バイナリデータの XML ビューにおける DOM 更新サポートの検討

品川　徳秀†　　北川　博之††

† 千葉大学 環境リモートセンシング研究センター　〒 263–8522 千葉県千葉市稲毛区弥生町 1–33
†† 筑波大学 電子・情報工学系　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †siena@faculty.chiba-u.jp, ††kitagawa@is.tsukuba.ac.jp

あらまし　従来のアプリケーションではバイナリデータが主に扱われてきた．一方，異種データ統合利用を XML ベースで行なうアプローチがあり，バイナリデータをこの枠組みで利用可能にする事は，その応用範囲を広げる事に繋がり有用である．これまで，バイナリデータの動的・部分的な XML ビューをオフセット計算によって実現する方式について検討を行なってきた．この XML ビューにおいて，DOM API を通じてバイナリデータの参照を可能にする方式，XPath 問合せ処理の効率化手法について考察を行なってきた．本稿では更に，DOM API のサポート範囲を拡張し，DOM オブジェクトに対するデータ更新を実現するための機構について論じる．また，部分的に実装を行なってきており，この実装において実験を通じて確認された実行特性について併せて報告する．

キーワード　XML ビュー, バイナリデータ, XML データ統合, XPath 問合せ処理

Supporting Updates for XML Views over Binady Data

Norihide SHINAGAWA† and Hiroyuki KITAGAWA††

† Center for Environmental Remote Sensing, Chiba University
1–33 Yayoi-cho, Inage-ku, Chiba, 263-8522 Japan
†† Institute of Information Sciences and Electronics, University of Tsukuba
1–1–1 Tennodai, Tsukuba, Ibaraki, 305–8573 Japan
E-mail: †siena@faculty.chiba-u.jp, ††kitagawa@is.tsukuba.ac.jp

Abstract　Ordinary applications have mainly used application-specific binary data, and XML has been widely used for exchange, integration and management of data. When we can expand the scope of XML technologies to such binary data, they can be shared by many areas beyond the original application area. We have proposed a scheme to realize XML views over binary data which can be materialized dynamically and patrially based on offset calculation, and then, a read-only DOM API and XPath query optimization scheme for such XML views. This paper describes how to support updates in the DOM API. We have been implementing such DOM API. This paper also shows experimental results with the implementation.

Key words　XML View, Binary Data, XML-based Data Integration, XPath Query Processing

1. はじめに

近年，急速な XML 応用の展開と共に，各種サービスやアプリケーションで XML 対応が進んでいる．これらのサービスや XML データの活用では，XML によるデータ交換の効率化や XML 中心のデータ統合利用が重要な課題であり，様々な研究が行なわれている [1] [2] [3].

多くのソフトウェアでは，格納効率やアクセス効率，既存資産との互換性等の理由から，バイナリフォーマットを標準データ形式としている．これらのデータは，そのアプリケーションとそのデータ形式に個別に対応した周辺ソフトウェアでの利用が前提である．しかし，XML 中心の異種データ統合に見られるように，それらに限定されずにより広範囲なデータと併せて利用したいという要求がある．XML 中心のデータ統合では，各種データは XML データとして表現され，XML 問合せ・変換言語や XML 対応ライブラリ等で処理される．これらの内部では，DOM 等の共通化された API でアクセスが行なわれる．即ち，これらの API でアクセス可能であれば，必ずしも利用対象は物理的に XML データである必要はない．例えば，RDB に XML ビューを与え，DOM を用いて論理的な XML データとして扱う等の研究がある．

この統合利用の枠組みで任意のバイナリデータを利用可能に

する事は，その応用範囲の容易な拡大に繋がる．例えば，(1) デジタルカメラ等での画像リポジトリ管理に用いられる Exif フォーマットのバイナリデータから，画像一覧の XHTML 文書生成や，撮影条件に合う画像に他の XML 文書から選択した文章を付与を行なう，(2) ネットワーク機器管理プロトコル SNMP による収集データを機器障害履歴や既知の脆弱性リストの XML 文書等と組み合わせてレポート作成を行なう，(3) バイナリの PIM データを直接参照し，ウェブログに予定等を記載させる，等の応用が考えられる．また，設定ファイルがバイナリであるようなアプリケーションの周辺ツールの作成においても，XML ビューを用いる事でより手軽に扱え，XML の特性から本体のバージョンアップに伴うデータ構造の変更に影響されにくくなるという利点も想定される．

XML 中心のデータ統合での非 XML データの利用方式として，(A) 事前に XML 形式に変換する，(B) 利用時に DOM 等のデータ構造で主記憶上に一括展開する，(C) ビューを通じて各種 API でアクセスする，等がある．テキストファイルに対しては [4] 等があるが，バイナリデータについては考慮されていない．バイナリデータに対しては，一般のバイナリデータを扱う BinX [5] や，ASN.1 を XML に変換するもの等がある．これらは前二者の手法によっているが，現存のものは対応可能なデータ形式が限定されたり，汎用的でも対象データを一括変換するために必要以上に資源を消費する事があったりする．特に (A) の手法には，同じデータが異なる表現で重複し，使用の手間やデータの同期，格納容量の増加等の問題がある．これらは，アクセス時に必要な部分のみを変換する (C) の方法で解決できる．バイナリデータについては，前述のもの等 (A), (B) の方式によるものが存在するが，(C) のアプローチによるものは知るところ存在しない．また，XML のバイナリ表現の検討も行なわれているが [6]，本研究の目的は既存データの活用であり，XML のコンパクトな表現ではない．

