document.

```
<?xml version="1.0" encoding="utf-16"
standalone="yes"?>
```

Processing Instructions

- An XML file may include processing instructions.
- Specific applications reading the document might interpret the instructions as commands to be executed.
- Generally, processing instructions are used to inform the parser to associate with the XML document a particular XSL or CSS file for formatting.
- Unlike the declaration, processing instructions can appear anywhere in the document.
- Example of processing instructions:

```
<?xml-stylesheet type="text/css"
href="mysheet.css"?>
```

Comments

- Comments can be included in an XML document to provide additional information to a human reader.
- Applications ignore comments.
- Comments can be included in the XML document anywhere outside other markup with the following syntax.

```
<!-- Comment text comes here.  -F->
```

CDATA

- One can force the text in an XML document to be treated like character data, by enclosing it in a CDATA section.
- For instance, if you want to an element to contain text with many illegal characters and do not want to replace them with entity references, you may want to use CDATA.

```
<example>
<![CDATA[
  <html>
    <head> <title>21st Century
       Technologies</title>
    </head>
    <body bgcolor="blue">
       <p>Computers & more </p>
    </body>
  </html>
]]>
</example>
```

Namespaces

- XML namespaces allow the user to divide structured data further into application-specific groups.
- Namespaces exist to overcome two complications:
  - If different applications concern with different sections of an XML document, namespaces allow to indicate which information pertains to which application.
  - Namespaces help to eliminate collisions.

Uniqueness

- Uniqueness is the most important aspect of a namespace.
- Namespace must be identified by a unique Uniform Resource Indicator, URI.
- When you define a namespace, you should associate it with a URI to which only you have a right.
- For instance, if you are creating a series of XML documents that contain product information for an e-store, `http://www.shoppingisfun.com`, you could associate your namespace with a URI as `http://www.shoppingisfun.com/product-info`
- Namespaces do not have to be associated to real URIs. They are just unique strings.

Declaring Namespaces

- Namespaces should be declared.
- Namespaces can be declared within any element in a document, using the reserved attribute name `xmlns`.
- A namespace will apply only to the element (and all of its children) in which it is declared.

Declaring Namespaces

- To associate an element with a namespace, we attach to the element name a prefix that represents that namespace.
- A prefix is a kind of alias (used for compactness) for the namespace string.
- For example, to create a prefix `foo` to act as an alias for the namespace `http://www.stuff.com/foo`, we would have the following:
  `<text xmlns:foo="http://www.stuff.com/foo">`
- It declares that the namespace `http://www.stuff.com/foo` is associated with the prefix `foo`.

Declaring Namespaces

```
<text xmlns:foo="http://www.stuff.com/foo">
  <foo:word>Somelongword</foo:word>
</text>
```

- This code places `<word>` into `foo` namespace.
- Note that `<text>` is not in the `foo` namespace.

Declaring Namespaces

Three namespace declarations:

1. `<author xmlns="http://www.authors.org/namespace">`
2. `<author xmlns:aut="http://www.authors.org/namespace">`
3. `<aut:author xmlns:aut="http://www.authors.org/namespace">`

- The first defines a new global (default) namespace. The `author` element and all its descendants are part of it.
- The second defines a new namespace, but anything that one wants to associate with it must use the `aut` prefix, including the `author` element itself.
- The third does the same, but `author` associated with the namespace.

Default Namespaces

- When a default namespace is assigned to an element, all its descendants automatically get fall within the same namespace.
- No prefix is necessary.

Prefixing Elements and Attributes

- To place an element in a namespace, add the prefix associated with the namespace to the elements start- and end-tags:

```
<foo:person
    xmlns:foo="http://www.stuff.com/foo">
<!- details of this person ->
</foo:person>
```

- Attributes must be prefixed explicitly.

Scope

- The section of XML to which a namespace declaration and its prefix apply is called a scope.
- Namespace declarations can be made within any element, and are available only to the element in which they are made, and to all its descendants.
- That means, the scope of a namespace encompasses only the element and its descendants.
- If a prefix is used outside the scope, the XML may still be well-formed, but the prefixed elements will be considered as stand-alone elements and not in the namespace.

Summary

- Namespaces are unique strings used to avoid conflicts between different elements that may have the same name in an XML document.
- Namespaces also allow applications to recognize specific data subsets of an XML document and ignore others.
- A namespace will be associated with a prefix when it is declared.
- The prefix acts as an alias for the namespace and can be attached to a name of an element or an attribute to indicate that the element or the attribute is within the namespace.
- If you do not want to use the prefix, you can use the default namespace.
- The default namespace automatically includes the current element and all the descendants of the current element in itself.