データベースシステム 114－10
（1998. 1. 19）

効率的な更新が可能な構造化文書の索引

金本博隆 加藤弘之 絹谷弘子 吉川正俊

奈良先端科学技術大学院大学 情報科学研究科

本稿では，効率的な更新が可能な構造化文書の索引を提案する．この索引は，文書の論理構造を管理するStructure Indexと，文書の内容，属性名や属性値などを管理するContent Indexの二種類の索引から構成される．したがって，この索引を用いることで，構造化文書に対するあらゆる種類の問合せを効率的に処理することが可能となる．我々は，問合せ能力，検索コスト，更新コストの観点から既存の索引と我々の索引との比較を行ない，それぞれ点で提案する索引の方が優位であるか，もしくは同等であることを示す．更に，我々の索引を実装するための枠組を提案する．

An Efficiently Updatable Index Scheme for Structured Documents

Hirotaka KANEMOTO, Hiroyuki KATO, Hiroko KINUTANI and Masatoshi YOSHIKAWA

Graduate School of Information Science
Nara Institute of Science and Technology (NAIST)

In this paper, we propose an efficiently updatable index scheme for structured documents. This index scheme consists of two types of indices. One is called *structure index*, and manages logical structure of documents. The other is called *content index*, and manages contents, elements names, attributes names and attributes values of documents. Using our indices, therefore, a wide range of queries over strucuted documents can be processed efficiently. We compare our indices with existing indices in terms of expressive power of queries, update costs and retrieval costs. As a result of these comparisons, we show that our indices are superior than existing indices at each of the points of views. Moreover, we have designed a scheme of the indices for implementation.

1. はじめに

1.1 研究の背景と動機

電子化文書の共有や再利用の観点から SGML に代表される構造化文書が利用されつつある. また, SGML のサブセットに当たり SGML 文書を Web 環境で利用することを考慮した文書記述言語 XML(eXtensible Markup Language) の仕様の策定作業が最終段階に入った. XML 対応のブラウザ等のアプリケーションが実装され普及すると, 構造化文書がさらに広く利用されるものと考える.

従来は文字列の追加・削除などの更新がない場合 (もしくは追加のみ) の構造化文書の索引手法が考えられ提案されてきた. しかし文字列の更新を考慮した場合の索引手法の研究はあまりされていない. 文字列の更新を考慮しない索引手法をそのまま文字列の更新を考慮する場合に適用したとすると, 索引の更新コストが高くなり不適当であると考える. そのため, 文字列の効率的な更新を目指した構造化文書の索引手法を研究する必要がある.

1.2 研究の目的

本研究では, 効率的な更新が可能な構造化文書の索引手法の提案を目的としている. 索引のアーキテクチャを示し, さらに実装手法を議論する. 第 2 章では構造化文書について述べる. 第 3 章で Structure Index と Content Index を用いる索引手法を提案する. 第 4 章ではこれまで用いられてきた構造化文書の索引手法を紹介する. 第 5 章でこれまでの索引手法と提案する索引手法を比較する. 第 6 章で実装手法を議論し, 第 7 章でまとめと今後の課題について述べる.

2. 構造化文書

本章では, 構造化文書の中で広く利用されている SGML(Standard Generalized Markup Language)[ISO86] [JIS92] と, 現在仕様の策定が進められている XML(eXtensible Markup Language)[Wor97] を簡単に紹介する.

2.1 SGML

SGML は 1986 年に国際規格 (1992 年に JIS 規格) となった, 文書データの多角的利用と異機種間の文書交換を目的とした文書記述言語である.

SGML 文書は三つの部分から構成される.

- SGML 宣言 … 文字コード, 宣言や定義に使う記号, タグ名に使う文字などを宣言する.
- 文書型定義 (DTD) … 文書の論理構造を定義する.
- SGML 文書インスタンス … 文書型定義に従って記述されたタグ付き文書.

2.2 XML

XML は, SGML のサブセットに当たり SGML 文書を Web 環境で利用することを考慮した文書記述言語である. XML の特徴は, SGML の仕様から実装が困難なものや仕様に記述されているが意味のないものを取り除き, HTML と比較した場合のリンク機能の強化やスタイルシートを追加することなどが挙げられる.