本研究では，以上の観点からバイナリデータに対する XML ビューの検討を行なってきた．内部構造に半構造性を持つバイナリデータを対象とするが，問題の複雑化を避けるため，ひとまずワープロやプレゼンツール等の表現力が高く複雑なデータは対象外とする．提案手法では，バイナリデータのフォーマットは既存の XML スキーマ言語をベースとしたフォーマット定義言語で与え，これに従ってデータ格納場所のオフセットを計算し，XML ビューに部分的・動的に対応付ける．ファイル中の位置情報を用いるという点でインデックスを用いたデータアクセスに似ているが，インデックスを生成する事なく，フォーマット定義と若干のデータアクセスで実現できる．

具体的な API として DOM のインタフェースを提供する．これまでの提案ではデータ参照のみのサポートに限られていたが，本稿では更に更新操作のサポートについて論じる．これを用いる事で，既存ソフトウェアでもデータローダの置き換えでバイナリデータを XML データとして利用可能となる．XML 表現は冗長度が高くなる傾向があり，一括変換では過剰に記憶空間を必要とする可能性がある．これを避けるため，データアクセス時に必要な部分に限定した動的・部分的なノードの実体化をサポートし，これに基づいた XPath 問合せ処理の効率化を説明する．動的・部分的な実体化に関しては XML DBMS や PDOM [8] をはじめとした各種 Persistent DOM に関する多くの提案があるが，本提案では予めパースした結果を保存したりする必要はなく，バイナリデータに直接アクセスする．

また，提案手法に基づく DOM API の実装を部分的に進めてきており，この実装において実験を通じて確認された実行特性について併せて報告する．

2. XML ビュー定義

2.1 バイナリデータの例

説明上，簡単なバイナリデータの例として BMP 画像を扱う．画像情報ヘッダとパレットの構造が異なる Windows 版と OS/2 版が存在し，画像情報ヘッダの大きさで判別される．パレットの色数と画像の大きさは画像情報ヘッダに記述され，これによりフィールドの繰返し回数が変化する．このように，BMP ファイルは部分構造の選択や繰返しといった半構造性を持つ．より複雑なデータフォーマットでも一般に，バイナリデータでの半構造性は選択基準や繰返し回数を特定のフィールドに格納しておく事で与えられる．これらを**構造決定情報**と呼ぶ．データ構造の決定には，既知のデータフォーマットに加え，実データ中の構造決定情報が必要となる．

2.2 バイナリフォーマット定義

バイナリデータのフォーマット定義言語で XML ビューを定義する．本稿ではバイナリデータの構造に直接的に対応する XML 構造を与え，構造変換や導出値の利用等は範囲外とする．

a） フォーマット定義概要

BMP 画像に対するフォーマット定義の例を図 1 に示す．定義言語は XML スキーマ言語 RELAX NG [7] をベースとしているが，一部の語彙を変更してある．例えば，`format`, `record`, `field` は，RELAX NG の `grammer`, `define`, `element` にほぼ対応する．

`format` は定義の最外郭要素である．`start` はデータフォーマットの最外郭の構造を，`record` は定義中で参照可能な名前付き部分構造を定める．`field`, `attribute` は，それぞれ XML ビュー中の `name` で与えられる名を持つ要素，属性に対応付ける．省略された場合は XML ビュー上には出現しないため，語境界の調整やダミーフィールドを表現できる．`ref` は名前付き部分構造を参照する．

図 1 において，全体 `bitmap` の概構造は `header`, `infosize`, 後に述べる選択可能な部分構造, `image` からなり，更に細部の構造を持っている．

b） データ型の指定

`bintype` 属性はバイナリデータ型を，
`xmltype` 属性はその文字列表現のデータ型を与える．使用可能なデータ型は RELAX NG と同様に別途定義され，`bintypeLibrary`，`xmltypeLibrary` 属性で導入される．データ型ライブラリはその URI で識別される．`bintype` は一般にはバイナリデータを作成したプラットフォームに依存し，バイトオーダの差異等はこれに含まれる．`xmltype` には XML

```
<format xmlns="http://.../bmp"
        bintypeLibrary="http://.../ia32"
        xmltypeLibrary=
         "http://www.w3.org/2001/XMLSchema-datatypes">
  <typedef type="long" bintype="long" xmltype="int"/>
  <!-- 他の型定義は省略 -->

  <start>
    <field name="bitmap">
      <ref name="header.def"/>
      <field name="infosize" type="ulong"/>
      <choice>  <!-- 情報区画サイズで選択 -->
        <when test="../infosize[1]=40">
          <ref name="bmpinfo-win.def"/>
          <ref name="palette-win.def"/>
        </when>
        <when test="../infosize[1]=12">
          <ref name="bmpinfo-os2.def"/>
          <ref name="palette-os2.def"/>
        </when>
      </choice>
      <ref name="image.def"/>
    </field>
  </start>

  <record name="header.def">
    <field name="header">
      <field name="filetype" type="uint"/>
      <field name="filesize" type="ulong"/>
      <field type="int"/>   <!-- 未使用 -->
      <field type="int"/>   <!-- 未使用 -->
      <field name="img-pos"  type="ulong"/>
    </field>
  </record>

  <record name="bmpinfo-win.def">
    <field name="bmpinfo">
      <field name="img-width"  type="long"/>
      <field name="img-height" type="long"/>
      <!-- 中略 -->
      <field name="color-num" type="ulong"/>
      <field name="important" type="ulong"/>
    </field>
  </record>

  <record name="palette-win.def">
    <field name="palette">
      <repetition number="preceding::color-num[1]">
        <field name="color">
          <field name="red"   type="uchar"/>
          <field name="green" type="uchar"/>
          <field name="blue"  type="uchar"/>
          <field              type="uchar"/>
        </field>
      </repetition>
    </field>
  </record>

  <!-- record name="bmpinfo-os2.def"
       record name="palette-os2.def"
       record name="image.def"  は省略 -->
</format>
```

