Beyond Babel - Simplifying Translation with XML

Robin **La Fontaine** <robin.lafontaine@deltaxml.com>
Thomas **Nichols** <thomas.nichols@deltaxml.com>
Martin **Bryan** <martin@is-thought.co.uk>

Abstract

Localization of text to multiple target languages has always presented unique challenges. Although it is easy to translate a single version of a document into different languages, it is much more difficult to maintain translations over multiple versions of a document. First you need to find out exactly what has changed, then find the equivalent places in the translated document and only then begin the update.

With the move to representing documents as structured XML comes a new set of technical solutions based around intelligent three-way structured merging - synchronization. This paper presents a comprehensive and generic solution to the problems of managing changing "source language" documents and propagating those changes to translated versions in multiple languages.

By processing the original source-language document, the original translated version and the updated source-language text, we demonstrate how a "template" document can be generated which is very much easier to process for re-translation. This merged document has the original translation for any unchanged elements, and a template for any modified or new sections. By identifying precisely what needs attention and possible re-translation, and providing source-language and original-translation versions of the text, re-translation of complex documentation sets managed by large teams becomes practicable. End-user interactive editing tools can be built which simply process the resulting XML document.

We discuss the technical advantages of using Unified Deltas to archive files as well as the cost reductions and reduced time to market made possible by giving the translation team what they want - clear and simple indications of what has changed with "in situ" display of context. By identifying both what has changed and the exact location of the change, our technique allows translators to concentrate on what they do best - translating. Updating translations becomes less expensive, less prone to error, and a lot less tedious.

We present examples from real-world translations, and discuss extensions to the basic scheme outlined here. We show how this work builds upon existing intelligent XML change control software, allowing merging of multiple XML documents into a single document as well as tracking of changes in document structure and content. This technique offers a revolution in text translation which should be of interest to anyone involved in test translation - all made possible by the adoption of XML.

Table of Contents

1. Beyond Babel - Simplifying Translation with XML

1.1. Translation: the hidden cost of globalization

As markets become globalized, more and more companies are facing the problem of maintaining multilingual documentation. Where, in the past, one person would update a single file when a small error was noted in a document, today procedures have to be in place to ensure that any changes made to source language documentation are reflected in all translations of the document. Translation is typically outsourced, outside the immediate control of the documentation department, and timescales for translation will differ from supplier to supplier. Instructions need to be issued to a whole team of translators, and the results of their work need to be integrated back into the company's document repository before new releases of the documentation can be considered.

The first problem encountered when translating updates to a source file is how to identify where changes have been made. Even if programs with change tracking facilities are used to update the source, identifying where each change has occurred, and ensuring that a matching change is made to the translated files, is a task that can be prone to error. For structured documentation, techniques that ensure the correct update of affected elements are essential. Yet many XML-based editing tools do not provide even basic facilities for tracking change. Users of such programs must rely on XML-aware differencing engines, such as DeltaXML, to identify where changes have been made.

The second problem faced by translators is to minimize re-translation of already translated text. First the translators need to identify the point in the original translation file where the change occurred. Then they need to ensure that any altered text is accurately replaced by the relevant translation. Here users of XML have an advantage. Typically translations have the same basic structure as the source file. If you know the XML path (XPath) to the translated element there is a strong possibility that the XPath for the translation is the same. Synchronization of changes at paragraph level is safer in an XML environment than in an unstructured environment.

But what happens if the changes involve the addition of elements, perhaps resulting in the introduction of a new level in the markup tree or the addition of a new sibling at a higher level? Synchronizing changes to the structure of a source file is a much more complicated process. Some XML-based differencing engines report any changes to the path of an element as a difference. Advanced XML-aware differencing engines, such as DeltaXML, are able to resynchronize changes even when elements have changed their position within the parent element. By adopting such tools, translators can accurately identify the minimal set of changes needed to adjust the structure of their translations, a vital first step to ensuring that a matching change is made in the translated file.

To correctly update a structured document, a translator needs to be able to identify:

1. all elements in the original file whose contents have changed in the updated file
2. any elements that have been added to the updated file
3. any elements which have been removed from the original file
4. any words that have been added to the original file
5. any words that have been removed from the original file
6. any words whose position in the updated file differs from that in the original file.