現在 W3C(World Wide Web Consortium) で仕様の策定が進められている.

```
<!-- DTD for Article -->
<!ELEMENT Article
      -- (Title, Author+, Abstract, Document)>
<!ELEMENT Title      -- (#PCDATA)>
<!ELEMENT Author     -- (#PCDATA)>
<!ELEMENT Abstract   -- (#PCDATA)>
<!ELEMENT Document   -- (#PCDATA|Section)+>
<!ELEMENT Section    -- (#PCDATA|Paragraph)*>
<!ATTLIST Section       Update #PCDATA #REQUIRED
                        Status #PCDATA #REQUIRED>
<!ELEMENT Paragraph  -- (#PCDATA|Keyword|Word)*>
<!ATTLIST Paragraph     Update #PCDATA #REQUIRED
                        Status #PCDATA #REQUIRED>
<!ELEMENT Keyword    -- (#PCDATA)>
<!ELEMENT Word       -- (#PCDATA)>
```

図 1 論文の文書型定義 (Article.dtd)

2.3 構造化文書の例

本稿では, 構造化文書の例として XML を想定して議論していくことにする[1]. 図 1の論文の文書型定義 (DTD) と図 2の XML 文書インスタンスを用いる. 文書型定義 (図 3) および XML 文書インスタンス (図 4) は木構造で表現できる.

3. 提案する索引手法

本章において, XML 文書インスタンスの階層構造と element の属性を対象にした索引である Structure Index と, element 中の属性や属性値と文書中の部分文字列を対象にした索引である Content Index を提案する.

構造化文書の問合せは文字列 (content), 構造 (structure), 属性 (attributes) を対象とした問合せに分類できる [SDDTZ97]. Structure Index と Content Index を利用することによりこれらの問合せに対応可能である.

3.1 Structure Index

Structure Index は文書の階層構造と element の属性に関する問合せに適した木構造の索引である (図 5).

本研究では XML 文書インスタンス中の element の入れ子の発生や文書の階層構造が変化する場合も考慮している. そのため索引の葉とそれ以外の節の構造を基本的に同じにする必要がある. したがって索引は

- element 名
- element 内に含まれる文字数
- XML 文書インスタンス中の出現位置 (ページ番号, スロット番号)
- 親・子・次に出現する element へのポインタ
- 属性や属性値 [2]

を管理する必要がある. 子 element へのポインタを持つものが節, 持たないものを葉として区別する.

索引中の element 名には element で囲まれた文字列の場合は element 名を, element で囲まれていない文字列の場合は NULL をそれぞれ格納する.

属性や属性値は, 検索効率の向上のために該当する element と関連づけて格納する.

[1] SGML についても XML と同様の議論が可能である.
[2] 複数回出現可能

```
<?xml virsion="1.0" encoding="Shift_JIS"?>
<!DOCTYPE Article SYSTEM "Article.dtd">

<Article><Title>XML の紹介</Title>
<Author>金本博隆</Author>
<Abstract>現在話題になっている
XML(eXtensible Markup Language) の紹介を行う.
</Abstract>

<Document>
<Section Update="1997-12-22"
Status="Unfinished">
<Title>はじめに</Title>

<Paragraph Update="1997-12-17"
Status="Finished">
私は<Keyword>構造化文書</Keyword>に
<Word>興味</Word>を持ち, 特に
<Keyword>XML</Keyword>の
<Word>調査</Word>を行っている.
</Paragraph>

<Paragraph Update="1997-12-22"
Property="Unfinished">
<Keyword>XML</Keyword>とは, 現在
<Keyword>W3C</Keyword>で<Word>制定作業</Word>
が進められている<Keyword>マークアップ言語</Keyword>
である.<Keyword>XML</Keyword>は
<Keyword>SGML</Keyword>の<Word>サブセット</Word>
に当たり, 単に<Keyword>HTML</Keyword>の
<Word>拡張版</Word>であるという<Word>見方</Word>
は必ずしも<Word>正しい</Word>とはいえない.
</Paragraph>
</Section>
...
```

