情報処理学会論文誌　データベース　Vol. 3　No. 4　13–33 (Dec. 2010)

PPX：XML 整形出力のための出版言語

金　　哲[†1]　遠山元道[†2]

本論文では XML データを HTML 化する XML 整形出力言語 PPX（Pretty Printer for XML）を提案する．本言語はレイアウト式の指定によって XML データの構造変更と書式の設定を行い，様々な異なる表構造を持つ HTML を出力する．PPX は RDB に対して同様のことを行う SuperSQL に基づいているが XML データをレイアウト対象とするので次のことが異なる．1 つは文書中心の XML データでは要素ノードの現れる順序が重要な場合があり，それらに対して重複を許し元の順序のとおりに変換を行うオプションを与えたことである．もう 1 つは XML データの非定型木構造に対して条件分岐を用いて異なる変換方法を適用することである．実験では W3C の Query Use Cases に PPX を適用し，ユーザ定義関数の処理のためのユースケースを除いたすべてについて PPX によって等価な表現が可能であることを確認した．また，XML データの HTML 化において XSLT や XQuery よりも生産性が高い例を示した．

PPX: An XML Data Publishing Language

Zhe Jin[†1] and Motomichi Toyama[†2]

In this paper, a PPX (Pretty Printer for XML) publishing language for transforming XML data into HTML is proposed. The feature of this language enables XML data structure conversion, layout information setting by specifying the layout expression, and outputs various HTML table structure that is based on extended TFE of SuperSQL. However, in order to convert XML data, the following points are different. 1) Because the order of the element nodes is very important in same document-centric XML data, a new repeat operator has been implemented. The operators can guarantee the original data order even after sorting, grouping, and duplication elimination operation. 2) We introduced a conditional branch to process variable form tree structure by a different layout pattern. PPX is applied to the W3C query use cases, and the results show that PPX can express all the queries except for FNPARM use cases. In addition, by the experiments of the large-scale XML data and generation of different HTML table structures, PPX indicates higher productivity than XSLT and XQuery.

1. はじめに

XML（Extensible Markup Language）はデータの表現，交換や情報の記述・処理など様々に利用されており，XML で記述されたデータが大量に存在する．XML データを Web ブラウザで表示する場合はそれを HTML に変換する必要がある．たとえば，The University of Washington of XML repository [16] で提供する DBLP Computer Science Bibliography のデータを著者別，年度別などに様々なユーザの用途に応じて見やすくレイアウトすることが考えられる．このような処理を本研究では XML データ出版，または XML 整形出力と呼ぶ．広く普及している XQuery [3] や XSLT [1),2)] が上述の DBLP データから論文題目と著者を取り出し，著者を論文題目でソート，グルーピング，および重複排除を行い，HTML 化することを考えてみる．これを実現する XQuery を次に示している．ただし，この例では HTML タグの記述を省略している．

```
⟨results⟩ {
for $a in distinct-values( db2('bib.xml')//title )
    sortby ( title ascending )
let $b := $a//author
return
    ⟨result⟩
      ⟨book⟩ { $a/title/text() } ⟨/book⟩
      { for $c in distinct-values ( $b )
              sortby (author ascending)
        return ⟨name⟩ { $c/text() } ⟨/name⟩
      }
    ⟨/result⟩
} ⟨/results⟩
```

この問合せでは FOR 句がネストされており，外側の FOR 句にある SORT BY 関数によって title の一覧を昇順で作り出している．そして内側の FOR 句でそれぞれの title にい

†1 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†2 慶應義塾大学理工学部情報工学科
Department of Information and Computer Science, Keio University

る author を拾い上げ，DISTINCT-VALUES 関数によって重複を削除し，SORT BY 関数によって昇順にソートしてから結果を出力している．

また，XSLT 2.0 でこれに相当することを行う HTML タグの記述を省略したプログラムの一部を以下に示す．

```
...
<xsl:template match="papers">
<xsl:for-each-group select="xf:distinct(paper/title)">
<xsl:sort select="paper/title"/>
<xsl:copy-of select="current-group()/paper/title"/>
<xsl:apply-templates select="authors"/>
</xsl:for-each-group>
</xsl:template>

<xsl:template match="authors">
<xsl:for-each-group select="xf:distinct(author)">
<xsl:sort select="author"/>
<xsl:copy-of select="current-group()/author">
</xsl:for-each-group>
</xsl:template>
...
```

この例では 2 つの XSL のテンプレートを利用しており，1 つ目のテンプレートで重複排除，昇順にソートされた title 要素ノードのリストを取得し，2 つ目のテンプレートではそれぞれの title に対応する author をさらに重複を削除，昇順にソートしてから結果を出力している．

本論文ではこれらとは異なる方法で XML データから HTML への変換，出版を行う XML 整形出力言語 PPX（Pretty Printer for XML）を提案する．たとえば，上述した XSLT，XQuery などと同等の変換を行う PPX クエリを次に示している．

```
GENERATE html
  [ $a/title , [ $b ]! ]!
FOR
  $a in db2('bib.xml')//book,
  $b in $a//author
```

この例に見られるように，一般的な用途において PPX では XQuery や XSLT より簡潔な記述が可能である．

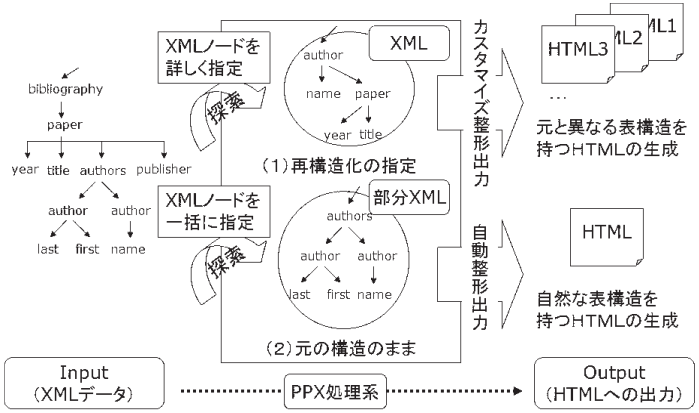


図 1　XML 整形出力
Fig. 1　Pretty printer for XML.

