社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ＦＩ－78 (9)
2005－ＤＤ－49 (9)
2005／3／25

基幹構造圧縮: コンテントアダプテーションに適した構文解析済 XML の圧縮

行友英記* 金野晃* 中山雄大* 竹下敦*

本研究では，構文解析済 XML の圧縮方法として基幹構造圧縮を提案する．基幹構造圧縮は，複数の XML 文書に共通する構造パターンをテンプレートして予め用意し，これらのテンプレートを用いてメモリ上の構文解析済 XML を圧縮する．部分木の再利用が可能な状態のまま構文解析済 XML を圧縮することから，メモリ上により多くの構文解析済 XML を格納でき，そのため，コンテントアダプテーションのように元となる XML 文書を多様なユーザの嗜好性や端末の特性に適応させ，組み替えたり集約させたりして配信するようなアプリケーションの高速化に適している．

Structure-based Compression of Parsed XML for Content Adaptation

Hideki Yukitomo*, Akira Kinno*, Takehiro Nakayama*, & Atsushi Takeshita*

This presentation proposes a technique, called structure-based compression, for compressing parsed XML. Structure-based compression registers common structural patterns as templates in advance, and uses these templates to represent the complete structural information of parsed XML. By compressing parsed XML through its structure, the same memory space can hold more parsed XML instances, and even after compression, parsed XML can be traversed without decompression. These features support the acceleration of content adaptation, in which original contents are adapted according to the users' preferences and device capabilities prior to delivery.

1. Introduction

XML and its variants (XHTML, WML and so on) are playing a key role in the handling of Internet contents, so many of them are represented as XML documents, and are stored as such in database systems.

To achieve features such as the partial addition, deletion, and alternation of XML documents, they must be parsed. However, once an XML document is parsed, its memory requirement is drastically increased because parsed XML must contain the relations between its elements, in addition to the original text.

Since XML document database have to hold quite a lot of XML documents, popular XML documents are cached as parsed XML in memory and the remainder are stored in a second storage area. If parsed XML is compressed, more parsed XML instances can be held in memory, the number of disk I/O and expensive parsing processes are reduced, and thus the overall system performance is enhanced [1].

Especially for the case of XML content adaptation, the delivery of the adapted content involves the integration of material from several source XML documents according to the user's preference and device capability. Thus keeping more XML documents in parsed form can accelerate the time taken to locate the source XML documents that matches the user's preference and devices capability, and also can accelerate the management process of delivering adapted content from source XML documents.

This presentation proposes a novel compression technique for parsed XML called 'structure-based compression'. With structure-based compression, the structural information of parsed XML is replaced by structural templates given in advance. Even after compression, parsed XML can be traversed without decompression. We conducted simulations to examine the space efficiency of structure-based compression and prove that it works well with real world XML documents.

* (株)NTT ドコモ マルチメディア研究所
Multimedia Labs, NTT DoCoMo, Inc

2. Related work

2.1. Un-parsed XML Compression

Liefke et al. proposed the first compression technique for un-parsed XML named XMill[2]. XMill can use general text compression techniques, such as LZW [3], and is useful when simply storing XML documents. However, it does not improve the performance of XML document database systems because the XML documents must be decompressed and parsed before use.

XPRESS [1], XGRIND [4], and XQzip [5] were proposed to cover the shortcoming of XMill. They are also text based approaches, but aim at query-enabling XML compression; XML documents that match a given XPath query can be found even in compressed form.

This feature is desirable for some application. However, it is not enough for content adaptation because content adaptation requires the partial addition, deletion, and alternation of XML documents. To support these features, the whole XML document should be kept in parsed form, and the compression of parsed XML should be considered.

2.2. Parsed XML Compression

Parsed XML can be represented by the document object model (DOM) standardized by W3C [6]. DOM implements each XML node in parsed XML as a single java/C++ object. Thus, it is a convenient data structure for maintenance functions such as partial addition, deletion, and alternation. This implementation approach offers a lot of useful features. Unfortunately, it has huge memory requirements [7]. However, not all applications need such sophisticated features. Some use simpler and more compact data structures. For instance, some XSLT[1]-based transformers, such as Xalan [7] and SAXON [9], handle parsed XML with proprietary data structures, DTM and Tiny Tree, respectively. Such approaches realize compact parsed XML by aggregating multiple XML nodes into a single java/C++ object and eliminating unnecessary features.

