Merging XML files: a new approach providing intelligent merge of XML data sets

Robin La Fontaine, Monsell EDM Ltd
robin.lafontaine@deltaxml.com
http://www.deltaxml.com

Abstract

As XML becomes ubiquitous so the need for powerful tools to manipulate XML data becomes more pressing. Merging XML is particularly tricky, but often necessary to consolidate data feeds from heterogeneous systems, or to synchronize submissions of XML fragments which make up a larger document. An automated mechanism for defining and controlling such merges has been developed and is demonstrated to provide a consistent, adaptable and resilient solution to this problem. Integration into an information pipeline allows limitless customization.

As XML tools become more powerful and able to handle many of the peculiarities of real data, so the possibility of achieving a genuine, intelligent merge of XML data sets becomes a reality. Increasingly users are wanting to apply concurrent engineering to XML, i.e. to allow multiple users to add to a single data set simultaneously.

This paper proposes a systematic approach to merging based on the use of an intermediate XML file that contains both of the files to be merged in a formal structure that clearly identifies data that is common to both files and data that is unique to one of the files. The advantage of this intermediate file is that many of the conflicts that typically emerge when XML data is merged can be identified and resolved. The resolution of these conflicts is a key to achieving a useful merge.

The paper addresses issues of real data in terms of how to control the correct correspondence between the data within the files. Finding this correspondence is a necessary step before a sensible merge can be executed. Applying these techniques means that XML files containing libraries of data, e.g. XML Schemas or SVG files, can be intelligently merged in an automated way. This improves quality and reduces the required human effort involved in these essential processes.

The proposed method provides XML users with a general-purpose merge operation for the amalgamation of XML data and thus gives another reason for adopting XML as the preferred format for documents and data.

Introduction

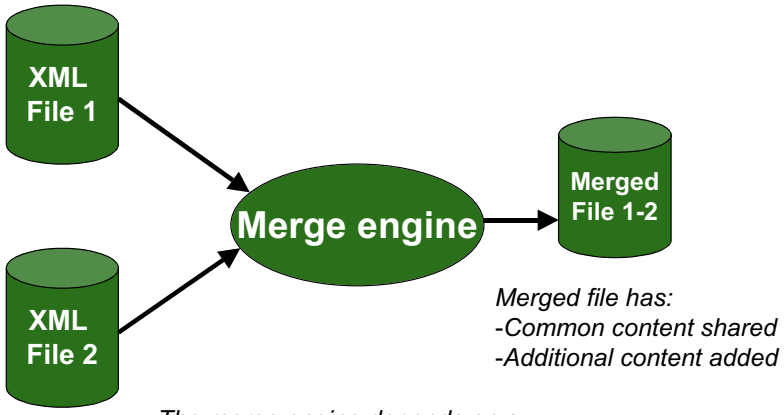
The requirement to merge XML data sets is becoming more widespread, and different solutions are emerging. It is evident that different problems exist in different usage scenarios, and there is a need for a flexible solution, which can be adapted to different needs. This paper presents a method based on a structured intermediate XML file which provides an opportunity to make adjustments to the data before the final commitment to the merged result. This enables specific requirements to be met using standard XML tools and access methods.

At XML Europe last year, 2001, Manger presented a paper on merging [Manger2001] and the discussion after this presentation made it clear that there is a large number of different use cases in this area. A more recent publication [Lindholm2001] provides a very thorough review of the current state together with an algorithmic solution for merging XML documents and data. The paper omits DeltaXML [LaFontaine2001]

as a possible implementation solution. DeltaXML is a commercial XML comparison method, developed over several years, and it provides a structured intermediate file very suitable as a basis for a merge operation. Independent assessments indicate that DeltaXML provides a fast and accurate implementation for XML comparison [Cobena2002].

Requirements for Merge

There are two separate but related merging problems: the 2-way merge and the 3-way merge. The difference between these depends on whether there are two files to be merged or whether there is also a 'base' file from which the others are derived. The 3-way merge has the potential to provide a more accurate solution where a base file exists, but it is more complex. There are use cases for both of these because it is often the case that only two files exist, and there is no original 'base' file. In this paper we shall deal first with the 2-way merge and then extend this to the more complex and powerful 3-way merge. In both cases structured intermediate XML files provide great flexibility to the solution.



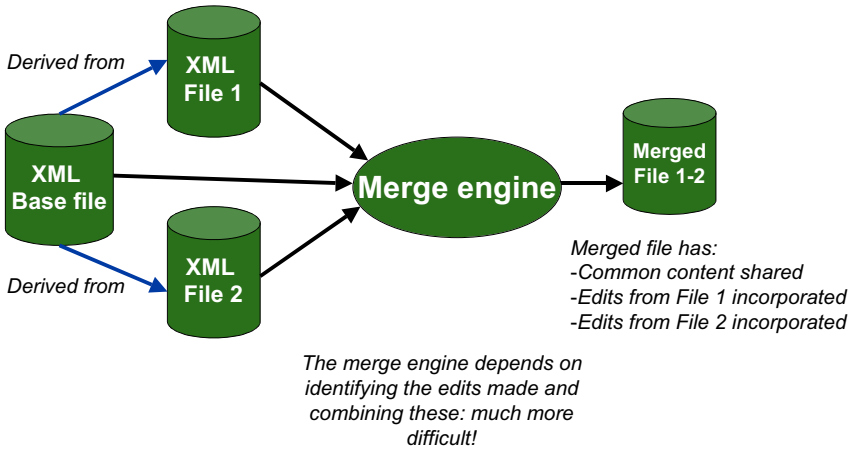
2-way merge

For a 2-way merge, the basic requirements can be simply stated in an informal way: the merged document should contain everything from both the original documents, without duplication where there are overlaps. The difficulty is in defining where these overlaps occur and ensuring that these are handled correctly. Although every user has a clear idea of his/her own requirements, these tend to differ according the usage scenario.

