AN AUTOMATIC GENERATION METHOD OF DIFFERENTIAL XSLT STYLESHEET FROM TWO XML DOCUMENTS

Takeshi Kato, Hidetoshi Ueno, Norihiro Ishikawa
Network Management Development Department, NTT DoCoMo, Inc., 3-5 Hikarino-oka, Yokosuka, Kanagawa, JAPAN

Keywords: XML, DOM, XSLT, Differential Data

Abstract: We propose a differential XSLT stylesheet generation method for arbitrary pairs of XML contents. It is possible to obtain the revised XML document by supplying the XSLT stylesheet with the differential data to the original XML document. Comparing with sending whole revised XML document, the original XML document can be updated by sending less information, the differential data. This paper introduces a difference detection algorithm based on the DOM tree and a difference representation method that permits the expression of difference information. We also discuss a new XSLT function for the proposed method. We also introduce prototype software implemented based on proposed method and evaluation result that shows the effectiveness of our method. An experiment shows that the proposed method is suitable for updating XML contents, especially for web service in the costly mobile network.

1 INTRODUCTION

XML (eXtensible Markup Language)[Bray, 2000] is an extensible meta-language that is being widely applied in description languages such as XHTML (eXtensible HyperText Markup Language), SVG (Scalable Vector Graphic) and CC/PP (Composite Capability / Preference Profiles) as well as communication protocols such as SOAP (Simple Object Access Protocol). Unfortunately, XML content is generally large compared to CSV (Comma Separated Value) and TLV (Tag-Length-Value) content because XML uses element tags, attributes, and XML declarations. In mobile web services, XHTML[Baker, 2000] mobile profile has become the standard markup language for mobile phones. Retransmitting the whole content wastes bandwidth and time, especially in the wireless network. It is becoming more and more important to promote efficiency in the update process and version control of XML data. An effective idea is updating content locally by using difference data because the difference between the old content and the updated content tends to be small. Mogul et al (1997) confirmed the benefit of transmitting just the difference data, for example HTTP Delta-encoding[Mogul, 2002]. If the differential data of two XML documents is generated and transmitted instead of the whole document, the amount of transmitted data can be greatly reduced. When a client receives the differential data, it regenerates the new content, by applying the differential content to the original content.

To realize the above scenario, we propose a technique for the automatic generation method of a differential XSLT stylesheet from two arbitrary XML documents and show the availability of description of difference data using a differential XSLT stylesheet. This technique can also be provided to web services such as push information delivery service, weblog service and so on, in which content need to be frequently and partially updated. The proposed technique can promote efficiency in the update process and version control of them.

In this paper, section 2 describes overview of the proposed method. In section 3, the prototype software and the results of experiments are mentioned. Section 4 proposes some extensions to XSLT for the proposed method and related works are addressed in Section 5. We conclude this paper in Section 6.

2 PROCESS OF DIFFERENTIAL XSLT STYLESHEET GENERATION

The proposed method uses XSLT (eXtensible Stylesheet Language Transformations) [Clark, 2000]

Kato T., Ueno H. and Ishikawa N. (2005).
AN AUTOMATIC GENERATION METHOD OF DIFFERENTIAL XSLT STYLESHEET FROM TWO XML DOCUMENTS.
In *Proceedings of the First International Conference on Web Information Systems and Technologies*, pages 5-12
DOI: 10.5220/0001229300050012

for updating XML document. XSLT is a language that provides the function of data structure transformation for XML documents. We adopted the Document Object Model (DOM) [Le Hors, 2000] to express the XML data structure. An original and updated XML documents are parsed and converted into two DOM trees. We generate differential XSLT stylesheets in three steps because it is difficult to perfectly extract differential data in one operation. The proposed method is shown in Fig. 1.

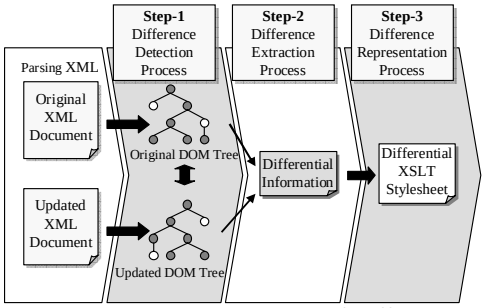


Figure 1: Generation Process of Differential XSLT Stylesheet.

Step-1 Difference Detection Process

The differences between the two DOM trees are detected. In this process, common parts and differential parts are identified.