This latter point can be difficult to determine accurately. Removal of an existing word, or addition of new words, can result in the position of existing words changing. It is not sufficient to check if the n^{th} word in an updated element matches the nth word in the original. Accurate difference identification requires the checking of word sequences to identify where the order of words has changed. Small changes, such as the addition of a punctuation symbol, can easily be overlooked when visually examining the amended text, yet to a translator the identification of such changes is vital.

Finding where translation is required is only part of the problem. Translators also need to ensure that changes in the document structure are accurately propagated into the translated file. This requires that elements whose structure needs to be updated are clearly flagged within the translation, and that the relevant structural changes are indicated in a form that makes updating the translation a simple process, involving minimal work on the part of the translator.

DeltaXML's unique capability to synchronize changes between multiple files to create "unified deltas" provides the basis for a simple-to-use technique for automating translation management. DeltaXML's ability to apply keys to align elements whose position has changed in the data structure ensures that only significant changes are reported. In addition, the DeltaXML Translation Assistant allows the "translation units" for which retranslation is to be requested to be controlled during data extraction.

1.2. Simplifying translation management using XML-savvy synchronization

DeltaXML's solution to simplifying translation management is based on three key operations:

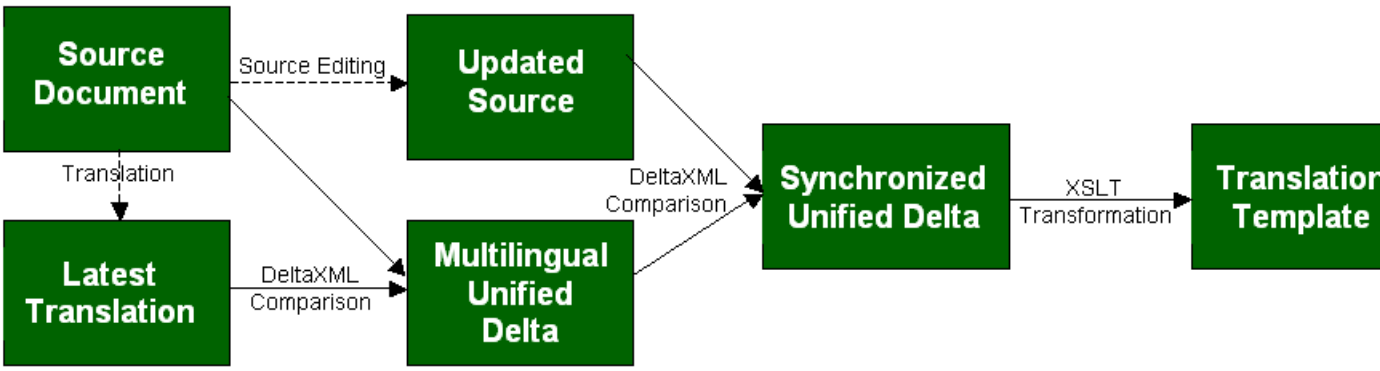
1. Comparing differences in document structures.
2. Re-synchronizing comparisons at word boundaries.
3. Merging changes in document structures and contents.

These three functions are integrated within DeltaXML's Unified Delta multiple-file archiving facility, as described in our XML 2003 paper on "Russian and XML: Designing Multi-Version XML Documents" [XML 2003]. As each new version of a file is produced changes (deltas) to the contents or structure of elements are identified to provide a minimal set of extensions to the Unified Delta file used to archive all versions of the document as a single file. Typically a single Unified Delta file will be used to archive versions of a document produced in a single language. But Unified Deltas can provide an alternative approach to document archiving, allowing a single file to contain all translations of a single version of a document. Using such multilingual Unified Delta files it is possible to store all current translations in a single source file without having to repeat the markup, or those parts of the document, such as code examples, that do not change during translation.

An XML-savvy translation management system can be built using Unified Delta files. The process involves:

1. Identifying the previous translation and using that as the base for a Unified Delta file.
2. Merging the translation with the original source document (which will typically have an identical structure but different contents).
3. Identifying changes made to the source document since the translation was completed.
4. Removing unchanged source document elements, leaving the existing translation as the element content.
5. Removing original and translation elements where an element has changed, leaving untranslated content from the updated source file.

