文書スキーマを利用したXMLファイル編成法

井ノ口伸人† 吉川 正俊††

† 名古屋大学 工学部 電気電子・情報工学科 情報工学コース
†† 名古屋大学 情報連携基盤センター
〒 464-8601 名古屋市千種区不老町
E-mail: †inoguchi@dl.itc.nagoya-u.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

あらまし スキーマ情報を持つ XML 文書をネイティヴ XML データベースに格納する場合，そのスキーマ文書から構造やデータ型に関する情報を取得することができるが，それらをファイル編成のために利用した研究例は少ない．本研究ではスキーマ情報を利用することで，一般性を失うことなく，データベースごとに異なる最適化が可能だと考え，スキーマ情報を利用したファイル編成法を提案する．ある一つのスキーマ文書に対して，多数の妥当な XML 文書が存在し，その中から目的の部分文書を検索することが可能となるような索引付けとデータの物理的配置を設計する．特にデータを配置する際の複数の XML 文書の間の関係について詳しく考察を加える．

キーワード XML, データベース一般, 最適化

An XML File Organization using Document Schema

Nobuto INOGUCHI† and Masatoshi YOSHIKAWA††

† Department of Information Engineering, School of Engineering
†† Information Technology Center
Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi, 464-8601, Japan
E-mail: †inoguchi@dl.itc.nagoya-u.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

Abstract When XML documents with schema information are stored in native XML database, we can utilize information about structure and data type from a schema document. However, there are less examples of study using it for file organization. We guess it is possible to optimize database refrecting design by useing information of schema, so we suggest a method of file organization using it. There are a schema document and many valid XML documents. We discuss an indexing method and physical storage which enable to search aiming partial documents from these documents. Particularly, we consider the relationships between XML dosuments.

Key words XML, Database, Optimization

1. はじめに

XML 文書にはタグの名前を自由に記述することが可能である．しかし各利用者が独自に記述していては，XML 文書をアプリケーションで利用することや，データ交換のためのフォーマットとして用いることは不可能である．そこで XML 文書には取り得る構造に関する文法を指定することができるようになっている．そのための言語が，DTD, W3C XML Schema [1] [2] [3], RELAX NG [4] などのスキーマ言語である．DTD は XML 1.0 の仕様書中で規定されているスキーマ言語であるが，名前空間をサポートしておらず，データ型に関する情報も記述できないという二つの大きな問題が存在する．W3C XML Schema と RELAX NG はそれぞれ W3C と OASIS が制定したスキーマ言語であり，これらはどちらも DTD が持つ二つの問題を解決している．特にデータ型に関する情報は，データの物理的配置を決めるために重要なものである．そこで，本論文では，ネイティヴ XML データベースに文書を格納する際に，スキーマ文書から得られる情報を利用することで，一般性を失わずに，データベースごとに異なる最適なファイル編成を行う方法について考える．

既存の研究では，索引付けの対象は主に単一の XML 文書であり，また，複数の XML 文書を対象としている場合でも各文書に ID を振るだけであるなど，単純なものが多い．しかし，単一のスキーマ文書について多数の妥当な XML 文書が存在し，

それらをネイティブ XML データベースに格納することを考えたとき，XML 文書の間の関係は無視することはできない問題になる．そこで本論文では，XML 文書の間の関係について考察し，格納する際に考慮すべき点について述べる．

本論文の残りは以下のように構成される．3. 節で木文法 [5] について簡単に説明する．4. 節で，木文法における非終端記号の親子関係について定義する．そして 5. 節で，4. 節で定式化した関係を用いてスキーマ文書から特徴的な構造やデータを抽出する方法について述べる．6. 節で，文書間の関係について考察を加え，7. 節で，5. 節，6. 節の内容を受けた効率的なデータの物理的配置について述べる．

2. 関 連 研 究

2.1 経路情報とコンテンツの切り離し

経路情報とコンテンツを切り離す手法は，Zhang らによって [6] で提案されている．[6] では，問題としているノードの子 (child) と，そのノードより後に現れる兄弟ノード (following-sibling) に高速にアクセスするためのパターンマッチング手法 next-of-kin(NoK) pattern matching と，その上で NoK パターンマッチングを効率的に機能させるためのデータの物理的配置方法 succinct XML phisical storage scheme を提案している．succinct XML phisical storage scheme では，コンテンツのデータと，構造に関するデータを別々に格納している．コンテンツは，Dewey ID [7] をキーに持つ B+木によってと構造と対応付けられている．また，補助的な索引として，コンテンツのハッシュ値をキーとし，Dewey ID を求める B+木も備えている．これは，問合せにコンテンツによる条件が含まれているときに，利用される．また，経路情報を，NoK パターンマッチングを機能させるための，親子と兄弟が計算可能な形の文字列に変換することで，索引のサイズを圧縮している．

2.2 ネイティヴ XML データベース

ネイティヴ XML データベースに関する包括的な研究として，Fiebig らによる Natix [9] が挙げられる．[9] では，XML 文書の管理を関係データベースの一種として見ることなく，データベースシステムの性能が何に影響されるかについて，データベースシステムの各構成要素ごとに述べている．問合せ処理だけでなく，ストレージやトランザクション管理の設計および最適化のため，取り組むべき問題についても示している．それらに対する解答として，ネイティヴ XML データベース Natix 中で用いられている種々の技術について検討している．

