An improved diff3 format for changes and conflicts in tree structures

Robin La Fontaine

Nigel Whitaker

Abstract

There are some pieces of software, and some formats, that are de-facto standards and have been around for decades. One of these is the diff3 format for representing changes and conflicts in text documents. Diff3 works well for unstructured text documents that are divided into lines. It works surprisingly well for pretty-printed source code and similar documents. But it has frustrating limitations when used for XML or JSON or similar tree-based data formats.

Can we improve on diff3 without making it too complicated? Can the existing representation of changes and conflicts be extended to handle tree-based data? This paper seeks to answer these questions and to describe how further benefits can be enjoyed by using XML or JSON as the basis for showing conflicts and changes.

Table of Contents

Introduction and Background

This paper is focused on the diff3 format rather than the diff3 executable application. In this paper we will consider the diff3 format from GNU diffutils [1]. 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.

The diff3 format can present information that is used in a three way merge. It is a derivative of the two-way diff change format which uses a subset of the change markers (it does not include the ancestor information, but does use left and right angle brackets to delimit the two inputs). Many users do not invoke diff3 directly, instead it is often invoked by a version control systems such as git or mercurial when the users merge a branch, cherry-pick, rebase or change branches with working directory changes.

The format can be used for resolving changes directly, perhaps using a simple text editor, and this was a common mode of operation with early version control systems. It can also be suitable for use with a GUI to provide accept/reject changes resulting in a new version of the file with the conflicts resolved.

We will provide some background to how the diff3 tool identifies areas of conflict in order to better understand the format itself. We will not go into any details about the limitations of using line-based comparison tools on tree-structured data, which is a subject that has been explored elsewhere and the

limitations are well known in principle if not in detail, as are a number of different ways to make a line-based comparison work better with tree-structured data, e.g. re-formatting into some canonical form.

It is possible to do a better job of comparison for XML and JSON if the comparison engine is aware of the tree structure. The issue then is how to represent the change in a way that is suitable for other systems, for example Visual Studio Code [2] which understands the diff3 format. With some ingenuity, certain changes can be represented so that accepting or rejecting the change results in a well-formed output. However this is not always possible when for example start and corresponding end tags have been added or deleted, or when changes are nested.

We will propose a way that the diff3 format could be extended to handle ‘connected changes’ where the acceptance of one change requires the acceptance (or rejection) of a connected change, for example to keep start/end tags or braces balanced. We will explore the difficulties in trying to extend it further to handle nested changes and propose a way to use XML or JSON to achieve this in a way that is more suited to those technology stacks.

How diff3 delimits the extent of changes and conflicts

The example is based on this paper, "A Formal Investigation of Diff3" [3]. This explains how the two two-way diffs are aligned. It is useful to understand this in order to see how it might affect tree-structured data.

The example consists of three text files with numbers on each line, A, B and the 'old' file O as shown below:

Table 1. XML Attribute value change

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

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

Diff3 alignment across two diffs

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

The last three lines 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
```

This shows that the '4 5' sequence has been accepted as the only possibility between the '1' and the '2', but between the '2' and the '6' we have three possible choices which are listed in the output. We do not want to get diverted into a discussion about alignment algorithms, nor whether or not this is appropriate for tree-structured data. The point here is that the positions in the files at which they all three align are considered 'anchor' points and all of the data between is considered to be a choice - and when there is some kind of conflict then the choice is left for the user to select.

There is an interesting consequence of this structure: it is not possible to have two consecutive choices without a separator that is due to a commonality between all three files, i.e. an anchor point. Although 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, the diff3 format dictates that two adjacent changes are seen as one choice. For structured data such as XML, it may be possible to get round this by artificially creating anchor points that are white space which is not relevant to the result, but this is not ideal, partly because diff3 would not do this and therefore the subsequent change to the layout of the files is not expected by the user.

The diff3 format provides a way to delineate the three choices, though not all of them may be present. Each choice is independent of any other choice, there is no connection between them. This presents a problem for tree-structured data because there is a dependency between, for example, inserting a start tag and inserting the corresponding end tag - unless these are done as a single choice the result will not be well-formed. This problem can always be overcome by duplicating some of the data, and the argument here is very similar to that presented at this conference last year regarding change to both content and structure [4]. Duplication can work well when the span of the change is small because very little data needs to be duplicated, but when the span is larger then more data needs to be duplicated and the nature of the change is lost in this duplication. In the extreme, duplication of the entire contents of an XML file will always yield a choice between well-formed fragments because each fragment is the entire file. This is correct but of course of little practical use.

We will look at some examples of changes to attributes where we can, with some manipulation of the data, present choices where the selection of any one of the two or three choices will provide a well-formed result.

Preserving well-formed tree structure in diff3

In this section we explore the issues of preserving the well-formed structure of XML or JSON when presenting choices in diff3.

Representing XML Element Tag Change in diff3

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. Here is an example of a change of structure.

Table 2. XML tag change

A.txt	O.txt	B.txt
This is a long paragraph where 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.	This is a long paragraph where 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.	This is a long paragraph where <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>.

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..

