An improved diff3 format using XML: diff3x

Robin La Fontaine

Nigel Whitaker

Abstract

There is no doubt that the diff and diff3 format has established itself as a well-used de-facto standard. It might seem presumptuous to suggest that it could be improved, or indeed that it needs to be improved. However, the original premise of line-based text files as the subject matter is now out of date with more structured information being the norm. Often this is in the form of programming source code where the layout tends to remain fairly consistent through edit cycles, but increasing use of JSON and XML pose particular difficulties for the simple line-based structure of diff3.

In our paper at MarkupUK in 2019, we discussed some of the issues and suggested some minor improvements to diff3. These changes suffered from the common complaints of a retro-fit in that they did not sit comfortably with the original and only did half a job. The prevalence of GUIs also suggest that the actual syntax of a diff file is not as important as it was in that the emphasis has changed from human readability to interchange between two applications, for example between a git 'merge driver' and a git 'mergetool'. For these reasons it seemed better to consider a different approach using the tools and formats that are now in common use, for example XML or JSON.

What might diff3 look like as an XML format? Would the advantages of a new format make it worth swapping from the tried and tested diff3? Could existing GUI software easily adapt to a new format and, perhaps, even be simpler as a result?

Table of Contents

Introduction and Background

This paper is a sequel to "An Improved diff3 Format for Changes and Conflicts in Tree Structures" [1] and again is focused on the diff3 format rather than the diff3 executable application. In this paper we will develop an XML alternative to the diff3 format from GNU diffutils [2]. There are many possible outputs from diff3 but the one we are interested in is the one that provides a merged file result with conflicts marked up, i.e. the '-m' option on the command line.

Many users do not view diff3 data directly or invoke diff3 itself, instead it is often invoked by a version control systems such as git or mercurial when the users merge a branch, graft or cherry-pick, rebase or change branches with working directory changes.

A characteristic that we seek from the start is that as the number of changes tends to zero, so the diff file tends to resemble the original files. This is desirable in that minimal processing is needed for few changes and human understanding is improved simply because when the diff and the original files are very similar they will look very similar.

We distinguish between the 'carrier' syntax which is the diff3 alternative, and the 'payload' which is the content of the file(s) being compared or merged.

In our previous paper, we considered both XML and JSON as a candidate for the carrier syntax and established that XML is a more natural fit, with the payload of the original files being, typically, not XML. However, of course this could be applied to XML itself and then there is likely to be at a human readability level a confusion between the payload and the carrier, and we will look at that in this paper.

Developing an XML syntax for diff3x

First we look at the trivial but important example of a diff3 for three identical files, i.e. there are no changes. As we are using XML there is a minimum overhead of the start and end tags, so if the example files are all the same, as shown below:

Table 1. Three identical files

A.txt	O.txt	B.txt
1 2 3 4 5 6	1 2 3 4 5 6	1 2 3 4 5 6

The XML representation would be as shown below and note that white space needs to be preserved, or CDATA could be used.

```
<diff3x>1
2
3
4
5
6</diff3x>
```

Now we can move on to represent actual changes.

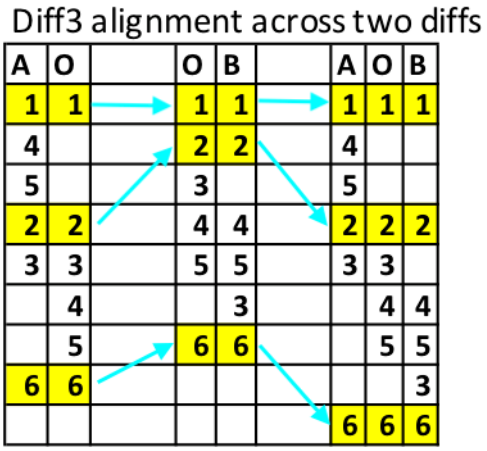
We need an example to show the syntax and we will use the same example as in the previous paper because it also allows us to illustrate how the XML syntax can potentially represent a richer view of the differences. For clarity we repeat the example here. The example is based on this paper, "A Formal Investigation of Diff3" [3]. The example consists of three text files with numbers on each line, the files are denoted A.txt, B.txt and the 'old' file O.txt as shown below:

Table 2. Mismatched sequences

A.txt	O.txt	B.txt
1 4 5 2	1 2 3 4	1 2 4 5

A.txt	O.txt	B.txt
3 6	5 6	3 6

The way these are combined into the two diffs, A+O and O+B are shown in the table below.