The requirements must be defined in a more formal way, and for the purposes of this paper can be defined in a way that is specific to XML, although not specific to any particular use of XML. The basic requirements can be summarized as follows:

1. The attributes of any element should be the union of the attributes in the two files for the corresponding elements. Where there are conflicts, these need to be handled (see later discussion).
2. The child elements of any given element should be the ordered union of the child elements for the corresponding elements in the two files.
3. An intelligent comparison algorithm is required to identify corresponding elements in the two files at each level in the tree structure. It is preferable that this correspondence should be controllable, e.g. using a key, to avoid any ambiguity in situations where an accurate correspondence is required.
4. The merge should, preferably, be able to take account of both ordered and unordered elements.

5. Where PCDATA (text) nodes have changed, there is a choice between executing a merge of the contents or choosing one or other of the text representations.
6. Conflicts should be identified.



3-way merge

Additional requirements can be identified in the case of the 3-way merge, as follows:

1. Any change to an element from either branch should be present in the merged file.
2. Elements deleted in either branch should be deleted in the merged file.
3. Attributes deleted in either branch should be deleted in the merged file.
4. Conflicts should be identified, and these will be different from those of the 2-way merge.

Other merge solutions do have other requirements and there are two major influences on these.

The first is the use case for the merge and the results expected or required. For example, merging data with a very strict hierarchical structure tends to be a more rigorous process than merging documents, where the textual information may move within the XML tree structure.

The second is the effect on the result of the matching algorithm chosen to work out the basic correspondence between the two files: this is a fundamental driving force. An example of this is whether a 'move' operation is to be handled. Some systems will handle this and others do not. In this paper we will not consider the move operation because it introduces additional complexity and is to some extent overcome by the ability to perform unordered comparisons, i.e. to allow the child elements of a given element to appear in any order. Discussions with the author of a paper analysing the change in some 40,000 XML files from the web show that the move operation is not a common one [Cobena2002]. A move operation is more likely to apply to XML documents than XML data, where the structure tends to be more fixed.

Tree matching algorithm

The algorithm employed to match the two input XML trees has, as indicated above, an important influence on the result. In this paper we are using the DeltaXML comparison tool described at XML Europe 2001 [LaFontaine2001]. This comparison process is operating on the data as well-formed XML without knowledge of the structure from the DTD or Schema.

In order to understand the situations where a merge can be effective and where there are likely to be problems, it is useful to understand how the comparison process works. DeltaXML performs a tree-structured comparison walking through 'corresponding' nodes in the two XML trees. A key part of this process is how it finds, at any given level in the trees, which nodes 'correspond'.

The correspondence is established at any level in the tree as follows. First, the two lists of XML nodes in the two files are regarded as ordered unless a **deltaxml:ordered="false"** attribute is present on the parent element; this provides fine control, at each level, of how successive levels in the tree should be treated. Elements with PCDATA and element content are always considered ordered.

The problem, in the case of an ordered list of elements (possibly including PCDATA nodes) is to find the best, ordered, match between these at a given level in the tree. This is achieved using the Wu[1] algorithm for the LCS (Longest Common Subsequence). This algorithm is optimized for situations where most of the nodes correspond.

In order to provide full control over this process, any element may have a key attached to it. A keyed element will only be matched with an element of the same type (local name and namespace) that also has the same key value. The matching is applied to find the longest match of consecutive nodes in the two files at a given level, thus finding the LCS. This provides an anchor point and then the same process is applied to the list of nodes before the match, in both files, and the list of nodes after the match.

When there are no nodes with the same key values, the comparator looks for nodes that are exactly equal, i.e. they are nodes of the same element type and all their child nodes down through the tree match up exactly. In the case of text nodes, the text stings must be the same.

When there are no nodes that are equal, the comparator looks for nodes that are of the same type, i.e. they are nodes with elements of the same name, or they are text nodes. The longest common subsequence of these is then found and these nodes are considered to correspond between the two lists.

One of the consequences of this process is that two white space nodes, e.g. two newlines, will give an exact match and will be considered to be corresponding nodes. If these whitespace nodes are in fact insignificant then incorrect results may be obtained. In general, it is best to remove all white space using a simple XSL filter before comparing the files.

This approach executes in linear time and memory, which is especially important for larger files, and in the majority of cases produces an optimal matching between the two files [Cobena2002].

Intermediate delta file

Following the comparison process, we do not produce an edit script, which may be difficult to manipulate. Rather, a delta file is generated which has the following properties:

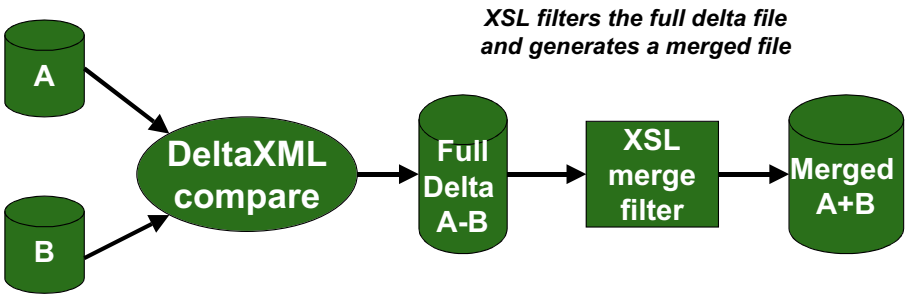
1. The delta file contains all of the data from the two files.
2. Data that is common between the two files is represented only once in the delta file.
3. Data that is present in one file and not the other is identified accordingly.
4. Changes to attributes are identified and represented as attributes.
5. A simple XSL script can generate either of the original files from the delta file.

This delta file provides a very good basis for the merge operation – much of the difficult work has been done at this stage in working out the correspondence between the two files. The delta file also has the advantage that, because it is a single XML file containing all the information from both original files, it is possible to traverse it and make adjustments or decisions about situations that may cause a merge conflict.