3. 木 文 法

木文法については基本的に [5] によるが，導入として本節でも簡単に説明する．木文法には，記述力の高い順に，正規木文法，単一型木文法，局所木文法の三つのクラスがある．これらは，それぞれ RELAX NG, W3C XML Schema, DTD にほぼ相当する．

3.1 正規木文法

まず，基本的なクラスである，正規木文法と呼ばれる木文法のクラスを導入する．

［定義 3.1］ 正規木文法は四つ組 $G = (N, T, S, P)$ である．ただし，

- N は非終端記号の有限集合
- T は終端記号の有限集合
- S は開始記号の集合で，N の部分集合
- P は $X \to \mathbf{a}(r)$ の形をした生成規則の集合で，$X \in N$, $\mathbf{a} \in T$，かつ r は N 上の正規表現である

生成規則 $X \to \mathbf{a}(r)$ について，X を左辺，$\mathbf{a}$ と r を内容モデルと言う．このクラスは，ほぼ RELAX NG に相当する．ただし，RELAX NG には，属性－要素間制約とインタリービングが拡張されている．これらの詳細については, [4] に述べられている．

［例 3.2］ 以下の木文法 $G_1 = (N, T, S, P)$ は正規木文法である．

［木文法 1］

```
N = {Books, Book, Title, Author, Section1,
     Section2, Counter, Integer, Subsection,
     String, Pcdata}
T = {books, book, title, author, section,
     subsection, counter, integer, string, pcdata}
S = {Books}
P = {Books      → books(Book*),
     Book       → book(Title, Author+,
                       (Section1*|Section2*)),
     Title      → title(Pcdata),
     Author     → author(Pcdata),
     Section1   → section(String?, Title, Pcdata),
     String     → string(Pcadata),
     Section2   → section(Counter*, Title, Pcdata,
                          Subsection*),
     Counter    → counter(Integer),
     Integer    → integer( ε ),
     Subsection → subsection(Title, Pcdata,
                             Subsection*),
     Pcdata     → pcdata( ε )}
```

3.2 単一型木文法

単一型木文法を導入する前に，その定義に必要な，非終端記号の競合を導入する．

［定義 3.3］ 二つの異なる非終端記号 X, Y が，以下の三つの条件を全て満たしているとき，X と Y は互いに競合していると言う．

（1） ある生成規則が X を左辺に持つ

（2） もう一つの生成規則が Y を左辺に持つ

（3） これらの生成規則が同じ終端記号を右辺に持つ

ここでは, [5] からの拡張として，条件 3 の同じ終端記号が $\mathbf{a}$ であるとき，X と Y は終端記号 $\mathbf{a}$ について競合していると言うこととする．

［例 3.4］ 例 3.2 の正規木文法 G_1 において，非終端記号 Section1 と Section2 は，それを左辺に持つ生成規則が，右辺に同じ終端記号 section を持つので，競合している．

非終端記号の競合の定義を用いて，単一型木文法を導入する．

単一型木文法は同一内容モデル内での競合を禁止している．このクラスはほぼ W3C XML Schema に相当する．しかし，厳密には W3C XML Schema の方が記述力がやや小さく，また一部の機能は単一型木文法のクラスの能力を越えている．

［定義 3.5］ 単一型木文法とは，以下の条件を全て満たす正規木文法である．

- どの生成規則も，互いに競合する複数の非終端記号を内容モデルに持つことはない
- どの開始記号も互いに競合しない

［例 3.6］ 例 3.2 の正規木文法 G_1 は，非終端記号 Section1 と Section2 が競合しており，かつ，Book を左辺に持つ生成規則の内容モデルにそれらが同時に現れているので，単一型木文法ではない．

3.3 局所木文法

局所木文法の定義は以下の通りである．このクラスはほぼ DTD に相当する．

［定義 3.7］ 局所木文法は，競合する非終端記号を持たない正規木文法である．

［例 3.8］ 例 3.2 の正規木文法 G_1 は，非終端記号 Section1 と Section2 が競合しているので，局所木文法ではない．

3.4 解　　釈

ここで，各文法に対して妥当な木と妥当でない木を定義するために，木の解釈を導入する．

［定義 3.9］ 木 t の正規木文法 G における解釈 I とは，木の各ノード e から G の非終端記号 $I(e)$ への以下のような写像である．

- e が t のルートなら $I(e)$ は開始記号である
- 各ノード e とその子ノード列 $e_0, e_1, \ldots, e_m$ に対して，生成規則 $X \to \mathbf{a}(r)$ が G に存在して以下の条件を全て満たす
  - $I(e)$ が X
  - e の終端記号が **a**
  - $I(e_0), I(e_1), \ldots, I(e_m)$ が r にマッチする