図 1　フォーマット定義例 (BMP)

Fig. 1　An Example of the Format Definition (BMP)

```
<!-- (a) value による選択 -->
<choice>
  <field name="sub">
    <field name="ver" type="...">
        <value>1</value> </field>
    <ref name="sub-1"/>
  </field>
  <field name="sub">
    <field name="ver" type="...">
        <value>2</value> </field>
    <ref name="sub-2"/>
  </field>
</choice>
<!-- (b) when による選択 -->
<field name="sub">
  <field name="ver" type="..."/>
  <choice>
    <when test="preceding::ver[1]=1">
        <ref name="sub-1"/> </when>
    <when test="preceding::ver[1]=2">
        <ref name="sub-2"/> </when>
  </choice>
</field>
```

図 2　選択定義の例

Fig. 2　Examples of Choices

Schema Part 2: Datatypes[9] 等が使用される．使用可能なデータ型ライブラリは処理系に依存し，相互変換できない bintype と xmltype の指定はエラーとする．便宜のため，typedef で bintype と xmltype の組を定義し，type 属性で参照できる．

図 1 では，bintypeLibrary に IA32 のものを，xmltypeLibrary に XML Schema Part 2: Datatypes を用い，long 等のデータ型の組を定義して field 定義に使用する．

c) 構造選択

選択構造は RELAX NG と同様，value による値制約を持つ構造を choice に列挙して定義する．また，より一般の条件記述のため，choice 直下に XSLT と同様の when, otherwise を許す．その test 属性に XPath 式が記述されるが，参照できるノードはその箇所よりも先頭方向のノードのみとする．

図 2 は，sub の内部構造を ver の値で sub-1，sub-2 から選択する．図 1 では，プラットフォーム別の同名の異なる構造 bmpinfo, palette を持つが，infosize の大きさで判別して選択する例である．

d) 繰返し構造

繰返し構造は，終端の判断に構造決定情報を必要とする．このため，RELAX NG とは異なり，repetition 内に繰り返す構造を記述し，number 属性に XPath 式で回数，もしくは until 属性に終端条件を明示的に指定する（図 3）．

図 1 では，palette 中の color が繰返し構造を取り，その回数を color-num で決定している(注1)．

本稿ではこれ以上の詳細は述べないが，外部モジュール化されたフォーマット定義の利用や，XML 名前空間の扱い，文法の入れ子等については RELAX NG のそれに準じる．

(注1)：実際にはピクセルの色深度に応じて決定される事もある．

```
<!-- (a) 固定回数の繰返し -->
<repetition number="3">
  <ref name="sub"/>
</repetition>

<!-- (b) XPath 式での繰返し回数指定 -->
<field name="times" type="...">
<repetition number="preceding::times[1]">
  <ref name="sub"/>
</repetition>

<!-- (c) 終端指定による不定回数の繰返し -->
<repetition until="point/x = -1 and point/y = -1">
  <field name="point">
    <field name="x" type="..."/>
    <field name="y" type="..."/>
  </field>
</repetition>
```

図 3　繰返し定義の例
Fig. 3　Examples of repetitions.

3. XML ビュー構築

本節では，バイナリデータと XML データの物理的な対応付け方法を述べる．始めにバイナリデータの参照に必要な機構を説明し，その後，データ更新のための機構について説明する．

3.1 アクセス API

フォーマット定義による階層構造は XML ビューでの階層構造に対応付けられ，`format`，`field`，`attribute` に対して DOM の `Document`，`Element`，`Attribute` ノードが生成される．値は `xmltype` による文字列表現となる．また，バイナリ表現のままアクセス可能にするための `Node.getNativeValue()` メソッド等，幾つかの拡張機能を提供する．

XML データ統合利用環境では主に通常の XML データを利用するため，本機構は既存の XML プロセッサと同様に使用できる必要がある．本機構ではフォーマット定義が与えられたバイナリデータでない場合に既存の XML プロセッサに処理を委譲する事で，これを実現する．例えば Java でならば，JAXP の DOM パーサに本機構を用いれば良い．

フォーマット定義とバイナリデータの対応付けは，外部の設定ファイルによる．例えば，ファイル拡張子やバイナリデータ先頭に含まれる特定のシーケンスの出現 (注2) に応じてフォーマット定義を対応付ける等である．

3.2 オフセット計算

DOM オブジェクト生成時に全データを主記憶上に展開するのは必ずしも得策ではない．DOM ノードは親・子・兄弟要素や属性へのポインタ，ノード名・内容テキスト等から構成され，時として実ファイルサイズの数十倍の記憶領域が必要になる．バイナリデータではフィールド長は変動要因がなければ固定長であり，フォーマット定義のみから対象フィールドのバイナリデータ中のオフセットを計算可能である．また，変動要因を含んでいても構造決定情報を併せて参照する事で計算可能である．本機構では，この特徴を用いて動的・部分的なノードの実体化を実現する．尚，明示的な構造決定情報がないナル終端文字列や図 3 のような終端指定による不定回数の繰返しが含まれる場合には，実際にデータを走査する必要がある．