PPX による XML 整形出力は次の 2 種類のことを行う．1 つはユーザがレイアウト式の詳細な指定によって XML データの構造変換，および書式情報の設定を行い，表構造を持つ HTML を出力することを目的とするカスタマイズ整形出力（**図 1** (1)）である．もう 1 つはユーザがレイアウト指定をせず，システムが XML データをオリジナルのデータ構造に基づいて自然な階層構造を持つ HTML 表形式やインデント形式などで出力する自動整形出力（図 1 (2)）である．この 2 つは排他的ではなく XML データの一部をカスタマイズ整形し，一部を自動整形することもできる．

提案する PPX の基礎技術である SuperSQL [9] は TFE（Target Form Expression）を用いて RDB から HTML 表形式を含む XML 形式など様々な媒体を出力する．SuperSQL は入力として RDB を扱うが，XML データは構造の不規則性など RDB と異なる性質を持つためこれに対応する拡張が必要である．

本言語は XML データの性質に基づき，特に次の 3 点について SuperSQL の TFE を拡張する．

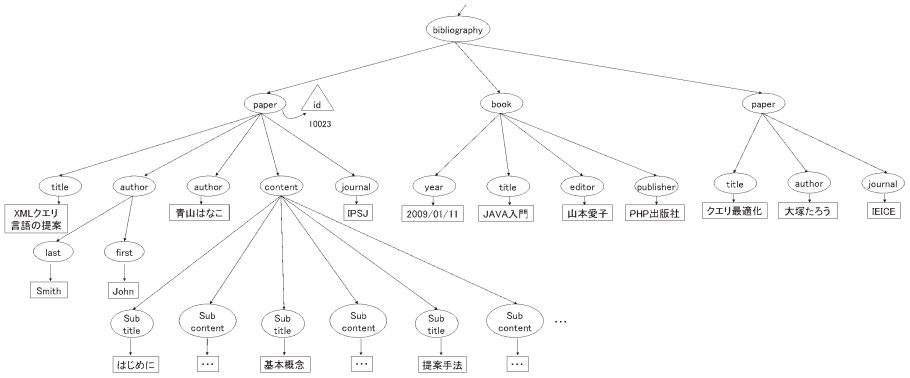


図 2　XML データの木構造表現
Fig. 2　Tree representation of XML data.

(1) 要素ノードの順序の保存

データ中心の XML データでは処理効率のために順序を無視することがある．しかし，文書中心の XML データでは要素ノードの順序が重要である．たとえば，**図 2** で示している XML データから著者名と論文題目名を選択し，論文題目名を著者名でグルーピングするとグルーピングのために著者名でソートが行われ，出力される結果は以下のようになる．

寺田いちろう	データの格納方法
青山はなこ	XML クエリ言語の提案
大塚たろう	クエリ最適化
	データの格納方法
	索引技術の開発
山本ももこ	規模データストリーム処理
	索引技術の開発大
…	…

ここでは兄弟要素ノードの順序を任意にして出力しても問題にはならない．しかし，さらにサブタイトル，サブコンテンツを選択し，それをグルーピングするときに出力される結果が以下のようになってしまうと，論文の構成が乱れてしまう．

青山はなこ	
XML クエリ言語の提案	
関連研究	データを格納する既存の方法には …
基本概念	XML データとは …
実験・評価	様々なデータを適用し，…
提案手法	本研究では新たな方法を …
はじめに	XML ではデータ中心の XML と …
まとめ	本論文では …
大塚たろう	
…	

このような場合は兄弟要素ノードの順序保存が必要になる．この例だけを見ると順序保存を標準にすればよいと考えられるが順序保存では同一値の重複の除去ができない．このため，重複を除去して順序を保存しないレイアウトと順序を保存して重複を除去しないレイアウトを使い分けるために 2 種類の演算子を提供する．

(2) 条件分岐の導入

2.4 節で後述するイレギュラ XML データから構成された非定型木構造に対して異なるレイアウト方法を適用するために条件分岐の指定を導入した．たとえば，図 2 で示している XML データで出現する paper 要素ノードと book 要素ノード以下にある部分 XML データは異なるデータ構造を持つので，条件分岐の適用によってそれぞれに応じたレイアウトを指定する必要がある．条件分岐を持たない従来の TFE では paper に合わせたレイアウトでは book の情報は無視され出力できない．

(3) 自動整形出力

XML データを HTML に変換する際に出力の表構造をきっちりと定義して望む結果を得るというニーズのほかに，その全体もしくは一部について元の XML データの木構造を自然に表現する出力を自動的に得たいというニーズが存在する．PPX では後者の目的のために XML データの部分木の自動整形を指示する演算子を提供する．自動整形にはインデント形式で出力してしまう簡易な方法と木構造の出現パターンを解析してレイアウトの最適化を行う高度な方法が考えられる．後者については独立の論文で扱うものとし，ここでは簡易な自動整形のみを取り扱う．特に不規則性が予見できず，前項の条件分岐で対処できない場合

に必要である．

本論文の貢献の 1 つは元の TFE が持つ結合演算子，反復演算子などのレイアウト指定演算子に加えて要素ノードの順序を保存する反復演算子，部分 XML に対して自動変換を行うための自動レイアウト演算子，イレギュラ XML データの処理を指定するための条件分岐構文の 3 点を PPX に導入したことである．もう 1 つの貢献は PPX の実装を行い，これに基づいて XSLT や XQuery との比較実験を実施して記述力と生産性の評価を行ったことである．

1.1 関 連 技 術

XML データを HTML 化する既存の技術を次のように分類する．

(1) 汎用プログラミング言語による変換

JAVA，PERL，PHP，C++などの汎用プログラミング言語の場合，XML パーサと DOM や SAX などの API を使ってプログラミングを行い，取り出した XML データのタグを HTML タグに変換して Web ブラウザで表示する．

