Automatic Generation of a Differential XSL Stylesheet From Two XML Documents

Norihiro Ishikawa, Takeshi Kato, Hidetoshi Ueno, Hiromitsu Sumino and Hideharu Suzuki
Multimedia Laboratories, NTT DoCoMo
3-5, Hikari-no-oka, Yokosuka, Kanagawa, 239-8536, Japan
TEL: +81-468-40-3819
E-mail: {ishikawa, t_kato, hueno, sumino, hideharu} @mml.yrp.nttdocomo.co.jp

ABSTRACT

We propose a new technique for generating a differential XSL stylesheet from arbitrarily two XML documents. The Document Object Model (DOM) is adopted to express the data structure of XML documents. We propose several algorithms to detect difference information between the two DOM trees. The difference information, which consists of its difference type, position and content, is generated for each differential node by using proposed algorithms. We use the XSL Stylesheet as the format of expressing difference information, and we also propose the way to create XSL Stylesheet from the difference information.

Keywords

XML, XSL, XSLT, XPath, DOM

1. Introduction

Push information delivery services have been emerging as new services over the Internet. Contents such as news, advertisement information and weather information are provided from push servers. It is desirable to provide difference data of a content if the content is updated because the difference between the old content and the updated content tends to be small. The retransmission of a whole original content is waste of bandwidth when transmitting the content over the wireless network. The paper [1] also showed that the benefit to transmitting difference data only if contents are updated. W3C has standardized a technology for transforming XML documents that is called XSL Transformations (XSLT) [2]. If a differential content between two XML documents (e.g. XHTML document) is generated as an XSL stylesheet and the differential content is only transmitted instead of transmitting a whole document, the amount of transmitted data can be reduced. When a client receives the differential content, it can reconstruct a whole content, by applying the differential content to the original content by using the XSLT engine. To realize the above scenario, we propose a technique for the automatic generation of a differential XSL stylesheet from arbitrarily two XML documents.

2. Proposed Technique

We adopted the Document Object Model (DOM) [3] to express the XML data structure. Two XML documents are parsed and converted into two DOM trees. Differential parts of the two DOM trees are detected by using proposed algorithms as described in section 2.1. After detecting deferential information, the XSL Stylesheet is created from the information (Figure 1).

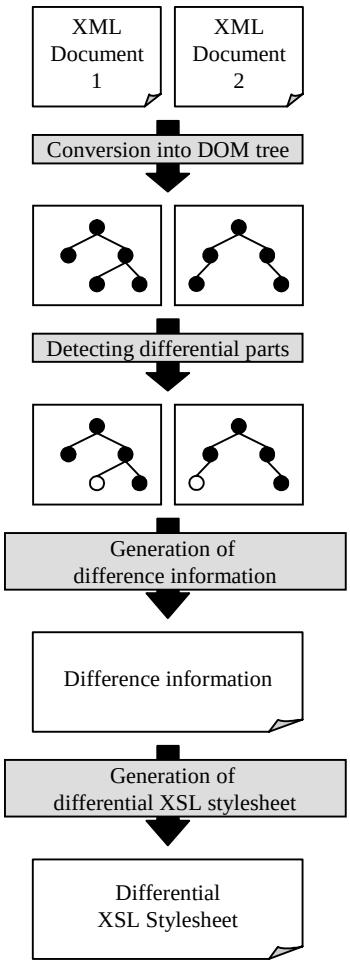


Figure 1: Generation Process of Differential XSL Stylesheet

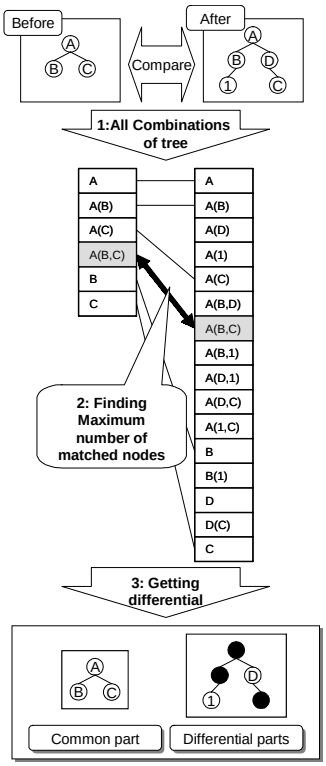
2.1 Detection of Differential Parts from Two DOM Trees

Two DOM trees are scanned by using difference-detection algorithms. In this process, deleted and inserted nodes are detected. We have considered three algorithms to detect difference parts as described below.