あるフィールド f_i について，その親フィールドを $p = parent(f_i)$，同じ親を持ちファイル先頭方向に存在する先行フィールドを $preceding(f_i) =< f_1, f_1, ..., f_{i-1} >$ と表記し，p の絶対オフセット $position(p)$ は既知とする．f_i の長さ $length(f_i)$ は，フォーマット定義から既知，もしくは事前に構造決定情報を読む事で確定される．p 先頭からの f_i の相対オフセットは明らかに $offset(f_i) = \sum_{j=1}^{i-1} length(f_j)$，絶対オフセットは $position(f_i) = position(p) + offset(f_i)$ である．ここで，どの f_j についても同様に，$offset(f_j), length(f_j)$ は $offset(f_i)$ 計算前に確定可能である．また，構造決定情報が XPath 式で与えられている場合も，2.2 節にあるように指定できるノードは先頭方向のもののみであり，これらノードの変動要因も同様に予め確定可能である．

3.3 ノード管理

ノードは，確定された絶対オフセットを起点として，そのノードのフォーマット定義に従って構築される．この時点では，子孫ノードや属性ノードの構築は行なわず，それらへのアクセスが発生した時に同様にノードを構築する．これにより，不要なノードの実体化処理を抑制する．

また，ノードが既に主記憶上に構築されている場合にそれを直接得られるようにするため，$< Offset, Length, Node >$ をスキーマとする**実体ノードテーブル**に登録しておく．

割り当てられた記憶領域を超えて使用しようとした場合，構造決定情報となるノードは値が繰返し参照される場合に備えて主記憶上に残し，それ以外の使用されなくなったノードの部分木を優先して破棄する．更に，オフセットはデータ構造が変更されない限り一定であり，また，その構造決定情報は繰返し参照されなければ不要と見做せるため，構造決定情報となるノードについても過剰に増加した場合には破棄する．

頻繁なノードの実体化・破棄は実行効率を低下させる要因となる．例えば，広範なフィールドへランダムアクセスを繰り返すような状況に発生する．これは十分な記憶領域を使用して破棄頻度を減らす事で軽減できる．適切な割り当て量は応用に依存するため，利用者が設定可能なパラメータとする．将来的には使用状況に応じた自動調整を検討する．また，実装上の考慮点として，オブジェクトプーリングやバイナリデータのバッファリングを行なう事が望ましい．

3.4 データ更新

データ更新では，次の点を考慮する必要がある．

- DOM でのデータ更新はパース後の DOM オブジェクトの更新であり，パース前の XML データを書き換えない．
- 構造決定情報やそれに依存する部分構造に対する更新によってフォーマット定義と矛盾する状態が発生しうる．
- 構造の変更に伴い，フォーマット定義に従わない構造になったり，バイナリ表現でのデータ長が変化したりする．

これらから，更新された部分は元データとは別に管理，参照

(注2)：ファイル先頭の文字列の例: BMP の “BM”，GIF の “GIF89a”

する．本稿では簡単化のために，更新されたデータは主記憶に保持するものとするが，一時ファイルにマッピングする事も可能である．

3.4.1 値の更新

フィールド値，即ち `Text` ノードの値を更新する場合，そのノードに対して必要な更新に加え，$< Offset, NewValue, DataType, OldValue >$ をスキーマとする**値更新テーブル**に登録する．ここで，$Offset$, $NewValue$ のみが必須である．値更新テーブルは，バイナリデータ参照時のフックとして機能する．

拡張 API として，新しい値と共にそのデータ型を指定できるメソッドを提供する．これは新しいデータ型を $DataType$ に設定する．従来の API を使用する場合は，$DataType$ は $null$ とし，フォーマット定義で指定されたデータ型であるものとして扱う．新しい値として与えられた文字列がそのデータ型に変換可能であれば，そのバイナリ表現が $NewValue$ に設定される．変換不可能な場合には，$DataType$ には “$string$” が，$NewValue$ には与えられた文字列そのものが設定される．更に，そのノードの値が構造決定情報である場合に限り，元ファイル中での値を $OldValue$ に設定する．

データ参照時は，このノードが主記憶上から破棄されている場合には，再構築が行なわれる．その際，3.2 節のオフセット計算を行ない，そのオフセットに対応する更新が値更新テーブルに登録されていなければ，従来通りファイルから読み出す．値更新テーブルに登録されている場合，通常の API で値を参照しているならば $NewValue$ に基づいてノード生成が行なわれる．また，オフセット計算中に構造決定情報として参照しているならば，$OldValue$ を返す．尚，オフセット計算時には，$DataType$ の如何にかかわらず，フォーマット定義で指定されたデータ型に従ってデータ長を返す．

3.4.2 構造の変更

ノードの追加・削除等で構造が変更された場合，それ以後は，それらのノードを破棄の対象としないようにマークする．

構造が変化した部分以降のノードは依然として破棄の対象となりうるが，変化部分のデータ長が変化するため，3.2 節のオフセット計算をそのまま適用できなくなる．これを解決するため，変化部分と無変化部分を混在して子に持つノードは，無変化部分の先頭ノードそれぞれについて，元のファイル中でのオフセットを保持する．同時に，構造変更前の自身のデータ長を保持し，そのノード以降のオフセット計算にはこれを提供する．これらによって，構造変更後も無変化部分のオフセット計算を可能とする．

