Russian Dolls and XML: Handling Multiple Versions of XML in XML

Keywords: Change Management, Content Management, Data representation, Electronic Publishing, Enterprise Content Management, Fragment, Legal publishing, Markup, Publishing, SAX, Structure, XML

Robin La Fontaine
CEO
DeltaXML (Monsell EDM Ltd)
Upton-on-Severn
Worcestershire
UK
robin.lafontaine@delaxml.com
http://www.deltaxml.com

Biography

Robin La Fontaine has a degree in Engineering Science from the University of Oxford and a Masters degree in Computer Science. He views the ability to handle change intelligently as essential for a successful company. His company has developed a method for finding and representing changes in XML documents and data and this is implemented as DeltaXML. Robin has contributed to the STEP XML Working Group within ISO (ISO10303). He has been project manager of several European research projects including the XML/EDI European Pilot Project. His background is in CAD data exchange and Lisp programming.

Thomas Nichols
CTO
DeltaXML (Monsell EDM Ltd)
Upton-on-Severn
Worcestershire
UK
thomas.nichols@delaxml.com
http://www.deltaxml.com

Biography

Thomas Nichols has been developing commercial software since he caught the OO bug in 1986, and saw the advent of XML in 1998 as his escape from EDI. His interest throughout his career has been in practical implementation of rocket-science technology to solve real-world problems, he is now greatly enjoying putting his commercial experience into practice steering the technical implementation of DeltaXML, which he sees as a revolutionary technology. Thomas is a skilled Java and C++ programmer, working with cross-platform technologies and happiest in a "heterogeneous environment". XML is one of his very favourite beasts.

Abstract

Flexible as it is, XML is an ephemeral format. An XML document holds a snapshot of data, unlike a repository such as CVS/RCS which can store a history of such snapshots, a versioned archive. Such an archive format has many uses, providing an auditable record of revisions, a switchable set of configuration options or a document allowing "track changes" for editorial control.

Ideally such an archive should itself be expressed in XML, allowing the archive to be processed by the same tools as the data it contains. A CVS revision history cannot be transformed by XSLT, cannot be referenced with XPath or pointed to with XPointer.

There is currently no generic solution to this problem.

What are the challenges? The multi-version XML file should of course record just the changes and share unchanged data between versions. Handling attributes presents a particular problem because each may be present or absent in a particular version and could have different values for different versions. Ideally, when dealing with documents changes should be recorded at the word level. It must be easy to extract a version, add a new version into the archive, delete a version from the archive or clone a version within the archive. An overriding requirement is to keep to the spirit of XML, i.e. that the archive is not only easily processed by computer but is also inherently understandable when viewed in a text editor.

This paper proposes a generic XML format to handle multiple versions of an XML document, using the same basic structure as the files which are being archived. The paper discusses the technical issues and challenges, and alternative approaches. Benefits of the proposed format include the ability to extract any version with a simple XSLT stylesheet, and similarly to use XSLT for deletion of a specific version and cloning a version for minor modifications. The most interesting challenge is how to automate the addition of an arbitrary new version into the multi-version file.

The proposal is built on many years of research and commercial experience in identifying and handling changes to XML data. The proposed format will enable development of a new class of applications, which are briefly discussed.

Table of Contents

Introduction

There is something very appealing about a Russian "matryoshka" doll, whose outer case opens to reveal another smaller doll, inside which are smaller and yet smaller nested dolls. There is something appealing, also, about holding many XML documents within a single XML file. It seems on first inspection that this should be quite possible, but in fact there are some significant challenges. If it can be done effectively and efficiently, new applications can be built, and old applications can be simplified - this will be a significant development, furthering the usefulness of XML.

Many content management systems hold multiple versions of a text file within a single archive file, and extract from this a specific version as required. It is well understood that comparing XML files is a lot more complex than comparing text files [DeltaXML] , and similarly the problem of holding multiple XML files in a single file is considerably more complex.