For users familiar with revision control systems such as CVS this process, illustrated in the following diagram, is comparable to the conflict resolution process when multiple users checkout the same version and the commit their (conflicting) changes.



1.3. How did we manage that?

Before demonstrating how Unified Delta files can be used to simplify the translation process it is important to understand how changes are represented within Unified Delta files. The following set of "source files" illustrate the types of changes that typically occur in XML documents. They demonstrate:

1. changes to element content

2. changes to element structure.

In this paper we are not illustrating changes to attribute values, whose handling was explained in the paper presented at XML 2003 [XML 2003].

Initially a Unified Delta file is created by adding two attributes to the root element of the base (first) version of a file. The first attribute is a namespace identifier assigned to files processed using the Unified Delta suite:

```
  xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
```

The second attribute, `dxu:vset`, stores the identifier of the base version of the document, between a pair of vertical bars (the pipe symbol). As each revision is made to the file a vertical bar and a new version identifier are added to the start of the list. For example, when three versions of a file have been produced the attribute value on the root element might have the form:

```
  dxu:vset="|1.2|1.1|1.0|"
```

The last version identifier in the list always indicates the base version of the document, while preceding version indicators identify the order in which changes have been added to the archive.

During processing a `dxu:vset` attribute is assigned to each embedded element, the values being inherited from the parent element if none have been assigned locally. The archive file, however, only records version information in those cases where the value of the attribute differs from that of the `dxu:vset` attribute of its parent element. In all cases the value of a `dxu:vset` attribute will be a subset, or implied copy, of the values assigned to the parent element.

The following base version will form the basis of the short examples used to demonstrate the creation of Unified Delta files:

```
<statement made-by="Robin" approved-by="Thomas"
           xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
  <p id="p1">DeltaXML allows you to:
   <list type="bulleted">
    <item>Identify and display changes in
     <acronym meaning="Extensible Markup Language">XML</acronym>
     files</item>
    <item deltaxml:key="last">Manage changes to XML documents.</item>
   </list>
  </p>
</statement>
```

We have added a `deltaxml:key` attribute to the last item in the list as we want it to remain the last item, irrespective of changes made to the contents of the list. The `deltaxml` namespace prefix definition has, therefore, been added to the root element.

This `statement` must be assigned, during the creation of the archive file, an identifier, such as `1.0`, that is to be used to distinguish this version from others in the archive. When the archive file is initially created it will have the form:

```
<statement xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
           xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1" dxu:vset="|1.0|"
```

```
            approved-by="Thomas" made-by="Robin">
  <p id="p1">DeltaXML allows you to:
   <list type="bulleted">
    <item>Identify and display changes in
     <acronym meaning="Extensible Markup Language">XML</acronym>
     files</item>
    <item deltaxml:key="last">Manage changes to XML documents.</item>
   </list>
  </p>
</statement>
```

Note that the attributes of the root element have been augmented by the Unified Delta namespace declaration and a `vset` attribute.

1.3.1. Recording changes to element content

If one or more words in a section of the document whose model is PCDATA (parsed character data) in the XML file are changed, a pair of `dxu:PCDATA` elements are generated to record the change(s) made. In the case of mixed content, where one or more elements are embedded within data streams, as shown in the first item of the example file, each segment of parsed character data is compared separately. For example, if the second version of our statement, identified as version `1.1`, is altered to read:

```
<statement made-by="Robin" approved-by="Thomas">
  <p id="p1">DeltaXML allows you to:
   <list type="bulleted">
    <item>Identify, display and process changes in
     <acronym meaning="Extensible Markup Language">XML</acronym>
     files</item>
    <item deltaxml:key="last">Manage changes to XML documents.</item>
   </list>
  </p>
</statement>
```

the resulting archive will have the form:

```
<statement
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
            xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1" dxu:vset="|1.1|1.0|"
            approved-by="Thomas" made-by="Robin">
  <p id="p1">DeltaXML allows you to:
   <list type="bulleted">
    <item>
     <dxu:PCDATA dxu:vset="|1.1|">Identify, display and process changes in </dxu:PCDATA>
     <dxu:PCDATA dxu:vset="|1.0|">Identify and display changes in </dxu:PCDATA>
     <acronym meaning="Extensible Markup Language">XML</acronym>
     files
    </item>
    <item deltaxml:key="last">Manage changes to XML documents.</item>
   </list>
  </p>
</statement>
```

Note that the second part of parsed character data for the first item, like the text for the second item, has not been changed. As it applies to all versions of the archived file it is not enclosed in a `dxu:PCDATA` element.