(2) 問合せ言語・変換言語による変換

XSLT 1.0[1]，XSLT 2.0[2]，XQuery[3] などは XML データを HTML に変換する目的で広く使われている．XML 変換言語である XSLT は変数とテンプレートをうまく用いることにより変換を行う．SQL の影響を強く受けている XQuery は問合せ関数の作成や FOR 句，LET 句，WHERE 句，ORDER BY 句，RETURN 句をネストした記述によって変換を行う．静的に型付けされた XML データ処理用の関数型言語である XDuce[8] は正規表現のパターンマッチを行う能力を組み込みで持っており，マッチングに応じて変換を行う．また，これらの言語は XML データを HTML 化するために，HTML タグの記述も同時に行う必要がある．提案する PPX は XML データの探索部分には XQuery の機能を用いてフラットなリスト構造のデータを取り出す．また，レイアウト式の指定によってフラットなリスト構造のデータに対する構造化，および HTML タグの付与や書式情報を設定する．

(3) スタイルシート言語による変換

XSL-FO[4] や CSS[5] などのスタイルシート言語は XML データを Web ブラウザに表示する際のマージンや色，文字サイズなどの書式情報を与える．XSL の 1 つの機能である XSL-FO は XSLT によって XML データの構造変換を行い，その際に Web ブラウザで表示するための書式情報を付加できる．CSS は XML データの構造を変換する機能はないので簡単なスタイル付けだけで済む場合に利用できる．

論文構成　2 章では基本となる概念を簡単に述べる．3 章では PPX の概要，条件分岐構文と変換モデルなどについて述べる．4 章では実装，5 章では実験・評価について述べ，最後にまとめと今後の課題を述べる．

2. 基 本 概 念

2.1 データモデル

XPath 2.0[11] のデータモデルを採用して XML データを記述する．XML データはルートノードを根とする木構造であり，ルートノード，要素ノード，テキストノード，属性ノード，名前空間ノード，処理命令ノード，コメントノードなど 7 種類のノードから構成される．PPX では要素ノード，テキストノード，属性ノード以外に追加した NID（Node identifier）ノードを含めた 4 つのタイプを使用する．これらを**図 3** で示している．ここで NID ノードはそれぞれの要素ノードを識別するための深さ優先順序のラベリング方法[12] を用いてユニークな番号を与えるものであり，要素ノードの属性ノードとして扱う．テキストノードには NID を付与しない．NID は独自の名前空間を利用する．**図 4** はそれぞれの要素ノードに NID 番号が付けられた XML データを示している．

2.2 パス表現式

PPX で XML データの検索に用いるパス表現式は XPath 2.0 に基づいており，ルートから始まる絶対パス表現式とそれ以外の相対パス表現式がある．以降はこれらを区別する必要がなければパス表現式と略する．PPX がサブセットとして利用する XPath 表現式の文法は付録 A.2 で示している．

完全なパス表現式はルートノードからテキストノードまでを表示した絶対パス表現式をいう．たとえば，3.2 節の PPX 1 の 1 番目の FOR 句で利用したパス表現式と GENERATE

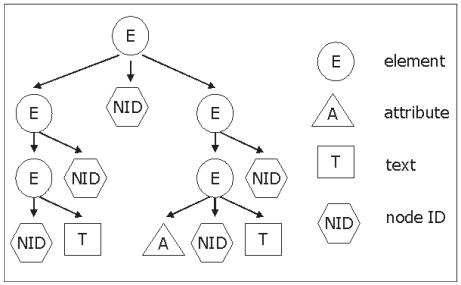


図 3　利用した XML ノードの種類
Fig. 3　The four used node types.

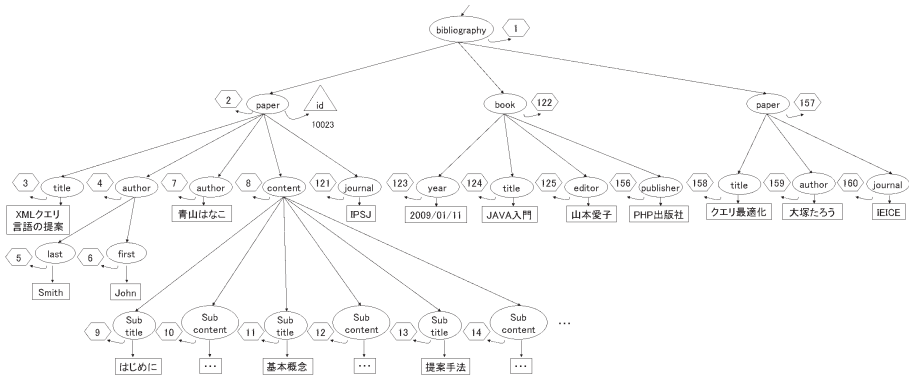


図 4　NID を付与したデータモデル
Fig. 4　Data model with NID.

句で利用した相対パス表現式を連結した次のパス表現式

/biblio/paper/title/text()

は title 要素ノードのテキストノードの値を探索する完全なパス表現式である．これをレイアウト式で指定する場合，text() は省略して指定する．

不完全なパス表現式はルートノードからテキストノード以外の任意の要素ノードまでを表示した絶対パス表現式をいう．たとえば，3.2 節の PPX 3 の FOR 句で利用したパス表現式と GENERATE 句で利用した相対パス表現式を連結した次のパス表現式

/biblio/paper

は paper 要素ノード以下にある部分 XML を探索する不完全なパス表現式である．ここでいう部分 XML とは XML データの全部，または一部である．すなわち，不完全なパス式によって探索された任意の要素ノードを根ノードとし，その根ノードのすべての子孫ノードを含む部分木である．

2.3　イレギュラ XML データ

イレギュラ XML データとは XML データで任意の要素ノードを親ノードとする部分木の集合が 2 種類以上の異なるデータ構造を持つ非定型木構造である．たとえば，図 2 の XML データで bibliography 要素ノード以下には paper に関する XML データだけではなく，データの種類や構造が異なる book に関する XML データも出現する．ここで paper を book に対してイレギュラ XML データといい，逆に book は paper に対してイレギュラ XML データという．