Algorithm-1: (1) All combinations of nodes in both trees are created. (2) Each combination of nodes is compared in all cases, (3) we can find the DOM trees that have maximum number of matched nodes. This algorithm is shown at figure 2.

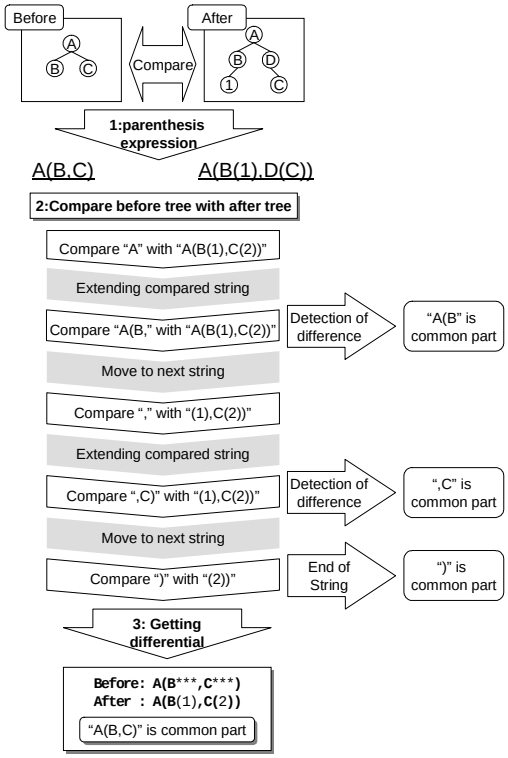
Algorithm-2: (1) The parenthesis expressions of each DOM tree are compared as follows. (2) Each parenthesis expression of tree is compared with the other one from top of a string of character. If the maximum common strings between them are detected, those strings are regard as common parts. Then each parenthesis expression of tree is compared with the other one again from next character. (3) This procedure is repeated until end of string of character and common parts are got. This algorithm is shown at figure 3.

Algorithm-3: (1) Corresponding nodes of the two DOM trees are compared from the root to the leaf. (2) This procedure is repeated until the difference between two corresponding nodes is detected for the first time. (3) Then the rest of sub trees are regarded as differential parts. This algorithm is shown in figure 4.



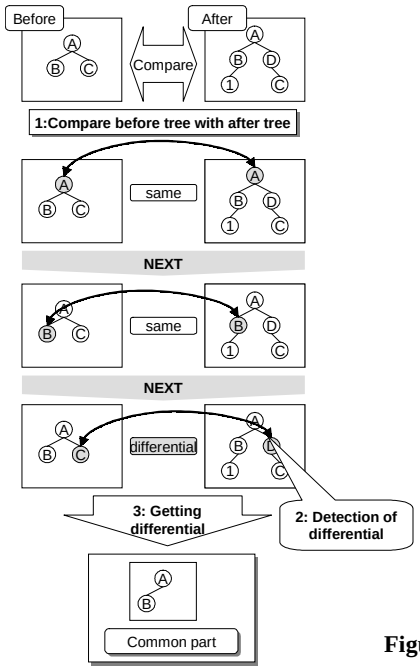

```
tree subtree1[] =  new tree[count of all combinations of tree1];
tree subtree2[] =  new tree[count of all combinations of tree2];
tree commonsubtree;

subtree1[] = "All combinations of tree1";
subtree2[] = "All combinations of tree2";
commonsubtree = nothing;

for (i = 1 to (count of all combinations of tree1)) {

   for (j = 1 to (count of all combinations of tree2)) {
      if (subtree1[i] = subtree2[j]) {
         if (Size(commonsubtree) < Size(subtree1[i])) {
            commonsubtree = subtree1[i];
         }
      }
   }

}

Save commonsubtree;
```

