Difference Output Format (EDUL)

This section describes the output format of the differencing tool, named EDUL. EDUL stands for Extended Delta Update Language, and it is an extension of Adrian Mouat's difference output format DUL [19]. The first section discusses the differences between those two formats. The four following sections thoroughly describe the insert, delete, update and move edit operations. Finally, the last section gives an example of the EDUL output format.

Throughout this section the XPath standard [35] and DOM Level 2 Core Specification [32] are heavily utilised. For the sake of clarity, the rest of this chapter will refer to the node that an edit operation was applied to as the *changed_node*. Moreover, the elements of the *treePatchFile* that describe the four edit operations will be referred as <insert>, <delete>, <update> and move respectively.

The Differences between EDUL and DUL

Although DUL is well documented, it is not adequate enough to satisfy the requirements of this thesis. As a result, the difference output format is extended in order to be able to be successfully applied to an arbitrary document, as well as handle all the special cases that were discussed in the previous chapter. The following paragraphs describe all the changes from DUL to EDUL. The full description of EDUL output format can be found in sections 5.2.2 to 5.2.5.

Insert

In order to be able to handle all the special cases related to the order of the children after the insertion of the *changed_node*, information regarding the previous and the next sibling of the node was needed. So, the name and the value of those two nodes were added in the <insert> element, which represents the insert edit operation.

Delete

Chapter four defines that if the value of the *changed_node* is updated in *file 2* it should not be deleted. In order to satisfy this condition, an attribute that encloses the value of the *changed_node* in the *base file* is added.

Update

Similarly to the delete edit operation, in case the value of the *changed_node* is updated in *file 2*, it should not be updated again. So, an attribute that keeps that value in the *base file* is added.

Move

Finally, information on the previous and the next sibling of the *changed_node* in the *base file* is added in the move element. Thus, the order of the children can be checked.

Insert Edit Operation

Sections 5.2.2 to 5.2.5 describe in detail the four edit operations that are considered in this thesis. The first XML element that is discussed is <insert>. This element represents the insertion of a leaf node. All of the attributes of this element are presented below.

nodetype

The *nodetype* attribute determines the type of the *changed_node*. Its value equals to the value of the DOM method getNodeType(). Figure 5-2 shows the values that the DOM method returns.

Node Name	Return value of DOM getNodeType()
Element	1 - ELEMENT_NODE
Attribute	2 - ATTRIBUTE_NODE
Text	3 – TEXT_NODE
Processing Instruction	7 - PROCESSING_INSTRUCTION_NODE
Comment	8 – COMMENT_NODE

Figure **Error! No text of specified style in document.**-1 – The DOM getNodeType() method

name

The second attribute of the <insert> element represents the name of the *changed_node*, and it is equal to the value returned by the DOM method getNodeName(). The *name* attribute will be omitted in cases where the *changed_node* is either a TEXT_NODE or a COMMENT_NODE, because the value that the DOM method getNodeName() returns for all of those nodes is *#text* and *#comment* respectively.

parent

This attribute uniquely identifies the XPath parent of the *changed_node* in *file 1*. The attribute is not used exclusively to locate the parent node of the *changed_node* in *file 2*, assuming that the position of the parent node might not be the same. The algorithm to locate the parent node, it is discussed in section 5.4.

childno

Similarly to the *parent* attribute, this is another attribute that it is not used directly. It only indicates the position where the *changed_node* should be inserted. The attribute is used in conjunction with four other attributes, named *prev_sib_name*, *prev_sib_value*, *next_sib_name* and *next_sib_value* that are discussed below.

prev_sib_name – prev_sib_value

Although *prev_sib_name* and *prev_sib_value* are two separate attributes, they are considered together, because they both keep information about the previous sibling of the *changed_node* (this node is returned by the DOM method getPreviousSibling()). In case there is no previous sibling, both of the attributes are omitted.

The first attribute stores the name of the previous sibling of the *changed_node*, in the same way that the attribute *name* works. The second attribute represents the value of the previous sibling. The value of this attribute is equals with the value of the DOM method

getNodeValue(). So, in case that the previous sibling is an ELEMENT_NODE, the return value of the method will be *null* and the attribute will be omitted.

next_sib_name – next_sib_value

Equally, the *next_sib_name* and *next_sib_value* attributes store information on the next sibling of the *changed_node*. *Next_sib_name* stores the name of the next sibling, whilst *next_sib_value* its value as they are defined by the DOM methods getNodeName() and getNodeValue() respectively. If the value of the next sibling is *null*, *next_sib_value* attribute will be excluded.