```
<p>This is a long paragraph
where
<<<<<<< A.txt
<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>
||||||| O.txt
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>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.txt
. </p>
```

We can improve this but at the cost of some intelligence on the part of the user to make consistent choices.

```
<p>This is a long paragraph
where
<<<<<<< A.txt
<strong>
||||||| O.txt
```

```
=======
<italic>
>>>>>>> B.txt
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
<<<<<<< A.txt
</strong>
||||||| O.txt

=======
</italic>
>>>>>>> B.txt
. </p>
```

What we really need here is some way to connect the relevant consistent choices so that it the start tag is selected then the appropriate choice of the end is also made automatically. One simple way to achieve this would be to add a choice id into the format. In this case we have given the three choices id value of 42. This is shown below.

```
<p>This is a long paragraph
where
<<<<<<<42< A.txt
<strong>
||||||| O.txt

=======
<italic>
>>>>>>> B.txt
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
<<<<<<<42< A.txt
</strong>
||||||| O.txt

=======
</italic>
>>>>>>> B.txt
. </p>
```

There are many ways this could be achieved syntactically, this is just one. The rules here would be:

1. A conflict may be labelled with an id.

2. For any labelled conflict, there must be at least one other labelled conflict with the same id value.

3. The selection of a choice within a conflict with an id automatically results in the selection of the corresponding choice, i.e. the choice with the same source file, within conflicts with the same id.

This is not a big change but would make a significant difference to the ease of use of diff3 format for structured data,.

Representing XML Attribute Change in diff3

XML attributes present a particular challenge for diff3 format. Here is an example of a change of value for an attribute.

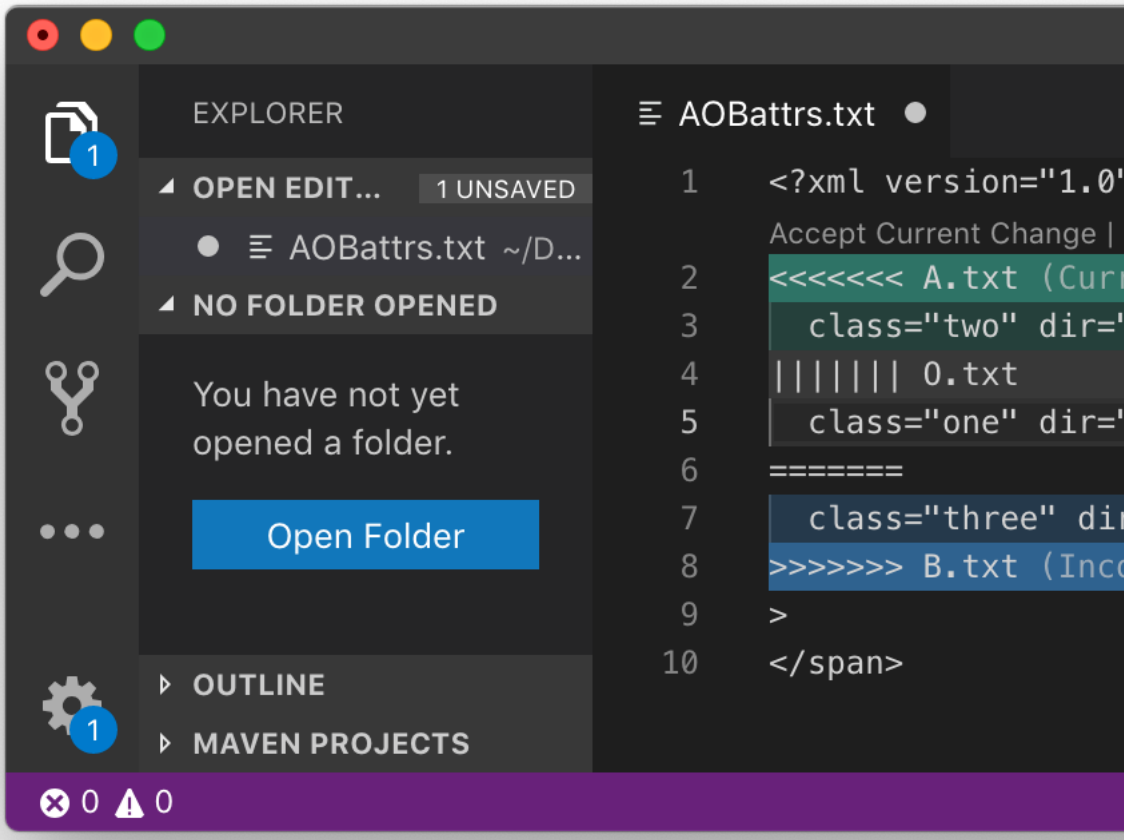
Table 3. XML Attribute value change

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

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.

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

Figure 1. Attribute example in Visual Studio code



In Figure 1, “Attribute example in Visual Studio code” we see how this can be displayed and managed in Microsoft Visual Studio code.

The above will produce syntactically correct results though it is not ideal because it would be more natural to choose the attributes separately rather than as a pair. This can be achieved by inserting additional white space so that we get two choices as shown below.