1.3.2. Recording changes to element structure

Changes in element structure take place in three clearly clearly distinguishable circumstances:

1. The addition of a new element between existing elements.

2. The addition of a new element within PCDATA to create mixed content.

3. The removal of an existing element.

The following version of our statement, identified as `1.2`, illustrates how Unified DeltaXML handles the three main types of changes to element structure:

```
<statement made-by="Robin" approved-by="Thomas">
  <p id="p1"><emphasis>DeltaXML</emphasis> allows you to:
   <list type="bulleted">
    <item>Identify, display and process changes in XML files</item>
    <item>Synchronize concurrent edits</item>
    <item deltaxml:key="last">Manage changes to XML documents.</item>
   </list>
  </p>
</statement>
```

In this example a new element has been used to emphasize part of the contents of the paragraph heading. In the embedded list the embedded `acronym` element in the first item has been removed, and a new item has been added between the two items listed in the first two versions.

When this version is used to update the archive containing versions `1.0` and `1.1`, the following unified archive is created:

```
<statement
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
            xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1" dxu:vset="|1.2|1.1|1.0|"
            approved-by="Thomas" made-by="Robin">
  <p id="p1"">
   <emphasis dxu:vset="|1.2|">DeltaXML</emphasis>
   <dxu:PCDATA dxu:vset="|1.2|"> allows you to: </dxu:PCDATA>
   <dxu:PCDATA dxu:vset="|1.1|1.0|">DeltaXML allows you to: </dxu:PCDATA>
   <list type="bulleted" >
    <item>
     <dxu:PCDATA dxu:vset="|1.2|">Identify, display and process changes in XML files</dxu:PCDATA
     <dxu:PCDATA dxu:vset="|1.1|">Identify, display and process changes in </dxu:PCDATA>
     <dxu:PCDATA dxu:vset="|1.0|">Identify and display changes in </dxu:PCDATA>
     <acronym  dxu:vset="|1.1|1.0|" meaning="Extensible Markup Language">XML</acronym>
     <dxu:PCDATA dxu:vset="|1.1|1.0|"> files</dxu:PCDATA>
    </item>
    <item dxu:vset="|1.2|">Synchronize concurrent edits</item>
    <item deltaxml:key="last">Manage changes to XML documents.</item>
   </list>
  </p>
</statement>
```

There are now three versions for the first fragment of PCDATA in the first item, and the second fragment of PCDATA, and the embedded `acronym` element, are now flagged as only applying to versions `1.0` and `1.1` of the file. There are two versions of the text of the paragraph heading, the first applying only to version `1.2` while the second applies to both version `1.0` and `1.1`.

The last item, however, has no `dxu:vset` attribute as it shares the value assigned to the `list` element, which shares the value assigned to its parent, `p`, which inherits its values from the root element, `statement`. Note that, though this text was originally the second item in the list, no change to the text of the second item has been reported. This is because DeltaXML keys allow changes in order within a set of siblings to be ignored.

1.3.3. Why the additional markup?

At this stage you may be wondering about the efficiency of the archives when so many new elements have been added to the file. In practice few files will contain the number of changes deliberately made in this single paragraph

example. We have set out to demonstrate four different types of change to a file within a single paragraph and have, only left one element unchanged.

Typically only a small percentage of paragraphs or embedded elements are changed during a revision cycle. Most elements will be unchanged. Elements that never change are stored only once, in their original form (as the last `item` shows). Elements that are removed have an attribute added that shows which versions of the file they were used in. Elements that are added have a single attribute added to their contents, whose value is extended each time a new version is added. It is only changes to content that produce extra markup, and this is minimized to a single namespace-controlled element and attribute per altered PCDATA segment or attribute value. The result is a very compact representation of the differing versions.