Figure 2: Algorithm-1




```
int k,m,p;
string treestring1 = Character expression of tree1;
string treestring2 = Character expression of tree2;

for (i = 0 to treestring1.length) {
k = 0; p = 0; m = 0;
for (j = 0 to treestring1.length() - i) {
   m = p;
   p = treestring2.indexOf(treestring1.substring(i,i + j + 1));
   k = j;
   if (p = -1) {
      break;
   }
}
if (i + 1 != treestring1.length()) {
   if (k > 0) {
      Save as Common Part (treestring1.substring(i,i + k));
      if (m != 0) {
         Save as Differential Part in After one (treestring2.substring(0,m));
      }
      treestring2 = treestring2.substring(m + k);
      i = i + k - 1;
   }
   else {
      Save as Differential Part in Before one (treestring1.substring(i,i + k + 1));
      i = i + k;
   }
}
else {
   Save as Common Part (treestring1.substring(i,i + k + 1));
   treestring2 = treestring2.substring(m + k);
   i = i + k;
}
}
```

Figure 3: Algorithm-2




```
string treeroute1[] = new tree[count of leaf in tree1];
string treeroute2[] = new tree[count of leaf in tree2];
string commonpart;

treeroute1[] = "All route from root to leaf of tree1";
treeroute2[] = "All route from root to leaf of tree2";
leafcount = Select smaller one (count of leaf in tree1,count of leaf in tree2);

for (i = 1 to leafcount) {
   routecount = Select smaller one (treeroute1[i].length,treeroute2[i].length);
   for (j = 1 to routecount) {
      if (treeroute1[i].substring(0,j) = treeroute2[i].substring(0,j)) {
         commonpart = treeroute1[i].substring(0,j);
      }
      else {
         break;
      }
   }
Save commonpart;
}
```

Figure 4: Algorithm-3

2.2 Generation of Difference Information Using Elements and Attributes of Nodes and XPath

This process generates the difference information between two DOM trees. The difference information is derived from the information on common and differential parts of two DOM trees. The difference information consists of its difference type (i.e. deletion, insertion and alteration), position and content for each differential node. The position of a node within a DOM tree is represented by XPath [4].

Description	Abbreviate syntax
child::	Omissible
attribute::	@
/descendant-or-self::node()/	//
Self::node()	.
Parent::node()	..

Table 1: Sample of Abbreviate Syntax

2.3 Generation of Differential XSL Stylesheet

This process generates a differential XSL stylesheet from the differential information. Some considerations should be given to this process, to minimize a differential XSL stylesheet. For example, a node in the same level should be designated by "node ()", and copying a node without having attributes should be carried out by "<xsl:copy/>". Additionally, the use of an abbreviate syntax (See Table 1) and an abbreviate XPath (e.g. relative path) should be considered.

3. Consideration

3.1 Evaluation on Difference-Detection Algorithms

Future research should look at the following issues and developing prototype software of the proposed technique.

(1) Validation of proposed algorithms
(2) Comparative evaluation on the performance of proposed algorithms
(3) Generation of the minimal differential XSL stylesheet from the difference information

Using Current XSLT
<xsl:template match="/A/C"/> <xsl:template match="/A/C/node()"/>
Using Generation Function
<xsl:copy-of select="/A/node()[generation()=2]"/>

Table 2: Sample of Using Generation Function

3.2 Proposal on New Elements and Functions for XSLT

XSLT is not optimized for generating a differential XSLT stylesheet. We propose new elements and functions of XSLT for this purpose. In XSLT, a sub-tree can be regarded as a node-set using “<xsl:copy-of>”. However a node-set must include all nodes from the root of a sub-tree. Therefore a node-set cannot be used to designate the part of a sub-tree. The part of a sub-tree can not be copied at once using “<xsl:copy-of>”. We propose a new function “generation” that designates nodes from the root of a sub-tree to the n-th generation descendants. We think that introducing such function can optimize a differential XSLT stylesheet (See Table 2).

4. References

1. Jeffrey C. Mogul, et al.: Potential benefits of delta-encoding and data compression for HTTP, Proceeding of SIGCOMM 97 (September 1997).
2. J. Clark: XSL Transformations (XSLT) Version 1.0, W3C Recommendation (November 1999).
3. Arnaud Le Hors et al.: Document Object Model (DOM) Level 2 Core Specification, W3C Recommendation (November 2000).
4. J. Clark and S. DeRose: XML Path Language (XPath) Version 1.0, W3C Recommendation (November 1999).