この定義を用いた正規木文法に対する木の妥当性を導入する．

［定義 3.10］ ある正規木文法 G において，木 t の解釈が存在するとき，t は G に対して妥当である．

次に具体的な解釈の計算アルゴリズムを示す．

（1） 木の各ノードに対して，そのラベルを右辺の終端記号に持つ生成規則を探す．

（2） この生成規則の左辺の非終端記号を，この解釈におけるこのノードの非終端記号として割り当てる．

3.5 解釈の一意性

解釈の定義に基づいて，木文法の各クラスの下で，木が複数の解釈を持ち得るか否かについて述べる．

［命題 3.11］ [5] 局所木文法の下では，木の解釈は一意に定まる．

局所木文法では非終端記号の競合が許されないので，右辺に同じ終端記号を持つ生成規則は存在しない．そのため手順 1 で発見できる生成規則はただ一つである．従って，局所木文法では木の解釈は一意に定まる．

［命題 3.12］ [5] 単一型木文法の下では，木の解釈は一意に定まる．

```
<books>
 <book>
  <title>Example of XML Document</title>
  <author>Nobuto Inoguchi</author>
  <section>
   <title>XML Document</title>
   This is a exmaple of xml document.
  </section>
 </book>
</books>
```

図 1　XML 文書の例

単一型木文法では，根ノードは競合しないので，一意に非終端記号を決定できる．根ノード以外のノードについては，親ノードに対して定まった内容モデルを利用する．手順 1 では複数の生成規則を発見し得るが，内容モデル内の非終端記号間には競合が存在しない．従って，手順 1 で発見した生成規則の左辺の非終端記号中で，親の内容モデルにも出現するものを選べば，木の解釈は一意に定まる．

［命題 3.13］ [5] 正規木文法の下では，木の解釈は複数存在し得る．

［例 3.14］ 図 1 は，例 3.2 の正規木文法 G_1 について妥当な XML 文書の例である．この文書の終端記号 section の解釈は，Section1 と Section2 の二通りの可能性があり，どちらかに定まらない．

4. 木文法における非終端記号の親子関係

本節では，木の親子，祖先，子孫に相当する概念を，木文法において定義する．先に親子を定義し，その定義を用いて，祖先，子孫を定義する．

4.1 親　　子

まず，最も基本的な関係として，親と子を導入する．これは二つの非終端記号の間の関係である．

［定義 4.1］ ある正規木文法 G の二つの非終端記号 X, Y が，次の条件を満たすとき，G において，X は Y の**親 (parent)** であると言う．また，このとき同時に Y は X の**子 (child)** であると言う．

- 生成規則 $X \to \mathbf{a}(r)$ が存在し，r 中に Y が現れる．

これ以降では，便宜上，非終端記号 X の子である非終端記号の集合を $child(X)$ とする．すなわち $child(X)$ は X についての生成規則 $X \to \mathbf{a}(r)$ に対して，内容モデル r 中に現れる全ての非終端記号の集合である

［例 4.2］ 例 3.2 の木文法 G_1 において，Books は Book の親であり，Book は Books の子である．

次に，ノードの親子と非終端記号の親子の対応について調べるが，その前に準備として，木の解釈 (定義 3.9) で用いられている $I(e)$ を，ノードの解釈 E(e) として以下のように拡張定義する．

［定義 4.3］ 木文法 $G = (N, T, S, P)$ において，ノード e の終

端記号が x で，かつ $X \to x(r) \in P$ ならば，$X \in E(e)$

さらに非終端記号の親子を，次のように定義する．

［定義 4.4］ 妥当な木 t のノード e_x, e_y の終端記号をそれぞれ x, y とする．木 t において，e_y が e_x の子ならば，終端記号においても，y は x の子である．これを $y \in child(x)$ と表す．

定義 4.3，定義 4.4 より，ノードの親子と非終端記号の親子の対応は，終端記号の親子と非終端記号の親子の対応に置き換えられる．

［命題 4.5］ 木文法 $G = (N, T, S, P)$ と木 t があり，木 t のノード e_x, e_y の終端記号が $x, y \in T$ だとする．このとき，

$$X \in E(e_x), Y \in E(e_y)(X, Y \in N), y \in child(x)$$
$$\Rightarrow Y \in child(X)$$

命題 4.5 は，木文法 $G = (N, T, S, P)$ と木 t において，終端記号 $x, y \in T$ が木 t において親子であることの，解釈 $E(e_x), E(e_y)$ が親子であることに関する十分性について述べている．

a） 局所木文法

競合の許されない局所木文法においては，ノードの解釈は一意である．従って，$\mid E(e_x) \mid = \mid E(e_y) \mid = 1$ となり，ノードはただ一つの非終端記号と対応づけられる．以上のことから，命題 4.5 が成り立つのは明らかである．

b） 単一型木文法

単一型木文法においては，木と木の解釈の対応は一対一である．しかし，3.5 節で述べたように，各ノードの解釈の要素数が 1 になるには，親ノードの解釈の要素数が 1 である必要がある．そのため，x と y のどちらについても競合が発生している場合，命題 4.5 は成り立たない．

［例 4.6］ ある単一型木文法 $G_2 = (N, T, S, P)$ が以下の生成規則を含む．

［木文法 2］

```
X  → x(Y),
X′ → x(Y′, Z),
Y  → y(Pcdata),
Y′ → y( ε ),
Z  → z(Pcdata),
```