3.5 更新のコミット

全ての更新部分が主記憶に保持されるため，データの更新が多く行なわれると主記憶を浪費してしまう可能性がある．そこで，拡張 API として，更新のコミットをサポートする．コミットを行なう場合には，更新内容とフォーマット定義が矛盾しない事を要件とする．これらが矛盾するにもかかわらずコミットを行なった時の結果は未定義とする．但し，実装によっては，明らかに矛盾している事が分かる場合には，例外を発生させる事もありえる．

コミット時には，主記憶に保持されたノード群をバイナリデータに逆変換し，一時ファイルに書き出す．更に，これらのノードから破棄不能マークを消去し，主記憶使用量を勘案して必要ならばノードの破棄を行なう．また，併せて値更新テーブルの $NewValue$ も同ファイルに書き出され，$DataType$ が与えられている場合はそれに従ってフォーマット定義を書き換える．その後，値更新テーブルをクリアする．

コミット後のバイナリデータは，元ファイルと一時ファイルをマージした仮想的なストリームによって表される．このため，オフセット計算は仮想ストリームのものとなり，実ファイル中の物理アドレスへの変換が必要になる．これを $< LogOffset, Length, FileName, PhysOffset >$ をスキーマとする**アドレス変換テーブル**に保持し，参照する．

3.6 更新を考慮したノード取得手順

以上のデータ更新の管理を考慮すると，オフセット計算に基づくノードへのアクセスに対して，次のような手続きでノードが取得される．

（1） オフセット計算
（2） 実体ノードテーブルまたは値更新テーブルの参照
（3） アドレス変換テーブルの参照
（4） ノード構築

4. XPath 問合せ処理

3. 節では DOM API での XML ビュー探索の実現手法を述べた．より一般には，ノードの取得に XPath 問合せを併用する事も多く，XPath ライブラリを用いたソフトウェア開発だけでなく，XSLT や XQuery 等でも基本的な問合せ言語として利用される．また，DOM Level 3 でも XPath 対応が行なわれている．このため，XPath 問合せ処理の効率化は重要である．

DOM オブジェクトに対して一般の XPath 処理系を適用すると，単純にはノードを一つ一つ辿って行くという処理になる．これに対し，本提案の XML ビューでは，XPath 式の複数のステップをまとめてオフセット計算する事で途中のノードへのアクセスを省略できる場合がある．得られたオフセットを用いて 3.6 節の手順でノードが取得されるため，この簡略化はノードの更新状況によらず適用可能である．提案機構の DOM オブジェクトに対する XPath 問合せ処理について考察を行なう．

4.1 ノード実体化処理の埋込み

通常の DOM 実装では全 DOM ノードが主記憶上にある．しかし，本機構のような部分的な実体化を行なう場合，いずれのノードも任意の時点で主記憶上から破棄される可能性がある．このため，ノード間の移動を行なう際に起点ノードの実体化を保証しておかねばならない．コンテキストノードを起点として指定パスに適合するノード集合を実体化する処理を α とする．

パスを p，ステップを $s = x :: t[e]$（x, t, e は各々 s の軸，ノードテスト，修飾式）とする．また，$x :: t$ のフィールド定義が変動要因を含む時に XPath 式や `value` によって与えられる構造決定情報 r を明示するため，表記 $x :: t|_r$ を用いる．特に依存しない場合には r を空式とし，$x :: t|_r = x :: t$ と見做す．

この時，α の適用は式 (1), (2) で変形した上で処理される．

$$\begin{aligned}\alpha(\underline{s/p}) &= \alpha(\underline{x :: t|_r[e]/p}) \\ &\to \alpha(\alpha(\alpha(x :: t|_{\alpha(\underline{r})})[\alpha(\underline{e})])/\alpha(\underline{p}))\end{aligned} \tag{1}$$

$$\alpha(\underline{e}) \to \begin{cases} e & \underline{e}\text{が定数式} \\ \text{変形 (1) を適用} & \underline{e}\text{がパス式} \\ \alpha(op(\alpha(\underline{e_1}), \alpha(\underline{e_2}), ..., \alpha(\underline{e_n}))) & \\ \qquad \text{op が n 項演算子もしくは関数} & \end{cases} \tag{2}$$

更に，式 (1) の $\underline{r}, \underline{e}, \underline{p}$，式 (2) の $\underline{e_1}, ..., \underline{e_n}$ についても α を再帰的に適用する．

4.2 ノード実体化処理の簡略化

XPath 問合せ処理は 4.1 節の結果に従って行なわれるが，問合せ中の部分式がフォーマット定義から決定できる場合，即ち，構造決定情報の取得のためにバイナリデータ中の値を参照しない場合には，中間ノード集合を実体化する事なく適合ノードを直接実体化可能である．

ここで α を機能分割し，コンテキストノードを起点としてノード集合のオフセット計算を行なう λ と与ノード集合をオフセットに従って実体化する μ を用いて，$\alpha = \mu \circ \lambda$ とする．以下に，これらを削減可能な場合を検討する．

始めに，単ステップのパス式 $p = \alpha(\alpha(x :: t|_{\alpha(r)})[\alpha(e)])$ について考える．$\alpha(r)$ はこのステップの評価に先立って評価される必要があるので，他の部位とは独立に簡易化を行なえる．