Step-2 Difference Extraction Process

We define added nodes, deleted nodes and change nodes as differential data. In this process, added nodes, deleted nodes and change nodes are classified as described later.

Step-3 Difference Representation Process

Differential data is mapped to XSLT templates and a differential XSLT stylesheet is generated from them.

The following subsections describe these processes in detail.

2.1 Difference Detection Process

In the first step, accordant, appearing and disappearing nodes are detected. We propose a new detection algorithm that changes the DOM tree structures, maximizes the accordant tree and regards the remaining nodes as differential nodes. The assumptions of the algorithm are as follows.

i) A virtual node is assumed at the head of an actual root node and regarded as a temporary root node. It is treated as a common node. (Fig. 2)

ii) The goal is to maximize the number of common nodes.

iii) The term "common" means that the nodes have the same element names, same attribute names, and same attribute values.

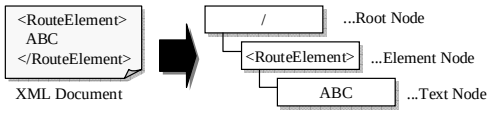


Figure 2: Structure of XML Document.

Note that it is not necessary to compare an element node, an attribute node and a text node if they are treated as DOM nodes. Nodes are compared type-by-type for efficiency.

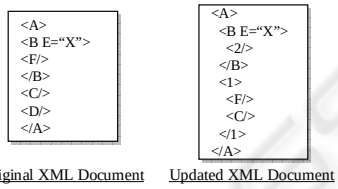


Figure 3: Example of original and updated XML Document.

The proposed difference detection method is described below with reference to Fig. 4 using the XML document in Fig. 3.

(a) Finding maximum number of common nodes (Fig. 4(a))

The method compares the components of the DOM trees to find the maximum number of common nodes.

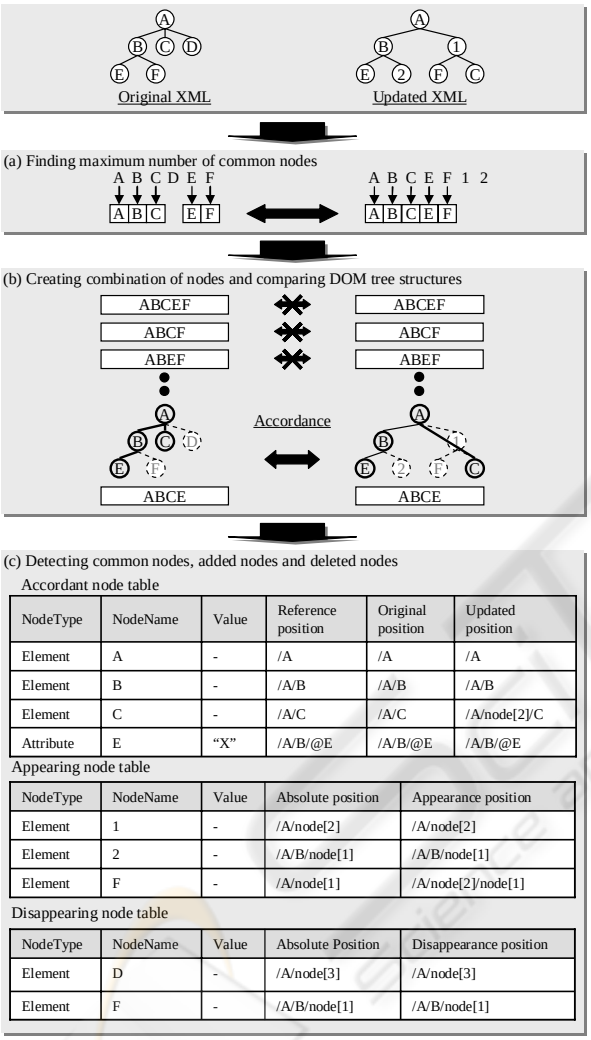
(b) Creating combination of nodes and comparing DOM tree structures (Fig. 4(b))

Combinations are compared in decreasing order of the number of nodes.