It should be noted that the basic form of the archive is controlled by the form of the base document. Once the elements controlled by the DeltaXML namespaces are removed the tree structure of the document becomes a combination of all the trees used for the multiple versions. Each version of the embedded file can be recreated by selecting those elements whose actual or implied `dxu:vset` attribute contains the relevant version identifier.

1.3.4. Storing translations as different versions

Our integrated approach to version archiving means you can transmit all translations of a particular document as a single XML file. In this scenario the source language is stored as the base version of the document, and each translation is stored as a delta to the source. Version identifiers show how many languages the document has so far been translated into. The following example shows how version 1.2 of our statement could be stored as a multilingual version:

```
<statement
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
            xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
            dxu:vset="|German|French|English|"
approved-by="Thomas" made-by="Robin">
  <p id="p1">
   <emphasis>DeltaXML</emphasis>
    <dxu:PCDATA dxu:vset="|German|"> erlaubt: </dxu:PCDATA>
    <dxu:PCDATA dxu:vset="|French|"> vous permet: </dxu:PCDATA>
    <dxu:PCDATA dxu:vset="|English|"> allows you to: </dxu:PCDATA>
   <list type="bulleted">
    <item>
     <dxu:PCDATA dxu:vset="|German|">Kennzeichnung, Anzeige an und verarbeite von Anderungen an
     <dxu:PCDATA dxu:vset="|French|">d'identifier et de visualiser les changements dans les fich
     <dxu:PCDATA dxu:vset="|English|">Identify, display and process changes in XML files</dxu:PC
    </item>
    <item>
     <dxu:PCDATA dxu:vset="|German|">Synchronisieren Sie Gleichlaufendes redigiert</dxu:PCDATA>
     <dxu:PCDATA dxu:vset="|French|">de synchroniser les modifications faites simultanement</dxu
     <dxu:PCDATA dxu:vset="|English|">Synchronize concurrent edits</dxu:PCDATA>
    </item>
    <item deltaxml:key="last">
     <dxu:PCDATA dxu:vset="|German|">Handhabung von Anderungen an den XML-Dokumenten.</dxu:PCDAT
     <dxu:PCDATA dxu:vset="|French|">de gerer les changements dans les documents XML.</dxu:PCDAT
     <dxu:PCDATA dxu:vset="|English|">Manage changes to XML documents.</dxu:PCDATA>
    </item>
   </list>
  </p>
</statement>
```

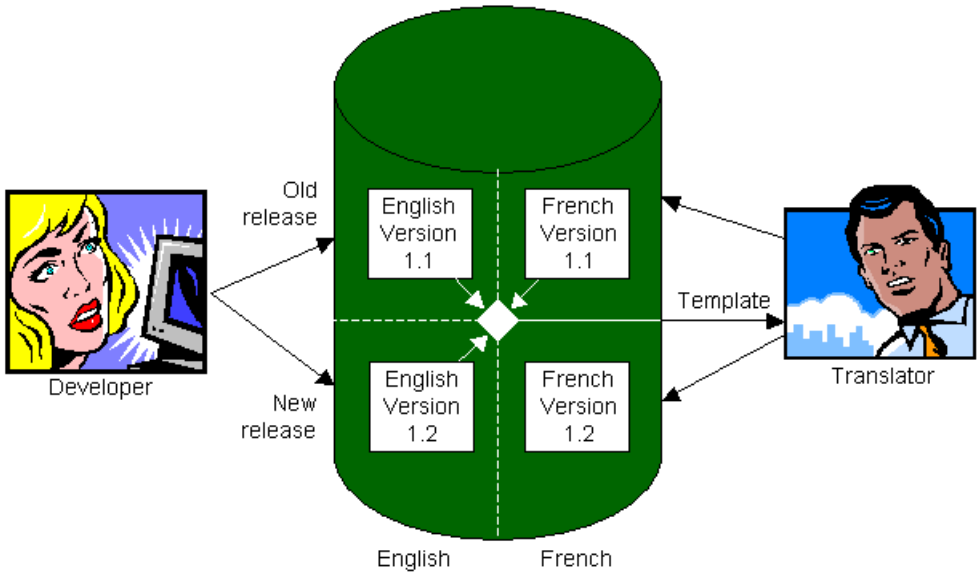
1.3.5. Updating a specific language version

When we need to update translations to reflect changes made to the master language all we need to do is:

1. Extract the current version of the foreign language to be updated, and make that the base version for the Unified Delta file.