An example of such a problem, cited in [Manger2001] is where ID values need to be checked. If ID attributes are present in both files, then there is no guarantee that the merged file will have correct ID values. Checks need to be made first to ensure that the IDs in the merged file will be correct. The structured

[1] "An O(NP) Sequence Comparison Algorithm", Sun Wu, Udi Mander, Gene Myers, Univ. of Arizona, Tuscon, 15th September 1990, Information Processing Letters 35 (1990) 317-323

delta file provides an ideal environment for finding and addressing such problems. This is because it is simple to identify all the ID attributes from each file, and those that are shared, using Xpath statements, and thus to solve any anticipated conflicts.



Using the DeltaXML full delta file as the basis of a merge

The use of this intermediate file enables almost any specific requirements to be met by processing this file in some way, or changing the way the merged file is derived from the intermediate file.

Worked Examples for 2-way Merge

This section describes some worked examples to illustrate the merge process on small test files. This is not an exhaustive set of examples but will serve to illustrate the methodology and show how most cases can be handled automatically and special cases can be handled quite simply using special XSL filters.

Insertion

The first example is a simple insertion of an element. This example is from [Lindholm2001], example I1.

The two input files are as follows:

```
<R>
        <a>
                <i1 />
                <c />
        </a> <b>
                <d />
                <e />
        </b>
</R>
```

i1-t1.xml

```
<R>
        <a>
                <c />
        </a>
        <b>
                <d />
                <i2 />
                <e />
        </b>
</R>
```

i1-t2.xml

The result of merging these two files is as follows:

```
<R>
        <a>
                <i1 />
                <c />
        </a>
        <b>
                <d />
                <i2 />
                <e />
        </b>
</R>
```

Merge of i1-t1.xml and i1-t2.xml: i1-m.xml

This shows that <i1> has been added from the first file and <i2> has been added from the second file. If we look in a little more detail at this process we can see how it works. First, the two input files are compared and a delta file generated. This delta file is what is termed, in DeltaXML, a 'full delta' in that it has not only the changed data but also the unchanged data. All the examples are pretty-printed for clarity but in general DeltaXML does not generate any additional whitespace in delta files.

```
<R xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
deltaxml:delta="WFmodify">
        <a deltaxml:delta="WFmodify">
                <i1 deltaxml:delta="delete" />
                <c deltaxml:delta="unchanged" />
        </a>
        <b deltaxml:delta="WFmodify">
                <d deltaxml:delta="unchanged" />
                <i2 deltaxml:delta="add" />
                <e deltaxml:delta="unchanged" />
        </b>
</R>
```

Full delta comparing i1-t1.xml and i1-t2.xml: i1-full.xml

The full delta file represents all the data from both input files in such a way that:

- the common data is shared,
- data from i1-t1.xml which is not present in i1-t2.xml is shown as deleted (using an attribute deltaxml:delta="delete" on the element), and
- data from i1-t2.xml which is not present in i1-t1.xml is shown as added (using an attribute deltaxml:delta="add" on the element)

This structured representation of the two files is an excellent basis for a merge. All that is required is to generate a file from this where both the deleted and added data is present, i.e. the merged file contains all the data from both the input files. This can be achieved using a simple XSL script, and gives the result shown above.

XHTML Example

We will now consider a slightly larger example using XHTML to show how the addition of elements is handled to generate a merged result with all additional elements included.

Consider a base file in XHTML as follows:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p>This is the first paragraph</p
><p>This is the second paragraph</p
><p>This is the third paragraph</p
></body>
</html>
```

File A.xml

Notice that there is no whitespace between the <p> elements and this is deliberate because whitespace here is significant and would normally be considered part of the data. The removal of the whitespace could also have been achieved using an XSL stylesheet on input, as mentioned above.

In our example, the above file has been edited by two separate people who have each added elements. There are therefore two new files derived from A.xml as follows:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p>This is the first paragraph</p
><p>This is the second paragraph</p
><p>This is the third paragraph</p
><p>This is a paragraph added after the third paragraph</p
><p>This is another paragraph added after the third paragraph</p
></body>
</html>
```

File B.xml

And:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p>This is the first paragraph</p
><p>This is the paragraph added after the first paragraph</p
><p>This is the second paragraph</p
><p>This is the third paragraph</p
></body>
</html>
```

File C.xml

Our task now is to merge these two files so that we get the additional paragraphs from each in the merged output file.

First, we use DeltaXML to do a comparison and produce a full delta output. The result of this is the following file:

```
<p0:html xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
xmlns:p0="http://www.w3.org/1999/xhtml" deltaxml:delta="WFmodify">
   <p0:head deltaxml:delta="unchanged">
      <p0:title>Untitled</p0:title>
      <p0:meta name="generator" content="BBEdit 6.1.2"/>
   </p0:head>
   <p0:body deltaxml:delta="WFmodify">
      <p0:p deltaxml:delta="unchanged">This is the first paragraph</p0:p>
      <p0:p deltaxml:delta="add">This is the paragraph added after the first
paragraph</p0:p>
      <p0:p deltaxml:delta="unchanged">This is the second paragraph</p0:p>
      <p0:p deltaxml:delta="unchanged">This is the third paragraph</p0:p>
      <p0:p deltaxml:delta="delete">This is a paragraph added after the third
paragraph</p0:p>
      <p0:p deltaxml:delta="delete">This is another paragraph added after the
third paragraph</p0:p>
   </p0:body>
</p0:html>
```

Full delta file showing differences between B.xml and C.xml, pretty-printed

DeltaXML always uses a prefix for any element with a namespace, and in this case the 'p0' prefix has been generated because no prefix was present in the input files. Of course the prefix is not important to the XML parser, which will handle it automatically to present the correct namespace to the application.

In this full delta file you can see where the differences are noted in the <p> elements. The elements that have been added between A.xml and B.xml are shown with a **deltaxml:delta="add"** attribute and those that have been added between A.xml and C.xml are shown with a **deltaxml:delta="delete"** attribute. They are 'deleted' because they are present in C.xml but not in B.xml.

We can now process this file to generate a merged file as follows:

```
<p0:html xmlns:p0="http://www.w3.org/1999/xhtml"
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
        <p0:head>
                <p0:title>
                        Untitled
                </p0:title>
                <p0:meta name="generator" content="BBEdit 6.1.2" />
        </p0:head>
        <p0:body>
                <p0:p>
                        This is the first paragraph
                </p0:p>
                <p0:p>
                        This is the paragraph added after the first paragraph
                </p0:p>
                <p0:p>
                        This is the second paragraph
                </p0:p>
                <p0:p>
                        This is the third paragraph
                </p0:p>
                <p0:p>
                        This is a paragraph added after the third paragraph
                </p0:p>
                <p0:p>
                        This is another paragraph added after the third paragraph
                </p0:p>
        </p0:body>
</p0:html>
```

Merged file for B.xml and C.xml, pretty-printed

In this case, we have achieved what we needed in terms of a merged file. If any other edits had been made then the merge would not have worked so well because the comparator would not have been able to identify the equivalent nodes between the two files (see later discussion for how this problem can be solved). And in some other cases, for example where both files have additions in the same place, the order of the elements in the merged file needs to be considered: it may or may not be important.

In the above example the deltaxml attributes have been left in, but these could easily have been stripped out during the merge process.

Adding Attributes

Attributes can be added and the merged file will contain the attributes added in both files. Consider additional attributes as follows. First, a new class attribute has been added to the <body> element.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body class="overview"><p>This is the first paragraph</p
><p>This is the second paragraph</p
><p>This is the third paragraph</p
><p>This is the fourth paragraph</p
><p>This is the fifth paragraph</p
></body>
</html>
```

File E.xml with new class attribute on <body> element

Similarly, another edit adds an attribute to a <p> element:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p>This is the first paragraph</p
><p>This is the second paragraph</p
><p class="important">This is the third paragraph</p
><p>This is the fourth paragraph</p
><p>This is the fifth paragraph</p
></body>
</html>
```

File F.xml with new class attribute on <p> element

Now the full delta output for the comparison of these two files yields this result:

```
<p1:html xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
xmlns:p1="http://www.w3.org/1999/xhtml" deltaxml:delta="WFmodify">
        <p1:head deltaxml:delta="unchanged">
                <p1:title>
                        Untitled
                </p1:title>
                <p1:meta name="generator" content="BBEdit 6.1.2" />
        </p1:head>
        <p1:body deltaxml:delta="WFmodify" deltaxml:old-
attributes="class=&amp;quot;overview&amp;quot;">
                <p1:p deltaxml:delta="unchanged">
                        This is the first paragraph
                </p1:p>
                <p1:p deltaxml:delta="unchanged">
                        This is the second paragraph
                </p1:p>
                <p1:p deltaxml:delta="WFmodify" deltaxml:new-
attributes="class=&amp;quot;important&amp;quot;">
                        This is the third paragraph
                </p1:p>
                <p1:p deltaxml:delta="unchanged">
                        This is the fourth paragraph
                </p1:p>
                <p1:p deltaxml:delta="unchanged">
                        This is the fifth paragraph
                </p1:p>
        </p1:body>
</p1:html>
```

Full delta file showing differences between E.xml and F.xml

The added attributes are denoted by the **deltaxml:new-attributes** attribute for class="important", i.e. the attribute added in F.xml but not present in E.xml, and by **deltaxml:old-attributes** for the attribute class="overview" which was in E.xml but not in F.xml.

The result of applying the merge XSL stylesheet to this full delta file gives the following merged file:

```
<p1:html xmlns:p1="http://www.w3.org/1999/xhtml"
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
        <p1:head>
                <p1:title>
                        Untitled
                </p1:title>
                <p1:meta name="generator" content="BBEdit 6.1.2" />
        </p1:head>
        <p1:body class="overview">
                <p1:p>
                        This is the first paragraph
                </p1:p>
                <p1:p>
                        This is the second paragraph
                </p1:p>
                <p1:p class="important">
                        This is the third paragraph
                </p1:p>
                <p1:p>
                        This is the fourth paragraph
                </p1:p>
                <p1:p>
                        This is the fifth paragraph
                </p1:p>
        </p1:body>
</p1:html>
```

Merged file for E.xml and F.xml, pretty-printed

This result is as expected and contains both additional attributes.

Controlling the Merge with Keys

In the above examples, the merges were performed as expected. This is because the initial comparison process was able to match up, correctly, the corresponding elements in the two files. In cases where many changes have been made to the files, it is not always possible to achieve, in all cases, the 'correct' correspondence between the two files. Applying more sophisticated algorithms and more processing power can help this situation, but in some cases it is very difficult to automate. A better approach is to add some control information into the files, in the form of keys, to aid this process. With keys it is possible to guarantee that corresponding elements are matched up, and then the result of the merge will be as expected.

The example below shows how keys can be added to assist the process. Consider merging the file B.xml with this new file J.xml:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p class="important">This is the first paragraph</p
><p class="important">This is the paragraph added after the first paragraph</p
><p class="important">This is the second paragraph</p
><p class="important">This is the third paragraph</p
></body>
</html>
```

J.xml, to be merged with B.xml

The result of the merge with B.xml is not what is expected because the comparison program is not able to match up the <p> elements correctly because they have all been changed by the addition of an attribute.

```
<p0:html xmlns:p0="http://www.w3.org/1999/xhtml"
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
        <p0:head>
                <p0:title>
                        Untitled
                </p0:title>
                <p0:meta name="generator" content="BBEdit 6.1.2" />
        </p0:head>
        <p0:body>
                <p0:p class="important">
                        This is the first paragraph
                </p0:p>
                <p0:p class="important">
                        |#[This is the second paragraph]#[This is the paragraph
added after the first paragraph]#|
                </p0:p>
                <p0:p class="important">
                        |#[This is the third paragraph]#[This is the second
paragraph]#|
                </p0:p>
                <p0:p class="important">
                        |#[This is a paragraph added after the third
paragraph]#[This is the third paragraph]#|
                </p0:p>
                <p0:p>
                        This is another paragraph added after the third paragraph
                </p0:p>
        </p0:body>
</p0:html>
```

J.xml, incorrectly merged with B.xml

What has happened here is that the <p> elements are all different between the two files so they have been matched by position, and the result of the merge is not what was required. (The new and old text data items are delimited by |#[,]#[and]#|, to show this, though these could be changed easily by modifying the merge XML stylesheet.)