(c) Detecting accordant nodes, appearing nodes and disappearing nodes (Fig. 4(c))

The node-set in the DOM tree that has the maximum number of accordant nodes, are regarded as determinate accordant node-set. Other nodes in the DOM trees than determinate accordant node-set are taken to be the determinate differential nodes, which are then classified into two types, appearing nodes and disappearing node. Appearing nodes are the determinate differential nodes in the original DOM tree. Disappearing nodes are the determinate differential nodes in the updated DOM tree. The information of each node is arranged into three tables. In the accordant node table, "Reference position" column indicates the position of each accordant node. "Original position" column means the position of the accordant node in the original DOM tree and "Updated position" column similarly indicates the position in the updated DOM tree. In the appearing node table, "appearance position" column indicates the position in the updated DOM tree. "Absolute position" column means the relative position of correspondent node in the accordant DOM tree. In the disappearing node table, "disappearance position" column indicates the

position in the original DOM tree. "Absolute position" column similarly means the relative position of correspondent node in accordant DOM tree. In each table, other nodes than accordant nodes are represented using *node[n]* in which *n* is a position in the same siblings in the updated DOM tree. A leftmost node is expressed as *node[1]*. Attribute nodes are expressed as *@ + attribute name*.



Accordant node table

NodeType	NodeName	Value	Reference position	Original position	Updated position
Element	A	-	/A	/A	/A
Element	B	-	/A/B	/A/B	/A/B
Element	C	-	/A/C	/A/C	/A/node[2]/C
Attribute	E	"X"	/A/B/@E	/A/B/@E	/A/B/@E

Appearing node table

NodeType	NodeName	Value	Absolute position	Appearance position
Element	1	-	/A/node[2]	/A/node[2]
Element	2	-	/A/B/node[1]	/A/B/node[1]
Element	F	-	/A/node[1]	/A/node[2]/node[1]

Disappearing node table

NodeType	NodeName	Value	Absolute Position	Disappearance position
Element	D	-	/A/node[3]	/A/node[3]
Element	F	-	/A/B/node[1]	/A/B/node[1]

Figure 4: Difference Detection Process.

The purpose of this process is to detect the maximum number of accordant nodes and the remaining differential nodes.

2.2 Difference Extraction Process

In this step, detailed differential data is extracted from the tables created in the previous step. The proposed method classifies appearing and disappearing nodes into three kinds of nodes.

i) Change node

Comparing an appearing node table and a disappearing node table, two nodes that have the same absolute position and the same type are regarded as change nodes. In Fig. 4, node "2" in the appearing node table and node "F" in the disappearing node table are change nodes.

ii) Added node

Other nodes than change nodes in an appearing node table are regarded as added nodes. In Fig. 4, node "1" and node "F" in the appearing node table are added nodes.

iii) Deleted node

Other nodes than change nodes in a disappearing node table are regarded as deleted nodes. In Fig. 4, node "D" in the disappearing node table is a deleted node.

Reference Position	Node Type	Classification of difference	Additional Data		
			Node Type	Node Name	Node Value
/A/B/*[1]	Element	Change	**Element**	**2**	-
/A/*[2]	Element	Addition	**Element**	**1**	-
/A/*[2]/*[1]	Element	Addition	**Element**	**F**	-
/A/*[3]	Element	Deletion	-	-	-

Figure 5: Example of Differential nodes.

Fig. 5 shows a table of differential nodes created from the XML documents in Fig. 3.

2.3 Difference Representation Process

In this step, differential data generated in the previous step are mapped to XSLT templates from which a differential XSLT stylesheet is generated. The proposed difference representation method consists of the following sequences.

(a) Determining a transformation position as a pattern of XSLT template rule

A XML data structure transformation is described specifying the following two items.

- Which position of XML data is transformed? (Determining a transformation position as a pattern of XSLT template rule)
- How is the corresponding part of XML data transformed? (Transformational description)

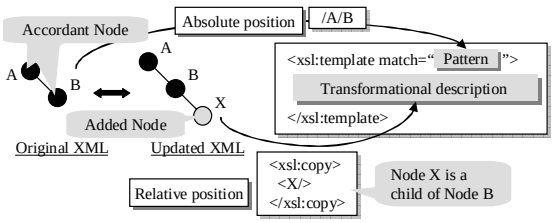


Figure 6: XSLT Template rule.

These items constitute a template rule and are represented as *xsl:temlate* element in the XSLT stylesheet. (Fig. 6) In a template rule, a pattern is an XPath[Clark, 1999] expression of the position of a node in an original DOM tree. We must specify transformation position as a pattern in a template rule. The position of an added node in an updated DOM tree is separated into a reference position and relative position and the reference position is specified as a pattern in a template rule.