Neumüller [10] introduced a compression technique for parsed XML. His approach uses a dictionary for storing node names, and each parsed XML node holds an index to that dictionary. Since data centric XML tends to re-use the same element/attribute names in the same document, data size can be shrunk.

[1] XSLT is a XML transformation language and standardized by W3C [8].

Each XML document must be parsed and represented by a data structure that allows prompt access to its data. The approaches of [7][9][10] can compress parsed XML while offering this feature, however, structural information is not fully utilized in compression, thus, there is still room for improvement.

3. Proposed compression technique

3.1. Preparation

A sample XML document is shown in Figure 1. As shown, XML is a kind of ordered rooted tree. Thus the parsed XML for this sample XML document can be represented as shown in Figure 2. Figure 2 shows that there are two types of nodes in the tree. Element nodes, the circles, have element names as values. The other, text nodes (the rectangles), have string data as node values. A sequential number is placed on the top right corner of each node in the depth first manner. The relationship between nodes is represented by arrows.

Logically, the parsed XML is represented as shown in Figure 2. A naive implementation might equate each node with a single java/C++ object, and the relationship between nodes is implemented by their references to each other. However, this kind of implementation consumes too much memory, and building this tree data structure is slow because the creation/deletion of objects in C++/Java is quite an expensive operation [7]. Thus, advanced implementations usually take a slight different approach for space and speed efficiency reasons. Let us now explain one such implementation, Apache Xalan's DTM; we explain structure-based compression with DTM in Section 3.2. In DTM, parsed XML is represented by a table, and each node is represented by a row that has several columns. Table 1 shows the DTM

```
<R> <A> <E>abcd</E>
        <E>efgh</E>
        <E>ijkl</E>
        <E>mnop</E>
    </A>
    <B />   <B />
    <C> <D>qrst</D>
        <D>uvwx</D>
        <D>yzab</D>
        <D>cdef</D>
    </C>
</R>
```

Figure 1: A sample XML document

representation of the sample XML document in Figure 2[2].

In Table 1, the first column represents the identical number of each node. The second column shows the node type, the value is either 'Element' or 'Text'. The third column shows node name. If node type is 'Element', this field represents element name. If node type is 'Text', the third column is meaningless. The fourth column shows the node's value (string data) and has meaning only for text nodes. The fifth to eighth columns represent the relationship between nodes. The columns show child, previous sibling, next sibling, and parent reference, in that order. Some cells are grayed-out in the last column. This indicates a reference to a parent node; such references are not shown in Figure 2 for better visibility. The terms used hereafter are defined below.

Structure is a set of relationships between nodes. A structure involves node type and node name. This means that element nodes and text nodes can be distinguished. Furthermore, the name of an element node is meaningful. No text nodes have names, and their values are meaningless with regard to the structure.

Pure Structure is a set of relationships between nodes. In contrast to structure, it is concern with neither node type nor node name.

Open-join information is a set of references between a node included in a structural template and an outside node. It specifies the connectivity of nodes in a structural template. That is, it describes which nodes of the pure structure in the structural template can be connected from/to outside nodes.

Compressed DTM represents a parsed XML after structure-based compression. It inherits Xalan's DTM interface, however, it consists of

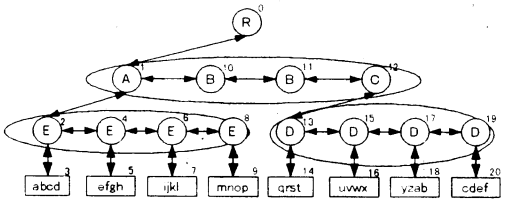

Figure 2: The Parsed XML of the sample XML document shown in Figure 1

[2] Real implementation is more complicated. For instance it includes 'COMMENT' and 'ATTRIBUTE' nodes. However, in this presentation, we explain about the essence of DTM, and details only parts which relates to structure-based compression. For details about DTM, please refer to [7].

Table 1: DTM representation of sample XML document

Index	Type	Name	Value	c	ps	ns	p
0	Element	R	-	1	-	-	-
1	Element	A	-	2	-	10	0
2	Element	E	-	3	-	4	1
3	Text	-	abcd	-	-	-	2
4	Element	E	-	5	2	6	1
5	Text	-	efgh	-	-	-	4
6	Element	E	-	7	4	8	1
7	Text	-	ijkl	-	-	-	6
8	Element	E	-	9	6	-	1
9	Text	-	mnop	-	-	-	8
10	Element	B	-	-	1	11	0
11	Element	B	-	-	11	12	0
12	Element	C	-	13	11	-	0
13	Element	D	-	14	-	15	12
14	Text	-	qrst	-	-	-	13
15	Element	D	-	16	13	17	12
16	Text	-	uvwx	-	-	-	15
17	Element	D	-	18	15	19	12
18	Text	-	yzab	-	-	-	17
19	Element	D	-	20	17	-	12
20	Text	-	cdef	-	-	-	19

Root DTM and several template instances.