To improve this, keys can be added using the **deltaxml:key** attribute. Consider the example above but with the addition of some **deltaxml:key** attributes that identify the paragraphs. The **deltaxml:key** attribute must be unique within the parent element. It could be an attribute of type ID, but this is not necessary. In formal documents, paragraphs are often numbered and whatever attribute holds this number could be used as a basis for generating the **deltaxml:key** attribute. This can be automated using XSL as an input filter, so the user does not need to change their way of working – an important consideration.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p deltaxml:key="1">This is the first paragraph</p
><p deltaxml:key="2">This is the second paragraph</p
><p deltaxml:key="3">This is the third paragraph</p
><p deltaxml:key="3a">This is a paragraph added after the third paragraph</p
><p deltaxml:key="3b">This is another paragraph added after the third paragraph</p
></body>
</html>
```

File K.xml, formed from file B.xml with deltaxml:key attributes

Now we can also add **deltaxml:key** attributes to file J.xml as follows.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html SYSTEM "xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
<head>
        <title>Untitled</title>
        <meta name="generator" content="BBEdit 6.1.2" />
</head>
<body><p class="important" deltaxml:key="1">This is the first paragraph</p
><p class="important" deltaxml:key="1a">This is the paragraph added after the
first paragraph</p
><p class="important" deltaxml:key="2">This is the second paragraph</p
><p class="important" deltaxml:key="3">This is the third paragraph</p
></body>
</html>
```

File L.xml, formed from file J.xml with deltaxml:key attributes

Now when the files are compared the elements will be matched using their **deltaxml:key** attributes, and a correct merged file will be generated as shown below.

```
<p0:html xmlns:p0="http://www.w3.org/1999/xhtml"
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
        <p0:head>
                <p0:title>
                        Untitled
                </p0:title>
                <p0:meta name="generator" content="BBEdit 6.1.2" />
        </p0:head>
        <p0:body>
                <p0:p class="important">
                        This is the first paragraph
                </p0:p>
                <p0:p class="important">
                        This is the paragraph added after the first paragraph
                </p0:p>
                <p0:p class="important">
                        This is the second paragraph
                </p0:p>
                <p0:p class="important">
                        This is the third paragraph
                </p0:p>
                <p0:p>
                        This is a paragraph added after the third paragraph
                </p0:p>
                <p0:p>
                        This is another paragraph added after the third paragraph
                </p0:p>
        </p0:body>
</p0:html>
```

Final result of controlled merge between K1.xml and L1.xml

In this case, the result is what was required in that the <p> elements are in the correct order and the attributes have been added as expected.

The extra effort in writing an XSL input filter to add the **deltaxml:key** attributes can mean that the results of the merge require no further processing, as has been shown above.

Using a Base File for 3-way Merge

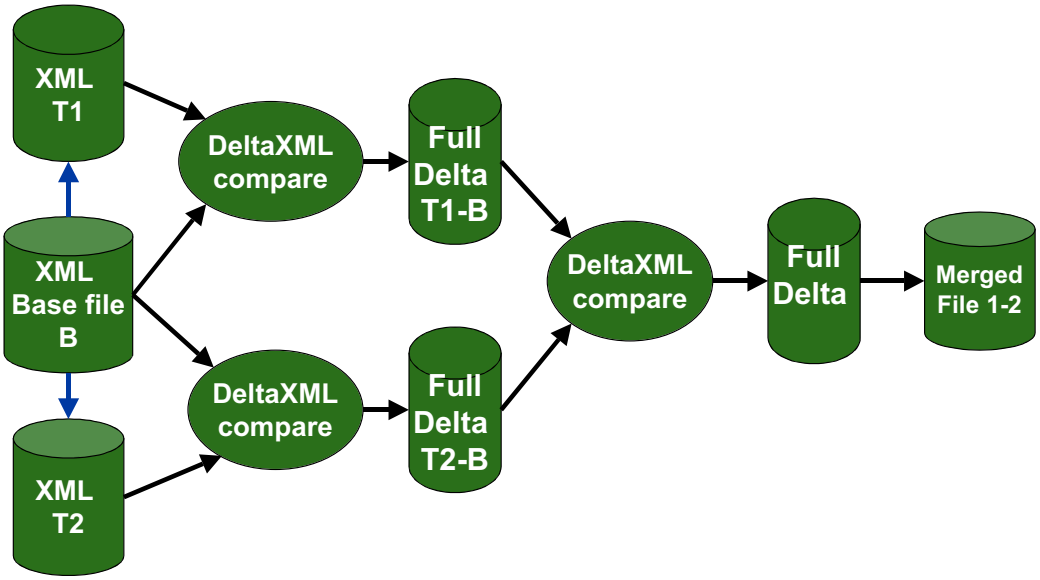
Although there are use cases for a 2-way merge, the 3-way merge is more powerful in cases where a base file does exist. In particular, the limitations of the 2-way merge in these cases include (for a base file TB and two derived files T1 and T2):

- it is not possible to distinguish between an element added from TB to T1 and one that has been deleted in TB to T2: in both cases they will be in T1 but not in T2 (also vice versa).
- where there is conflict between the two files, e.g. for some PCDATA, it is not possible to know which one should be selected, i.e. it is not possible to know whether T1 or T2 contains the 'updated' text.
- Similar problems apply to attributes.