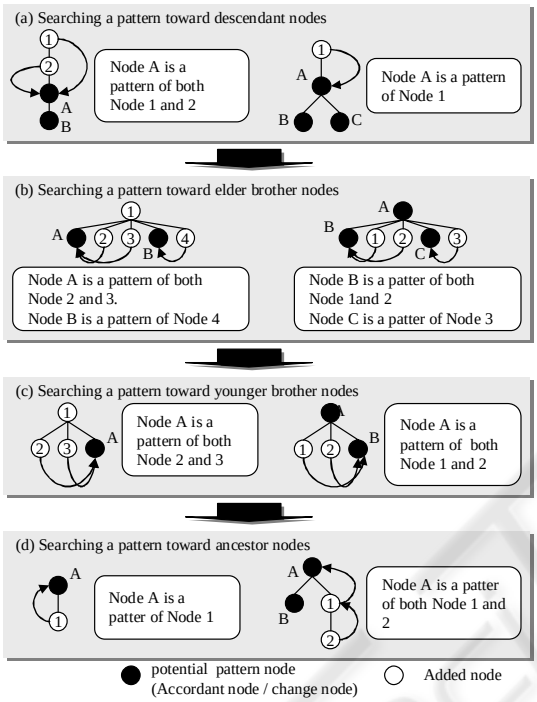


Figure 7: Searching a transformation position as a pattern of XSLT template rule.

In the proposed method, a pattern of a template rule is determined by tracing an updated DOM tree from top to bottom (from a parent to a child), and from left to right (from an elder brother to a younger brother) As a result, templates are generated in a tree structure that branch toward the descendant direction. Additionally, the positions of change and accordance nodes can be used as a pattern of a template rule because their own position can be specified in an original DOM tree. The method used to determine a pattern of a template rule is described with reference to Fig. 8. For change nodes and accordance nodes, their own positions are patterns. In case of added nodes, patterns are determined according to following sequences.

i) Searching a pattern toward descendant nodes (Fig.7 (a))

A change node or an accordance node is searched for starting from the corresponding node and moving toward descendant nodes. The position of the nearest descendant appropriate node is regarded as a pattern. If there is no appropriate node, there are multiple appropriate nodes, or there is a branch in the tree, the next procedure is applied.

ii) Searching a pattern toward elder brother nodes (Fig. 7(b)

A change node or an accordance node is searched starting from the corresponding node and moving toward elder brother nodes. The position of the nearest elder brother appropriate node is regarded as a pattern. If there is no appropriate node, the next procedure is applied.

iii) Searching a pattern toward younger brother nodes (Fig. 7(c))

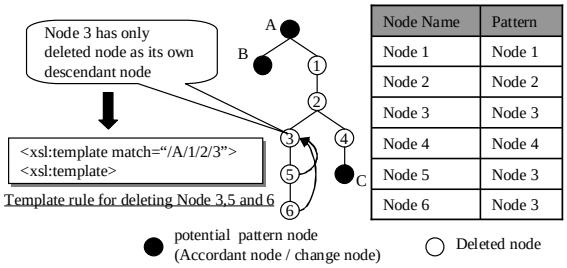
A change node or an accordance node is searched starting from the corresponding node toward younger brother nodes. The position of the nearest younger brother appropriate node is regarded as a pattern. If there is no appropriate node, the next procedure is applied.

iv) Searching a pattern toward ancestor nodes (Fig. 7(d))

A change node or an accordance node is searched from the corresponding node toward ancestor nodes. The position of the nearest ancestor appropriate node is regarded as a pattern. If there is no appropriate node, the pattern of a parent node is taken as that of the corresponding node.

The reason why the search pattern direction begins with descendant nodes and leaves ancestor nodes to the last is that if a pattern search commences with ancestor nodes, a pattern is underspecified since XML has a tree structure.

The search process for deleted nodes starts with ancestor nodes. As shown in Fig. 8, if all descendant nodes are deleted nodes, the position of the top node is regarded as the pattern of these nodes and all deleted nodes can be deleted using the same template. Therefore, the position of the farthest ancestor node which has only deleted nodes is regarded as the pattern. If there is no appropriate node, the position of the corresponding node is regarded as the pattern.



Node Name	Pattern
Node 1	Node 1
Node 2	Node 2
Node 3	Node 3
Node 4	Node 4
Node 5	Node 3
Node 6	Node 3

Figure 8: Searching pattern of deletion node.

The search for attribute nodes also starts toward ancestor nodes, because attribute nodes basically have only parent nodes.

(b) Generating mapping information

Differential data is organized according to the patterns identified in the previous step. The differential nodes are classified according to pattern because differential nodes that have the same pattern can be described using the same XSLT template. Fig. 9 shows the differential data for the XML documents in Fig. 3.