3.　PPX（Pretty Printer for XML）

PPX の構文は以下のように表せる．このうち，WHERE 句は省略可能である．

GENERATE ⟨ 媒体指定 ⟩ ⟨ レイアウト式 ⟩
FOR ⟨ ”$” ＋ 変数名 in db(’XML’)/パス表現式 ⟩
WHERE ⟨ 条件式 ⟩

GENERATE 句では出力する媒体（HTML，XML etc.）の指定と FOR 句で探索されたフラットな構造のデータを木構造に構造化するレイアウト式を指定する．本論文では出力媒体として HTML のみを扱う．PPX の FOR 句，WHERE 句などは XQuery と同様のため説明を省略し，主にレイアウト式を記述する拡張 TFE について説明する．

3.1　拡張 TFE

拡張 TFE は SuperSQL の TFE（Target Form Expression）を拡張したものであり，演算子とオペランドを持つ一種の式である．

3.1.1　オペランド

オペランドはテキストノードを持つ要素ノード，または部分 XML を持つ要素ノードを表す．以下に示すように変数名とパス表現式の組合せからなる．ここでパス表現式は要素ノード以下にあるテキストノードの値や部分 XML に対して探索を行う．

オペランド ::= ”$” ＋ 変数名/パス表現式

3.1.2　演　算　子

演算子にはレイアウト指定演算子と自動レイアウト演算子がある．

(1) レイアウト指定演算子

- 結合演算子

結合演算子は両辺の整形結果をいずれかの方向（次元）に結合する二項演算子である．**図 5** は左から右にデータを横に結合して出力する横結合（,），縦に結合して出力する縦結合（!）と 3 次元方法へ結合（出力が HTML ならばリンクとなる）を表す深度結合（%）演算子を示す．

- 反復演算子

反復演算子には 2 種類の単項演算子がある．1 つは重複を排除し，要素ノードの現れる順序を並べ替えて出力を行う反復演算子であり，横反復（[ ],），縦反復（[ ]!）と深度反復（[ ]%）などがある．これらは SuperSQL の反復演算子と同様である．もう 1 つは本研究で

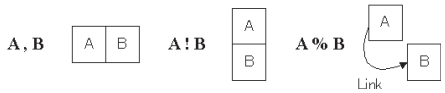


図 5　結合演算子
Fig. 5　Connect operators.

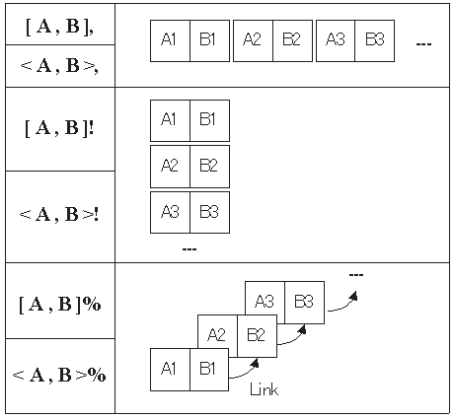


図 6　反復演算子による反復連結
Fig. 6　Repeating by repeat operators.

拡張した重複を許し，要素ノードの現れる順序のとおりに出力を行う反復演算子であり，横反復（⟨ ⟩,），縦反復（⟨ ⟩!）と深度反復（⟨ ⟩%）などがある．2 種類の反復演算子は異なる方法でデータに対して構造変換（ソート・グループ・重複データの処理）を行い，反復的に連結を行う．

(a) データの反復連結

2 種類の反復演算子は得られたデータがある限り，指定する方向に繰り返し結合する．**図 6** は横に順に表示する横反復（[ ], と ⟨ ⟩,），縦に順に表示する縦反復（[ ]!と ⟨ ⟩!）と奥行き方向（リンク）に順に表示する深度反復（[ ]%と ⟨ ⟩%）などの種類と意味を上から下の順に示している．

(b) データのソート・グルーピング・重複処理

元の順序を無視する反復演算子は要素ノードの値の順で並べ替え，元の順序を重視する反復演算子は要素ノードの位置の順を保存する．前者はデータによるソート，グルーピング，

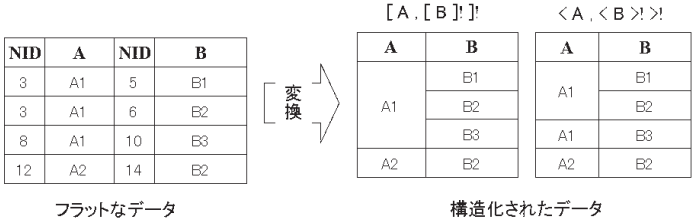


図 7　反復演算子によるグルーピング
Fig. 7　Grouping by repeat operators.

重複データの処理などを行い，後者はノード番号（NID）によるソート，グルーピング，重複データの処理などを行う．ソートについて反復演算子に DESC というキーワードを指定（たとえば，[(desc) $i/author]!）すると降順でソートを行い，ASC と指定するか，または何も指定しない（たとえば，[$i/author]!）場合は昇順でソートを行う．また，重複の判断について前者は要素ノード名の同一性とテキストノードの値の同値性を満足する必要がある．後者は要素ノード名の同一性，テキストノードの値の同値性，および NID の値の同値性を満足する必要がある．

図 7 では探索されたフラットなリスト構造のデータを 2 種類の反復演算子によってグルーピングした際に異なる結果となる例を示している．

- 装飾演算子

装飾演算子により出力する文字サイズ，フォント，横幅などの指定を付加できる．これらは@{ 装飾指定式 } の構文による装飾演算子によって指定する．装飾指定式は（項目名 = 値）として指定する．複数指定するときは各々を “,” で区切ったリストとする．たとえば，$A@\{width = 300\}$ という装飾子を用いるとオペランド A のセル幅を 300 px にする．