このとき，終端記号が x, y である親子ノード e_x, e_y に対して，$E(e_x) = \{X, X'\}, E(e_y) = \{Y, Y'\}$ である．e_x, e_y は親子なので，$y \in child(x)$ であるが，$Y \notin child(X'), Y' \notin child(X)$ である．

c） 正規木文法

正規木文法の記述力は，単一型木文法よりも高い．従って，正規木文法においても，命題 4.5 は成り立たない．

4.2 祖先，子孫

本節では，先に定義した親子関係を用いて，非終端記号の祖先と子孫を導入する．

［定義 4.7］ ある正規文法 G の二つの非終端記号 X, Y が以下の条件を満たすとき，X は Y の**祖先 (ancestor)** であると言う．また，このとき同時に Y は X の**子孫 (descendant)** である．

- $Y \in desc(X)$

$desc(X)$ は次に示す方法で再帰的に定義される集合である．この再帰的計算は新しい要素が加えられなくなった時に終了する．

$$desc(X) = child(X) \cup \bigcup_{Y \in child(X)} desc(Y) \quad (1)$$

［例 4.8］ 例 3.2 の木文法 G_1 において，Section1 の子孫は String, Title, Pcdata である．

祖先，子孫関係についても，親子関係と同様に，木文法の各クラスに対して，ノードの子孫，祖先が非終端記号の子孫，祖先と対応するか否かについて調べる．そのための準備として，終端記号の子孫，祖先を導入する．

［定義 4.9］ 妥当な木 t のノード e_x, e_y の終端記号をそれぞれ x, y とする．木 t において，e_y が e_x の子孫ならば，終端記号においても，y は x の子孫である．これを $y \in child(x)$ と表す．

［命題 4.10］ 木文法 $G = (N, T, S, P)$ と木 t があり，木 t のノード e_x, e_y の終端記号が $x, y \in T$ だとする．このとき，

$$X \in E(e_x), Y \in E(e_y)(X, Y \in N), y \in desc(x)$$
$$\Rightarrow Y \in desc(X)$$

a） 局所木文法

局所木文法のクラスでは，終端記号の解釈は一意である．従って，$\mid E(e_x) \mid = \mid E(e_y) \mid = 1$ となり，ノードはただ一つの非終端記号と対応づけられる．以上のことから，命題 4.10 が成り立つのは明らかである．

b） 単一型木文法

単一型木文法のクラスでは，親の解釈の要素数が 1 になれば，終端記号の解釈の要素数が 1 になる．しかし，祖先，子孫を考えるときは，各ノードの親は考えていない．そのため，x, y のいずれかについて競合が発生している場合，命題 4.10 は成り立たないことがある．

［例 4.11］ 以下の条件を全て満たす単一型木文法 G_3 があるとする．

［木文法 3］

- $Y_1 \in desc(X)$
- $Y_2 \notin desc(X)$
- 終端記号 X の解釈は x に定まる．
- 非終端記号 Y_1 と Y_2 が終端記号 y について競合している．

ここで，終端記号が $x, y \in T$ であるノード e_x, e_y の解釈がそれぞれ $X \in E(e_x), Y_2 \in E(e_y)(X, Y_2 \in N)$ を満たし，e_x が e_y の親であるとする．このとき，$X \in E(e_x), Y_2 \in E(e_y)(X, Y_2 \in N)$ かつ $y \in desc(x)$ であるが，$Y_2 \notin desc(X)$ である．

c） 正規木文法

正規木文法の記述力は，単一型木文法よりも高い．従って，正規木文法においても，命題 4.10 は成り立たない．

5. スキーマ情報の利用

スキーマ文書を解析して，経路に関する情報を取得する．基本的な方針として，根ノードから葉ノードへの経路を求める．ただし，XML 文書中でラベルを持たないテキストノードに相当する非終端記号は，経路として考えない．

最も簡単な場合は，葉ノードの解釈である非終端記号を生成

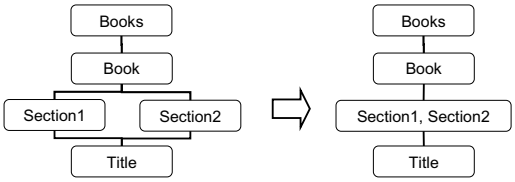

図 2　Tempral Branch

するまでの，開始記号からの生成規則の適用順序が一意に定まる場合である．この種のノードを検索する場合には，経路の正誤だけで検索が可能である．また XPath 式の self-or-descendant 軸を一意に展開することも可能である．

経路が一意に定まらない時の原因として，経路が枝分かれする場合と，スキーマ情報に再帰的な構造が含まれている場合が考えられる．まず枝分かれした経路は，基本的に個別に扱う．しかしある特殊な枝分かれに関しては，グループ化する意味があると考えられる．これについては，5.1 節で詳述する．次に再帰構造を持つ場合を考える．この場合は，取り得る構造が無限に存在するので，スキーマ解析時は変数で置換しておく．そして XML 文書を解析した際に，実際の構造をコンテンツと共に保持させる．詳しくは 5.2 節で詳述する．