The last three columns show how the two diffs are combined. Note that the yellow match shows where all three files align - and this is important because it is the data between these alignment points that are considered as units of change. Now we can look at the diff3 output using the -m option:

```
1
4
5
2
<<<<<<< A.txt
3
||||||| O.txt
3
4
5
=======
4
5
3
>>>>>>> B.txt
6
```

How might this look as XML? Because we are only looking at a maximum of three files it seems reasonable to have a specific element to represent each one. However, we also need to record the original file names and these could be shown as attributes on the root element. So, the above might be represented in XML as follows.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">1
4
5
2<choice3><a>
3</a><o>
3
4
5</o><b>
4
5
3</b></choice3>
```

```
6</diff3x>
```

The element <choice3> introduces a three-way choice between the three original files.

Showing Non-conflicting Changes

Now that we have moved into an XML syntax world, we have the opportunity for a richer representation to show non-conflicting changes. It is often useful to see these to understand how a merge has been handled and provides more context when resolving conflicts. In this example we could show the origin of the '4 5' data that was added by A.

To show this we need to have elements defined for the limited combinations of the three input files. We can do this as <ao> for data in both O and A, and similarly <ab>, <ob>. We do not need <aob> because any lines matched across all three will be present as a child of the root element. Any non-conflicting change will have one of these dual origin elements as one of the two options. As these are non-conflicting changes, they are a resolved choice and we can indicate this with an attribute to show which option has been included. This attribute could also be used to indicate which option or options in a conflicting choice is to be included in the final result. A GUI might provide a user with the ability to undo this choice.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">1<choice2><a include="true">
4
5</a><ob/></choice2>
2<choice3><a>
3</a><o>
3
4
5</o><b>
4
5
3</b></choice3>
6</diff3x>
```

Showing linked changes to JSON Structure in diff3x

For JSON, the issue of handling curly braces (for objects) and square brackets (for arrays) is similar to XML start and end tags. Again, some representation of connected change is needed to maintain syntactic correctness.

Object members and array members are comma separated and this is tricky to get right in some situations. The syntax is shown below.

```
object = begin-object [ member *( value-separator member ) ]
                end-object
array = begin-array [ value *( value-separator value ) ] end-array

These are the six structural characters:
  begin-array = ws %x5B ws ; [ left square bracket
  begin-object = ws %x7B ws ; { left curly bracket
  end-array = ws %x5D ws ; ] right square bracket
  end-object = ws %x7D ws ; } right curly bracket
  name-separator = ws %x3A ws ; : colon
  value-separator = ws %x2C ws ; , comma

Insignificant whitespace is allowed before or after any of the six
structural characters.
ws = *(
       %x20 / ; Space
```

```
%x09 / ; Horizontal tab
%x0A / ; Line feed or New line
%x0D ) ; Carriage return
```

Here is an example of a change to an array of strings.

Table 3. JSON structural change

A.txt	O.txt	B.txt
[[12,13,14],20,21,22]	[12,13,14,20,21,22]	[[12,13,14,20,21,22]]

This could be represented in the diff3 format as shown below, but note that this is not how diff3 would process the above files which each have a single line. However the result below could be generated by careful use of line breaks in the input files.

```
[
<<<<<<<< A.txt
[
||||||| O.txt

=======

>>>>>>> B.txt

<<<<<<<< A.txt

||||||| O.txt

=======
[
>>>>>>> B.txt
12,13,14
<<<<<<<< A.txt
]
||||||| O.txt

=======

>>>>>>> B.txt
,20,21,22
<<<<<<<< A.txt

||||||| O.txt

=======
]
>>>>>>> B.txt
]
```

The above will only produce syntactically correct results if the correct choices are made, which is not easy. It is also not easy to see what is going on in the diff3 file because of the extra whitespace lines that have to be inserted to make this work.

We introduce here a way to connect the relevant consistent choices so that if the '[' is selected then the appropriate choice of the end ']' is also made automatically.

A choice consists of one or more options. An option may have an id. An option may also have a select attribute which provides a boolean value made up of one or a combination of other ids and if these are 'true', i.e. have been chosen to be included, then this option is selected automatically.

This could be represented in diff3x as shown below. We show here the id attribute on an option to identify it uniquely within the file. We then use that id to reference that option where another option needs to be linked to it. In this case, the '[' in the A.txt has `id="a42"` and this is then referenced in the select attribute of the corresponding ']' option later in the file. Similarly the pair of square brackets in B.txt are linked with the id "b44".