図 2 XML 文書インスタンス

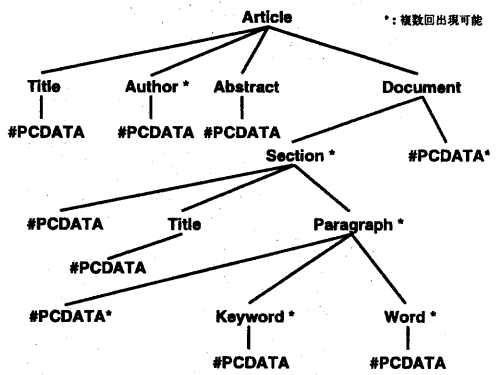

図 3 文書型定義 (DTD) の木構造表現

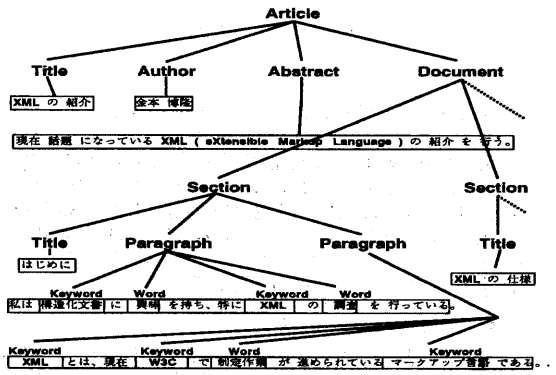

図 4 XML 文書インスタンスの木構造表現

XML 文書インスタンス中の出現位置には, XML 文書インスタンスが格納されているページ番号とスロット番号を格納する.

構文解析した XML 文書インスタンスは, ディスクのページ内に図 6の形式で格納されている. タグつきの文字列が格納されているデータ部と, 文字列を指している Slot Directry からなる. Slot Directory はページの先頭からの文字数を管理しており, XML 文書インスタンスは索引から Slot Directory を通して参照される. 初期状態では図 6のように, Slot Directory のスロット番号は右下から左上に向かい順番に割り当てられている.

Structure Index と構文解析した XML 文書インスタンスを格納しているページとの関連をみるため次の問合せを考える：

第 1 章第 1 パラグラフの中で<word>でタグづけされた 1 番目に出現する文字列を見つけなさい.

はじめに Structure Index を根から
<Document> → <Section>(1) → <Paragraph>(1) → <Word>(1)
の順でたどる (( ) は出現順序). Structure Index から, 出現位置は (10, 33) であることが分かる. 次に 10 ページの Slot Directory の 33 番目を参照する. 33 番目の Slot Directory には 355 が格納されているので, 355 番目の文字列を参照する. 開始タグを無視し 2 文字取り出すと, 求める文字列"興味"が求められる.

3.2 Content Index

Content Index は element と文字列に関する問合せに適した転置索引で, 検索対象に対応した四つの索引とそれに対応した Postinglist からなる (図 7). Postinglist には出現位置が格納されている.

四つの索引は XML 文書インスタンス中の部分文字列, element 名, 属性名, 属性値を管理し, Word Inverted Index, Element Inverted Index, Attribute Inverted Index, Attribute Value Inverted Index と呼ぶ. これらの索引は,

- 文字列, element 名, 属性名, 属性値のいずれか一つ
- 出現回数
- Postinglist へのポインタ

を管理する.

XML 文書インスタンス中の部分文字列は, Content Index から次のように検索される. 例として次の問合せを考える：

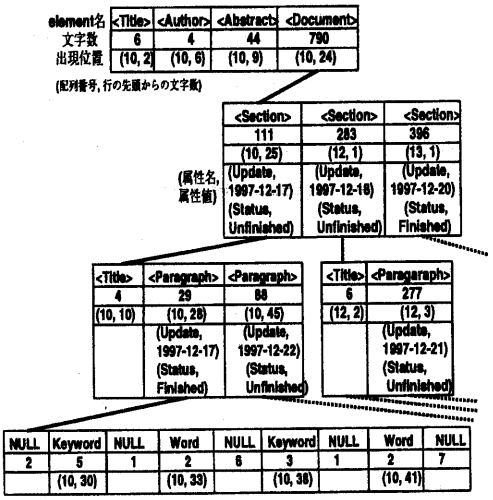

図 5 Structure Index

文字列"XML"の出現位置を求めなさい.

Word Inverted Index から出現位置は (10, 3), (10, 10), (10, 39) なので, ページ番号 10 の Slot Directory の 3, 10, 39 番目を参照する. Slot Directory には 99, 144, 385 が格納されているので, ページの先頭から 99, 144, 385 文字目を参照すれば求める文字列"XML"が検索できる.