Structural template is a data structure to represent a fragment of the pure structure. It consists of pure structure, template ID and open-join information. Template ID is used to differentiate the structural templates. A structural template is referenced from template instance, and is used to describe a part of the pure structure that matches the parsed XML.

Non-compressed DTM represents a parsed XML without compression. While basically the same as Xalan's DTM, its data structure is slightly different from that of the original Xalan's DTM. The difference is shown in Section 3.3.

Root DTM is non-compressed DTM included in compressed DTM. With structure-based compression, compressed DTM's pure structure is partly replaced by template instances. However, there will be several nodes that do not belong to any template instance. Root DTM contains such all stray nodes.

Template instance is a data structure to represents the part of parsed XML in compressed DTM. It consists of instance ID, a reference to the structural template, node information, and outer join information. Instance ID is used to identify each template instance in the same compressed-DTM. The reference to the structural template shows which structural template is used in the template instance.

Node information is a set of node types, node names, and node values in a template instance. Since the pure structure contains only structural relationships between nodes, node information represents type, name, and value of each node in the template instance.

Join information: Set of references between nodes in the template instance and outside nodes. To build compressed DTM with template instances, each template instance and node in Root DTM must be connected. Join information contains such relationships and provides the concrete value of open-join information in the structural template.

Structural information is either pure structure, or open-join information, or join information.

3.2. Structure-based compression

The basic idea of structure-based compression for parsed XML is the substitution of pure structure with structural templates. For instance, in Figure 2, each area surrounded by an ellipse holds four nodes, and according to the definition of structure (See Section 3.1), these structures surrounded by ellipses differ from each other (since the node's names within each structure are different). However, if you neglect node name, and concentrate only on 'pure structure', these three ellipses look the same. This means that if we can separate the pure structure from the parsed XML, divide it into fragments, and re-use them as structural templates, we can compress the parsed XML.

Figure 3 shows the concept of structure-based compression more precisely. In Figure 3, the XML document database holds many parsed XML instances, which is represented by compressed DTM, and one parsed XML, '#1' is described. Parsed XML #1 contains a root DTM and three template instances; those are represented by a DTM-like data structure. The part that contains meaningful data is represented by the solid line, and the unused part is indicated by the thin dotted line.

If there are at least two template instances which refer to the same structural template in three template instances, they can share structural information, and total data size can be reduced.

Note that not all structural information is redundant, because join information should be described. Furthermore, template instances have to contain some extra data, such as instance ID, and the reference to the structural template. The overheads incurred by using structural templates are, however, much smaller than the structural information, and so can be ignored. Thus, total size of compressed DTM is decreased as more

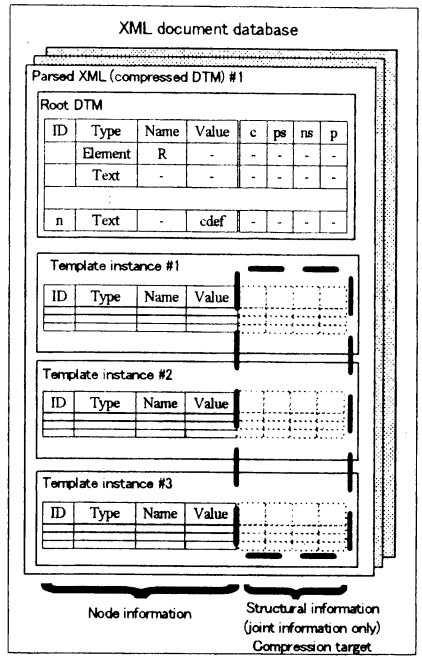

Figure 3: The concept of structure-based compression

structural information is replaced by more template instances (the area surrounded by the thick dashed line).

3.3. Memory consumption

To evaluate the efficiency of structure-based compression, we show the memory consumption of parsed XML treated by structure-based compression, and compare it with that of non-compressed parsed XML.