(2) 自動レイアウト演算子

自動レイアウト演算子は不完全なパス表現式によって探索された部分 XML に対して自動変換する単項演算子である．

- &−，&+演算子

&− 演算子は部分 XML からタグを除いてインデント形式[⋆1]で表示する単項演算子であ

⋆1 これは図 13 でも示したようにある階層に属する 1 つのデータから下位階層に位置する複数のデータが枝分かれした状態で配置されているものである．

り，&+演算子はタグ付きのまま同様にインデント形式で表示する単項演算子である．

- &演算子

&演算子は部分 XML のインスタンスの統計的な性質に基づいて出力する表構造を決定し，HTML への自動変換[6] を行う単項演算子である．本論文では詳細については扱わない．

3.1.3　演算子の優先順位

通常，結合演算子における連結の優先順位は左が高く，左から右へと処理される．この優先順位を変更する場合には優先的に処理したい部分を明示的に中括弧（{ }）で括ればよい．

3.1.4　条件分岐構文

PPX のレイアウト式によって規定される変換規則は条件分岐構文を含むことができる．ここで条件分岐構文を表現する if-then-else 構文で if 句の指定をパターンといい，then 句や else 句のレイアウト式の指定をレイアウト方法という．

(1) 条件分岐構文の記述規則

イレギュラ XML データから構成された非定型木構造をパターンで表現することによって異なる XML データ構造に対してそれぞれのレイアウト式を記述する．

```
if (条件式) then (レイアウト式) else (レイアウト式)
```

または

```
if (条件式) then (レイアウト式)
elseif (条件式) then (レイアウト式)
else (レイアウト式)
```

ここで else 句を省略することも可能である．

(2) 条件分岐構文の記述例

- オプショナル記述

次の条件分岐は name 要素ノードがテキストノードを持つ場合にそれを出力することを条件分岐構文で指定する．

```
GENERATE html
  [ $i//title ! [ $j/affiliation ,
    if ( $j/name/text() ) then ( [ $j/name ]! )
  ]! ]!
FOR
  $i in db('bib.xml')/biblio/*,
  $j in $i//( author | editor )
```

ここで 3 行目の if 句の条件が TRUE の場合，次の then 句のレイアウト式を適用し，FALSE の場合は空になることを示している．

- 合成記述

イレギュラ XML データから構成された 2 つ以上の非定型木構造をそれぞれのパターンに対応し，異なるレイアウト方法を生成する条件分岐構文を指定することができる．たとえば，次の条件分岐は author（または editor）要素ノード以下にテキストノードか，あるいは要素ノードか，あるいは属性ノードが現れるかによって異なるレイアウト方法を生成する条件分岐構文を指定している．

```
GENERATE html
  [ $i//title ! [ $j/affiliation , [
    if ( $j/name/text() ) then ( $j/name )
      else if ( $j/@name ) then ( $j/@name )
        else ( $j/name/first , $j/name/last )
  ]! ]! ]!
FOR
  $i in db('bib.xml')/biblio/*,
  $j in $i//( author | editor )
```

ここで 3 行目の if 句の条件が TRUE の場合，次の then 句のレイアウト式を適用し，成立しなければ下の elseif 句の判定へと移る．また，elseif 句の条件が成立すれば次の then 句のレイアウト式が採用される．成立しなければ次の else 句に続くレイアウト式が採用される．

3.2　応　用　例

図 8 は XML データインスタンスの一部であり，それを利用して PPX が整形出力を行う応用例を示す．

(1) カスタマイズ整形出力

カスタマイズ整形出力はオペランドをレイアウト演算子と組み合わせて指定によって整形出力を行うものであり，オペランドで指定する相対パス表現式は FOR 句で指定するパス表現式を連結して完全なパス表現式となる必要がある．たとえば，以下に示す PPX 1 は GENERATE 句のレイアウト式でオペランドに演算子を組み合わせて記述した完全指定によって FOR 句から探索されたフラットなリスト構造のデータに対して構造化を行い，HTML タグの付与によって**図 9** のような HTML への変換を行う．ここでは著者の所属と著者名で論文タイトルを 2 段階にグルーピングしている．

```
<biblio>
  <paper>
  <date>2006</date>
  <title>関係データベースを利用した XML 文書検索システムの開発</title>
  <author>山田敏之
  <info><age>27</age><sex>男性</sex><member>正会員</member></info>
  <email>yamada@gmail.com</email>
  <add>名古屋市鶴見区 10-3-2</add>
  <affiliation>四国電気電子大学</affiliation>
  </author>
  <author>吉野早人
  <info><age>35</age><sex>男性</sex><member>正会員</member></info>
  </author>
  </paper>
  <book>
  <date>2004</date>
  <title>FP 教科書 FP 技能士 2 級・AFP 完全ガイド</title>
  <editor>高橋親政</editor>
  <publisher>太郎出版社</publisher>
  </book>
  <paper>
  <date>2009</date>
  <title>WEB データ観測のための各種可視ビューの開発</title>
  <author>吉野早人</author>
  <author><last>羽多野</last><first>貴子</first></author>
  </paper>
  …
</biblio>
```

図 8　XML データインスタンス
Fig. 8　An XML data instance.

奈良都市大学	
藤本哲哉	XML文書からPDFファイルを生成する方法の提案
	XMLデータ処理を効率化するコンパクト化法に関する研究
静岡女子大学	
山本純一郎	XML文書からPDFファイルを生成する方法の提案
筑波工学院大学	
天野茂樹	XML文書からPDFファイルを生成する方法の提案
横浜理工大学	
絹谷忠一	XML文書からPDFファイルを生成する方法の提案
四国電気電子大学	
波多野幸子	XMLデータ処理を効率化するコンパクト化法に関する研究
吉野早人	XMLデータ処理を効率化するコンパクト化法に関する研究
	WebにおけるXMLを活用したアクセス制御の効率化
	親子関係を用いたXMLデータの効率的な構造解析
	WEBデータ観測のための各種可視ビューの開発
川崎総合大学	
天野茂樹	XMLデータ処理を効率化するコンパクト化法に関する研究

図 9　PPX 1 によるフォーマット結果
Fig. 9　Format results by PPX 1.