3.3 提案する索引の検討

前節で提案した二つの索引を問合せ能力, 更新コスト, 問合せコストの面から検討する. 本稿におけるコストとは, ディスクへのアクセス回数のことであるとする.

3.3.1 問合せ能力の検討

構造化文書の問合せは文字列 (content), 構造 (structure), 属性 (attributes) を対象とした問合せに分類できる [SDDTZ97].

文書の階層構造と element の属性を対象とする問合せは Structure Index を, element の属性と文字列を対象とする問合せは Content Index を利用する. 文字列や文書の階層構造, 属性に関する複合した問合せについては, これらの索引を組み合わせて利用することにより構造化文書上の問合せすべてに対応可能である.

3.3.2 更新コストの検討

本節では更新コストを検討する. 例として 1997 年 12 月 22 日に, 経路式<Article>.<Document>.<Section>(1).<Paragraph>(1) でたどることのできる<Keyword>XML</Keyword>の直後に, <Keyword>Hytime</Keyword>を挿入した場合を考える (図 8).

Structure Index

Structure Index の葉と, 葉から根に向かう経路の途中の節を更新するので, 索引の更新回数は XML 文書インスタンスの階層構造に依存する. 文書の階層構造の高さを h とすると, 更新コストは高々$O(h)$ である.

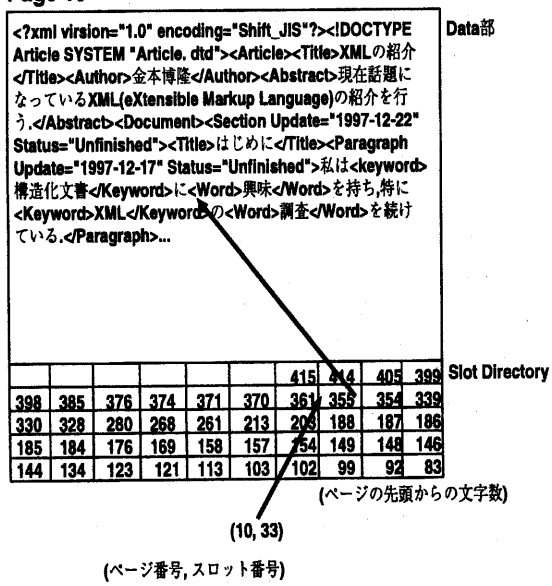

図 6 XML 文書インスタンスのディスクにおける格納形式

Content Index

Word Inverted Index の Hytime と, Element Inverted Index の<Keyword>とその Postinglist を更新する. Content Index をハッシュ法で実現した場合, 更新コストは $O(1)$ である.

この場合, 提案する索引の更新コストは一般的に $O(h)$ となる.

3.3.3 検索コストの検討

本節では提案する索引の検索コストの検討を行う. 検索例として二つの索引を利用する次の問合せを考える:

文字列"XML"を含み, 1997-12-17 に更新した<Paragraph>を含む<Section>を求めなさい.

Structure Index

索引の葉と, 葉から根に向かう経路の途中の節を検索するので, Structure Index の検索回数は文書の階層構造の高さに依存する. 文書の階層構造の高さを h とすると, 検索コストは $O(h)$ である.

Content Index

Word Inverted Index から XML を, Element Inverted Index から属性<Paragraph>, <Section>を, Element Value Inverted Index から属性値 1997-12-17 を検索する. Content Index をハッシュ法で実現した場合, 検索コストは $O(1)$ で

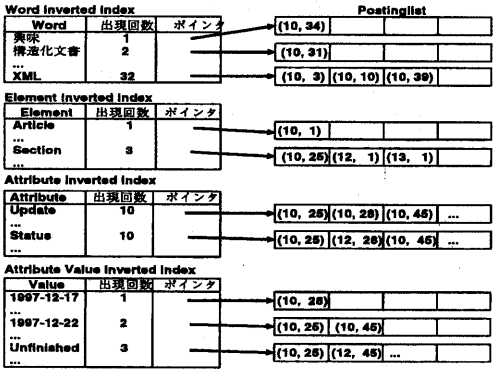

図 7 Content Index

```
...
<Document>
<Section Update="1997-12-22"
Status="Unfinished">
<Title>はじめに</Title>

<Paragraph Update="1997-12-17"
Status="Finished">
私は<Keyword>構造化文書</Keyword>に
<Word>興味</Word>を持ち，特に
<Keyword>XML</Keyword>
<Keyword>Hytime</Keyword>          <- 挿入
の
<Word>調査</Word>を続けている．
</Paragraph>
...
```