In [Lindholm2001] the author suggests that this information can be found if the base file is processed and first matched with the two derived files. Markers are then added to distinguish these cases. This can be achieved using DeltaXML because the delta file that results from comparing TB with T1 can be used to generate a new version of T1 which has additional XML attributes to denote:

- any element that has been deleted
- any attribute that has been modified or deleted
- any element where the text content (directly contained within it) has been changed

Given an annotated version of T1, denoted T1a, and an annotated version of T2, it is then possible to compare these and generate, as before, a full delta between T1a and T2a. This will have sufficient information to enable a merged file to be generated such that the additional requirements of the 3-way merge are met.



Overview of 3-way merge based on two comparison stages

Let us consider first an example showing a deletion, taken from [Lindholm2001], example D1. The base file is as follows:

```
<R>
        <s1>
                <p1 />
                <p2 />
        </s1>
        <s2>
                <p3 />
                <p4 />
        </s2>
</R>
```

Base file d1-b.xml

```
<R>
        <s1>
                <p1 />
                <p2 />
        </s1>
</R>
```

Modified file d1-t1.xml

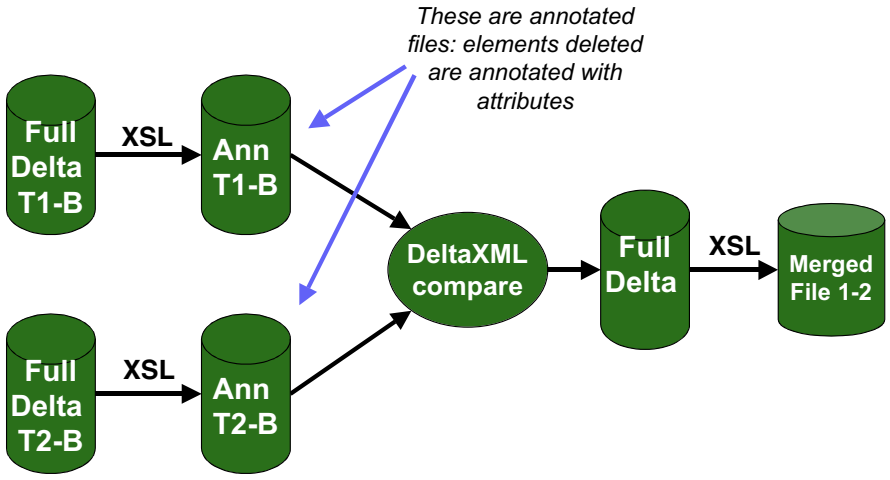
Note that <s2> has been deleted.

```
<R>
        <s1>
                <p1 />
        </s1>
        <s2>
                <p3 />
                <p4 />
        </s2>
</R>
```

Modified file d1-t2.xml

Note that <p2> within <s1> has been deleted.

From these three files we can generate the deltas by comparing d1-t1.xml with the base file d1-b.xml and similarly d1-t2.xml and d1-b.xml. From these we can generate an annotated version of d1-t1.xml and d1-t2.xml as shown below, again using an XSL script to do this. These annotated files are then compared again, and from the resulting full delta a merged file can be generated. We will work though this example in full to see how this works.



Refinement to 3-way merge adding annotated files before final comparison

The first step is to compare the modified files with the base file, obtaining a full delta in each case.

```
<R xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
deltaxml:delta="WFmodify">
        <s1 deltaxml:delta="unchanged">
                <p1 />
                <p2 />
        </s1>
        <s2 deltaxml:delta="delete">
                <p3 />
                <p4 />
        </s2>
</R>
```

Full delta file comparing d1-b.xml with d1-t1.xml

Note that <s2> has been marked as deleted.

```
<R xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
deltaxml:delta="WFmodify">
        <s1 deltaxml:delta="WFmodify">
                <p1 deltaxml:delta="unchanged" />
                <p2 deltaxml:delta="delete" />
        </s1>
        <s2 deltaxml:delta="unchanged">
                <p3 />
                <p4 />
        </s2>
</R>
```

Full delta file comparing d1-b.xml with d1-t2.xml

Note that <p2> within <s1> has been marked as deleted, and <s2>is unchanged.

Next, we process these files with XSL to generate an annotated version with the information we need about elements which have been deleted.

```
<R xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
        <s1>
                <p1 />
                <p2 />
        </s1>
        <s2 xmlns:dxann="http://www.deltaxml.com/ns/annotated-for-3-way-merge"
dxann:deleted="true">
                <p3 />
                <p4 />
        </s2>
</R>
```

Annotation file showing deletions from d1-b.xml to d1-t1.xml

Note that <s2> has been marked as deleted using a new namespace so that this does not get confused with the deltaxml namespace.

```
<R xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1">
        <s1>
                <p1 />
                <p2 xmlns:dxann="http://www.deltaxml.com/ns/annotated-for-3-way-
merge" dxann:deleted="true" />
        </s1>
        <s2>
                <p3 />
                <p4 />
        </s2>
</R>
```

Annotation file showing deletions from d1-b.xml to d1-t2.xml

Note that <p2> within <s1> has been marked as deleted, and <s2> is not marked because it is unchanged.

Now we compare these two annotated files to generate a full delta. As the annotation attributes are in a different namespace from the deltaxml attributes, there will be no confusion and the annotation will be compared in the usual way.

```
<R xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
xmlns:dxann="http://www.deltaxml.com/ns/annotated-for-3-way-merge"
deltaxml:delta="WFmodify">
        <s1 deltaxml:delta="WFmodify">
                <p1 deltaxml:delta="unchanged" />
                <p2 deltaxml:delta="WFmodify" deltaxml:new-
attributes="dxann:deleted=&amp;quot;true&amp;quot;" />
        </s1>
        <s2 deltaxml:delta="WFmodify" deltaxml:old-
attributes="dxann:deleted=&amp;quot;true&amp;quot;">
                <p3 deltaxml:delta="unchanged" />
                <p4 deltaxml:delta="unchanged" />
        </s2>
</R>
```

Full delta file comparing the two annotated files

Finally, we run an XSL script on this to generate a merged file, which takes into account the two deletions.

```
<R>
        <s1>
                <p1 />
        </s1>
</R>
```

Final result: 3-way merged file

This result is as expected, and is in agreement with [Lindholm2001].