At first, we define the memory consumption of non-compressed parsed XML. Let N_x denote the number of nodes in XML document x. The memory consumption of non-compressed parsed XML is given by

$$U(x) = 4 \cdot 4N_x + 4 \cdot 2N_x + D(x)$$

The first term represents pure structural information. In the case of uncompressed parsed XML, memory is needed to store all references (parent, child, previous sibling, and next sibling) of all nodes. This requirement is given by $4N_x$. Since we assume that all of these are represented by integers consisting of a total of four bytes, $4N_x$ is multiplied by four.

The second term represents two pointers those represent node name and node value. Each pointer is represented by four bytes, and thus is represented by $4.2N_x$.

The last term $D(x)$ is the sum of all string data. Since each node has its own name and value, string data are stored in the dictionary, and each node has its own index to the dictionary for name and value, respectively.

Second, the memory consumption of parsed XML treated by structure-based compression is given by

$$C(x) = 4\left(\sum_{i=0}^{T} A(t_i, x)\left(J(t_i)+2\right)+4M_x\right) + 4 \cdot 2N(x) + D(x)$$

The second and third term have the same definition as in the case of $U(x)$. In the first term, M_x denotes the number of nodes in Root DTM. T is the total number of structural templates, and t_i is the *i-th* structural template. $A(t,x)$ is the number of times structural template t occurs in document x. $J(t)$ is the size of join information for structural template t. The first term also represents pure structural information. Compressed parsed XML needs only the join information plus some overhead for each template instance. The size of join information is represented by $J(t)$, and the overhead consists of two factors: template instance ID and the references to structural templates. Thus, the memory consumption of each template instance is represented by $J(t,x)+2$. The total memory consumption for pure structure of compressed parsed XML is the sum of that for all template instances, and is thus given by

$$\sum_{i=0}^{T} A(t_i, x)\left(J(t_i)+2\right)$$

Since we have to consider the pure structure included in Root DTM, and have to represent memory consumption in bytes, $4M_x$ is added to the above equation (multiplication of four represents the four references) and multiplied by four in $C(x)$ as memory consumption of the pure structure.

Note that the notation $I(x)$ is used as the memory consumption for ideal compression by structure-based compression. If all nodes are replaced by some structural template, the ability of structure-based compression is maximized. $I(x)$ denotes the memory consumption of this case, and is approximated by supposing that there is a virtual structural template which can replace all nodes in Root DTM.

4. Simulation

We conducted a simulation to examine the space efficiency of structure-based compression. In this section, we compare the memory consumption of compressed parsed XML produced by structure-based compression with that of non-compressed parsed XML. They are represented by compressed DTM, and non-compressed DTM, respectively.

4.1. Conditions

As the input XML documents, we collected HTML documents from three web sites, A, B and C, by following hyperlinks to the third level. Strictly speaking, HTML is not a variety of XML. By complementing the illegal notation in HTML documents, we translated them to valid XML documents.

Site A contains a lot of rumors, tips, and information about miscellaneous topics. Site B is an IT news site, containing a lot of technology related news articles. Site C is a software vendor's site, and most documents are product information. Some elements or tags of the collected documents were complemented in advance to yield validated XML. Table 2 shows miscellaneous information about the documents from the three web sites. Note that we treated a single HTML file as a single document in this simulation.

The compression ratio strongly depends on the suitability of the set of structural templates for the documents. Since we have not yet developed a strategy for choosing the 'optimum' structural

Table 2: Sample data information

Site	Number of contents	Ave. File size [KB]	Ave. Number of nodes
A	933	27.4	1143.7
B	629	17.0	698.2
C	333	22.1	1117.5

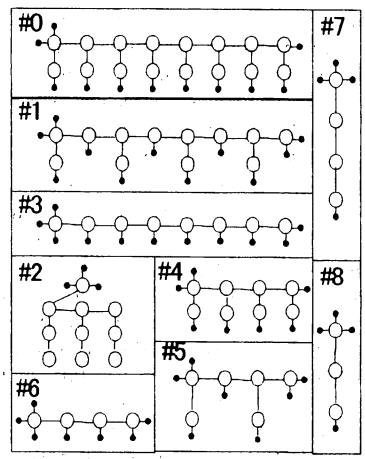

Figure 4: Structural templates used in simulation

template, we chose the structural templates after a rough manual examination of the documents of these three sites. We chose nine structural templates manually.. See Figure 4 for details. The number of each structural template is set on the top left corner of the template. Filled circles and blank circle represent open join information, and nodes, respectively. The relationship between nodes is represented by lines.