The last sour attributes determine where the *changed_node* should be inserted and will handle several conflicts that were discussed in chapter 4.

content

The content of the insert element stores the value of the *changed_node*. In case that the node is an ELEMENT_NODE, its value will be *null*, and the <insert> element will be empty.

example

The following example demonstrates an insertion of an ELEMENT_NODE with name equals to “genre”. As we can observe, the previous sibling is a TEXT_NODE and the next one is a “year” element.

```
<insert nodetype="1" name="genre" parent="/node()[1]/node()[4]" childno="6" prev_sib_name="#text" prev_sib_value="This is the previous sibling" next_sib_name="year"> </insert>
```

Delete Edit Operation

The second XML element discussed is <delete>, and represents the deletion of a leaf node (remember that the differencing algorithm only allows deletions of leaves). The attributes of <delete> element are described below.

node

This attribute uniquely identifies the XPath node in *file 1* to be deleted. However, assuming that the *changed_node* might have been moved in *file 2*, this attribute will not be used exclusively to locate the node. The detailed algorithm that describes how the *changed_node* is being located is described in section 5.4.

charpos - length

The *charpos* and *length* attributes are considered together since they can only be used in conjunction. Those two attributes are only applicable in case that character data is deleted. The value of *charpos* attribute sets the first character of the text to be deleted, while *length* attribute specifies the number of character to be deleted. So, in case that a <delete> element has attributes *charpos="5"* and *length="4"*, *treePatch* should delete the 5^{th}, 6^{th}, 7^{th} and 8^{th} character of the node.

In case that the node we want to delete is an ELEMENT_NODE (the type of the node is determined by the return value of the DOM method getNodeType()) the attributes will be omitted.

value

Attribute *value* stores the value of the *changed_node* in the *base file*. The value of the node is determined by the DOM method getNodeValue(). In case the *changed_node* is an ELEMENT_NODE, the attribute will be omitted because the value of all element nodes is *null*.

This attribute was added in order to satisfy the major special case of the delete edit operation that was discussed in the fourth chapter. Before the delete operation is applied

to the *changed_node*, *treePatch* needs to check that the value of the node in *file 2* is equals to the value of the same node in the *base file*. In case that those two values are not identical, the operation will not be applied.

example

An example that shows all the attributes of the <delete> operation is illustrated below. The edit operation of this example is applied to a text node, and will delete the whole text of the node, in case that its value in *file 2* is equals to “This is a text node”.

```
<delete charpos="1" length="19" node="/node()[1]/node()[2]/node()[4]"
value="This is a text node"></delete>
```

Update Edit Operation

At the beginning of this chapter, all the bugs of Adrian Mouat’s implementation were clearly stated. One of those was that the differencing tool does not support the update edit operation (contrary it interprets all the updates as a deletion followed by an insertion). The current implementation uses Adrian Mouat’s differencing tool, so it does not support the update operation. However, all attributes of the <update> element will be described in detail, because the difference output format can be used independently of the rest of the tool.

The update edit operation is not applicable to element nodes, because their value is always *null*.

node

The *node* attribute is used in exactly the same way as the <delete> element. It uniquely identifies the XPath node that the update operation is applied to. Similarly to the <delete>

element, this attribute will not be used exclusively to locate the *changed_node* in *file 2*, because the position of the node might have been changed.

charpos - length

The first of those two attributes, named *charpos* defines the first character of the text to be updated. The latter stores the number of characters that will be updated.

oldvalue

This attribute was added in order to handle some of the special cases that were discussed in the previous chapter, and stores the value of the *changed_node* in the *base file*. Before an update operation is applied to the *changed_node*, *treePatch* should ensure that *file 2* has not also changed the value of that node. In case that both of the authors of *file 1* and *file 2* have updated the value of the same node, the final version after the patching will contain the node with the value in *file 2*..

content

Correspondingly to the <insert> element, the content of the <update> element stores the new value of the *changed_node*. The value of the node is equals to the value that the DOM method getNodeValue() returns.

example

The following example illustrates the case where the value of a text node is updated. The value of the node is equals to the value of the *oldvalue* attribute, so the update operation will be applied.