Pattern of template rule

Absolute position	Node Type	Difference	Transformation
/A	Element	Accordance	No
/A/B	Element	Accordance	Yes
/A/C	Element	Accordance	Yes
/A/D	Element	Deletion	Yes
/A/E	Attribute	Accordance	No
/A/B/F	Element	Change	Yes

Pattern node table

Node Name	Difference	Pattern
/A/1	Addition	/A/B
/A/B/2	Change	/A/B/F
/A/1/F	Addition	/A/C
/A/D	Deletion	/A/D

Transformational description of template rule

Absolute position	Relative position	Node Type	Difference	Addition Data		
				Node Type	Node Name	Node Value
/A/B	Parent	Element	Addition	Element	1	-
/A/C	Elder brother	Element	Addition	Element	F	-
/A/B/F	itself	Element	Change	Element	2	-
/A/D	itself	Element	Deletion	-	-	-

Figure 9: Differential Information according to Patterns.

(c) Mapping to XSLT template

Differential data is mapped to an XSLT template according to node type and classification of difference.

i) Mapping to XSLT template for accordant, changed and deleted nodes

Nodes that have the same pattern are collectively described by the same XSLT template. As shown in Appendix A at the end of this paper, each node is mapped to one XSLT template.

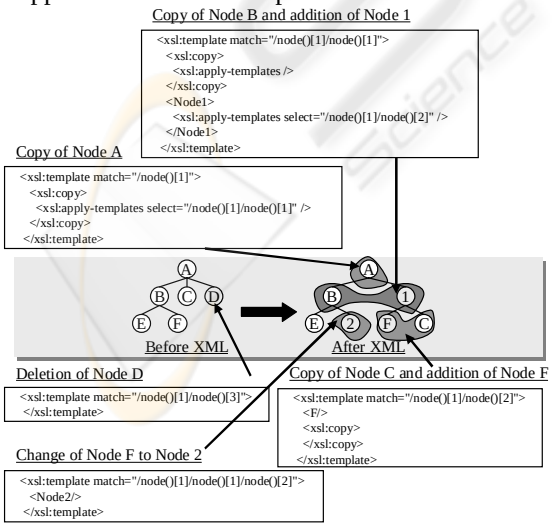


Figure 10: Example of XSLT Templates.

ii) Mapping to XSLT template for added nodes

Nodes that have the same pattern are collectively described by the same XSLT template. The added node is described as Appendix B. If a node regarded as a pattern has a descendant template, *xsl:apply-template* element is added in correspond template. If a node which share the same pattern are described by the same template. If there are multiple descendant templates of the correspond added node, multiple *xsl:apply-template* elements are used as shown in Appendix C.

Fig. 10 shows XSLT templates for the XML documents in Fig. 3.

(d) Generating a differential XSLT stylesheet

A Differential XSLT stylesheet is generated from XSLT templates as shown in Fig. 11. The header part includes a XML declaration and various XSLT parameters. The footer part includes a template to copy those accordance nodes that are not regarded as transformation position (pattern node).

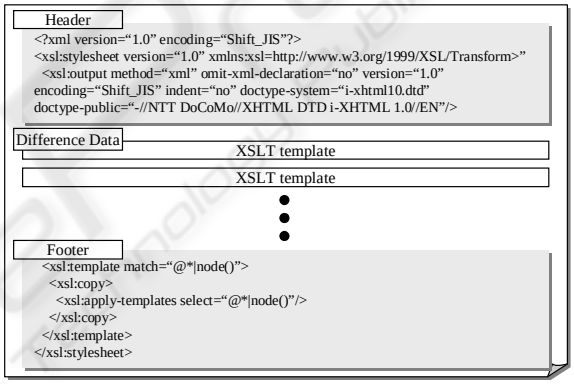


Figure 11: Differential XSLT Stylesheet.

3 EXPERIMENTS

We have implemented prototype software in Java (J2SE 1.3.1).We used Apache Xalan-Java 2.4.0 (XSLT Engine) to verify the generated XSLT stylesheets and IBM XML Parser for Java to parse XML documents. Our prototype software has functions to generate differential XSLT stylesheets from arbitrary pairs of XML documents and verify them. Additionally, it provides a graphical viewer of the differential data of two XML documents and shows some values such as compression ratio, difference ratio and so on. We have examined the validity of our proposed method using this software.

3.1 Stock Price Information

We have investigated the relationship between compression ratio and difference ratio using the following stock price information. In Fig. 12, stock price is updated every minute; old price data are

deleted and new price data are added. We changed XML data size and differential ratio and determined the resulting compression ratio.