We first choose simple horizontal (#0, #3, #4, #6) and vertical (#7, #8) templates. As for the other templates, they were inspired by the sequence of anchor (#1 and #5) and table (#2) structure in site A.

The structural templates are sorted by the number of nodes in them. We used a simple algorithm to find every place where the structural template matched the XML documents. For each XML document, every structural template was tested as to whether it matched the sub-tree rooted by each node in the depth first manner.

4.2. Simulation results

Figure 5 shows the memory consumption of non-compressed and compressed parsed XML for each site. The Y axis plots memory consumption for pure structure (thus dictionary size *D(x)* is subtracted) in Kilo bytes and the X axis plots document number. To improve readability, documents are sorted by size of non-compressed parsed XML. Figures 5(a), 5(b), and 5(c) present the simulation results for sites A, B, and C, respectively. The top curve (denoted by *U(x)-D(x)*) shows the size of non-compressed parsed XML. The second curve (denoted by *C(x)-D(x)*) represents compressed parsed XML with structure base compression. The bottom curve (denoted by *I(x)-D(x)*) represents the case where whole nodes are replaced by some structural template. The difference between the top and second curve indicates the size of memory saving achieved by structure-based compression. Figures 5(a), 5(b), and 5(c) show that structure-based compression works well for each site. Table 3 shows the average reduction rate of total and structural memory consumption for each site. Even considering the total memory consumption, at least a 5.5% reduction is possible with structure-based compression. The compression efficiency of structure-based compression strongly depends on the documents and the set of structure templates. As shown in Table 3, each site has a different average reduction size even though the same structure templates were used. In other words, the contribution of each structure template to memory reduction size depends on the documents. For

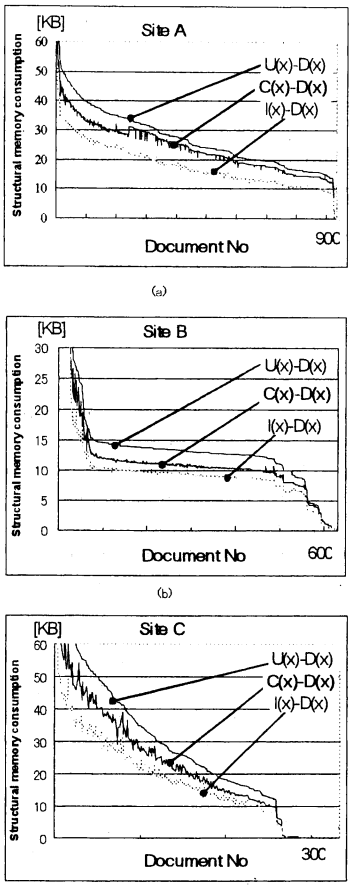

Figure 5: Memory consumption

Table 3: Average reduction rate of total and structural memory consumption

Site	total memory consumption (U(x)-C(x))/U(x)	structural memory consumption (U(x)-C(x))/U(x)-D(x)
A	5.5%	11.0%
B	7.4%	16.0%
C	7.8%	13.5%

example, further investigation shows that structural template #4 matches some document in site A more frequently than that in the other sites. In this case, we can say that structural template #4 is 'effective' for compressing site A's documents because it works quite well on several documents in site A. To

compress documents more efficiently, we have to find and add other structural templates like #4.

In these simulations, we roughly chose the structural template, so total memory consumption is reduced by at most 7.8% with structure-based compression. However, the difference between *U(x)-D(x)* and *I(x)-D(x)* in Figure 4 implies that there is still room for further improvement if we choose adequate structural templates. The upper-bound of the reduction in total memory consumption is estimated to be about 18 % for these examples. In other words, the number of parsed XML instances in the memory can be increased by 18%. Structure-based compression can be made to work more effectively by finding a lot of effective structure templates for each site.

5. Conclusion

In this presentation, we proposed structure-based compression, a novel compression technique for parsed XML. With structure-based compression, parsed XML can be compressed while still supporting the ability of prompt traverse. Since structure-based compression focuses only on pure structures, it is more powerful than conventional approaches like [10]. Simulations results proved the efficiency of structure-based compression, and showed that it can achieve 5.5% to 7.8% compression with real XML documents. Our future research goals are as follows:

1. Evaluate the efficiency of structure-based compression by implementing system

This presentation assessed the space efficiency of compression by just simulation. We will conduct trials to investigate the degree of memory reduction when actually real implemented.