```
<span id="23"
<<<<<<< A.txt
class="two"
||||||| O.txt
class="one"
=======
class="three"
>>>>>>> B.txt

<<<<<<< A.txt
dir="rtr"
||||||| O.txt
dir="TBA"
```

```
=======
dir="ltr"
>>>>>>> B.txt
>
```

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. This is shown below.

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

Representing JSON Structure Change in diff3

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 4. 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 as shown below.

```
[
<<<<<<<42< A.txt
[
||||||| O.txt

=======

>>>>>>> B.txt

<<<<<<<61< A.txt

||||||| O.txt

=======
[
>>>>>>> B.txt
12,13,14
<<<<<<<42< A.txt
]
||||||| O.txt

=======

>>>>>>> B.txt
,20,21,22
<<<<<<<61< A.txt

||||||| O.txt

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

The above will produce syntactically correct results though it is not intuitive and requires careful allocation of id values for the conflicts to ensure correct behaviour. Note that the '[' in A has to be a separate conflict to the '[' in B although they are at the same position in the array. Note that it could be argued that these changes are not conflicts but this does not matter here, the point is that if we do want to represent them as choices for a user to select, then we are able to do so.

Representing JSON Separator Change in diff3

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 5. 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 by inserting additional white space so that we get two choices as shown below.

```
[
<<<<<<< A.txt
"one"
||||||| O.txt
"one"
=======

>>>>>>> B.txt

<<<<<<< A.txt
, "two"
||||||| O.txt

=======

>>>>>>> B.txt

<<<<<<< A.txt

||||||| O.txt

=======
, "three", "four"
>>>>>>> B.txt
]
```

However this does not work correctly because if the "one" is rejected by accepting the B.txt choice in the first conflict, then we do not need a comma before the next item. Unfortunately we cannot get round that using a connected choice. The problem here is to do with a combination of choices rather than one choice. We can just be pleased that XML attributes are not comma separated!

diff3 format as XML or JSON

An obvious question about diff3, when we are looking at XML and JSON, is whether or not we would get a significantly better result if we used XML or JSON instead of the fairly basic format of diff3.