Other deletion examples from [Lindholm2001] were run and the results were in agreement except for D4. The difference appears to be due to the different comparison algorithms and whether a move operation is allowed. The files are shown below.

```
<R>
        <a>
                <b>
                        <c />
                </b>
        </a>
</R>
```

Base file d4-b.xml

```
<R>
        <a>
                <i>
                        <b>
                                <c />
                        </b>
                </i>
        </a>
</R>
```

Modified file d4-t1.xml

Note that a/b has been deleted and a/i has been added.

```
<R>
        <a>
                <c />
        </a>
</R>
```

Modified file d1-t2.xml

Note that a/b has been deleted and a/c has been added.

```
<R>
        <a>
                <i>
                        <c />
                </i>
        </a>
</R>
```

Merged file for d4: Lindholm results

```
<R>
        <a>
                <c />
                <i>
                        <b>
                                <c />
                        </b>
                </i>
        </a>
</R>
```

Merged file for d4: DeltaXML results

The DeltaXML result performs the edits as delete/add operations whereas the Lindholm results incorporate a move. This is interesting in that it shows the influence on the result of the different comparison methods.

Text (PCDATA) content modifications

The next example looks at modifications to text or values within the file to show that these are propagated through to the merged file in a 3-way merge.

```
<test1>
        <p>To show 3-way merge for list of tagged text items</p>
        <data>
                <list>
                        <i>100</i>
                        <i>200</i>
                        <i>300</i>
                        <i>400</i>
                        <i>500</i>
                        <i>600</i>
                        <i>700 (is deleted in t2)</i>
                        <i>800</i>
                        <i>900 (is deleted in t1)</i>
        </list>
        </data>
</test1>
```

Base file rl1-b.xml

```
<test1>
        <p>To show 3-way merge for list of tagged text items</p>
        <data>
                <list>
                        <i>101 (modified in t1)</i>
                        <i>200</i>
                        <i>300</i>
                        <i>400</i>
                        <i>500</i>
                        <i>550 (added in t1)</i>
                        <i>600</i>
                        <i>700 (is deleted in t2)</i>
                        <i>800</i>
                </list>
        </data>
</test1>
```

Modified file rl1-t1.xml

Note that <i>900<i> has been deleted, 550 has been added and 100 has been changed to 101.

```
<test1>
        <p>To show 3-way merge for list of tagged text items</p>
        <data>
                <list>
                        <i>100</i>
                        <i>200</i>
                        <i>275 (added in t2)</i>
                        <i>300</i>
                        <i>402 (modified in t2)</i>
                        <i>500</i>
                        <i>600</i>
                        <i>800</i>
                        <i>900 (is deleted in t1)</i>
                </list>
        </data>
</test1>
```

Modified file rl1-t2.xml

Note that <i>700<i> has been deleted, 275 has been added and 400 has been changed to 402.

When these are processed, the resultant merged file is as follows:

```
<test1>
        <p>To show 3-way merge for list of tagged text items </p>
        <data>
                <list>
                        <i>101 (modified in t1)</i>
                        <i>200 </i>
                        <i>275 (added in t2)</i>
                        <i>300 </i>
                        <i>402 (modified in t2)</i>
                        <i>500 </i>
                        <i>550 (added in t1)</i>
                        <i>600 </i>
                        <i>800 </i>
                </list>
        </data>
</test1>
```

Final result: 3-way merged file for rl1 example

Note here that because there is always an unedited item between the changed items, the comparison algorithm is able to find the correct correspondence. With adjacent edits this may be more difficult and some changes may be misinterpreted. However, this is can be solved by adding keys to the elements so that correct correspondence is guaranteed.

Handling Attributes in a 3-way merge

Attributes are handled in a similar way to elements in that the annotated file contains details of which attributes have been deleted and which have been modified. This information is used to write out the correct set of attributes in the merged file.

```
<test1>
        <p>To show 3-way merge for attributes</p>
        <data attr1="unchanged" attr2="base" attr3="base" attr4="deleted in t1"
                attr5="deleted in t2">
                <list>
                        <i>100</i>
        </list>
        </data>
</test1>
```

Base file rl2-b.xml

```
<test1>
        <p>To show 3-way merge for attributes</p>
        <data attr1="unchanged" attr2="modified in t1" attr3="base"
                attr5="deleted in t2" attr6="added in t1">
                <list>
                        <i>100</i>
        </list>
        </data>
</test1>
```

Modified file rl2-t1.xml

Note that attr4 has been deleted, attr6 has been added and attr2 has been modified.

```
<test1>
        <p>To show 3-way merge for attributes</p>
        <data attr1="unchanged" attr2="base" attr3="modified in t2" attr4="deleted
in t1" attr7="added in t2">
                <list>
                        <i>100</i>
        </list>
        </data>
</test1>
```

Modified file rl2-t2.xml

Note that attr5 has been deleted, attr7 has been added and attr3 has been changed.