The challenges are greater, but so are the rewards. A multi-version XML file will contain the data in a structured and processable form, opening up the opportunities for more intelligent applications. Several papers have been written discussing querying of multiple versions of XML data, for example [Temporal Queries] and [Pathexpression Queries] .

This paper looks at the requirements and proposes a format for storing multiple XML documents within a single XML archive document. It is applicable equally to documents or data held in XML.

Applications of a Multi-version XML Archive

Applications include the following:

1. Content management and version control systems.
2. Single-source publishing applications, for example with multiple language versions held within a single document.
3. Control systems comparing multiple data feeds and generating output for further processing or for visual display.
4. Multiple configuration sets stored within an archive as versions of a base document, selectable at runtime.
5. Audit control systems.
6. Innovative applications such as complex SVG animations, interactive collaborative editing environments and business intelligence systems running queries across multiple document versions - all from a single XML document.

Nomenclature

NOTE: We refer to such a "multi-version XML archive document" as a "unified delta" - it unifies the differences, or deltas, between the inputs into a single document.

Requirements

What are we looking for in this unified delta? An initial set of requirements should include:

1. The unified delta must itself be well-formed XML.
2. The unified delta must be capable of holding any well-formed XML data.
3. The unified delta must hold one or more XML documents with the same root element.
4. It must be easy to extract a particular version from a unified delta.
5. It must be easy to delete a version from a unified delta.
6. The unified delta must keep to the spirit of XML, in that it is not only easily processed by computer but is also inherently understandable when viewed in a text editor.
7. The unified delta must be as small as possible without compromising readability.
8. Common data within the unified delta must be shared, repetition of data must be minimized.

Issues in Designing a Multi-version "unified delta" XML Format

A number of different approaches have been considered before arriving at the proposed format. There is a wealth of experience in the area of versioned text files and relational database files and some of these studies apply to XML. However, XML presents its own particular challenges and opportunities. Some papers address XML versioning specifically and [Version Management of XML Documents] and [A Comparative Study of Version Management Schemes for XML Documents] provides an overview of different approaches.

Is it possible to have a unified delta that looks very much like the original document?

It is tempting to shoe-horn the input documents into a rigid archive format, to radically modify the structure of the original data in order to archive it. However, there is an inherent benefit in having a unified delta that is as close in structure as possible to the original. A feature of the proposed format is that to turn a given XML document into a unified delta file (containing one version) involves only the addition of a single attribute on the root element.

What about whitespace and all that jazz?

It is evident that documents will need to be held in a canonical form, for example they must not be pretty-printed and namespace prefixes must be consistent. Such rationalization is necessary before any sensible comparison between XML data files can be made, and comparison is the basis of adding a new document into an existing unified delta.

Does the order of adding versions matter?

Documents will be added to the unified delta in a determinate sequence - should this sequence be apparent in the resultant document? What effect does this order of addition have? Given that a comparison process is involved to merge a new document into a unified delta, the order of addition of files will affect the final representation - though the original documents can still be extracted unscathed.

In this proposal, we hold the versions as a logical stack, so that the most recently added version always appears first in document order. Different unified deltas holding the same set of inputs but in differing orders will be distinct. This is logical and intuitive: the unified delta is a history, and so chronology is important. The critical feature is that snapshots from different points in this chronology may be extracted and compared, and compared also with snapshots from other unified deltas.

How should different versions be identified?

We use a simple identifier rather than a number, for readability and ease of understanding. In principle identifiers of any length may be used, but for efficiency and simplicity we use minimally short identifiers. Another approach, descibred in [Temporal Queries] , proposes the use of time stamps to handle the versions, thus supporting temporal queries. This requires considerably more manipulation and logic both for queryiing the data and for adding new versions into the versioned XML.

How can versions of attributes be stored?

Attributes are perhaps the most difficult area to handle. Although it is possible to handle changes to attributes between two versions using attributes themselves, such as the mechanism used by [DeltaXML] , it becomes difficult to extend this to more than two versions. For this reason, where there are changes in attributes these are converted and held in elements. This is an overhead, and weakens the connection with the original document structure, but the result is easier to understand and to process. Where there are no changes, attributes stay as attributes.