<update node="/node()[1]/node()[2]/node()[4]" charpos="1" length="19" oldvalue="This is a text node">This is a text node</update>

Move Edit Operation

The last XML element that will be discussed here represents the move edit operation. Although Adrian Mouat's implementation was unable to successfully apply move operations, the addition of four attributes in the <move> element fix that bug. Those attributes contain information regarding the left and the right sibling of the *changed_node* in *file 1*. The results of that extension are discussed in chapter 7. All of the attributes of the <move> element are described in this section.

node

The *node* attribute uniquely identifies the XPath node to be moved. Once more, the attribute will not be used directly to locate the *changed_node* in *file 2*.

parent

This attribute uniquely identifies the parent of the *changed_node* in *file 1*. The reader should notice that the *parent* attribute contains information about the parent of the *changed_node*, after the move operation is applied. This attribute is the only piece of information that *treePatchFile* contains about the target node.

charpos - length

Those two attributes were previously discussed. They are only applicable in cases where character data is moved. The value of *charpos* attribute sets the first character of the text to be moved, whilst the *length* attribute specifies the number of characters to be moved.

In case that the *changed_node* is an ELEMENT_NODE, both of the attributes will be omitted.

childno

The *childno* attribute is used in equally to the <insert> element. It only implies the position of the child node but in practice the following attributes, named *prev_sib_name*, *prev_sib_value*, *next_sib_name* and *next_sib_value*, are used in order to find the right position.

prev_sib_name - prev_sib_value

Those two attributes store information on the previous sibling of the *changed_node* in *file 1* (the previous sibling is the node that is returned by the DOM method getPreviousSibling() for the *changed_node*). If the *changed_node* is the first child in *file 1*, both of the attributes will be omitted.

The first attribute stores the name of the previous sibling of the *changed_node*. The latter attribute represents the value of the previous sibling. In case that the previous sibling of the *changed_node* is an ELEMENT_NODE, the attribute *prev_sib_value* will be excluded.

next_sib_name - next_sib_value

Likewise, the *next_sib_name* and *next_sib_value* attributes give information regarding the next sibling of the *changed_node* in *file 1*. Attribute *next_sib_name* stores the name of the node, whilst the second attribute stores its value, as they are defined by the DOM methods getNodeName() and getNodeValue() respectively.

If the *changed_node* does have next sibling in *file 1*, both of the attributes will be omitted, whereas in case the value of the next sibling is *null*, the *next_sib_value* attribute will be skipped.

example

The following example shows the case where a node is moved. Its previous sibling in *file 1* is a text node, and its value is equals to “This is the previous node”, whereas the next sibling is also a text node.

```
<move node="/node()[1]/node()[4]/node()[6]" parent="/node()[1]/node()[4]" childno="7" next_sib_name="#text" prev_sib_name= "#text" prev_sib_value="This is the previous node" next_sib_value="This is the next node"></move>
```

EDUL Example

A complete example of the EDUL output format is presented below. In figure 5-3, three edit operations are shown; move, insert and delete.

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

<delta>
  <move childno="7" length="0" next_sib_name="genre" next_sib_value=""
   node="/node()[1]/node()[4]/node()[7]" parent="/node()[1]/node()[4]"
   prev_sib_name="year" prev_sib_value=""></move>

  <move childno="7" length="0" next_sib_name="#text" next_sib_value=""
   node="/node()[1]/node()[4]/node()[6]" old_charpos="8"
   parent="/node()[1]/node()[4]" prev_sib_name="#text"
   prev_sib_value=""></move>

  <insert charpos="1" childno="1" name="#text" next_sib_name="#text"
   next_sib_value="Old text" nodetype="3" parent="/node()[1]/node()[4]
   /node()[4]" value="New text">New text</insert>

  <delete charpos="9" length="8" node="/node()[1]/node()[4]/node()[4]
   /node()[1]" value="Old text"></delete>
</delta>
```

Figure **Error! No text of specified style in document.**-2 – EDUL Difference Output Format