The compression ratio is defined as the ratio of generated XSLT stylesheet to the before XML data. The difference ratio is defined rate of existence of unique nodes. (See formula (1) and (2).)

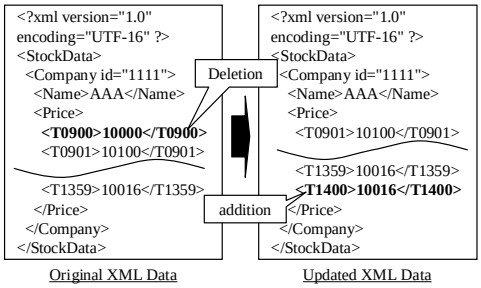


Figure 12: XML data of intra-day chart.

$$\text{Compression ratio} = \frac{\text{Size of Generated XSLT Stylesheet}}{\text{Size of updated XML data}} \times 100\,(\%) \quad (1)$$

$$\text{Difference ratio} = \frac{\text{Number of discordance nodes}}{\text{Number of all nodes in updated XML document}} \times 100\,(\%) \quad (2)$$

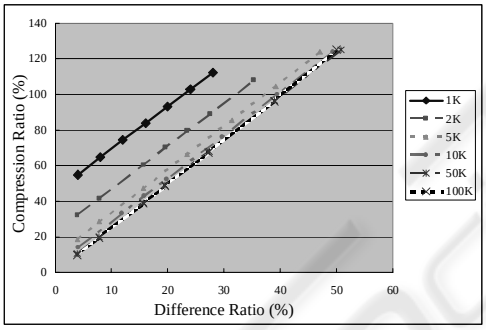


Figure 13: Relationship between compression ratio and difference ratio using stock price information.

Fig. 13 shows the experimental result using stock price information. In our experiments, the size of a XML document was changed from 1Kbyte to 100Kbyte, the difference ratio was changed from about 4% to 50%. As shown in Fig. 13, compression ratio becomes high when the size of a XML document increases. This means that the overhead of the header and footer of the XSLT stylesheet can be decreased for a large XML document. Even a small XML document (i.e. 1Kbyte document) can be effectively compressed if the difference ratio is below 25%. Therefore, the proposed method is suitable even for a small XML document if it has only minor updates.

3.2 News Flash Content

Then we have investigated the relationship between actual compression ratio and difference ratio using the following news flash content. In Fig. 14, news flash is updated; an old item is deleted and a new item is added. We changed difference ratio and compared the compression ratio of a differential XSLT stylesheet (D-XSLT), a differential XSLT stylesheet compressed by gzip (D-XSLT+gzip), a Diff [GNU, 2002] file compressed by gzip (Diff+gzip) and an entirely updated content compressed using gzip (HTML+gzip).

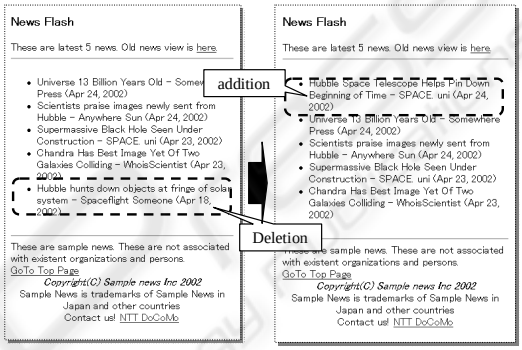


Figure 14: News Flash Content.

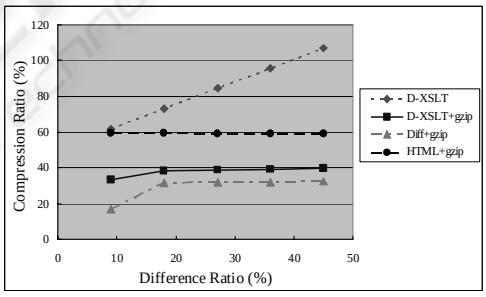


Figure 15: Relationship between compression ratio and difference ratio using news flash content.

Fig 15 shows the experimental result. The difference ratio is changed from about 10% to 45%. As shown in Fig. 15, a differential XSLT stylesheet complessed by gzip (D-XSLT+gzip) reduces about 63% of the size of entirely updated content compressed by gzip. Compared to Diff file compressed by gzip (Diff+gzip), the size of D-XSLT+gzip is about 38% larger. This result shows, although the proposed method is not optimised compared with Diff+gzip method, it realizes comparatively efficient compression.