図 8 更新例

ある．

この場合，提案する索引の検索コストは一般的に $O(h)$ となる．

4. 関連研究

本章では，これまで構造化文書の索引に用いられてきた二つの手法を紹介する．

Ron Sacks-Davis らによると，構造化文書の索引は Position-Based Index と Path-Based Index に分類可能である [SDDTZ97]．これら二つの索引の特徴を述べる．

4.1 Position-Based Index

文書中に出現する部分文字列の開始位置と長さ (または終了位置) を管理する索引を Position-Based Index という．たとえば文書の先頭から 100 文字目に文字列"SGML"(4 文字) が出現する場合，

(100, 4)

と表現できる．

4.1.1 リージョンモデル

先に示した (開始位置, 終了位置) は文書内の連続的な一部分なので，リージョン (region) であるという．

リージョンを互いに overlap しないように定義した場合，リージョンを階層的に指定することが可能である．XML 文書インスタンスにも同様の考え方を導入することにより，XML 文書インスタンス中の連続な一部分をリージョンとして扱うことが可能になる．

リージョンモデルとして次のものがある．

i) Generalized Concordance Lists(GC-lists) [CCB95a] [CCB95b]

Concordance Lists [Bur92](contiguous extents(文書中の連続した部分文字列) ごとに (開始位置, 終了位置) の二つ組で表現した表) を基本とする．contiguous extents の overlap は許すが，nest を許さないモデルである．

ii) Simple Concordance Lists(SC-lists)[DSDT97]

上で示した GC-lists を基本にし，contiguous extents の nest を許したモデルである．

したがって XML 文書インスタンスを考える場合，nest を許す SC-lists リージョンモデルを適用するのが良いと考える．

4.1.2 索引構造

Position-based Index の索引構造は，文字列と element の辞書とそれらの転置索引から構成される．

i) Text Index … 文字列の辞書と文字列の転置索引からなる．

ii) Element Index … element の辞書と element の転置索引からなる．

4.2 Path-Based Index

文書の階層構造を利用して，文書中に出現する部分文字列への経路式を管理する索引を Path-Based Index という．例えば `Section 1` の 1 番目の `Paragraph` の 7 `Word` 目に文字列"`SGML`"が出現した場合の経路式は，

`Section 1, Paragraph 1, Word 7`

と表現できる．

4.2.1 索引構造

Path-Based Index は (文字列, 経路式) の組で構成され，実装される．経路式をそのまま格納すると索引が巨大になる．経路式の格納効率を上げるために element 名を省略して同一 element 内における出現番号を格納する手法や，直前の経路式との差分をとる手法などが用いられる．

4.3 Position-Based Index と Path-Based Index の比較

Position-Based Index と Path-Based Index を問合せ能力，更新コスト，問合せコストの面から比較する．

4.3.1 問合せ能力の比較

Position-Based Index と Path-Based Index の問合せ能力を，文書の階層構造 (structure) と文字列 (content)，属性 (attribute) を対象とする問合せの面から二つの索引を比較する．Ron Sacks-Davis らによると，両方の索引は属性に関する問合せを考慮していないと指摘している [SDDTZ97]．

Position-Based Index

文書の階層構造を対象とする問合せの場合, Position-Based Index 自身から直接求めることはできない. しかし GC-lists や SC-lists リージョンモデルを適用することにより問合せ結果を間接的に求めることが可能である.

文字列の出現位置を対象とする問合せの場合, 索引から直接求めることが可能である.

Path-Based Index

次のような文字列の出現位置に関する問合せを考える:

文字列"XML"の 100 文字以内に文字列"SGML"が出現する<Section>を見つけなさい.

文字列"XML"と文字列"SGML"が Path-Based Index の同じ葉に出現する場合の問合せは可能であるが, 文字列がそれぞれ別な葉に出現する場合の問合せはできない. したがって Path-Based Index の複数の葉をまたぐ近接問合せは不可能である.