PPX 1:
GENERATE html
　[$j/affiliation ! [$j , [$i/title]!]!]!
FOR
　$i in db('bib.xml')/biblio/*,
　$j in $i//(author | editor)

- 異なる表構造への変換：

PPX はレイアウト式で演算子などの変更によって容易に様々な表構造を持つ HTML を生成する表現ができる．たとえば，PPX 1 を一部変更した次の PPX 2 はレイアウト式におけるオペランドの位置，および反復演算子によるグルーピングの変更などを行っている．

PPX 2:
GENERATE html
　[$i/title ! [$j/affiliation , [$j]!]!]!
FOR

XMLデータベースに基いた音楽データの検索方法	
名古屋情報科学大学	山田敏之
	野村一郎
XMLデータ処理を効率化するコンパクト化法に関する研究	
名古屋情報科学大学	青木憲正
	清水直毅
	鈴木一郎
四国電気電子大学	波多野幸子
	吉野早人
川崎総合大学	天野茂樹
奈良都市大学	藤本哲哉
WebにおけるXMLを活用したアクセス制御の効率化	
名古屋情報科学大学	鈴木一郎
四国電気電子大学	吉野早人
親子関係を用いたXMLデータの効率的な構造解析	
名古屋情報科学大学	山田敏之
四国電気電子大学	吉野早人

図 10　PPX 2 によるフォーマット結果
Fig. 10　Format results by PPX 2.

藤本哲哉	
XML文書からPDFファイルを生成する方法の提案	
はじめに	XMLの普及に伴い，XML文書を扱うアプリケーションが増加している．・・・
関連研究	XML文書からXMLやHTMLへの変換する研究は多くなされている．・・・
提案方法	本論文では，事例に基づくXML文書からPDFファイルへの変換手法を提案する．・・・
実装・評価	本研究で提案したPDFファイルの生成方法と亀岡ら[4]の比較を行った．本手法では，・・・
おわりに	・・・
山田達朗	
関係データベースを利用したXML文書の格納	
はじめに	近年，XMLで記述されたデータが増えている．XMLデータはタグを用いて・・・
従来方法の問題点	XML文書を関係データベースに格納する・・・
実験	実験にはワシントン大学で提供されているXMLデータを・・・
関連研究	・・・
結論	本論文では我々が開発したシステムを・・・

図 11　PPX 3 によるフォーマット結果
Fig. 11　Format results by PPX 3.

```
$i in db('bib.xml')/biblio/*,
$j in $i//( author | editor )
```

この結果，論文タイトルと所属で著者名を 2 段階にグルーピングした**図 10** のような HTML を生成する．

- 順序を保存する変換：

PPX では重複を許し，要素ノードの現れる順序のとおりにレイアウトを行う表現が可能である．次の PPX 3 では順序を保存したい部分（subtitle と subcontent 要素ノード）に元の順序を保存する反復演算子の指定によって変換を行う．

PPX 3:

```
GENERATE html
  [ $j ! [ $i/title !
    ⟨ $i/subtitle , $i/subcontent ⟩!
  ]! ]!
FOR
  $i in db('bib.xml')/biblio/*,
  $j in $i//( author | editor )
```

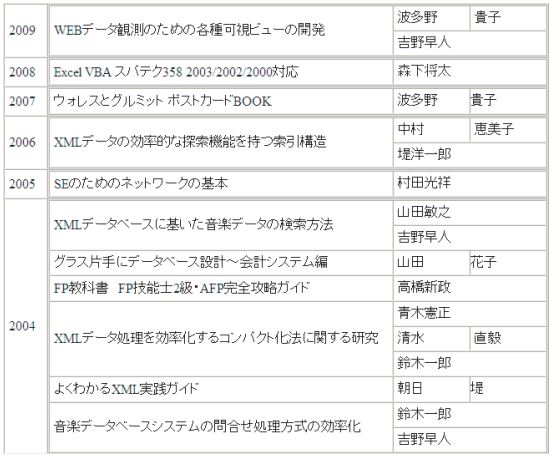
この結果，**図 11** のように subtitle と subcontent 要素ノードが持つテキストノードの値は元のデータ順を持つ HTML を生成する．

- 非定型木構造の変換：

PPX はイレギュラ XML データから構成された非定型木構造に対しては条件分岐構文に基づいた if-then-else 構文にオペランドを演算子と組み合わせて指定したものである．たとえば，以下に示す PPX 4 は paper で author 要素ノードがテキストノードを持つ場合と first，last 要素ノードを持つ場合が混在するのに対し，book では editor 要素ノードがテキストノードを持つというイレギュラ XML データを処理している．その結果，**図 12** のような HTML を生成している．

PPX 4:

```
GENERATE html
```



2009	WEBデータ観測のための各種可視ビューの開発	波多野	貴子
		吉野早人	
2008	Excel VBA スパテク358 2003/2002/2000対応	森下将太	
2007	ウォレスとグルミット ポストカードBOOK	波多野	貴子
2006	XMLデータの効率的な探索機能を持つ索引構造	中村	恵美子
		堤洋一郎	
2005	SEのためのネットワークの基本	村田光祥	
2004	XMLデータベースに基いた音楽データの検索方法	山田敏之	
		吉野早人	
	グラス片手にデータベース設計〜会計システム編	山田	花子
	FP教科書　FP技能士2級・AFP完全攻略ガイド	高橋新政	
	XMLデータ処理を効率化するコンパクト化法に関する研究	青木憲正	
		清水	直毅
		鈴木一郎	
	よくわかるXML実践ガイド	朝日	堤
	音楽データベースシステムの問合せ処理方式の効率化	鈴木一郎	
		吉野早人	