The table below shows an example in diff3 and the corresponding file in XML and JSON using a very simple syntax in each case. The purpose here is just to explore whether or not it makes sense to do this.

Table 6. diff3 format in XML or JSON

diff3	XML	JSON

1
4
5
2
<<<<<<< A.txt
3
||||||| O.txt
3
4
5
=======
4
5
3
>>>>>> B.txt
6
	<d:diff3>
	1
	4
	5
	2
	<d:change><d:content original="A.txt">
	3
	</d:content><d:content original="O.txt">
	3
	4
	5
	</d:content><d:content original="B.txt">
	4
	5
	3
	</d:content></d:change>
	6
	</d:diff3>
		{
		"diff3": [
		"1",
		"4",
		"5",
		"2",
		{
		"change": {
		"A.txt": ["3"],
		"O.txt": [
		"3",
		"4",
		"5"
		],
		"B.txt": [
		"4",
		"5",
		"3"
		]
		}
		},
		"6"
		]
		}

For JSON, we have represented the sequence of lines as an array of strings where each line is a string and a change is an object where each member name is the name of the original file. We could have concatenated the lines with a '\n' delimiter but this would have been very difficult to read.

This shows that JSON changes the look and feel significantly due to the way it represents strings. XML is similar to the original, though some detail is left out here, for example `xml:space="preserve"` or `<![CDATA[` to preserve the formatting. If the original data is XML, then representing the changes in XML in this way would be very confusing and it would be better to embed the changes within the original XML, though this would require the original to be well-formed.

The addition of the id (to represent connected changes) would be very simple in XML as an attribute, but a little harder in JSON because it would mean adding another member to the change object.

This is not intended as an alternative to diff3 and it is clear that there would be issues to resolve if JSON or XML is used. XML does look more appropriate but it lacks one desirable characteristic of diff3: a file with no conflicts is the file, there is no need to remove anything from it.

Nested Changes

Given an XML or JSON representation we can go one step further and make 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> <firstname>Nigel</firstname> <surname>Whitaker</surname> </personname> <address> <phone>+44 1684 532141</phone> <street>Geraldine Road</street> <city>Malvern</city> <country>UK</country> <postcode>WR14 3SZ</postcode> </address> </author>	<author> <personname> <firstname>Nigel</firstname> <surname>Whitaker</surname> </personname> <address> <street>Geraldine Road</street> <city>Malvern</city> <country>UK</country> <postcode>WR14 3SZ</postcode> </address> </author>	<author> <personname> <firstname>Nigel</firstname> <surname>Whitaker</surname> </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.

Let us now consider how this could be represented using the proposed XML format presented in the previous section:

```
<d:diff3>
<author>
  <personname>
    <firstname>Nigel</firstname>
    <surname>Whitaker</surname>
  </personname>
  <d:change>
    <d:content origin="A.xml">
      <address>
        <phone>+44 1684 532141</phone>
        <street>Geraldine Road</street>
        <city>Malvern</city>
        <country>UK</country>
        <postcode>WR14 3SZ</postcode>
      </address>
    </d:content>
    <d:content origin="O.xml">
      <address>
        <street>Geraldine Road</street>
        <city>Malvern</city>
        <country>UK</country>
        <postcode>WR14 3SZ</postcode>
      </address>
    </d:content>
    <d:content origin="B.xml"/>
  </d:change>
</author>
</d:diff3>
```

Here we can see the deletion of the address in `B.xml` and if we carefully look at `A.xml` and `O.xml` we can work out that a phone child element has been added. But is there a better representation we can use? Given we are now using a representation that follows the tree structure we can also make use of this structure in the result. Here is an alternative result where we allow change to nest:

```
<d:diff3>
```

```
<author>
  <personname>
    <firstname>Nigel</firstname>
    <surname>Whitaker</surname>
  </personname>
  <d:change>
    <d:content origin="A.xml, O.xml">
      <address>
        <d:change>
          <d:content origin="A.xml">
            <phone>+44 1684 532141</phone>
          </d:content>
          <d:content origin="O.xml"/>
        </d:change>
        <street>Geraldine Road</street>
        <city>Malvern</city>
        <country>UK</country>
        <postcode>WR14 3SZ</postcode>
      </address>
    </d:content>
    <d:content origin="B.xml"/>
  </d:change>
</author>
</d:diff3>
```