文書の階層構造に関する問合せの場合, 索引から直接求めることができる.

したがって Path-Based Index の問合せ能力より, Position-Based Index の問合せ能力のほうが高いことがいえる.

4.3.2 更新コストの比較

Position-Based Index と Path-Based Index の更新コストを比較する.

3.3.2 節の更新例 (図 8) を考える.

Position-Based Index

Position-Based Index は, 変更した文字列の出現位置以降の索引をすべて変更しなければならない. 索引づけされた連続した文字列の数を r とするとコストは r に依存するので, コストは $O(r)$ である.

Path-Based Index

Path-Based Index は葉と葉から根に向かい節を更新するので, 文書の階層構造の高さ h に依存する. コストは $O(h)$ である.

一般的な構造化文書を考える. 文書の階層構造の高さ h と索引づけされている部分文字列の数 r を比較した場合, 相対的に h が小さい. したがって, Path-Based Index の更新コストが Position-Based Index の更新コストより相対的に小さいことがいえる.

4.3.3 問合せコストの比較

Position-Based Index

Position-Based Index をハッシュ法で実現した場合, 問合せコストは $O(1)$ である.

Path-Based Index

Path-Based Index の場合, 文書の階層構造の高さ h に依存するために問合せコストは $O(h)$ である. したがって, Path-Based Index の問合せコストは Position-Based Index の問合せコスト以上であることがいえる.

表 1 索引の比較

	Position-Based Index	Path-Based Index	提案する索引
問合せ能力			
・文字列	○	△[3]	○
・構造	○[4]	○	○
・属性	−	−	○
更新コスト	$O(r)$	$O(h)$	$O(h)$
問合せコスト	$O(1)$	$O(h)$	$O(h)$

r: 索引づけされた文字列の数.
h: 索引の階層構造の高さ.

5. 既存の索引手法と提案した索引手法との比較

第 3 章で提案した索引手法と, 第 4 章で紹介した既存の索引手法 (Position-Based Index と Path-Based Index) を比較する (表 1).

Content Index と Position-Based Index はハッシュ法を用いて比較した.

既存の索引手法は属性に関する問合せを考慮していない. Path-Based Index は, 索引の複数の葉をまたぐ近接問合せは不可能である. Position-Based Index は索引の更新コストが高い.

今回提案する索引手法は, Content Index を利用することにより属性に関する問合せに対応している. 更新コストや問合せコストも $O(h)$(h: Structure Index の階層構造の高さ) であり, これらのコストは低いといえる.

したがって提案した索引手法は, 問合せ能力, 更新コスト, 問合せコストのすべての面で既存の索引手法と同等か, 優れているといえる.

6. 実装手法

本章では第 3 章で提案した索引のアーキテクチャを示し, 実装手法を議論する.

6.1 実験システムのアーキテクチャ

実験システムのアーキテクチャは, 図 9の通りである.

Index Generator は XML 文書インスタンスの構文解析, Structure Index と Content Index の生成機能を持つ. 構文解析された XML 文書インスタンスや Structure Index, Content Index はリポジトリに格納する.

Query Processor & Index Controller は, XML 文書インスタンスや索引の更新, XML 文書インスタンスへの問合せ機能を持つ. XML 文書インスタンスの更新や問合せはすべて Query Processor& Index Controller を通して行われる.

Index Generator や Query Processor & Index Controller 中の構文解析部に関しては LT XML ツールキット [MTT+97] を利用して実装することを考えている. 索引や XML 文書インスタンスの格納や更新に関して, トランザクションや永続性といったデータベース特有の機能を持つ必要がある. これら

[3] 索引の複数の葉をまたぐ近接問合せは不可能.

[4] GC-lists モデルや SC-lists モデル等を利用することにより間接的に求めることが可能.