図 12　PPX 4 によるフォーマット結果
Fig. 12　Format results by PPX 4.

```
[ (desc) $i//date , [
  if ( $i/paper ) then (
    [ $i//title , [
        if ( $j/text() ) then ( [ $j ]! ) else ( [ $j/first , $j/last ]! )
    ]! ]! )
  else ( [ $i//title , [ $j ]! ]! )
]! ]!
FOR
  $i in db('bib.xml')/biblio/*,
  $j in $i//( author | editor )
```

このレイアウト式では 3 行目の if 句の条件が TRUE の場合は paper 要素ノード以下の XML データに対して処理を行い，そうではない場合に 7 行目の else 句で book 要素ノード以下の XML データに対して処理を行う．その後，論文と書籍に関するデータを 2 行目の日付（date）でグルーピングを行う．

- テーブルヘッダ出力：

XML データの段階ではタグによりこのテキストは author である，そのテキストは title であるといった情報が分かるが出版時にそれらのメタ情報が失われる．次に示す PPX 5 はそれに関して文字列定数で見出しをつけて対処する 1 つの例である．出力結果のイメージ図は省略する．

```
PPX 5:
GENERATE html
  [ { "著者" , $j } ! [ { "論文" , $i/title } !
    { "内容" , ⟨ $i/subtitle , $i/subcontent ⟩! }
  ]! ]!
FOR
  $i in db('bib.xml')/biblio/*,
  $j in $i//( author | editor )
```

(2) 自動整形出力

自動整形はオペランドと自動レイアウト演算子との組合せによって部分 XML に対して自動変換を行うものであり，オペランドで指定する相対パス表現式は FOR 句で指定するパス表現式を連結して不完全なパス表現式となる必要がある．自動レイアウト演算子によって部分 XML を自然な表構造を持つ HTML 表形式や，階層構造を単純に実現するインデント形式に自動変換ができる．たとえば，以下に示す PPX 6 は GENERATE 句のレイアウト式で$i/author と&− 自動レイアウト演算子を組み合わせて記述し，部分 XML（paper 要素ノード以下にある XML データ）をインデント形式で表示する．この結果を**図 13** に示す．

```
PPX 6:
GENERATE html
  ⟨ $i/title ! ⟨ $i/date , ⟨ &− ( $i/( author | editor ) ) ⟩!
  ⟩! ⟩!
FOR
  $i in db('bib.xml')//( paper | book )
```

3.3　PPX 変換モデル

PPX の構文規則を付録 A.1 に示す．レイアウト式にはオペランド，単項式，二項式，または条件分岐式がある．

- オペランドは変数名とパス表現式の組合せで表現するか，または任意のレイアウト式を括弧 {} でくくったものである．

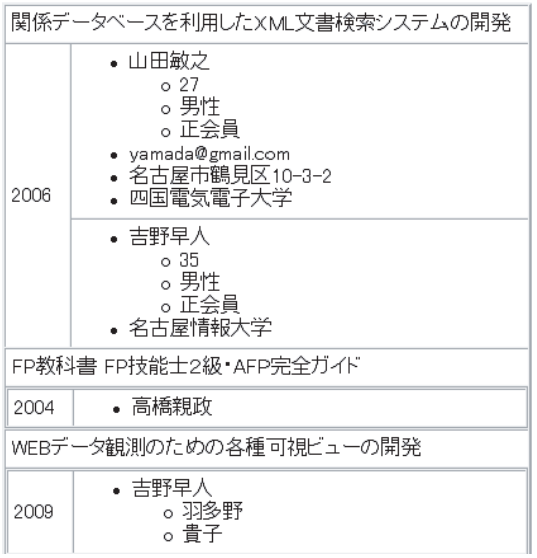


図 13　PPX 6 によるフォーマット結果
Fig. 13　Format results by PPX 6.

- 単項式は自動演算子，または反復演算子を用いて表現する．
- 二項式は 2 つのレイアウト式と二項演算子の組合せである．
- 条件分岐式は if-then-else 構文に論理式と 2 つのレイアウト式を組み合せたものであり，else 句は省略可能である．

PPX のレイアウト式の解析から変換パターンのそれぞれのデータ構造を表したものを条件分岐木と呼び，それらを再結合したものを変換スキーマ木と呼ぶ．これらは PPX 処理系における変換の内部モデルとなる．

条件分岐木：条件分岐木は中間変換木であり，内部ノードはレイアウト式に含まれる if 句の条件に対応する．葉はそれぞれのパターンに対応する if 句を含まないレイアウト式である．

ここでソース木（PPX 問合せ文のソースを構文解析して得られる木構造表現）から次項に述べる変換スキーマ木を直接作成することも可能である．しかし，実装においてソース木から条件分岐木という中間結果を経て変換スキーマ木を作成するようにしたため，条件分岐木の導入も述べている．また，次章の 4.2.1 項で述べる技術的なキーポイントの 1 つである目印の利用について条件分岐木との対応関係を明確にするためでもある．