2. Extract the version of the master file from which the last translation was taken (normally the one with the same version number as the translation) and add that to the Unified Delta file.

3. Extract a subsequent update to the master language and add this to the Unified Delta file.



If our source files are version 1.1 of our demonstration file, and we want to create a template that can be used to create a translation for the French edition of version 1.2, we can create the following Unified Delta:

```
<statement xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
           xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
           dxu:vset="|English-1.2|English-1.1|French-1.1|"
           approved-by="Thomas" made-by="Robin">
  <p id="p1">
   <emphasis dxu:vset="|English-1.2|">DeltaXML</emphasis>
   <dxu:PCDATA dxu:vset="|English-1.2|"> allows you to:
</dxu:PCDATA>
   <dxu:PCDATA dxu:vset="|English-1.1|">DeltaXML allows you to: </dxu:PCDATA>
   <dxu:PCDATA dxu:vset="|French-1.1|">DeltaXML vous permet: </dxu:PCDATA>
   <list type="bulleted">
    <item>
     <dxu:PCDATA dxu:vset="|English-1.2|">Identify, display and process changes in XML files</dx
     <dxu:PCDATA dxu:vset="|English-1.1|">Identify and display changes in </dxu:PCDATA>
     <dxu:PCDATA dxu:vset="|French-1.1|">d'identifier et de visualiser les changements dans les
     <acronym  dxu:vset="|English-1.1|French-1.1|" meaning="Extensible Markup Language">XML</acr
     <dxu:PCDATA dxu:vset="|English-1.1|">files</dxu:PCDATA>
    </item>
    <item dxu:vset="|English-1.2|">Synchronize concurrent edits</item>
    <item deltaxml:key="last">
     <dxu:PCDATA dxu:vset="|English-1.2|English-1.1|">Manage changes to XML documents.</dxu:PCDA
     <dxu:PCDATA dxu:vset="|French-1.1|">de gerer les changements dans les documents XML.</dxu:P
    </item>
   </list>
  </p>
</statement>
```

While this version of the file clearly displays both the original and updated versions of the source document, and the last version of the translation, it can be somewhat overpowering for translators! A number of techniques can be used to overcome this, including:

1. Converting the file to a browsable form with the different versions shown in different colours (e.g. green for unchanged source, red for changed source and blue for translated text).

2. Extracting the different versions for display in different windows, which can be synchronized given suitable editing tools.

3. Creating a revised source file containing the previously translated version of all unchanged elements and the revised source language version of all changed elements.

For our example file this last process could result in a file of the form:

```
<statement xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
           xmlns:dxu="http://www.deltaxml.com/ns/unified-delta-v1"
           xmlns:dxt="http://www.deltaxml.com/ns/translate-v1"
           approved-by="Thomas" made-by="Robin">
  <p id="p1" dxt:translated="no">
   <emphasis dxt:translated="no">DeltaXML</emphasis>allows you:
<list type="bulleted">
    <item dxt:translated="no">Identify, display and process changes in XML files</item>
    <item dxt:translated="no">Synchronize concurrent edits</item>
    <item deltaxml:key="last">de gerer les changements dans les documents XML.</item>
   </list>
  </p>
</statement>
```

The elements to be translated are clearly identified by a `dxt:translated="no"` attribute. In this heavily edited example many of the elements have been flagged as needing retranslation, but note that only those elements whose contents have been changed have been flagged in this way. In practice such flags will only be applied to a small proportion of the elements in a file, allowing changed elements to be quickly identified using the most basic of searching tools.