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
[
<choice2>
<a id="a42">[</a>
<ob/></choice2>
<choice2>
<b id="b44">[</b>
<ao/></choice2>
12,13,14
<choice2>
<a select="a42">]</a>
<ob/></choice2>
,20,21,22
<choice2>
<b select="b44">]</b>
<ao/></choice2>
]</diff3x>
```

Note that the '[' in A could be selected either instead of or as well as the '[' in B even though they are at the same position in the array. Whichever choice is made, the other choices with a select attribute identifying the same id are chosen and the result is syntactically correct. This is a powerful way to represent connected choices.

Representing JSON Separator Change in diff3x

The problem with separators is that they cannot consistently be associated with either the start or the end of each item (member for object and value for array) because if there is only one item then no separator is needed. Therefore maintaining correct syntax when items are added or deleted is not trivial. As mentioned above, the diff3 format does not allow consecutive choices without 'anchor' data between, so it is necessary to group consecutive items that may be added or deleted into one choice. This apparent restriction does lead to a greater likelihood of the syntax of each choice being consistent.

Here is an example of a change to an array of strings.

Table 4. JSON array value change

A.txt	O.txt	B.txt
`["one", "two"]`	`["one"]`	`["three", "four"]`

This could be represented as shown below. Note here that we are not showing the result of running 'diff3 -m' but rather we have run an XML aware comparison so we have results that we want to express in the diff3 format.

```
[
<<<<<<< A.txt
"one", "two"
||||||| O.txt
"one"
=======
"three", "four"
>>>>>>> B.txt
]
```

The above will produce syntactically correct results though it is not ideal because it would be more natural to choose the values separately rather than as a complete list. This can be achieved with diff3x as shown below. Here we have been able to perform the merge automatically, there are no conflicts but clearly it is useful for the user to see these and have the option to undo or change a choice.

We introduce here a third type of choice, the `<autoInclude>`. This is a choice that we do not expect the user to review but rather it is a single option that is chosen automatically based on one or more other selections. In this case, the commas are inserted only when they are needed.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
[
<choice2>
  <ao id="ao42">"one"</ao>
  <b id="b44" include="true"></b></choice2>
<autoInclude select="AND(ao42 a52)">, </autoInclude>
<choice2>
  <a id="a52" include="true">"two"</a>
  <ob id="ob53"></ob></choice2>
<autoInclude select="AND(OR(ao42 a52) b62)">, </autoInclude>
<choice2>
  <b id="b62" include="true">"three", "four"</b>
  <ao id="ao63"></ao></choice2>
]</diff3>

Result of this choice is:
["two", "three", "four"]
```

This is more complex in getting the logic correct for insertion of commas. XML users will be pleased that XML attributes are not comma separated!

Preserving well-formed tree structure in diff3x

In this section we explore the issues of preserving the well-formed structure of XML when presenting choices in diff3x.

Representing XML Element Tag Change in diff3x

XML tags present a problem for diff3 format in that it is in general not possible to ensure a well-formed result without unacceptable duplication of content. To handle tag changes in diff3x we need to treat the XML payload, i.e. the XML that is the subject of change, as text and so escape it with CDATA markers. Later in this paper we look at treating an XML payload as XML, which is more natural but as we shall see, it is not possible to represent tag changes in that approach.

Here is an example of a change of structure.

Table 5. XML tag change

A.txt	O.txt	B.txt
`<p>This is a long paragraph where <strong>most of it has been made either bold or italic, but the rest of the paragraph`	`<p>This is a long paragraph where most of it has been made either bold or italic, but the rest of the paragraph`	`<p>This is a long paragraph where <italic>most of it has been made either bold or italic, but the rest of the paragraph`

A.txt	O.txt	B.txt
remains unchanged - there is no change to the text so we do not want it duplicated. </p>	remains unchanged - there is no change to the text so we do not want it duplicated. </p>	remains unchanged - there is no change to the text so we do not want it duplicated</italic>. </p>

This could be represented as shown below, but there is duplication of unchanged text which is confusing because if there had been a small change the user would have found it difficult to see. Note that in this example the payload is XML but this is not seen as part of the XML of the carrier, the payload is treated as text because it is enclosed in the CDATA sections.

```
<diff3x a="A.txt" b="B.txt" o="O.txt"><![CDATA[<p>This is a long paragraph
where ]]>
<choice3>
<a><![CDATA[<strong>most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated</strong>
]]></a>
<o><![CDATA[most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated]]></o>
<b><![CDATA[<italic>most of it has
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated</italic>
]]></b></choice3>
<![CDATA[. </p>]]>
</diff3x>
```

