バイナリデータ上の XML ビュー機構と XPath 処理の提案

A Proposal of XML Views over Binary Data and XPath Processing

品川 徳秀† 北川 博之‡
Norihide SHINAGAWA Hiroyuki KITAGAWA

多くのアプリケーションでは固有の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，この枠組みでバイナリデータの扱いを可能にする事はその応用範囲の拡大に繋がる．本研究では，バイナリデータの XML ビューを提供し，DOM API として実現する機構について論じる．これにより，DOM ベースの XML 関連技術がバイナリデータに対しても適用可能となり，例えば XML 問合せの対象として利用可能となる．提案手法では，データアクセスの効率化をはかるため，与えられたフォーマット定義に基づいた動的・部分的な変換をサポートする．これは消費記憶領域とディスクアクセスを削減する．また，XPath 問合せでの処理の効率化について考察する．

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. This paper discusses XML views over binary data, and then provides DOM API. This approach makes DOM-based XML technologies applicable to binary data. Use of binary data as data source in XML queries is an example. We show an implementation technique of DOM API which enables partial instantiation of DOM nodes to limit the amount of consumed memories. The paper also shows techniques to realize efficient accesses to values based on properties of binary data. In addition, techniques to improve XPath query processing will be described.

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．

この枠組みでの非 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．バイナリデータでは，一般のバイナリデータを扱う BinX [5] や，ASN.1 を XML に変換するものなどがある．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，前述のものなど (A), (B) の方式で実現してい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 ビューを検討する．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．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．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．但し，現時点では更新操作はサポートしない．これにより，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について説明する．動的・部分的な実体化に関しては XML データベースや各種 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．

2. XML ビュー定義

2.1 バイナリデータの例

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できる．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

† 正会員, 千葉大学 環境リモートセンシング研究センター
siena@faculty.chiba-u.jp
‡ 正会員, 筑波大学 電子・情報工学系 kitagawa@is.tsukuba.ac.jp

フォーマットでも，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．

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 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 で，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．

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，バイトオーダ等はこれに含まれる．`xmltype` には XML Schema Part 2 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 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．

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使い分ける．

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[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．

3. XML ビュー構築

3.1 DOM 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

[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．

```
<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

```
<!-- (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 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

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[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，フォーマット定義と構造決定情報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．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．

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 f_i のフォーマット定義に従って構築される．その時点では，子孫ノードや属性ノードの構築は行なわず，アクセスが発生した時に同様にノードを構築する．使用可能な記憶領域を超過する時，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，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．

4.　XPath 問合せ処理の様子

XPath は，XSLT や XQuery 等での基本的なノード取得手段として使用され，DOM Level 3 でも XPath 対応が行なわれており，その処理の効率化は重要である．単純な実装ではノードを一つ一つ辿るという処理になるが，提案機構では XPath 式の複数の連続するステップをまとめてオフセット計算でき，その途中のノードの実体化を削減できる．

4.1　ノード実体化処理の埋込み

部分的実体化を行なうと任意の時点でノードが破棄される可能性があるため，ノード間を移動する時点でその実体化を保証する必要がある．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．未適用部分の式には下線を付す．

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

[2] ファイル先頭の文字列の例: BMP の “`BM`”，GIF の “`GIF89a`”

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

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，中間ノード集合を実体化せずに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．

単ステップのパス式 $p = \alpha(\alpha(x :: t|_{\alpha(r)})[\alpha(e)])$ について，$\alpha(r)$ は先立って評価されるため他の部位とは独立に簡易化を行なう．$\alpha(e)$ 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．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 全体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．

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

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．

$$\begin{aligned} \alpha(p) &\rightarrow \alpha(\alpha(s_1)/\alpha(p_1)) \rightarrow \alpha(\mu(\lambda(s_1)/\alpha(p_1))) \\ &\rightarrow \cdots \rightarrow \mu(\lambda(s_1)/\lambda(s_2)/.../\lambda(s_n)) \end{aligned} \quad (4)$$

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，$\alpha(p_i) \rightarrow \lambda(p_i)$ 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．

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

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

これから，実体化が必要なのは構造決定情報の `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だけで十分と分かる．

5. まとめ

バイナリデータを XML 中心のデータ統合で利用可能とするため，フォーマット定義に従った XML ビューを構築し，部分的な実体化が可能な DOM API と XPath 問合せ処理の効率化についての考察を行なった．バイナリデータの多くが固定長であるという特徴に着目してオフセット計算を主とする事で，主記憶消費とディスクアクセスの軽減を試みた。

今後，より複雑なデータ構造への適用可能性や更新操作のサポート，詳細な効率化手法の検討が必要である．また，実装による実現可能性の確認，実行性能の調査等の課題があり，これらについて取り組んで行く予定である．

[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 (2)(15017207) による．

[文献]

[1] V. Christophides, and J. Freire (Eds.).: International Workshop on Web and Databases (WebDB 2003) (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, and T. Milo.: “Querying and Updating the File”, Proceedings of 19th VLDB Conference, pp. 73–84 (1993).

[5] M. Westhead, and M. Bull.: “Representing Scientific Data on the Grid with BinX”, http://www.epcc.ed.ac.uk/~grid serve/WP5/Binx/sci-data-with-binx.pdf, EPCC (2003).

[6] The W3C Workshop on Binary Interchange of XML Information Item Sets, http://www.w3.org/2003/07/binary-xml-cfp.html (2003).

品川 徳秀　Norihide SHINAGAWA

千葉大学環境リモートセンシング研究センター助手．2001 年筑波大学博士課程工学研究科修了．博士（工学）．主に構造化文書や WWW の高度利用等の研究に従事．ACM，日本データベース学会，情報処理学会，各会員．

北川 博之　Hiroyuki KITAGAWA

筑波大学電子・情報工学系教授．1980 年東京大学大学院理学系研究科修了．理学博士（東京大学）．異種情報源統合，文書データベース，WWW の高度利用等の研究に従事．著者「データベースシステム」（昭晃堂），「Unnormalized Relational Data Model」（共著，Springer-Verlag）等．ACM，IEEE-CS，日本データベース学会，情報処理学会，電子情報通信学会，日本ソフトウェア科学会，各会員．