5.1　Temporal Branch

枝分かれした経路は基本的には異なる経路として個別に記憶する．しかし，一度分かれた経路が再び合流する場合には，それらの一時的な枝分かれ (Temporal Branch) をグループ化することで索引の圧縮が期待できる．このグループ化は，一時的な枝分かれが分類のために用いられている場合，言い換えれば，その子孫に位置するコンテンツが同じ実体集合として抽象化できる場合に効果的である．なお，本稿では以下の木文法に基づいた定義を Temporal branch とする．

［定義 5.1］ $child(X) = \{Y_1, Y_2, \ldots, Y_n\}$ であり，異なる二つの生成規則 $Y_i \rightarrow \mathbf{b_i}(r_i), Y_j \rightarrow \mathbf{b_j}(r_j)(i, j \leqq n, i \neq j)$ について，r_i, r_j に含まれる非終端記号の集合をそれぞれ N_i, N_j とする．$N_i \cap N_j \neq \phi$ となったとき，Y_i, Y_j のことを Temporal Branch と呼ぶ．

これらのノードは，検索条件として構造の細かな指定がない限り，同時にスキャンされるので，グループ化が有効だと考えられる．経路情報の中に現れる Temporal Branch を同じ変数で置換することでグループ化を実装し，その変数に代入すべきノード名はコンテンツとのペアにして記憶しておく．直感的には，図 2 に示すような，親と子を共有する非終端記号の集合をグループ化する．

［例 5.2］ 例 3.2 の木文法 G_1 において，Section1 と Section2 は Temporal branch である．従って Title から開始記号までの経路は，図 2 のように Title/(Section1, Section2)/Book/Books とまとめることができる．

5.2　再帰循環 (Recursive cycle)

スキーマ文書中に再帰的な構造が含まれる場合，そこから生成される木は理論上無限の深さを持ち得る．すなわち無限の経路を持つことになる．従ってスキーマ文書のみで，再帰的な構造を扱うことは現実的ではない．そこで再帰的な構造については変数で置換しておく．実際の文書を解析した時に，変数に代入すべき経路は Temporal Branch で用いた手法と同じく，コンテンツとのペアにして記憶しておく．再帰的な構造を作る非終端記号を以下のように定義する．

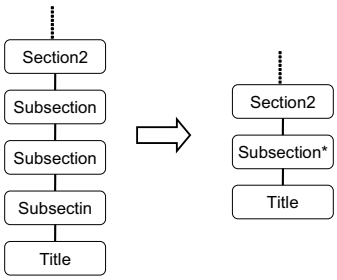

図 3　Rcursive Cycle

［定義 5.3］ 非終端記号 X について，$X \in desc(X)$ ならば，X は**再帰循環 (recursive cycle)** であるという．

［例 5.4］ 例 3.2 の木文法 G_1 において，Subsection が再帰循環である．従って，図 3 のように，Subsection の子である Pcdata, Title と，親である Section2 の間を変数で置換することで経路数を減らすことができる．

5.3　コンテンツへの構造埋め込み

次に既存の手法だが，構造のテキスト化について述べる．例えば XHTML における<b>タグや<i>タグは，混在モデルとして DTD では次のように表現される．

<!ELEMENT section (#PCDATA | b | i)* >

この場合の<b>タグ，<i>タグは表示の制御のために用いられており，文書の構造には直接関わっていない．このように混在モデルを形成しているタグを構造情報として含めることは，経路情報としては冗長である．従ってこれらのタグもテキストデータとしてコンテンツ内に埋め込んでしまった方が，効率的であると考えられる．こうした混在モデルを形成する非終端記号は，生成する木が複数のテキストノードと兄弟であることが多いので，これを利用してスキーマ文書から候補を検出する．

6.　XML 文書

複数の XML 文書を対象にすることから，本節では，XML 文書の間の関係について考察する．そのために，まず一文書の意味を考えることから始める．その後に，それらの文書の間の関係が何を表しているのかを考える．

6.1　一文書の意味

本節では，単一の XML 文書が，オブジェクトをどのようにモデル化して格納しているかについて述べる．伝統的な関係データベースでは，ER モデルに基づいて設計され，各テーブルがそれぞれ実体あるいは関係と一体一対応になっていることが多い．これに対して，ネイティヴ XML データベースでは，単一の XML 文書がより多くの情報を持つことが可能である．XML 文書は，それが表すオブジェクトの数から，図 4 に示した Single Object XML document(SO-XML) と Multi Objects XML document(MO-XML) の二種類に大別することができる．

なお，図 4 は四角が一文書，○が根ノード，△が一オブジェクトを構成する部分木を表現している．さらに，MO-XML の特殊な場合として，Database XML document(DB-XML) を加えた三種類の XML 文書について以下で説明する．

a） Single Object XML document (SO-XML)

単一の XML 文書が，一つのオブジェクトを表現している場合を考える．ER モデルに照らし合わせて言えば，XML 文書には複数の実体とそれらの関係，すなわち ER モデル全体が含まれていると言える．なお，この種の XML 文書のサイズは比較的小さなものとなる．