We can improve this significantly and use the id attributes as described above to ensure consistent choices. In this case we have given the options within the two connected choices an id value. Thus if option <a> is selected in one choice with id="a42" then the choice with id="a420" is automatically selected. We have also added 'vice versa' attributes so that the same would happen the other way round: if the end tag was selected then the corresponding start tag would also be selected.

```
<diff3x a="A.txt" b="B.txt" o="O.txt"><![CDATA[<p>This is a long paragraph
where ]]>
<choice3 >
<a id="a42" select="a420" include="true"><![CDATA[<strong>]]></a>
<o/>
<b id="b43" select="b430"><![CDATA[<italic>]]></b></choice3>
most of it has
```

```
been made either bold or
italic, but the rest of
the paragraph remains
unchanged - there is no
change to the text so
we do not want it
duplicated
<choice3>
<b select="b43" id="b430"><![CDATA[</italic>]]></b>
<o/>
<a select="a42" id="a420"><![CDATA[</strong>]]></a></choice3>
<![CDATA[. </p>]]>
</diff3x>
```

If the GUI tool allowed multiple options to be selected, then both `<strong>` and `<italic>` could be selected - provided of course they appeared in the correct nested order. We have assumed that the user would need to have the knowledge to know if the selection of two options was appropriate.

Representing XML Attribute Change in diff3x

XML attributes present a particular challenge for diff3 format, and this can be improved using an XML representation.

It is worth noting an issue with the diff3 format in this situation, because it is quite subtle but one that we can resolve in diff3x. The lines in the input files at which all three files align (the lines are all equal) are considered 'anchor' points. All of the lines between anchor points is considered to be a single choice - and when there is some kind of conflict then the choice is left for the user to select. The interesting consequence of this structure is that it is not possible to have two consecutive choices without a separator which requires a commonality between all three files, i.e. an anchor point. For structured data it would be natural, for example, to provide choices about attributes in a manner that allows each attribute to be chosen separately, but the diff3 format dictates that two adjacent changes are seen as one choice.

We can improve this with diff3x because we can easily have adjacent choices without any ambiguity.

Table 6. XML Attribute value change

A.txt	O.txt	B.txt
`<span class="two" dir="rtr" id="23">`	`<span class="one" id="23" dir="TBA">`	`<span id="23" class="three" dir="ltr">`

These changes can be represented in diff3x as shown below.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
<![CDATA[<span id="23"]]>
<choice3>
<a>class="two"</a>
<o>class="one"</o>
<b>class="three"</b></choice3>

<choice3>
<a>dir="rtr"</a>
<o>dir="TBA"</o>
<b>dir="ltr"</b></choice3>
<![CDATA[</span>]]></diff3x>
```

There is another representation that takes the common attribute name out of the choice but this may be less easy for a user to see what is happening by inspection of the diff3x but a GUI could provide an intuitive interface. For illustration, in the first class attribute we have included the attribute value

quotation marks in the options, and for the dir attribute we have put them outside - this is a matter of choice, either approach can be represented by diff3x.

```
<diff3x a="A.txt" b="B.txt" o="O.txt">
<![CDATA[<span
class=]]><choice3><a>"two"</a>
                  <o>"one"</o>
                  <b>"three"</b></choice3>
dir="<choice3><a>rtr</a>
            <o>TBA</o>
            <b>ltr</b></choice3>"
<![CDATA[</span>]]></diff3x>
```

Nested Changes

As diff3x is an XML representation we here consider going one step further and making use of the fact that the representation is hierarchical to support hierarchical or 'nested' change. A nested change is a change in one branch that modifies something that has been removed in another branch.

We will look at an XML example, showing nested changes.

Table 7. XML nested data example

A.xml	O.xml	B.xml
<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <ph>01684 532141</ph> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> <post>WR14 3SZ</post> </address> </author>	<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> <post>WR14 3SZ</post> </address> </author>	<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> </author>

In the above example one branch, `B.xml`, has deleted the `address` sub-tree which the other branch has modified with an added `phone` number. As we are now using an XML representation that has a tree structure we can also make use of this structure in the result.

In order to distinguish between a simple `<ao>` option and a nested choice, we introduce the `<ao-Choice>` element. This is a mixed content element that can include `<a>` or `<o>` elements as well as text.

