In LEOS, for comparison of two XHTML files, both texts are fed into a customized solution which injects the <span> elements with appropriate CSS classes to highlight the added, removed, or modified parts.

There are three major parts of the algorithm which compares two supplied XHTMLs

1) XML structure comparator (XMLContentComparator.java)

2) Identification of best match among nodes in same tree depth

3) Text comparator (JavaDiffUtilContentComparator.java)

XML structure comparator (XMLContentComparator.java)

**1) XML structure comparator (XMLContentComparator.java**)

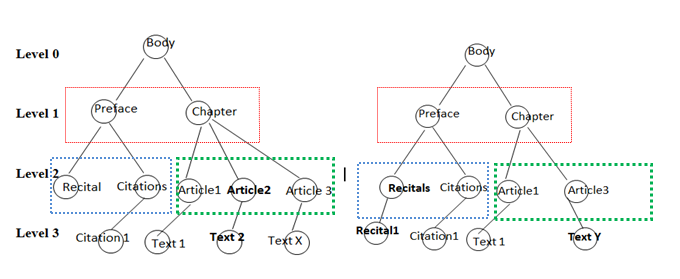
The processing starts in this component and it calls other parts as needed. The two XHTMLs are walked simultaneously using in-order traversal for which VTD parser is used.

In a single iteration, this component compares nodes from both XHTMLs for same depth and tries to find the best match from nodes in same depth.

**Step 1)** Let's take example of two docs as below. In second iteration we have:

oldNodes=[Preface, Chapter]

newNodes=[Preface, Chapter]



**Step 2)** Now from ***oldNodes***, *Preface* is taken and checked for best match with **Best Match Algorithm** in ***newNodes***=[*Preface*, Chapter]. If id of *Preface* node is the same then it is the best match else it is compared to *Chapter* node.

**Step 2A)** If no node matches *Preface* node in ***newNode*** then *Preface* node in ***oldNode*** is marked as **Removed** with appropriate CSS tag. And Step 1 is called with next node, *Chapter*.

**Step 3)** Let's assume ids are the same for both *Preface* nodes, then we check the equality of content.

**Step 3A)** If contents are same then we set the *Preface* node as not changed and terminate the node traversals.

**Step 4)** When node contents are different,

**Step 4A)** Then we check if node is of text type,

If yes then we call **Text Comparator**

If not then then we call step 1 again, with children taken of node *Preface*, i.e.

oldNodes=[Recitals, Citations]

newNodes=[Recitals, Citations]

Let's take another iteration from the same example:

**oldNodes newNodes**

↓ Article 1 Article 1

↓ Article 2

↓ Article 3 Article 3

*Article 1* is checked for best match in **newNodes**

Best match identified at position 1, so now it is checked for content equality.

Contents are same, so it is marked as no change and next iteration is started.

*Article 2* is checked for best match in **newNodes**

No match found, so it is marked as removed and next iteration is started.

*Article 3* is checked for best match in **newNodes**

Best match identified at position 2.

3A) If there are any nodes before it in **newNodes** that where not matched in previous iteration then they are marked as inserted.

3B) Content is compared with that of *Article 3* in **newNodes**.

3C) If contents are different then we check *Article 3* for text child (here we take it as Text for simplicity).

3D) If it has text then it is marked as modified and **Text comparator is** called to mark individual changes inside the text content.

**2) Best matching nodes in same tree depth.**

For finding best match, a numerical rank is assigned to **each unhandled** child in **newNodes**.

If id of node is equal then it is considered best match and is returned.

If ids of both sides are not found and old and new nodes have same TagName (for instance both are ***aknP*** or both are ***num***) then we calculate a numerical rank for each node and the highest ranking node is considered as best match.

(Basic idea of rank is, farther the node, lesser its rank, in combination with more similar the node then higher the rank.)

**3) Text comparator**

Text comparator uses patch functionality of google-diff-match-patch library.

1) All Authorial Notes are encoded into a single HTML node as these are special nodes containing text as attributes. A map is saved with original content.

<authorialNote >xx</authorialNote > 🡪 <Anode"+id+marker+textContent+"/>

2) All the text content is broken in words with space " " as splitter and all the HTML tags are placed in new lines.

3) These lines are fed to the google patch also which gives output as sets of inserted, removed and modified lines.

4) Now these patches are recombined in a buffer along with appropriate markers for added or removed content.

5) All encoded Authorial Notes are decoded and replaced with the original content.

6) Finally, the resulting string is returned to caller.