How can we ensure that any XML can be handled?

It is evident that in order to avoid interference with the original XML data being placed into the unified delta, we must adopt a new namespace for additional attributes and elements.

Proposed Format for Multi-version XML "Unified Delta"

The proposed format is best illustrated with a simple example.

We begin by considering an example with four versions of a text document to archive into a single file. Note that at this stage there are no changes to the attributes.

Example A

```
<doc name="example 1" date="2003-12-11">
 <body>
  <p font-type="times" font-style="bold">
    This is the content of the first and second example.</p>
 </body>
</doc>
```

Example B

```
<doc name="example 1" date="2003-12-11">
 <body>
  <p font-type="times" font-style="bold">
    This is the content of the first and second example.</p>
 </body>
</doc>
```

Example C

```
<doc name="example 1" date="2003-12-11">
 <body>
  <p font-type="times" font-style="bold">
   This is the content of the third example.</p>
  </body>
</doc>
```

Example D

```
<doc name="example 1" date="2003-12-11">
 <body>
  <h1>A new heading</h1>
  <p font-type="times" font-style="bold">
    This is the content of the fourth and subsequent examples.</p>
 </body>
</doc>
```

We allocate a namespace to cater for elements and attributes that we add for the archive file: `xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"`.

We first convert the first file into a unified delta with a version denoted 'A'. The result has been pretty-printed but in general all whitespace that is not significant is removed.

Example A, as a unified delta file

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

```
  <doc xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
    dxu:vset="|A|" date="2003-12-11" name="example 1">
  <body>
   <p font-style="bold" font-type="times">
     This is the content of the first and second example.</p>
  </body>
 </doc>
```

You can see here that the archive is almost identical to the original file except for the addition of a `dxu:vset` ('version set') attribute to indicate the version set for files in this archive.

Now we can add the B file to this unified delta:

Examples A and B as a unified delta

```
 <doc xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
   dxu:vset="|B|A|" date="2003-12-11" name="example 1">
  <body>
   <p font-style="bold" font-type="times">
     This is the content of the first and second example.</p>
  </body>
 </doc>
```

Again, this is just like the original files with all common data shared and the `dxu:vset` attribute set to indicate that the data is for versions B and A. In this case we have used a pipe character delimiter to separate these. Two files that are the same is not a particularly interesting example, but it shows that the archive file is sharing content and remains with a structure similar to the original files.

Now we can add in Example C to show that the changes will be recorded in the unified delta file.

Examples A, B and C in a unified delta

```
 <doc xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
   dxu:vset="|C|B|A|" date="2003-12-11" name="example 1">
  <body>
   <p font-style="bold" font-type="times">
    <dxu:PCDATA dxu:vset="|C|">
      This is the content of the third example.</dxu:PCDATA>
    <dxu:PCDATA dxu:vset="|B|A|">
      This is the content of the first and second example.</dxu:PCDATA>
   </p>
  </body>
 </doc>
```

You can see that where the text has changed, the two versions are present, each wrapped with a `dxu:PCDATA` element. This is necessary to separate the different versions of text. Each `dxu:PCDATA` has a `dxu:vset` attribute which indicates the version or versions that this data belongs to. If this `dxu:vset` attribute is not present, e.g. on the `<body>` element, then it means that the value is the same as for the parent element. Thus the `<body>` element and the `<p>` element are present for all versions.

To see more clearly how this works, we will add in Example D to the unified delta.