As the file is translated the `dxt:translated="no"` attributes can be deleted, leaving the file in the original state, except for the additional namespace declarations on the root element, which in no way affect the processing of the file. Alternatively a stylesheet can be used to remove the namespace-controlled attributes and the added namespace declarations once translation has been completed. The translated file is then in a form that can be used to update the Unified Delta archive for the language concerned, as well as providing the basis for the production of a revised version of the document.

While the simplified format shown above can be sufficient for basic translations there will be occasions where elements from both source files need to be retained for the translation template. In addition, some degree of control is required to ensure that only changes to relevant "translation units" are reported. The DeltaXML Translation Assistant will allow you to identify which elements should be treated as translation units, e.g. paragraphs and list items, so that changes to embedded elements do not get reported out of context.

1.4. The benefits of structured data comparison

Most permanent documents have multiple versions. Identifying the differences between versions can be a complex task, which is prone to error. With structured data, such as that found in files coded using the Extensible Markup Language (XML), the problem is compounded by the fact that changes in markup and inter-element spacing may not be significant. Most version control systems (for example CVS or RCS) operate on a textual, line-by-line basis, and do not provide facilities for accurately identifying differences between structured documents.

Change management of XML files can be simplified by using DeltaXML's Unified Delta API to create an archive that records the changes (deltas) made as each new version of a file is created. A Unified Delta file consists of an initial base version to which have been added a set of deltas that show changes made to elements, attributes or content required to generate alternative versions. By restricting change records to the smallest changed component (a string of parsed character data or the smallest containing element of mixed content), and only storing unchanged data in the base version, the size of Unified Delta files is minimized.

The Unified Delta suite optimizes the storage of multiple XML versions by using the DeltaXML differencing engine to identify true differences between structured documents. Differencing engines based on string comparison cannot identify when the only differences between two versions of a file are cosmetic ones such as the order of attributes or a change in an XML namespace prefix, which have no impact to the way in which XML-encoded

data can be processed. Differencing engines that do not understand structural context are unable to follow the structural correspondence between documents, and cannot be used to propagate structural changes across language versions.

Because Unified Delta files are XML files they can be processed using XML toolkits. In particular, they can be restructured and formatted using XML Stylesheet Language Transformations (XSLT). They can also be referenced using XPath or XPointer, XML's standard tools for identifying XML components. Such transformations can be used to:

1. Convert files into a form that can be displayed by any web browser.

2. Extract different versions for display in different windows.

3. Create a revised source file containing all unchanged elements and all changed elements.

While translation services can be developed using all three approaches, the third approach is the only one that will ensure that structural changes made to the XML tree will be accurately reflected in translated files, significantly reducing the possibility of human error during the identification of change points and their transfer to multiple translations of the source file.

The DeltaXML Translation Assistant will be supplied as a Java API. for use as part of a piped set of operations invoked during document processing. Integration with XML editors will provide easy-to-use editing environments based on automated integration of existing translations and new material.

1.5. Conclusions

This paper describes a new approach to managing translation that takes advantage of the power of XML, using XML-aware comparison tools to identify precisely what has changed between versions and to simplify subsequent re-translation. The process described removes the difficult and error-prone hand manipulation previously required to update translated documents, allowing a simpler and more creative translation process. Speeding the process of updating content in multiple target languages while improving quality and manageability, the strategy outlined here allows rigorous change control when delivering content to a global audience. Throughout, the process is powered by XML.

Bibliography

[XML 2003] "Russian Dolls and XML: Designing Multi-Version XML Documents", *Proceedings of XML 2003 Conference*, IDEAlliance, December 2003 (copy available from http://www.deltaxml.com/unified/multi-version-paper.html).

Biography

Robin **La Fontaine**
CEO
DeltaXML (Monsell EDM Ltd)
Upton-on-Severn
United Kingdom
robin.lafontaine@deltaxml.com

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
United Kingdom
thomas.nichols@deltaxml.com

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.

Martin **Bryan**
IS-Thought
Churchdown
United Kingdom
martin@is-thought.co.uk

Martin Bryan has been a technical author and software tester for more than two decades. He has represented the UK on international standards committees working on structured documentation standards such as SGML, DSSSL and Topic Maps for most of this period, and has been actively involved in promoting the use of XML and related standard within Europe for the last decade.