$\alpha(e)$ は $x :: t|_r$ が与えるノード集合を起点としたオフセット計算が可能であれば評価可能であり，$x :: t|_r$ が実体化されていなくとも式 (3) 前半の簡易化が成り立つ．但し，e 中でコンテキストノードの値が参照される場合には，$\alpha(e)$ を展開した式の中で改めてコンテキストノードが実体化される機会が存在する．以下も同様に，部分式中での値参照によってノードが実体化される事があるが，煩雑になるため特に明記しない．更に，e がパス式ならば $[e]$ は適合ノードの存在性の確認であるため，少なくともこの場合は e によって選択されるノードを実体化する必要はない．最後に，$[e]$ は選択条件であり，これを評価する以前に $x :: t|_r$ に適合するノード集合はオフセット計算済みなので，全体にかかる α は μ で良い．

$$\begin{aligned}\alpha(p) &\to \alpha(\alpha(x :: t|_{\alpha(r)})[\alpha(e)]) \\ &\to \alpha(\mu(\lambda(x :: t|_{\alpha(r)})[\alpha(e)])) \\ &\to \begin{cases} \mu(\lambda(x :: t|_{\alpha(r)})[\lambda(e)]) & e\text{がパス式} \\ \mu(\lambda(x :: t|_{\alpha(r)})[\alpha(e)]) & \text{それ以外} \end{cases}\end{aligned} \tag{3}$$

多段のステップ $s_i (i = 1, ..., n)$ からなるパス式 $p = s_1/p_1 = s_1/s_2/.../s_n$ では，p_i は s_i の適合ノードを起点としたオフセット計算が可能であれば十分である．

$$\begin{aligned}\alpha(p) &\to \alpha(\alpha(s_1)/\alpha(p_1)) \\ &\to \alpha(\mu(\lambda(s_1)/\alpha(p_1))) \\ &\vdots \\ &\to \mu(\lambda(s_1)/\lambda(s_2)/.../\lambda(s_n))\end{aligned} \tag{4}$$

n 項演算式 $\alpha(p) = \alpha(op(\alpha(p_1), \alpha(p_2), ..., \alpha(p_n)))$ について考える．各 p_i について，その選択ノードの値を op が参照しない場合，$\alpha(p_i) \to \lambda(p_i)$ と書き換えられる．更に，p 全体の結果が数値や文字列になる場合にはそれ以上の実体化は不要であり，最外殻の α を削除できる．尚，op がどの値を参照するかについて事前の知識がない場合，問合せ解析時にはこの変換を適用できないが，遅延評価等の実装により，実行段階で実体化を省略できる場合がある事を指摘しておく．

以上から，多くの場合に μ を削除可能である事が分かる．これは，パス中間のノードの不必要な実体化を抑え，記憶領域消費の軽減に繋がると期待される．

4.3 XPath 問合せ処理例

例えば，赤味の強い色を多くパレットに持つ画像を探す手段の一つとして，RGB 値が $R > G + B$ となる色の数を求める XPath 問合せ p を考える．この時，上記の変換規則を用いると，式 (5) の変形が得られる．

$$\begin{aligned}p &= count(/bitmap/palette/color[red > green + blue]) \\ &\to count(/bitmap/palette|_{../infosize[1]}/ \\ &\qquad color|_{preceding::color-num[1]}[red > green + blue]) \\ &\to \alpha(count(\alpha(/\alpha(\alpha(bitmap)/ \\ &\qquad \alpha(\alpha(palette|_{\alpha(\alpha(..)/\alpha(infosize)[\alpha(1)])\alpha(1)])})/ \\ &\qquad\quad \alpha(\alpha(color|_{\alpha(\alpha(preceding::color-num)[\alpha(1)])\alpha(1)])}) \\ &\qquad\quad\quad [\alpha(\alpha(red) > \alpha(\alpha(green) + \alpha(blue)))])))))) \\ &\to count(/\lambda(bitmap)/\lambda(palette|_{\mu(\lambda(..)/\lambda(infosize)[1])})/ \\ &\qquad \lambda(color|_{\mu(\lambda(preceding::color-num)[1])} \\ &\qquad\quad [\alpha(red) > \alpha(green) + \alpha(blue)]))\end{aligned} \tag{5}$$

式 (5) から，実体化が必要なのは選択構造と繰返し回数を確定するための構造決定情報である `infosize`, `color-num`, ピクセル値の `red`, `green`, `blue` であり，それ以外のノードはオフセット計算によって読み飛ばせる事が分かる．

5. 実験による性能特性の確認

提案機構を検証するため，部分的に実装を行なってきた．本稿執筆時点では，データ更新については未実装であり，受理可能な XPath 問合せは主に使われる物のみが実装されている．ここでは，現実装におけるデータ参照について，DOM API での巡回と XPath 問合せ処理の実行特性について述べる．

5.1 使用データ

データセットとして，50KB，800KB，1.5MB 程度の BMP ファイルをそれぞれ 100 個ずつ用いた．フォーマット定義には，図 1 に示したものより詳細なものを使用した．XML ファイルに変換し，それを Xerces で DOM オブジェクトとして読み込んだところ，各データセットの平均は表 1 のようになった．但し，DOM オブジェクトの項は，空の XML ファイルを読んだ