Examples A, B C and D in a unified delta

```
 <doc xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
   dxu:vset="|D|C|B|A|" date="2003-12-11" name="example 1">
```

```
  <body>
   <h1 dxu:vset="|D|">A new heading</h1>
   <p font-style="bold" font-type="times">
    <dxu:PCDATA dxu:vset="|D|">
      This is the content of the fourth and subsequent examples.</dxu:PCDATA>
    <dxu:PCDATA dxu:vset="|C|">
      This is the content of the third example.</dxu:PCDATA>
    <dxu:PCDATA dxu:vset="|B|A|">
      This is the content of the first and second example.</dxu:PCDATA>
   </p>
  </body>
</doc>
```

Handling Attributes

Now we will consider how changes to attributes are handled. Consider a file E which is similar to A but with different attributes:

Example E

```
<doc name="example 1" date="2003-12-25"
   original-date="2003-12-11">
  <body>
   <p font-style="bold">
     This is the content of the first and second example.</p>
  </body>
</doc>
```

The `name` attribute remains unchanged. The `date` attribute has been modified and `original-date` has been added. Witin the element `<p>` the `font-type` attribute has been deleted. When we add this to the archive we see how changes to attributes are represented.

```
<doc xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
  dxu:vset="|E|A|" name="example 1">
  <dxu:attributes>
   <date>
         <dxu:PCDATA dxu:vset="|E|">2003-12-25</dxu:PCDATA>
         <dxu:PCDATA dxu:vset="|A|">2003-12-11</dxu:PCDATA>
   </date>
   <original-date dxu:vset="|E|">2003-12-11</original-date>
  </dxu:attributes>
  <body>
   <p font-style="bold">
   <dxu:attributes>
    <font-type dxu:vset="|A|">times</font-type>
   </dxu:attributes>This is the content of the first and second example.</p>
  </body>
</doc>
```

Note first that any attributes that are unchanged remain as attributes. This is important because it adds to the readability of the unified delta and keeps the document size down.

Any changes to attributes, including additions and deletions, are held in a `dxu:attributes` element which is the first element in the content. The name of the attribute is used as a new element name and this element contains the value or values of the attribute. The actual value of the attributes is held as PCDATA in a similar way to the versioning of normal PCDATA. Note that where there is only one value, the `dxu:PCDATA` wrapper is not needed. Again, this aids clarity and keeps size down.

Some care needs to be taken with namespaces for attributes because the rules for namespaces on attributes is subtly different from those on elements. One of the reasons for converting the attribute names to element names is to make it easier to use normal XML processors to handle all the namespace declarations.

MIXED Content

Now let us look at mixed content, containing both PCDATA and elements, shown in a version F:

Example F

```
<doc>
 <p>This is the <em>last</em> version</p>
</doc>
```

We need to ensure that when we modify this the changes go into appropriate places. So consider G:

Example G

```
<doc>
 <p>This is the <em>ultimate</em> version.</p>
</doc>
```

Giving a unified delta of:

```
<doc xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
  xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1" dxu:vset="|G|F|">
 <p>This is the
  <em>
   <dxu:PCDATA dxu:vset="|G|">ultimate</dxu:PCDATA>
   <dxu:PCDATA dxu:vset="|F|">last</dxu:PCDATA>
  </em>
  <dxu:PCDATA dxu:vset="|G|"> version.</dxu:PCDATA>
  <dxu:PCDATA dxu:vset="|F|"> version</dxu:PCDATA>
 </p>
</doc>
```

Note here that the `<em>` element contains both versions of its contents. The initial text before the `<em>` has not been changed so remains as text with no `dxu:PCDATA` wrapper, whereas the text after the `<em>` element has been changed and is wrapped.

Operations on a Unified Delta

There are three main operations on an archive:

1. Adding a new version
2. Extracting a version
3. Deleting a version

Clearly adding a version is the most complex because it requires a full and accurate comparison to be made between a document in the archive and the new file being added. In these examples, DeltaXML has been used for this purpose because it is able to map accurately through the tree structure of the XML files and thus enable the mapping of a new version into the appropriate places in the existing archive.

Extracting a version proves to be very simple, and a short (60 line) XSLT stylesheet will achieve this. This could also be written as a filter on a SAX stream for increased speed.

Deleting a version is also fairly simply achieved with XSLT, being a 'negative image' of the extraction process, i.e. instead of writing out everything that is in a single version, we write out everything except this version.