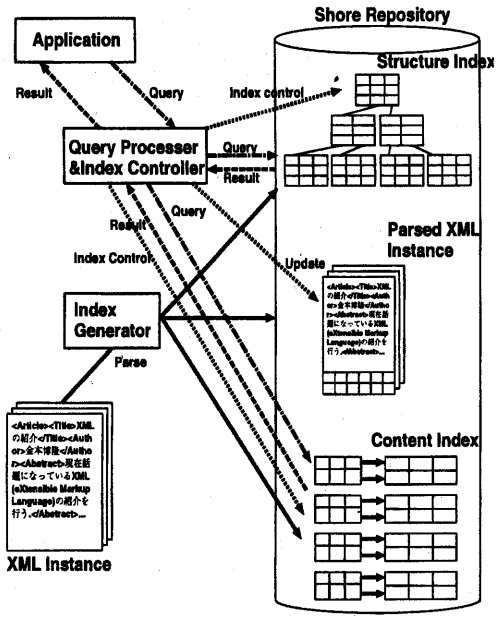

図 9 実験システムのアーキテクチャ

```
/* element を格納する構造体 */
struct SIDX_Element {
    char *label; /* element 名 */
    int number;  /* element に含まれる文字数 */
    int pageno;  /* ページ番号 */
    int slotno;  /* スロット番号 */
    SIDX_Attribute *first;
            /* SIDX_Attribute へのポインタ */
    SIDX_Element *nextelement;
            /* 次の element へのポインタ */
    SIDX_Element *parent;
            /* 親 element へのポインタ */
    SIDX_Element *child;
            /* 子 element へのポインタ */
};

/* 属性を格納する構造体 */
struct SIDX_Attribute {
    char *name;  /* 属性名 */
    char *value; /* 属性値 */
    SIDX_Attribute *next;
          /* 次の SIDX_Attribute へのポインタ */
};
```

図 10 Structure Index 実現のための構造体

の要件を満たすオブジェクト指向リポジトリである Shore[Uni96] を利用して実装することを考えている.

6.2 設計

前節で示したアーキテクチャを実現するため, 本節において Structure Index, Content Index を実現するための構造体を設計する. 構文解析した XML 文書インスタンスは, 図 6 を表現する構造体に格納される.

6.2.1 Structure Index

Structure Index の element を格納する構造体として `SIDX_Element` と, 属性と属性値を格納する構造体として `SIDX_Attribute`(図 10) を定義する.

element 名は `label` に格納する. 第 3 章で述べたように, element で囲まれていない文字列の場合の element 名には NULL を格納する. element で囲まれる文字数は `number` に格納する. element で囲まれている文字列の場合はその開始 element の先頭を, element で囲まれていない文字列の場合はその文字列の先頭を, ページ番号 `pageno` とスロット番号 `slotno` を用いて指す. 同じ階層に element が複数存在する場合, `nextelement` を用いて次に出現する element を指す. 検索や更新のため, 親 element を指す `parent`, 子 element を指す `child` ポインタを持つ. element の持つ属性と属性値は `SIDX_Attribute` へのポインタを持つことにより管理できる. 属性と属性値はそれぞれ `name`, `value` に格納する. 属性や属性値を複数個持つ場合は, `next` を用いて次の `SIDX_Attribute` を指す.

6.2.2 Content Index

Content Index を格納する構造体として辞書 `CIDX_Dictionary` と posting list `CIDX_Postinglist`(図 11) を定義する.

辞書 `CIDX_Dictionary` 中の `string` には Word Inverted Index の場合は XML 文書インスタンス中の部分文字列を, Element Inverted Index の場合は element 名を, Attribute Inverted Index の場合は属性名を, Attribure Value Inverted Index の場合は属性値を格納する. `no` には `string` の出現回数を格納する. `string` の出現位置は `CIDX_Postinglist` 構造体に格納し, `CIDX_Dictionary` から `first` ポインタにより指すことにする. 出現位置はページ番号と Slot Dictionary のスロット番号 `pageno`, `slotno` に格納する. 出現位置が複数個存在する場合は `next` を用いて次の `CIDX_Postinglist` 構造体を指す.

6.3 索引の格納方針

本節では提案した二つの索引の格納方針を示す. 索引の検索コストや更新コストを押えるためにディスクのページ内部の構造体の配置を意識する必要があるからである.

ディスクの 1 ページは数キロバイト程度であるという仮定に基づいて議論する. また, 近年のディスク価格の低下により, 索引の大きさは無視してよいという立場をとる.

6.3.1 Structure Index

同一階層に存在する element は同一ページで管理することを基本方針とする. 図 5 について索引の根に注目した場合, `<Title>`, `<Author>`, `<Abstruct>`, `<Document>` が Structure Index において同一階層であるという.

