2E-2

XML 文書を単一の 2 列のテーブルに変換する手法およびその効果

綿部　良介†

三菱電機(株)　先端技術総合研究所†

1. 概要

XML(Extensible Markup Language)文書は情報系システムに広く流通しており，今後は組込み機器を情報系システムに融合させるシステムが求められる．そのため，XML 文書を効率的に操作できる手法が求められている．XML 文書をテーブルに変換する手法は従来から提案されてきた[1][2]が，殆どの手法でスキーマを必須としており，しかもテーブルが複数にまたがってしまうという課題があり，組込み機器への適用が困難である．XML 文書の特徴の 1 つに半構造のデータを自由に記述できることが挙げられ，例えば組込み機器など，スキーマを必要としない XML 文書を扱う場面は多い．

そこで筆者は XML 文書の制約を利用して，スキーマを必要としない検証済み XML 文書を，元の XML 文書に復元可能な単一の 2 列のテーブルに変換する手法を提案する．また，本手法を用いたときの XML 文書の圧縮効果，S/W による XML 文書操作時のメモリ削減効果についてそれぞれ示す．

2. 手法

本手法では以下の 3 つの前提条件を設定する．

(1)登場するノードとして，基本的な 5 種類である「空でない要素」「空要素」「属性」「要素のテキスト」「属性のテキスト」を考える．「XML 宣言」「処理命令」「CDATA(CharacterDATA)セクション」「コメント」「文字参照」「エンティティ参照」のノードとして扱わない，これらは重要なノードでないので問題はないと見られる．

(2) タブ，改行などいわゆる ignorableWhiteSpace をノードとして扱わない．これらは本来無視される要素なので問題はない．

(3) XML 文書操作として一般的な API(Application Program Interface)である DOM(Document Object Model)では「属性」を「要素」の子ノードとして扱わない．本手法では「属性」を「要素」の子ノードとして扱うが，この扱い方により「属性」の重複カウントや 0 回カウントのような，XML 文書操作上の問題は発生しない．

```
<aaa>
  <bbb />
  <ccc ddd="eee">fff</ccc>
</aaa>
```

図 1: テーブルへ変換する XML 文書

How to Translate an XML Document into single Table which Has Only 2 Lines, and the Effect of This Translattion

† Ryosuke Watabe

† Mitsubishi Electric Corp. Advanced Technology R&D Center

本章では図 1 の XML 文書を例題に，XML 文書からテーブルへの変換，およびテーブルから XML 文書への復元をそれぞれ説明する．

2.1. 変換手法

まず図 2 の表 A のように，各行の 1 列目にノードの名称を，各行の 2 列目に XML 文書で最上位のノードであるルートノードの深さを 1，その子要素の深さを 2，さらにその子要素の深さを 3，としたときのノードの深さを，各行の 3 列目にノードの種類を，ノードが登場する順にそれぞれ代入する．

次に表の情報量を減らすため，図 2 の表 B のように，表 A のノードの種類をビット On/Off で置換する．具体的には「属性」「要素のテキスト」をビット On に置換し，「空でない要素」「空要素」「属性のテキスト」をビット Off に置換する．ここでノードの種類に関する情報が欠落してしまうが，後に示すようにこの情報は復元可能である．

最後にさらに表の情報量を減らすため，図 2 の表 C のように，表 B の 2 列目を 2 倍して，その値に表 B の 3 列目のビットが On だったら 1 を足した値を，表 C の 2 列目の値とする．この表 C が最終的な変換である．

表 A

aaa	1	空でない要素
bbb	2	空要素
ccc	2	空でない要素
ddd	3	属性
eee	4	属性のテキスト
fff	3	要素のテキスト

表 B

aaa	1	
bbb	2	
ccc	2	
ddd	3	*
eee	4	
fff	3	*

表 C

aaa	2
bbb	4
ccc	4
ddd	7
eee	8
fff	7

図 2: XML 文書からテーブルへの変換

2.2. 復元手法

まず図 2 の表 C の 2 列目の値を 2 で割った商と余りにより，表 C から表 B へ復元できる．

ここで，XML 文書には以下の制約がある．

・「要素のテキスト」「属性のテキスト」は子ノードを持たない．

・「属性」は必ず子ノードを持ち，それは「属性のテキスト」である．

この制約に基づき図 3 のようなフローチャートを作成すると，ビット On の「属性」「要素のテキスト」，ビット Off の「空でない要素」「空要素」「属性のテキスト」の 5 種類を一意に分類できる．