```
<d:diff3x a="A.xml" b="B.xml" o="O.xml"><![CDATA[
<author>
<personname>
 <first>Nigel</first>
 <last>Whitaker</last>
</personname>]]>
<d:choice3>
 <d:aoChoice include="true"><![CDATA[
   <address>]]>
     <d:a><![CDATA[
      <ph>+44 1684 532141</ph>]]>
     </d:a>
```

```
    <![CDATA[
    <st>Geraldine Rd</st>
    <city>Malvern</city>
    <country>UK</country>
    <post>WR14 3SZ</post>
   </address>]]>
  </d:aoChoice>
  <d:bChoice/>
 </d:choice3><![CDATA[
</author>]]>
</d:diff3x>
```

Here we can see that by allowing nested change we can avoid the repetition and make it easier for a human to understand. However, in order to support this in the format we have to introduce quite a number of new elements and different rules. This adds to the complexity of the format and moves us a lot further away from the original elegant simplicity of diff3. Keeping diff3x as simple as possible seems preferable to including the ability to represent a subset of changes as nested changes.

For this reason we have elected not to include the ability to represent nested changes in diff3x.

Should an XML Payload be represented as XML or text?

If our payload is XML, and our carrier is XML, does it make sense to embed the payload as XML rather than text? At first sight this seems to be an attractive proposition, and an example is shown below. For clarity we have put the diff3x elements into their own namespace, using a d: prefix.

Table 8. XML data example

A.xml	O.xml	B.xml
<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> </address> </author>	<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> <post>WR14 3SZ</post> </address> </author>	<author> <personname> <first>Nigel</first> <last>Whitaker</last> </personname> <address> <st>Geraldine Rd</st> <city>Malvern</city> <country>UK</country> <post>PQ9 5XY</post> </address> </author>

This could be embedded in the XML diff3x carrier as shown below.

```
<d:diff3x a="A.xml" b="B.xml" o="O.xml">
<author>
<personname>
  <first>Nigel</first>
  <last>Whitaker</last>
</personname>
<address>
 <st>Geraldine Rd</st>
 <city>Malvern</city>
 <country>UK</country>
 <d:choice3>
  <d:a/>
```