表 1　平均データサイズの比較
Table 1　Comparison of Avarages of Data Sizes

	小	中	大
BMP バイナリデータ	52KB	780KB	1.5MB
XML ファイル	820KB	13MB	29MB
DOM オブジェクト	9MB	49MB	96MB

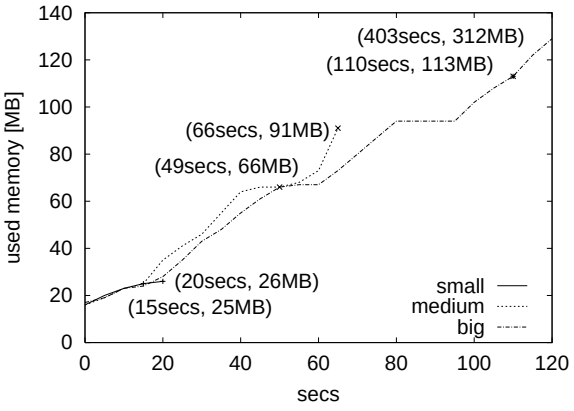

図 4　Xerces による全ノード巡回

Fig. 4　The Result of Traverses by Xerces

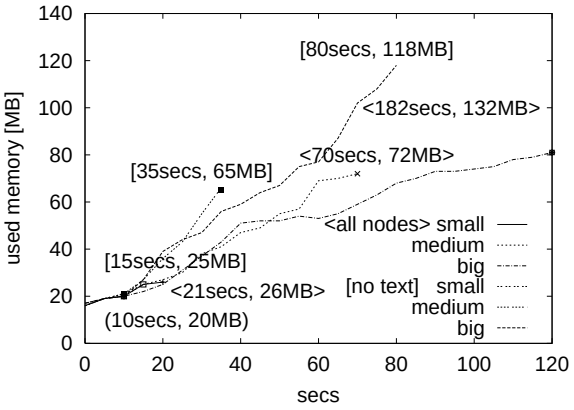

図 5　提案機構による全ノード巡回

Fig. 5　The Result of Traverses by Proposal

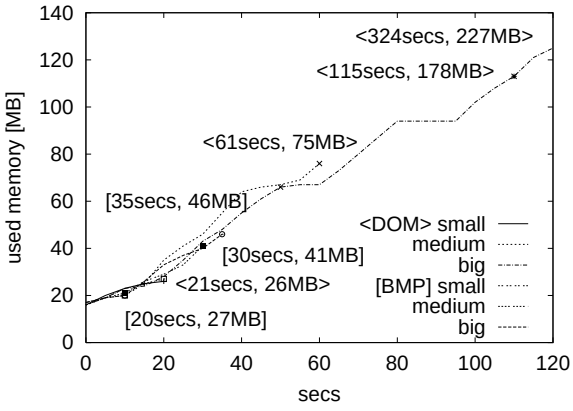

図 6　XPath 問合せ Q1

Fig. 6　The Result of XPath Query Q1

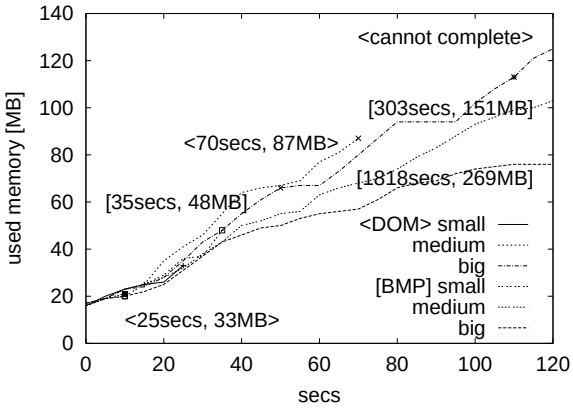

図 7　XPath 問合せ Q2

Fig. 7　The Result of XPath Query Q2

時との JVM による主記憶消費量の差であり，実際の DOM オブジェクトのサイズとは異なる．

これらの三つのデータセットに対して，提案機構でバイナリデータを処理した場合と，Xerces2 [10] 及び Xalan2 [11] で変換生成した XML ファイルを処理した場合を比較する．

実験環境は，Pentium MMX 300MHz, 物理メモリ 192MB, IBM OS/2 v4.51, IBM 版 JDK 1.3.1 である．また，実験プログラムの実行に不要なページを可能な限りスワップアウトさせた時の JVM 非起動時の物理メモリの空きは 137MB である．この環境で空の XML ファイルを処理した時の物理メモリの空きは 121MB であり，これが実質的なデータ領域として使用可能な量である．

5.2　DOM API でのアクセス

DOM API を用いてルートノードから深さ優先で全ノードを巡回する．巡回経路の全てのノードがアクセスされる．`Text` ノードの値を参照する場合としない場合の比較も併せて行なう．

図 4 に，Xerces を用いて `Text` ノードも含めて巡回した場合の結果を示す．図中の点は，XML ファイル全体のパースの終了時点と巡回終了時点の時刻と消費記憶容量である．ファイルサイズが一番大きいデータセットを巡回する過程で，主記憶を使い切って頻繁にスワップが発生したため，処理時間が非常に長くなった．`Text` ノードを含めない場合は，パース終了後の処理が多少速くなる程度の差のみであったため，省いている．