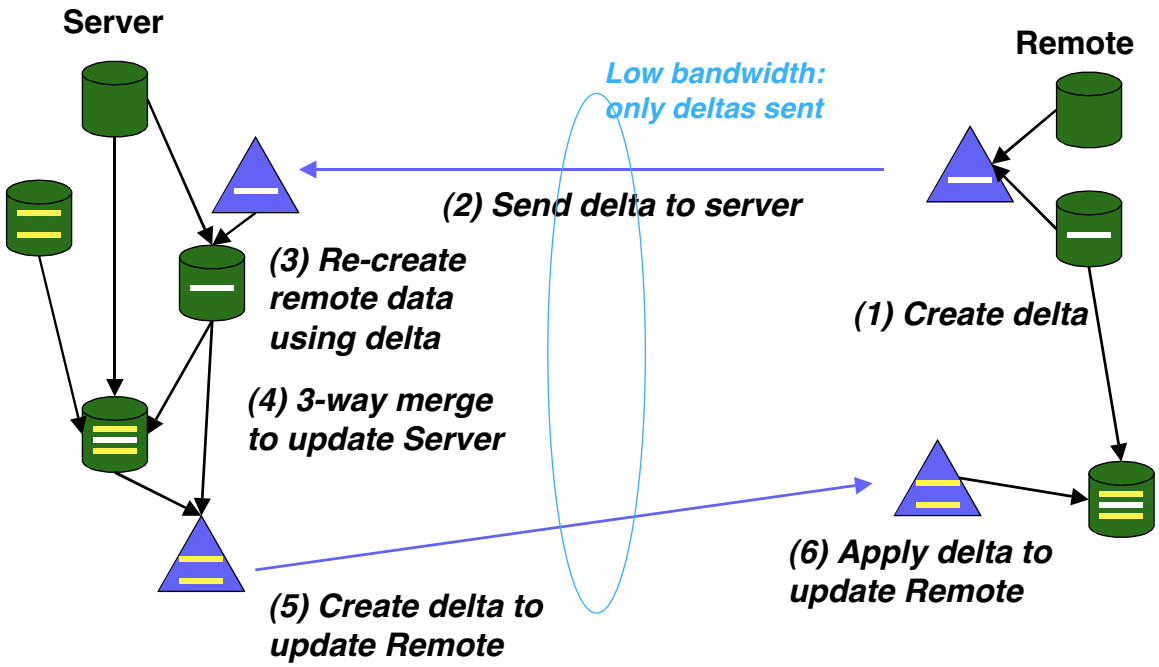
```
<test1>
        <p>To show 3-way merge for attributes</p>
        <data attr1="unchanged" attr2="modified in t1" attr6="added in t1"
attr3="modified in t2" attr7="added in t2">
                <list>
                        <i>100</i>
                </list>
        </data>
</test1>
```

Final result: 3-way merged file for rl2 example

The final merged result shows that the correct merge has taken place: modified attributes are modified, those deleted are deleted and those added are added to the final result. This is achieved using the annotated intermediate files and the delta generated by comparing these. There is some string manipulation within XSL to achieve this because the attribute values and annotation information has to be extracted from the **deltaxml:new-attributes** and **deltaxml:old-attributes** attribute values. Again, if there were specific requirements for handling specific attributes, these could be accommodated by changing the XSL styhlesheets.

Application of 3-way merge to remote data synchronization

Using the 3-way merge described in this paper, and taking advantage of the basic delta comparison capability, a scenario for synchronization of XML data between one or more remote (possibly off-line) users and a central server is shown.



3-way merge used for XML data synchronization

In this scenario, the two systems start with some data which is synchronized. These data sets are then edited both on the server and on the remote site to generate derived data files. In the diagram these are denoted with horizontal lines through the cylindrical data file representations. The following process is then executed to synchronize these.

1. The remote site creates a delta file (denoted by the triangular shape) to record all the changes made at the remote site.
2. The delta file (which typically is small) is then sent to the server.
3. The server then re-creates the status of the remote data by applying the delta to the original synchronized data file.
4. A 3-way merge is then performed at the server side to synthesise the edits made at the remote site and at the server. Using the intermediate files it would be possible to control which edits were 'allowed' from the remote site.
5. The server then creates a new delta from the difference between the new data set and the one created to be identical with the remote site data.
6. This delta can then be sent to the remote site and applied to bring the remote site up to date with the server data.

The delta files are not needed if bandwidth is not a problem: the full files can be exchanged or shared. The data synchronization process is not dependent on the structure of the XML files, so it does not need to be changed if the file format is changed. The process could also be amended to cope with asynchronous updates, though it would be more complex.

Conclusions

This paper has shown how two XML files can be merged using an intermediate delta file which contains a structured representation of the two files in XML such that common elements are identified as common and elements that are present in just one of the files are also identified. This intermediate file forms a good basis for performing any application-specific edits on the data before the final merged version is generated.

The paper also shows how this can be extended to perform a powerful 3-way merge where two XML files have been derived from a base file. Additions, modifications and deletions are faithfully reproduced in the merged file except where conflicts occur in which case one or other must be chosen. As with the 2-way merge, the use of intermediate files means that application-specific edits can be made at any stage giving great flexibility to the merge transformation.

The 3-way merge, combined with the use of normal delta files, can provide an XML system with data synchronization features normally only found in more complex database systems. This provides yet another reason for adopting XML as the preferred data format.

References

[Cobena2002] "A comparative study for XML change detection", Gregory Cobena, Talel Abdessalem, Yassine Hinnach, http://osage.inria.fr/verso/PUBLI/all-bykey.php?mytexte=cobena

[LaFontaine2001] "A delta format for XML: Identifying changes in XML and representing the changes in XML", Robin La Fontaine, XML Europe 2001, http://www.gca.org/papers/xmleurope2001/papers/html/s29-2.html

[Lindholm2001] "A 3-way Merging Algorithm for Synchronizing Ordered Trees – the 3DM merging and differencing tool for XML", Tancred Lindholm, MSc thesis, September 13, 2001. Helsinki University of Technology, Dept. of Computer Science. http://www.cs.hut.fi/~ctl/3dm/

[Manger2001] "A Generic Algorithm for Merging SGML/XML-Instances", Gerald W. Manger, XML Europe 2001, Berlin. http://www.gca.org/papers/xmleurope2001/papers/html/s29-1.html