［例 6.1］ 以下の木文法 G_4 は，本一冊を表す木文法である．G_4 に従う文書は，どれも SO-XML である．

［木文法 4］

```
N = {Book, Title, Author, Section1, Section2,
     Counter, Integer, Subsection, String,
     Pcdata}
T = {book, title, author, section, subsection,
     counter, integer, string, pcdata}
S = {Book}
P = {Book       → book(Title, Author+,
                       (Section1*|Section2*)),
     Title      → title(Pcdata),
     Author     → author(Pcdata),
     Section1   → section(String?, Title,
                          Pcdata),
     String     → string(Pcdata),
     Section2   → section(Counter*, Title,
                          Pcdata, Subsection*),
     Counter    → counter(Integer),
     Integer    → integer(ε),
     Subsection → subsection(Title, Pcdata,
                             Subsection*),
     Pcdata     → pcdata(ε)}
```

これは木文法 G_1 から非終端記号 Books と，それを左辺に持つ生成規則，そして終端記号 books を取り除き，開始記号を Books から Book に変えたものである．情報量は G_1 で生成される木の中で，Book が一度しか現れないものに等しい．従ってその木も SO-XML と言える．

b） Multi Objects XML document (MO-XML)

次に，単一の XML 文書がデータベースの部分集合を形成している場合を考える．例えば，問合せの解は，このような形をしている．これは SO-XML と後述の DB-XML の中間的な性質を持つと考えられる．

［例 6.2］ データベースを構成する複数の XML 文書が，木文法 G_1 について妥当なである時，それらのの内，SO-XML でないものは MO-XML である．

c） Database XML document (DB-XML)

次に，MO-XML の特殊な場合として，単一の XML 文書がデータベース全体を形成している場合を考える．このときの XML 文書は，根ノードの下に多くの兄弟ノードが存在し，それらを根とする部分木がそれぞれ一オブジェクトを表現している．従って，この種の XML 文書のサイズは非常に大きなものとなる．従って，一文書を扱うコストも巨大である．なお本稿では複数の XML 文書を対象としているため，以下ではこの種の XML 文書については考えない．

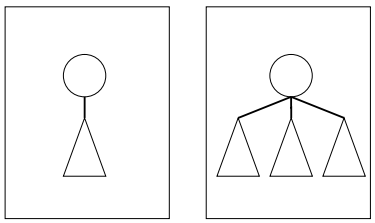

図 4　SO-XML と MO-XML の例

［例 6.3］ データベースが単一の XML 文書で構成されており，それが木文法 G_1 について妥当ならば，その XML 文書は DB-XML である．

6.2　データベースと文書

本節で文書の種類がデータベースに与える影響について考察する．複数の XML 文書を格納するデータベースの場合，それらの文書の種類については以下の二通りが考えられる．

a） SO-XML

どの XML 文書も，一つのオブジェクトを表している特殊な場合である．この状態では，各文書のサイズの差は小さく，また構造も近いため，定型的に扱うことが比較的容易である．

b） SO-XML, MO-XML 混在

各 XML 文書が，一つ以上のオブジェクトを表している場合である．この状態では，含まれるオブジェクトの数によって，各文書のサイズが大きく異なり得る．従って定型的に扱うことが難しい．さらに，文書間から正確に一オブジェクトを取り出すことが難しいという問題も現われる．

6.3　XML 文書の間の関係

本節では，XML 文書の間の関係について述べる．一般に XML 文書どうしに直接の関係は存在しない．しかし，同じスキーマ文書に従った文書の集合がデータベースを形成する場合，文書を単位として物理的に配置すると，クラスタ化が不可能になるという問題がある．例えば，クラスタ化を保ちながら，XML 文書をデータベースに挿入したい時に，挿入すべき場所が，MO-XML の中に存在するような場合である．あるいは，データベースに MO-XML を挿入する場合もこの例は発生する．つまり，クラスタ化を保持するためには，MO-XML を分割する必要がある．

7.　物理的配置

本節では，XML 文書の物理的配置方法について考察する．ここでは，木文法 5 を例として用いる．木文法 5 は，XMark [8] の DTD 文書から一部を抜粋し，さらに説明のために簡略化したものである．XMark の DTD 文書はオークションに関する情報を表現しているが，木文法 5 によって生成される木を解釈に持つ XML 文書は，オークションで売買される商品の集合を表

現している．従って，木文法 5 によって生成される木を解釈に持つ XML 文書は，MO-XML である．

［木文法 5］

```
N = {Site, Regions, Africa, Asia, Australia,
     Europe, Namerica, Samerica, Item, Id,
     Location, Quantity, Name, Payment,
     Shipping, Pcdata}
T = {site, regions, africa, asia, australia,
     europe, namerica, samerica, item, id,
     location, quantity, name, payment,
     shipping, pcdata}
S = {Site}
P = {Site      → site(Regions),
     Regions   → regions(Africa|Asia|Australia
                    |Europe|Namerica|Samerica),
     Africa    → africa(Item),
     Asia      → asia(Item),
     Australia → australia(Item),
     Europe    → europe(Item),
     Namerica  → namerica(Item),
     Samerica  → samerica(Item),
     Item      → item(Id, Location, Quantity,
                    Name, Payment, Shipping),
     Id        → id(Pcdata),
     Location  → location(Pcdata),
     Quantity  → quantity(Pcdata),
     Name      → name(Pcdata),
     Payment   → payment(Pcdata),
     Shipping  → shipping(Pcdata),
     Pcdata    → pcdata(ε)}
```