従って，図 2 の表 B から表 A へも復元できる．図 2 の表 A から図 1 へ復元できることは自明である．

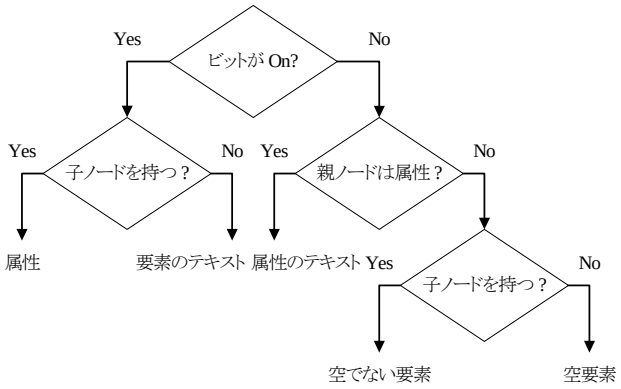

図 3: ノードの種類を復元するフローチャート

3.1. 圧縮効果

XML 文書を図 2 の表 C のような形式に変換することにより，XML 文書の圧縮が期待できる．

ここで，例えば変換したテーブルを RDB に格納しようとすると，DB のスキーマ作成が必要になる．

図 2 の表 C の 1 列目では一番長いノードの名称にあわせる必要があり，短いノードの名称を格納する場合に無駄が発生する．そこで，今回は図 2 の表 C のテーブルを改行の無い CSV(Comma Separated Values)文書に変換し，元の XML 文書と比較する．図 2 の表 C を改行なし CSV 文書に変換すると次のようになる．

aaa,2,bbb,4,ccc,4,ddd,7,eee,8,fff,7

図 1 を含めた 3 種類の XML 文書について，変換前 XML 文書と変換後 CSV 文書のサイズとを比較した結果を表 1 に示す．なお変換前 XML では，本手法で対象としている 5 種類のノード以外を削除し，正規化を行い，ignorableWhiteSpace を削除している．表 1 より，XML 文書の性質によらず圧縮を実現できることが示せた．[2]などの手法で XML 文書を CSV 文書に変換すると，本手法より高い圧縮を実現できるが，元の XML 文書に復元できない．本手法では圧縮を実現でき，かつ元の XML 文書に復元できる．

表 1: 圧縮効果

	ア	イ	ウ
変換前 XML	5,271byte	4,718byte	41byte
変換後 CSV	5,089byte	4,434byte	35byte
圧縮率	96%	94%	83%

ア: http://api.google.com/GoogleSearch.wsdl (2005/12/16 現在)
イ: http://www.mozilla.org/news.rdf (2005/12/16 現在)　ウ: 図 1

3.2. メモリ削減効果

従来 XML 文書の操作には DOM ツリーが必要で，大量のメモリを消費していた．しかし，XML 文書を図 2 の表 C のような形式に変換することにより，XML 文書の操作をテーブルの操作に置換できる．そのため，省メモリで XML 文書を操作できる．

ノードの追加・削除は，テーブルの行の追加・削除に相当する．ノードの編集は，テーブルの 1 列目を編集すればよい．例えば，図 1 のノード <bbb> に子要素として新たにノード群

<ggg><hhh>iii</hhh><jjj>kkk</jjj></ggg>

を追加する場合を考える．まず **2.1.**節の手法により，図 4 の表 D のように追加元のノード群でテーブルを作成する．次に，追加先のノードの深さに合わせて，追加元のノード群の深さを変更する．この場合，<bbb> の深さが 2 なので表 E のように表 D の 2 列目の値をそれぞれ 2 大きくする．最後に，追加元のテーブルを追加先のノードの行の直後に挿入する．この場合，表 F のように表 E を図 2 の表 B の 2 行目の直後に挿入する．

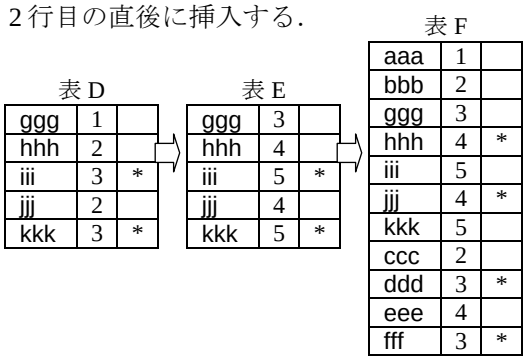

図 4: ノード追加の例

表 1 と同様の 3 種類の XML 文書について，Java の Document クラスでの DOM ツリー展開に必要なメモリと，本手法のテーブルに必要なメモリとを比較した結果を表 2 に示す．なお，測定に利用した PC の CPU は PentiumIII 800MHz，RAM は 256MB，OS は Fedora Core 1 である．表 2 より，省メモリで XML 文書を操作できることが示せた．

表 2: メモリ削減効果

	ア	イ	ウ
DOM	204,624byte	194,168byte	172,856byte
本手法	33,454byte	26,928byte	60byte

4. まとめ

スキーマを必要としない検証済み XML 文書を，元の XML 文書に復元可能な単一の 2 列のテーブルに変換する方法，および復元する方法をそれぞれ示した．また，本手法により XML 文書を圧縮できること，省メモリで XML 文書を操作できることをそれぞれ示した．

本手法ではノードの名称と構造を完全に分離できる．今後はこの特徴を用いて，本手法による XML 文書操作時の性能について検証を進める予定である．

参考文献

[1] 川口浩司他，インメモリ XML データベースの開発，情報処理学会研究報告 Vol.2003, No.78, pp9-16, 2003
[2] 小田切淳一他，大容量 XML 文書に対するデータ処理効率化手法の評価，2003 年電子情報通信学会ソサイエティ大会, B-16-8