Modifying Data in a Unified Delta

To what extent is it possible to modify data in a unified delta file? By this, we mean the changing, adding or deleting of something in the unified delta file with the purpose of changing a pre-determined number of versions in the file.

For a text-based file containing multiple versions it is generally not advisable directly to edit the repository; indeed the CVS manual offers dire warnings against doing so. With the proposed unified delta format this becomes quite possible, although it requires some care. It is safer to extract, modify and replace a version into the unified delta. However, one of the benefits of the proposed format is that because it is XML, it is possible to view the data in an editor and change it, making de-bugging and emergency data repair possible.

Quality Considerations

How do we measure the "quality" of a unified delta file? The simplest and most obvious metric is to consider the size of the file relative to the sizes of the original files. Compressed size should also be considered because tags of any length are typically compressed to a few bytes in most compression algorithms. We could use shorter element names but this goes against the spirit of XML.

Moving beyond this in terms of a measure of quality, if we want to work with the data in the unified delta then it is important that the common data is shared. There will always be some judgement involved in adding a new version to a unified delta and the unified delta will only be as good as the comparator used when adding new versions.

Summary of Unified Delta Format Rules

As shown in the above examples, the proposed unified delta format is really very simple. The following rules apply:

1. A new namespace is declared: `xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"`
2. The root element has a `dxu:vset` attribute with a value showing all the versions of data in the file.
3. In the `dxu:vset` attribute, the versions are separated using, in this case, a pipe symbol. Another delimiter can be used instead, note that the first and last characters will be this character.
4. The order of the versions in the `dxu:vset` attribute of the root element is the reverse order of addition to the archive, i.e. most recent first.
5. All other elements will have a `dxu:vset` attribute unless its value is the same as that of the parent node, in which case it is omitted.
6. The value of each `dxu:vset` attribute will always be a subset of the actual or implied value of the `dxu:vset` attribute on its parent.
7. Any versioned PCDATA is wrapped in a `dxu:PCDATA` element, which itself has a `dxu:vset` attribute.
8. Unchanged attributes of any element remain as attributes.
9. Changed attributes are held in a `dxu:attributes` element whose child elements are elements with the same names as the original attributes.
10. The value of changed attributes are held as `dxu:PCDATA items`.

Conclusions

The format proposed in this paper allows multiple XML files having the same root element to be represented in a single, well-formed XML file. Extraction of a particular version is a simple operation, accomplished with a short XSLT stylesheet. Deletion of a version is also a simple operation. The process of adding a new version into the archive is inevitably more complex and relies on access to an intelligent XML comparator.

In the proposed unified delta format, only one additional attribute, the `dxu:vset` attribute, is added to the original file. Two additional elements are needed, `dxu:attributes` and `dxu:PCDATA`. The attribute names are also converted to element names for use within `dxu:attributes`.

This simple unified delta format will allow new applications in a number of areas, and will provide XML-aware version support for content management and version control systems that operate on XML data.

Acknowledgements

This paper was prepared using XMetaL, the XML content editor from Corel.

Bibliography

[DeltaXML]
"A Delta Format for XML: Identifying changes in XML files and representing the changes in XML", from XML Europe 2001www.gca.org/papers/xmleurope2001/papers/pdf/s29-2.pdf
[Temporal Queries]
"Temporal Queries in XML Document Archives and Web Warehouses"
http://www.cs.ucla.edu/~zaniolo/papers/time-ictl03.pdf
[Pathexpression Queries]
"Pathexpression Queries over Multiversion XML Documents"
http://www.cse.ogi.edu/webdb03/papers/09.pdf
[Version Management of XML Documents]
"Version Management of XML Documents"
http://www.research.att.com/conf/webdb2000/PAPERS/5c.ps
[A Comparative Study of Version Management Schemes for XML Documents]
"A Comparative Study of Version Management Schemes for XML Documents", Chien et al, Sept 2000
http://www.cs.auc.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-51.pdf