```
  <d:o>
   <post>WR14 3SZ</post>
  </d:o>
  <d:b>
   <post>PQ9 5XY</post>
  </d:b>
 </d:choice3>
</address>
</author>
</d:diff3x>
```

This looks good and useful, but it has limitations. The main one is that it would not be possible to show changes to attributes in situ, even if we resorted to XML processing instructions. Other approaches to this include representing attribute change by introducing an XML element immediately following the end of the start tag.

Also, it would not be possible to show changes to element tags, e.g. if a `<bold>` was changed to `<italic>`.

These limitations could be overcome by making the diff3x more complex but there is no good reason to do this because it takes away the primary purpose of the diff3x carrier, which is to represent changes to a text file. As an XML payload can be considered to be a text file we should treat it as such to be consistent with the handling of JSON or any other payload.

Saving Selected Options

Some choices are for a user to decide which option(s) to include, others can be included automatically either as non-conflicting changes or as a direct result of other selections. If we are able to indicate in the format which options are currently selected, then we can save the state of the document during the resolution process. In addition, this would allow a user to change which option is selected in a choice. It would be advantageous for a GUI to allow not only a single option to be selected but two or more options - for example if A and B have added something different at the same position it may be correct to select neither, one or both options.

Comparing diff3 format with diff3x

Although human readability is not the prime concern, it is worth looking at how the files might appear. The table below shows the example first used in Table 2, “Mismatched sequences” as diff3 and the corresponding file in diff3x. We have not included the CDATA markers in XML here, but have added an xml:space="preserve" attribute indirectly via the schema.

Table 9. diff3 and diff3x representations

diff3	diff3x (XML)
1 4 5 2 <<<<<<< A.txt 3 ||||||| O.txt 3 4 5 ======= 4 5	<diff3x a="A.txt" b="B.txt" o="O.txt">1 4 5 2<choice3><a> 3<o> 3 4 5</o> 4 5 3</choice3> 6</diff3x>

diff3	diff3x (XML)
3 >>>>>>> B.txt 6	

It is worth noting that another advantage of the diff3x representation is that choices within a line can be represented quite naturally, so this overcomes the significant limitation of diff3 which is only able to show changes to a complete line.

Table 10. Characteristics of diff3 and diff3x

Characteristic	diff3	diff3x	Comment
No processing needed for unchanged file	***	**	
Preserve line structure	***	***	XML needs CDATA
Good for text editor (by hand)	**	*	
Intelligent changes	**	***	
Nested changes	*	***	
Changes within a line	*	***	
Show all resolved merges	*	***	
Show changes to JSON data	**	***	
Show changes to XML data	*	***	
Process the changes in the format of the native payload	-	*	Important to note that this is really not possible - a delta format in the native format is needed.

The table does show significant advantages of having an XML representation of diff3 when communicating merge results from a merge driver to a GUI mergetool.

Future work

In previous conferences and events we have shown and discussed different approaches to resolving and processing merge results using XML based technologies. We have shown a conflict resolver using an OxygenXML plugin and one based on Schematron Quick Fixes (SQF) 4. These work well with an XML payload, but aren't directly usable with other forms of structured content such as JSON. At DeltaXML we are finding that customers want to evaluate our tools in the browser and fewer users are willing to download and install a ZIP package with our code/APIs. We have tools/services for XML and JSON and display comparison and merge results in the browser, but we do not provide an interactive merge resolver. Using diff3x would help to provide this capability for both XML, JSON and potentially other structured formats.

In 5 we discussed what we feel are inefficiencies/shortcomings in many git workflows where a mergetool would repeat the work done in the merge driver, often inconsistently. The format presented here is one possible solution that could be used to address these issues. This would allow effort spent understanding structured merged techniques to be concentrated/shared and not repeated in different merge tools. We would welcome feedback from the wider git user community and also from those interested in mergetools.

Conclusions

In this paper we have introduced an XML update to diff3 which we have called diff3x. With diff3x, we can integrate more modern structure-aware comparison tools and represent choices that are consistent with the structure and more likely to provide well-formed or valid results.

We have considered the representation of nested changes but concluded that the additional complexity to the diff3x format is unlikely to be worth the small gain.

The diff3x format can take plain text as its payload. If the payload is XML it would need to be escaped for example with CDATA and treated as text. We considered whether, if the payload is also XML, this could be treated as XML and although this is possible we concluded that treating an XML payload as text has the advantage that tag changes and attribute changes can be represented directly in situ. Our experience with XML tools and technologies has allowed us to develop XML-centric resolvers, but these have the limitation that they do not work with JSON and other non-XML formats. They still have benefits and their applications, for example when considering nested change or n-way merge. As such they complement the benefits of diff3x.

In a related paper [5] we identified some issues in version control systems that caused inconsistency and confusion to users. One solution to those issues relies on separating the merge driver from subsequent conflict resolution tools or 'merge tools'. The diff3x format proposed here would work well for data exchange between these two steps. This is important because it separates knowledge of the format and structure of a file from a GUI designed to allow the acceptance of changes or the resolution of conflict thus enabling a single GUI to handle many different file structures in a consistent way.

The intention of diff3x is to provide a richer format than diff3 for the exchange of differences and conflicts in text files. The advantages of diff3x over diff3 include the following:

1. Recording user selections of options: this provides both the facility to save changes during a merge resolution session and providing an audit record of the choices made.

2. Connected options: this provides the ability for the selection of one option to trigger the selection of another connected option, e.g. when a start tag is selected the corresponding end tag is also included.

3. Auto include of text: this provides for certain text to be automatically included if certain options are selected, e.g. so that appropriate separators could be included.

We have shown that improvements to the representation of change for structured data is possible and desirable. Changing the existing diff3 format proved to be awkward and limited, so this move directly to a markup representation using XML is a better approach. In working this through in more detail in this paper we have identified other advantages to make it easier for users to accept or reject changes in a way that is consistent with the underlying structure of the data.

References

[1] *An Improved diff3 Format for Changes and Conflicts in Tree Structures. [https://markupuk.org/2019/Markup-UK-2019-proceedings.pdf#page=87]*

[2] *GNU DiffUtil - Comparing and Merging Files. [https://www.gnu.org/software/diffutils]*

[3] Khanna S., Kunal K., Pierce B.C. (2007) *A Formal Investigation of Diff3. In: Arvind V., Prasad S. (eds) FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science. FSTTCS 2007. Lecture Notes in Computer Science, vol 4855. Springer, Berlin, Heidelberg [https://link.springer.com/chapter/10.1007%2F978-3-540-77050-3_40]*

[4] *Schematron QuickFix [http://www.schematron-quickfix.com/]*

[5] Robin La Fontaine and Nigel Whitaker *Merge and Graft: Two Twins That Need To Grow Apart [http://archive.xmlprague.cz/2019/files/xmlprague-2019-proceedings.pdf#page=175]*