図 5 は，提案機構で生成されたノードの保持に使用可能な記憶空間を 30MB として BMP ファイルを処理した結果である．図中の点は，フォーマット定義のパース終了時点と巡回終了時点の時刻と消費記憶容量である．提案機構では，10 秒程度のフォーマット定義のパースの後，即座に巡回が開始される．また，`Text` ノードを含めた巡回では，散発的なディスクアクセス，ノードの生成，オフセット計算のオーバーヘッドのため，巡回速度が遅くなっている．しかし，大きなデータセットに対して特に顕著に見られるように，使用記憶容量の制限のため 40 秒辺りからの主記憶の消費速度が低下した．これによりスワップの発生が抑えられ，結果的に Xalan に比べて大幅な処理速度の向上が見られた．また，`Text` ノードを含めない場合は，ディスクアクセスがほとんど発生しないため，半分以下の時間で巡回を行なえた．

5.3　XPath でのアクセス

XPath 問合せの処理性能を比較する．尚，ここでは紙面の都合上，恣意的な問合せとなっているが，本質的には，ディスクアクセスを必要とするノード群の割合の影響や性能限界の特性等の確認を目的としている事に注意されたい．

問合せ Q1 として，「8 ビット画像のパレットのうち，赤の値が 128 以上の色の集合」を取得する．これは，次の XPath 問

合せで表される．

```
/bitmap[bmpinfo/color-num=8]/palette/color[red>=128]
```

これは，データ全体に比べて小さなパレット部分のみにアクセスする．ディスクアクセスを必要とするのは，構造決定情報と/bitmap/bmpinfo/color-num, /bitmap/palette/color/red に限られる．

問合せ処理結果を図 6 に示す．5.2 節と類似するが，走査するノードが限定されているため，より短時間で処理が完了した．

次に問合せ Q2 として,「16 ビット画像の画素のうち，赤の値が同じ画素が存在するものの集合」を取得する．これは，次の XPath 問合せで表される．

```
/bitmap[bmpinfo/color-num>=16]
    /image/pixel[red=../pixel/red]
```

これは，最後の選択条件のために画像全体を何度も走査するため，ノードが主記憶上に載り切らない場合にはディスクアクセスが多発する．これは大幅な性能低下に繋がる．

問合せ処理結果を図 7 に示す．小さいデータセットでは全てのノードが主記憶に収まるため，比較的短時間で処理が終わる．一方，大きいデータセットでは，Xalan では OS によるスワップが頻発し，最終的に 400MB 強の仮想記憶を使い尽くしてしまった．これに対し，提案手法ではノードの生成・破棄が多発し，ノードの値をディスクから何度も読んでいるため処理速度が大きく低下したものの，処理を完了する事ができた．

以上，5.2 節と 5.3 節の実験結果では，処理を行なうために十分な物理記憶容量がある場合は通常の DOM と比べて若干の性能は低下するものの，遜色ない結果であった．一方，主記憶が制約となる場合には提案手法の方が安定して処理可能であった．また，一部のデータにしかアクセスしない場合や中間のノードにしかアクセスしない場合には，XML ファイル全体をパースする処理が省かれ，提案手法の方が良い性能を示す事があると確認できた．

6. ま と め

XML 中心のデータ統合利用に既存のバイナリデータの統合を実現する事は，それらの応用範囲を拡大する事に繋がり有用である．この観点から，これまで，バイナリデータのフォーマット定義に従って，XML ビューを構築するというアプローチを取り，部分的な実体化方式をサポートした DOM API と XPath 問合せ処理の効率化についての考察を行なってきた．本機構では，あるデータの多くが固定長であるというバイナリデータの特徴に着目してオフセット計算を主とする事で，必要最小限のディスクアクセスでノードを取得する事が可能である．更に本稿では，DOM API におけるデータ更新をサポートした．また，実験によりデータ参照における性能特性を調べ，提案手法の利点を確認できた．

今後，主要なバイナリフォーマット群における適用可能範囲や，より複雑なデータ構造への適用可能性，Persistent DOM や XML DBMS も含めたより詳細な性能特性の調査，効率化手法の検討が必要である．データ更新についても実装を進め，その性能特性の確認についても取り組む予定である．

[謝　　辞]

本研究の一部は，科学研究費補助金基盤研究 (B) (15300027)，特定領域研究 (2)(15017207) による．

文　　献

[1] V. Christophides, J. Freire (Eds.). International Workshop on Web and Databases (WebDB 2003), San Diego, California, 2003.
[2] World Wide Web Consortium (W3C), http://www.w3.org/.
[3] Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/.
[4] S. Abiteboul, S. Cluet, T. Milo. Querying and Updaitng the File, 19th VLDB Conference, Dublin, Ireland, 1993.
[5] M. Westhead, M. Bull, Representing Scientific Data on the Grid with BinX, EPCC, January 2003.
[6] The W3C Workshop on Binary Interchange of XML Information Item Sets, http://www.w3.org/2003/07/binary-xml-cfp.html.
[7] J. Clark, M. Murata (Eds.), RELAX NG Specification, OASIS, http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.
[8] G. Huck, I. Macherius, P. Fankhauser, PDOM: Lightweight Persistency Support for the Document Object Model, Proc. 1999 OOPSLA Workshop “Java and Databases: Persistence Options” (OOPSLA’99), Colorado, USA, 1999.
[9] P. V. Biron, A. Malhotra (Eds.), XML Schema Part 2: Datatypes, W3C, http://www.w3.org/TR/xmlschema-2/.
[10] Apache Software Foundation, Xerces Java 2, http://xml.apache.org/xerces2-j/.
[11] Apache Software Foundation, Xalan Java 2, http://xml.apache.org/xalan-j/.