2. Develop an algorithm that can find effective structural templates

Mining frequent sub-trees in the forest is, in general, time consuming process [11, 12]. To make our compression technique practical, we have to find the structural characteristics of XML documents for each application domain, and find a fast way to identify a set of best (or quasi-best) structural templates for them.

References

[1]: J.K. Min, M.J. Park, and C.W. Chung, „XPRESS: A Queriable Compression for XML Data", Proc. ACM Symp. On the Management of Data(SIGMOD), 2003.

[2] H. Liefke, D. Sucia, "XMill: an efficient Compressor for XML Data", Proc. ACM Symp. On the Management of Data(SIGMOD), Dallas, Texas, 2000.

[3] J. Ziv, A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory 23(3): 337-343, 1977.

[4] P.M. Tolani, and J.R. Haritsa, "XGRIND: A Query-friendly XML Compressor", Proc. Int. Conf. Of Data Engineering(ICDE), 2002.

[5] J.Cheng, and W. Ng, "XQzip: Querying Compress XML Using Structural Indexing", In Proc. of Extending DataBase Technology, 2004.

[6] Document Object Model (DOM), http://www.w3.org/DOM/

[7] Apache XML Project: XSLT Processor Xalan, http://xml.apache.org/xalan-j/index.html

[8] XSL Transformations (XSLT), http://www.w3.org/TR/xslt

[9] The XSLT and XQuery Processor, SAXON, http://saxon.sourceforge.net/

[10] M. Neumüller, and J. N. Wilson, "Compact In-Memory Representation of XML", Technical report, Dept. of Computer and Information Science, University of Strathclyde, Glasgow, Scotland, UK, 2002.

[11] R. Agrawal, and R.Srikant. "Fast algorithms for mining association rules", In Proc. of Int. Conf. Very Large Data Bases (VLDB'94), pp. 487-499, Santiago, Chile, Sept. 1994.

[12] M. J. Zaki, "Efficiently Mining Frequent Trees in a Forest", In Proc. of SIGKDD 2002, ACM, 2002.

Appendix.

Figure 6 shows the sample compressed DTM for the data in Figure 1. It consists of (a) one root DTM and (b) three template instances, and (c) one structural template. The three template instances have references to structural template #1. Reference to a node is done using the X:Y notation; X is the template instance number and Y is the node's index in the template. Node name and node value are included in the table for better visibility though they are represented by dictionary's index.

Open-join information of structural template #1 is included in structural information, and each reference is indexed using a minus integer to distinguish it from inner structural information.

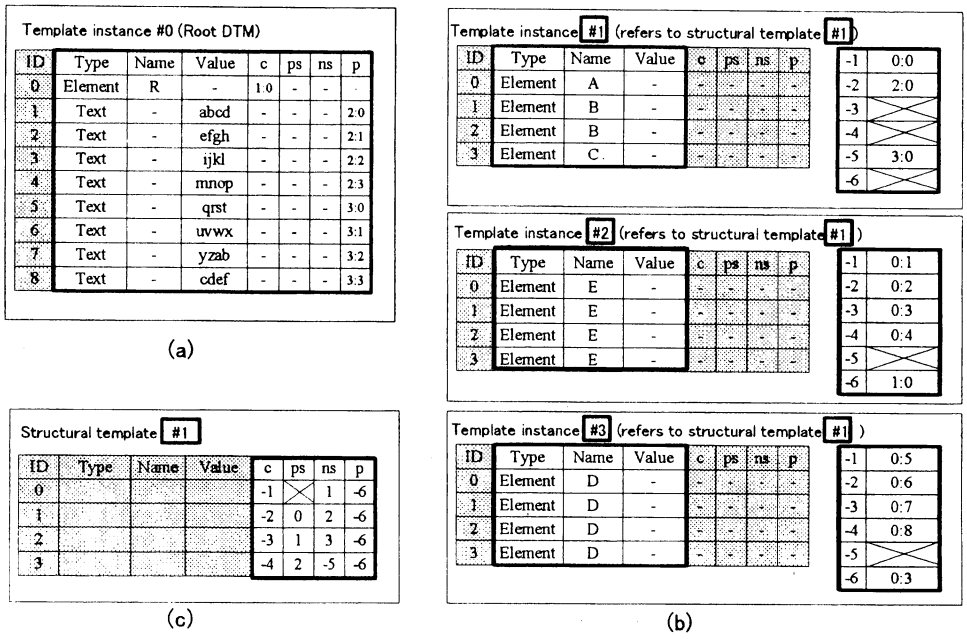

Figure 6: Compressed DTM representation of sample XML document