7.1 単　　位

まず何を単位として物理的配置するかを考える．第一に考えられるのが，XML 文書である．しかし，この場合は，6.3 節で述べた問題が発生する．各 XML 文書のサイズは広範囲に渡るため，定型的に扱うことが難しい．また，利用者は XML 文書を意識してデータベースにアクセスすることはないため，文書を単位として扱う積極的な理由は見当たらない．

次にオブジェクトを単位として扱う場合を考える．オブジェクトとは，ある非終端記号を開始記号として考えたときに生成される木である．生成された木は，いずれもある一つの実体を表現している．文書が SO-XML の場合，文書を単位として扱うことに等しい．これは，関係データベースでは，レコードごとに扱う手法 (N-ary Storage Model) に相当する．しかし，MO-XML の場合は，どの非終端記号をオブジェクトの開始記号とするかという問題がある．オブジェクトのサイズは文書ほどのばらつきはない．

［例 7.1］ 木文法 5 において Item を開始記号として生成される木は，一つの商品を表現している．そのため，オブジェクトと考えられる．

次にコンテンツを単位として扱う場合を考える．ここで言うコンテンツとは，あるオブジェクトが持つ，テキストノードとその親ノードのペアのことである．これは，関係データベースで，属性ごとに扱う手法 (Decomposite Storage Model [10]) に相当する．関係データベースでは属性ごとにデータ型が定義されているが，XML 文書に関しては，一般にその限りではない．しかし，スキーマ文書が W3C XML Schema か RELAX NG ならば，データ型に関する情報が含まれているので，データのサイズや，固定長と可変長のどちらなのかを知ることができる．これらの情報によって極めて定型的に配置することが可能である．ノードごとに配置した場合は，元々のオブジェクトを再現するための付加情報が必要になる．

［例 7.2］ 木文法 5 において，各 Item オブジェクトは，Id, Location, Quantity, Name, Payment, Shipping の六つのコンテンツを持つ．

オブジェクトと属性のどちらを単位として扱うかは，最終的に発行される問合せの種類に依存する．つまり問合せの解が，どちらになることが多いかによって，最適解もまた異なる．

複合的なアプローチとして，基本的にはオブジェクトを単位としながら，一部の属性を別に保存する方法が挙げられる．属性を選ぶための判断基準としては，スキーマ情報がある．スキーマ文書から各属性のデータ型を調べ，可変長のデータは，別に保存することで効率的なファイル編成が可能になる．また，7.2 節で後述するが，判断基準として，アクセス頻度を考えることもできる．

7.2 オブジェクト，コンテンツの抽出

a） オブジェクトの抽出

オブジェクトは，一塊の情報を表す部分木として表現される．木のどの部分を指してオブジェクトとするかは，スキーマの設計意図に依存するため，機械的な抽出は困難である．ただし，W3C XML Schema では，スキーマレベルで値の一意性が保証されているノードと，その値によって識別される範囲を指定する機能が用意されている．しかし，ここでは，文書スキーマの種類に関わらず，オブジェクトの抽出を行うために，識別子ノードを導入する．識別子ノードとは，その値の一意性が保証されているノードである．識別子ノードの存在を仮定すれば，オブジェクトは以下のように定義される．

［定義 7.3］ 次の二つの条件を満たす部分木を，オブジェクトと呼ぶ．

（1） ただ一つの識別子ノードを持つ．

（2） 含むノード数が最大である．

条件 1 によって，オブジェクトの一意性が保証される．条件 2 によって，情報の損失がないことが保証される．

識別子ノードとして，ID 型の属性ノードを利用することが考えられる．ID 型の属性ノードは，DTD で規定されているが，互換性のために，W3C XML Schema, RELAX NG においても利用可能である．ただし，ID 型には以下のような問題がある．

- 属性にしか指定できない．
- 組み合わせによる一意性の保証が不可能．
- 常に文書全体を一意性検証の範囲とする．
- 属性名に関わらず，ID 型のデータ全体で一意性を検証

する．

従って現状では，アプリケーションから識別子ノードを指定するアプローチも視野に入れる必要がある．

また，識別子ノードが指定されていない場合は，自動抽出のための足がかりとして，内容モデル中で，*が用いられている部分を利用することが考えられるが，決定的ではない．以下では，ある非終端記号をオブジェクトの開始記号として決定できたという仮定の下に議論を行う．一般に，XML 文書は複数種類のオブジェクトを持つことができる．オブジェクトの分類は，基本的には，開始記号からオブジェクトの開始記号への経路情報によって行われる．しかし，その経路情報の代わりに，再帰循環と Temporal Branch を検出し，置換したものを利用することで，構造による瑣末な分類によらず，オブジェクトを横断的に扱うことが可能となる．