Structure Index の検索や更新は, 同一階層を一つの単位として行われるので同一ページで格納すれば索引の格納されているディスクへのアクセスを減らすことができる. ディスクの 1 ページは数キロバイト程度であるならば, ほとんどの場合, Structure Index の同一階層を 1 ページで格納できるものと考える.

XML 文書インスタンスに依存するが, Structure

```
/* Dictionary 構造体 */
struct CIDX_Dictionary {
    char *string;
    /* XML 文書インスタンスの部分文字
     列または element 名または属性名ま
     たは属性値 */
    int no; /* string の出現回数 */
    SIDX_Postinglist *first;
      /* posting file へのポインタ */
};

/* Postinglist 構造体 */
struct CIDX_Postinglist
    int pageno;  /* ページ番号 */
    int slotno;  /* スロット番号 */
    CIDX_Postinglist next;
  /* 次の Postinglist へのポインタ */
};
```

図 11 Content Index 実現のための構造体

Index は XML 文書インスタンスと比較して相対的にデータ量が小さいため, 索引全体を一つのページで格納できる場合も考えられる.

6.3.2 Content Index

同一 element 名や属性, 属性値, 部分文字列の出現位置を管理する Postinglist は, 同一ページで管理することを基本方針とする.

Content Index の検索や更新は, それぞれの Inverted Index からポインタで指されている Postinglist を単位として行われる. 同じ文字列や属性などの出現位置を同一ページに格納すれば, ディスクへのアクセスを減らすことができる.

7. まとめと今後の課題

本稿では, 効率的な更新が可能な構造化文書の索引を提案した. 提案する索引の構造から効率的な更新が可能であることを述べた. また, 構造化文書へのすべての種類の問合せ (文書の階層構造, 文字列, 属性に関する問合せの三種類) に対応している.

本稿で提案した索引の実現のために構造体を設計し, 実装手法についても議論した.

今後の課題として, 構文解析した XML 文書インスタンスを格納する構造体を設計したり, 大量の XML 文書インスタンスに適するように索引手法を改良したりする必要があることが挙げられる.

今後の予定として, 第 6 章の議論に基づいて実験システムを実装し, 評価を行いたい.

謝辞

貴重で有益な意見や助言を与えて頂いたマルチメディア統合システム講座の皆様に厚くお礼を申し上げます.

参考文献

[Bur92] Forbes J. Burkowski. An algebra for hierarchically organized text-dominated databases. *Information Processing & Management*, Vol. 28, No. 3, pp. 333–348, 1992.

[CCB95a] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *Computer Journal*, Vol. 38, No. 1, pp. 43–56, 1995.

[CCB95b] C. Clarke, G. Cormack, and F. Burkowski. An algebra for structured text search and a flamework for its implementation. *The Computer Journal*, 1995.

[DSDT97] Tuong Dao, Ron Sacks-Davis, and James A. Thom. An indexing scheme for structured documents and its implementation. In *Proc. of the 5th International Conference on Database Systems for Advanced Applications (DASFAA'97)*, April 1997.

[ISO86] ISO 8879: 1986. *Information Processing - Text and Office System - Standard Generalized Markup Language (SGML)*, Oct. 15 1986.

[JIS92] JIS X 4151: 文書記述言語 SGML (Standard Generalized Markup Language), 日本規格協会, 1992.

[MTT+97] David McKelvie, Henry Thompson, Richard Tobin, Chris Brew, and Andrei Mikheev. *The XML Library LT XML version 0.9.5*. Language Technology Group, Human Communication Research Centre, University of Edinburgh, September 1997. http://www.ltg.ed.ac.uk/.

[SDDTZ97] Ron Sacks-Davis, Tuong Dao, James A. Thom, and Justin Zobel. Indexing documents for queries on structure, content and attributes. In *International Symposium on Digital Media Information Base*, Nov. 1997.

[Uni96] University of Wisconsin. Shore project home page. http://www.cs.wisc.edu/shore/, Mar 1996.

[Wor97] World Wide Web Consortium. Extensible markup language 1.0 (xml 1.0). Proposed Recommendation, http://www.w3.org/TR/PR-xml-971208, 1997.