4 EXTENSION TO XSLT FUNCTIONS

XSLT is not optimized for generating differential XSLT stylesheets. We have considered new XSLT function for the proposed method as described below.

i) Package copy of element and attribute nodes

In Fig. 16, node "e1" is entirely copied under the node "e3". However, XSLT can not copy any particular node. Thus the XSLT template repeats the same data of node "e1". The XSLT template must have the same data contained in the original XML document. This decreases the efficiency of compression.

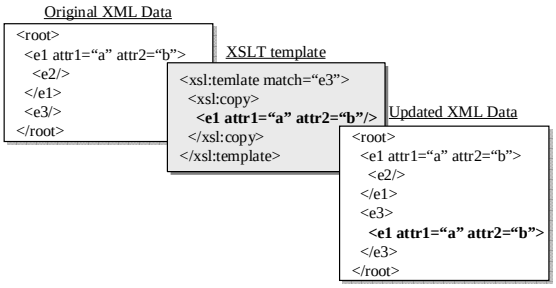


Figure 16: Example 1 of XSLT Stylesheet.

ii) Range copy

In Fig. 17, a sub-tree consists of the node "e1" and the node "e2", is copied under the node "e4". In XSLT, a sub-tree can be regarded as a node-set using *xsl:copy-of* element. However a node-set must include all nodes under the root of a sub-tree. Thus a node-set cannot be used to designate some parts of a sub-tree. Some parts of a sub-tree can not be copied using *xsl:copy-of* element.

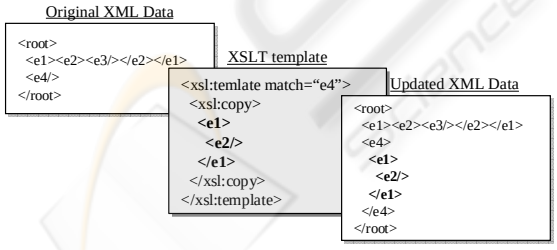


Figure 17: Example 2 of XSLT Stylesheet.

We propose a new XSLT function for *xsl:copy-of* element that designates nodes from the root of a sub-tree to the n-th generation descendants. If the proposed function is applied to above two examples, the resulting XSLT templates are generated as shown in Fig 18.

Introducing such a function can optimize the generation of a differential XSLT stylesheet.

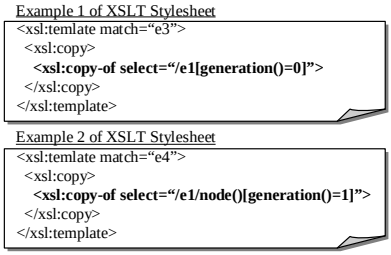


Figure 18: Example of XSLT template using proposed function.

5 RELATED WORK

Compared to existing methods of differential data generation [La Fontaine, 2001][Curbera, 1999], the proposed method has the advantage of easily adapting to expanded data as XML tree, and using existing XSLT engines. But there is a problem that mobile terminals have to deal with heavy process to expand XML navigation tree into memory. To lighten this process, XML processors for mobile terminals such as TinyTree[OSTG, 2004b] and DTM (Document Table Model)[Apache Project, 2004] have been developed. By expanding XML document as tree or array structure in the memory using these XML processors, it is more efficient to deal with difference data. In addition, the proposed method needs only XSLT engine, different from DUL (Delta Update Language) [OSTG 2004a] in which special language processing module is needed.

La Fontaine (2001) and Curbera (1999) described methods to generate difference data from arbitrary pairs of XML documents. These methods compare nodes of the original and the revised XML DOM trees from root to leaf. If a unique node is detected, all descendant nodes from that node are regarded as difference. They can not detect the most appropriate difference and no mention was made of this issue. On the other hand, the proposed method resolves this issue and so can extract the differences most effectively. It similarly compares the nodes of the original and revised XML DOM trees from root to leaf. When a unique node is detected, the corresponding node is regarded as difference if its descendant nodes are common. Therefore it is able to accurately detect the difference between original and revised XML documents.

6 CONCLUSION

We proposed an automatic method of generating differential XSLT stylesheets from arbitrary pairs of

XML documents. The proposed method consists of difference detection, difference extraction and difference representation; we proposed algorithms for each process. We have implemented prototype software and showed it to be effective using special contents such as stock price and news flash. We also proposed new XSLT function.

In future work, we will investigate processing time to generate differential XSLT stylesheet and continue to work on improving the performance of our algorithm. Future work includes quantitatively evaluating the proposed method in the case of general XML content. We will also confirm the effectiveness of the proposed XSLT function.