［例 7.4］ 開始記号からの経路によって分類した場合，木文法 5 には，以下の六種類の Itme オブジェクトが存在する．

- Site/Regions/Africa/Item
- Site/Regions/Asia/Item
- Site/Regions/Australia/Item
- Site/Regions/Europe/Item
- Site/Regions/Namerica/Item
- Site/Regions/Samerica/Item

Temporal Branch(Africa, Asia, Australia, Europe, Namerica, Samerica) を置換することで，これらの経路は等しくなり，いずれも同じオブジェクトを導くことが分かる．

b） コンテンツの抽出

コンテンツは含まれるオブジェクトと，オブジェクトの開始記号からコンテンツノードまでの経路によって区別される．オブジェクト同様に，各経路に対して再帰循環と Temporal Branch の検出，および置換を実行することで，コンテンツも横断的に扱うことができる．

7.3 統計情報の利用

スキーマ情報以外の情報，あるいはスキーマ情報を補う情報を統計情報から得ることで，スキーマ情報だけで索引付けを行う場合に欠点を補うことが可能である．ここでは統計情報として取得すべきものについて，問合せの種類と再帰循環の深さと属性の更新頻度の三種類について考える．

a） 問合せの種類

問合せの解が，オブジェクトを単位とするものと，属性を単位とするもののどちらが多いかによって，最適な物理的配置は異なる．しかし，統計情報は運用の中で初めて得られるので，最初から最適解を決定することは不可能である．しかし複合的なアプローチを選択した際，別に保存する属性を決定するためにこの情報が利用できる．最初はオブジェクトを単位として配置し，ある属性にアクセスが集中するならば，その属性を別に配置する．

b） 再 帰 循 環

再帰循環を含む経路は理論上は無限の深さを持ち得るため，スキーマ情報だけでは扱いが難しい．しかし実際のデータ中から分布を見出すことが可能である．例えば，深さはある範囲に収まり，その中でもごく狭い一部の範囲に分布が集中することが考えられる．そこで再帰循環の繰り返し回数を統計情報として取得することが有用である．

c） 更 新 頻 度

属性の更新頻度もデータの配置に利用できる情報である．更新が起こらなければ，ページに余白を持たせる必要がなくなる．

8. おわりに

本論文では，スキーマ文書から得られる情報を利用した，ネイティヴ XML データベースのファイル編成法について提案した．木文法で表現されるスキーマ情報と，木構造で表現される XML 文書の対応の理論的な裏づけとして，木文法を取り入れることによって，スキーマに現れる二種類の特徴的な構造，再帰循環と Temporal Branch を示した．また，あるスキーマ文書に関して妥当な多数の XML 文書を効率的に扱うために，含まれるオブジェクトの数によって XML 文書を，SO-XML, MO-XML, DB-XML の三種類に分類した．このことによって，多数の XML 文書を，スキーマ文書を通して類型的に扱うことができる．

今後の課題としては，以下の二点が挙げられる．

- XML 文書のファイル編成法の提案のために，経路情報のグループ化と，XML 文書の種類によるコンテンツの配置戦略について，主に理論的な側面から考察を加えてきた．今後は，経路情報とコンテンツを結びつけるための具体的な索引付け手法について考えることが今後の課題である．
- MO-XML 文書をオブジェクト単位に分割して格納することが有効だと述べたが，その際に文書をオブジェクトに分割するための具体的なアルゴリズムの提案にまでは至っていない．今後は，この分割手法について考えていくことが課題となる．

文　献

[1] David C. Fallside, Priscilla Walmsley, ”XML Schema Part0: Primer”, http://www.w3.org/TR/xmlschema-0/

[2] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, ”XML Schema Part1: Structures”, http://www.w3.org/TR/xmlschema-1/

[3] Paul V. Biron, Ashok Malhotra, ”XML Schema Part 2”, http://www.w3.org/TR/xmlschema-2/

[4] James Clark, MURATA Makoto, ”RELAX NG Specification”, http://www.oasis-open.org/committees/relax-ng/spec-20010811.html

[5] 村田 真, 河口 耕介, ”チュートリアル XML とスキーマ言語”, コンピュータソフトウェア, vol.21, no.6, pp.50-59, November 2004.

[6] Ning Zhang, Varun Kacholia, M. Tamer Özsu, ”A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML”, ICDE 2004, pp.54-65

[7] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita C. Zhang, ”Storing and Querying Ordered XML Using a Relational Database System”, Proceedings of ACM SIGMOD 2002, pp.204-215, June 2002.

[8] A. Schmidt, M. Kersten, D. Florescu, M. Carey, I. Manolescu, and F. Waas, ”XMark — An XML Benchmark Project” http://www.xml-benchmark.org.

[9] Thorsten Fiebig, Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, Julia Neumann, Robert Schiele, Till Westmann, ”Natix: A Technology Overview”, Web Databases and Web Services 2002, LNCS 2593, pp.12-33, 2003.

[10] G. P. Copeland, S. F. Khoshafian, ”A Decomposition Storage Model”, In proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 268-279, May 1985.