Here we can see that by allowing nested change and making some small adjustments to the format to allow multiple versions to be specified in origin attributes, we can avoid the repetition and make it easier for a human to understand. However we have moved further from the simple diff3 representation in this step. Rather than choose one of two or three possibilities at each step we now need to understand reuse of content and a more complex format is used for the origin attributes.

As well as being more compact and allowing reuse there is a further benefit, in that in some cases nested changes can be ignored. Suppose we decided to accept the change made in `B.xml`, the deletion of the address. In this case we would take the `B.xml` content, i.e. nothing, and immediately delete the other `d:content` alternatives at that level of the tree. We do not need to consider the nested change related to the phone element when we choose the outer `B.xml` alternative.

It is also possible to prove that for a three way merge process, as used by diff3, at most two levels of nested change/content structure is required. This can be generalized so that for n-way merge algorithms (akin to the idea of 'octopus merge' used in git) a maximum of n-1 levels of nested change are required.

We have not explored how the original diff3 representation could be enhanced to handle nested change, it could be done but it is more natural in the context of an XML representation of change.

Conclusions

In this paper we have explored whether we can integrate more modern structure-aware comparison tools with the existing diff3 format so that there is minimal change for users. We have shown that by laying out comparison results for structured representations such as XML or JSON we can make them easier to process and more likely to provide well-formed or valid results. We have shown that there are limitations, in particular the representation of connected changes, where some more intelligence in the diff format is needed to ensure the result is well-formed. Nested changes can also be represented with amendments to the diff format, but this is more complicated and is likely to be easier using XML rather than a variant of diff3.

Even if these things are possible that does not necessarily mean we should go down this route. It is worth considering some of the history and how we got here. Early version control systems were in use with 24 line, 80 column VDUs and with editing tools such as vi and emacs. In those days developers

intimately understood the representation and manipulated it directly. We are now used to using IDEs which directly support version control operations in their graphical user interfaces. In many cases these interfaces hide the change markers that we have been discussing and instead present the user with side-by-side alternatives and GUI control buttons to resolve differences. We could consider the display of the diff3 style change-markers akin to the concept of 'tag display' modes in word-processors and XML editors. In many of these systems it is either impossible to see any underlying markup or it is a feature for advanced (or perhaps 'older' ?) users that needs to be explicitly turned on, with the growing trend for the default being to hide the markup from the user. Is there a similar trend with change and conflict markers? This implies that the actual syntax used to represent the changes and conflicts is less important than it used to be, and there is less need to try to preserve it.

In our recent 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 algorithm from subsequent conflict resolution tools or 'merge tools'. In pursuit of the best way forward, we have further explored these possibilities and we have implemented the layout approaches discussed earlier.

We have shown that improvements to the representation of change for structured data is possible and desirable. Changing the existing diff3 format is awkward and limited, so it might be better to move directly to a markup representation using XML because the text will not be directly edited by users and mature tools are available to process the XML. Arguably it would be simpler to avoid these issues and present users with a merge user interface that understood structured content and provided operations which preserved the well-formed nature or validity directly. However, the value of a standard format for such conflicts and changes is that the merge tool is primarily a GUI and the user can choose the merge algorithm and the merge tool independently.

We have presented this paper to explore these ideas, but we are not suggesting that the best approach is extending or enhancing the current diff3 representation. An XML alternative to diff3 would have some advantages and should be explored as a longer-term improvement for representing and processing conflicts and changes in structured data.

References

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

[2] *Visual Studio Code User Guide: Using Version Control [https://code.visualstudio.com/docs/editor/version-control]*

[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] Robin La Fontaine *When Overlapping XML Meets Changing XML Does Confusion Reign? MarkupUK 2018 [https://markupuk.org/2018/Markup-UK-2018-proceedings.pdf#page=153]*

[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]*