TYPE	Mapping to XSLT (No continuing process to descendant of pattern)	Mapping to XSLT (Continuing process to descendant of pattern)
Accordance	No template	<xsl:template match="XPath of its own position"> <xsl:copy> <xsl:apply-templates/> </xsl:copy> </xsl:template>
Delete	<xsl:template match="XPath of its own position"> </xsl:template>	<xsl:template match="XPath of its own position"> <xsl:apply-templates/> </xsl:template>
Change	<xsl:template match="XPath of its own position"> <Data of Changed node/> </xsl:template>	<xsl:template match="XPath of its own position"> <Data of Changed node> <xsl:apply-templates/> </Data of Changed node> </xsl:template>

Appendix A: XSLT template mapping for accordance, delete and change node.

TYPE	Mapping to XSLT (No continuing process to descendant of pattern)	Mapping to XSLT (Continuing process to descendant of pattern)
Addition node in Parent	<xsl:template match="XPath of a pattern node"> <Parent addition node> <xsl:copy/> </Parent addition node> </xsl:template>	<xsl:template match="XPath of its own position"> <Parent addition node> <xsl:copy> <xsl:apply-template> </xsl:copy> </Parent addition node> </xsl:template>
Addition node in Child	<xsl:template match="XPath of a pattern node"> <xsl:copy> <Child addition node/> </xsl:copy> </xsl:template>	Same as left template
Addition node in elder brother	<xsl:template match="XPath of a pattern node"> <Elder brother addition node/> <xsl:copy/> </xsl:template>	<xsl:template match="XPath of a pattern node"> <Elder brother addition node/> <xsl:copy> <xsl:apply-templates/> </xsl:copy> </xsl:template>
Addition node in younger brother	<xsl:template match="XPath of a pattern node"> <xsl:copy/> <Younger brother addition node/> </xsl:template>	<xsl:template match="XPath of a pattern node"> <xsl:copy> <xsl:apply-templates/> </xsl:copy> <Younger brother addition node/> </xsl:template>

Appendix B: XSLT template mapping for addition node.

TYPE	Mapping to XSLT (No continuing process to descendant of addition node)	Mapping to XSLT (Continuing process to descendant of addition node)
Addition node	<Addition node/>	<Addition node> <xsl:apply-template select="XPath of lower pattern"/> </Addition node>

Appendix C: XSLT template mapping for addition node with descendant templates.

REFERENCES

Bray, T. el al (2000) *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation.

Baker, M. et al (2000) *XHTML Basic.* W3C Recommendation.

Mogul, Jeffrey C. et al (1997) *Potential benefits of delta-encoding and data compression for HTTP. Proceeding of SIGCOMM 97.* SIGCOMM 97.

Mogul, Jeffrey C. et al (2002) *Delta Encoding in HTTP. RFC3229.* The Internet Engineering Task Force.

Clark, J. (2000) *XSL Transformations (XSLT) Version 1.0.* W3C Recommendation.

Le Hors, A. et al (2000) *Document Object Model (DOM) Level 2 Core Specification Version 1.0.* W3C Recommendation.

Clark, J. (1999) *XML Path Language (XPath) Version 1.0.* W3C Recommendation.

The GNU Project (2002) *Diffutils* [Software]. Version 2.8.1. www.gnu.org. Available from: <http://directory.fsf.org/GNU/diffutils.html> [Accessed 25 January 2005].

Open Source Technology Group (2004a) *diffxml* [software]. Version 0.92A. SourceForge.net. Available from: <http://diffxml.sourceforge.net/> [Accessed 25 January 2005].

Open Source Technology Group (2004b) *SAXON* [software]. version 8.2. SourceForge.net. Available from: <http://saxon.sourceforge.net/> [Accessed 25 January 2005].

The Apache Software Foundation (2004) *xalan-J* [software] version 2.6.0. www.apache.org. Available from: <http://www.apache.org/dyn/closer.cgi/xml/xalan-j> [Accessed 25 January 2005].

La Fontaine, R. (2001) *A Delta Format for XML: Identifying Changes in XML Files and Representing the Change in XML.* XML Europe 2001.

Curbera, F. P. et al (1999) *Fast Difference and Update of XML Documents.* XTech'99 held in San Jose.

Ishikawa, N. et al (2002) *Automatic Generation of a Differential XSL Stylesheet From Two XML Documents. Proceeding of WWW Conference 2002 held in Hawaii.* WWW Conference 2002.