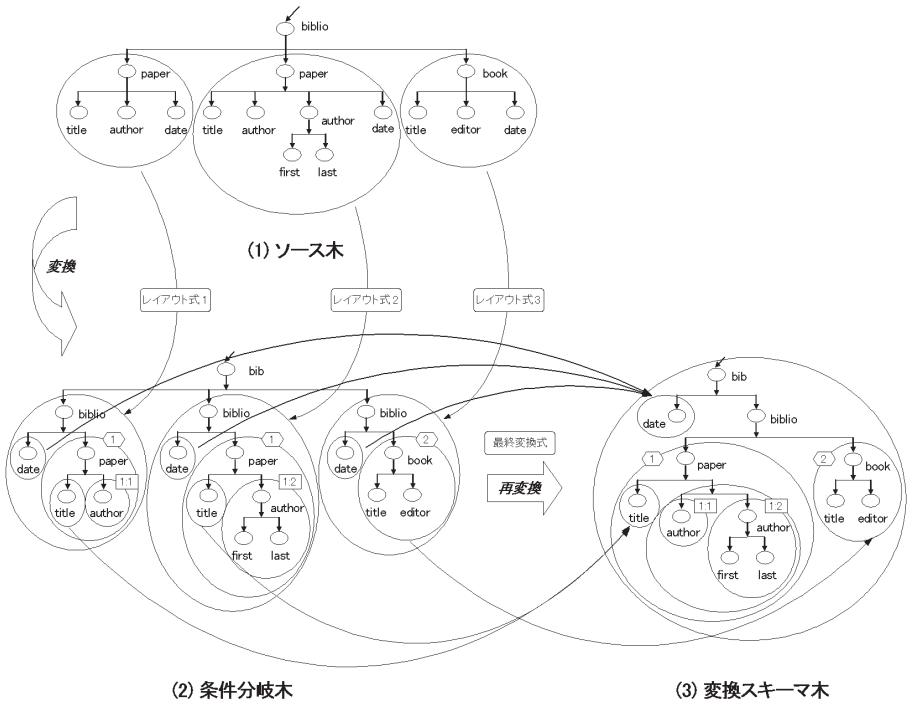


図 14　PPX 4 の変換モデル
Fig. 14　The transformation model of PPX 4.

変換スキーマ木：変換スキーマ木は出力結果の構造を表現し，条件分岐木において個別に表現されたレイアウトを再結合することによって表現される．

具体的な例として前節で示した PPX 4 の変換モデルを**図 14** に示す．条件分岐木では異なるレイアウトパターンに対応する XML データに対してそれぞれ構造化を行うことを図 14 の左側の下の (2) で示している．また，変換スキーマ木では条件分岐木によってそれぞれ構造化を行った XML データに対してそれぞれの結果を 1 つに結合することによって再構造化を行うことを右側の下の (3) で示している．

4. 実　　装

4.1 全体の流れ

本システムは**図 15** で示したように構文解析部，XQuery 生成部，レイアウト生成部，木構造生成部と出力媒体生成部からなる．図の中で実線は処理の流れを，点線はデータの流れを表している．PPX の問合せ文は構文解析部によってレイアウト式と XML データアクセス式に分ける．XQuery 生成部でこの XML データアクセス式から XQuery 文を生成し，それに基づいてフラットなリスト構造を持つ XML データが得られる．一方，レイアウト式から条件分岐木を生成した後，これらを再結合して変換スキーマ木を生成する．その後，変換スキーマ木は木構造生成部とレイアウト生成部にそれぞれ渡される．木構造生成部では変換スキーマ木に従ってフラットなリスト構造を持つデータの再構造化を行う．ここで不完全なパス式によって探索された部分 XML に含まれているデータは 1 つのタプルとして扱い，その部分のデータに対して構造化は行わず，元の構造まま出力媒体生成部に渡される．出力媒体生成部では変換スキーマ木に基づいてタグ付与方法を生成する．最後にこれらのデータとタグ付与方法は出力媒体生成部に渡されタグ付けを行い，HTML に変換される．

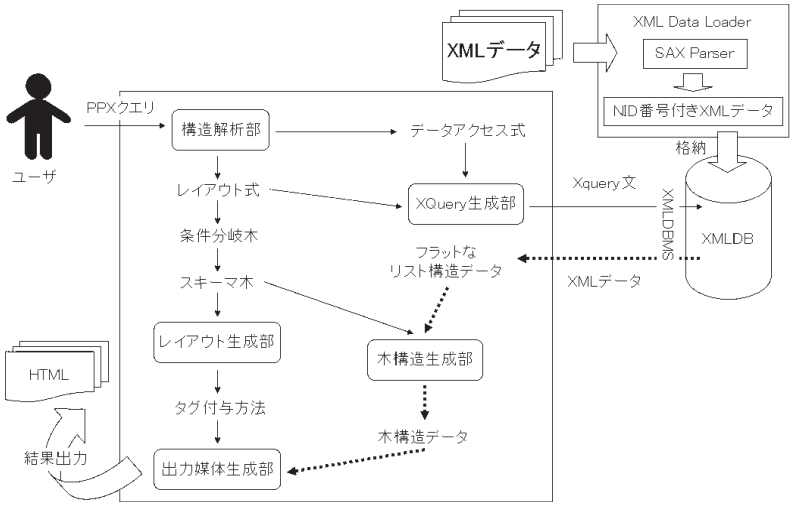


図 15　システムのアーキテクチャ
Fig. 15　Overview of system architecture.

4.2 内部処理アルゴリズム

本節では出力構造化を行うアルゴリズムの詳細を述べる．PPX 処理系の内部処理アルゴリズムは以下の 5 つのプロセスからなる．これらについて処理を行う順番に説明する．

- 構文解析部
- XQuery 生成部
- レイアウト生成部
- 木構造生成部
- 出力媒体生成部

4.2.1 構文解析部

PPX の問合せは構文解析部においてレイアウト式と XML データアクセス式の 2 つに変換される．すなわち，PPX の GENERATE 句で記述したレイアウト定義がレイアウト式であり，GENERATE 句，FOR 句，WHERE 句などで記述したパス式と検索条件から構成されたものが XML データアクセス式である．これらに対する変換の詳細と具体例を次に示す．

(1) レイアウト式の処理

レイアウト式はまず if 句による条件分岐のケースごとに分けた if 句を含まないレイアウト式群に変換される．この結果が条件分岐木である．次にそれぞれのケースを識別する目印を付けたうえでこれらのレイアウト式を結合する．この結果，得られるのが変換スキーマ木とそれに対応する目印付き変換レイアウト式である．たとえば，図 14 で示す変換モデルは 3.2 節で説明した PPX 4 の GENERATE 句のレイアウト式から条件分岐に基づいて 3 つのレイアウト式から構成された (2) の条件分岐木に変換される．次に示すものが条件分岐木に対応するレイアウト式であり，最初の 2 つが paper，3 番目が book に対応する．paper については name がテキストノードを持つ場合と要素ノード（first，last）を持つ場合に分かれている．

```
レイアウト式 1：/**（paper データを処理）**/
  [ $i//date , [ $i//title , [ $j ]! ]! ]!
レイアウト式 2：/**（paper データを処理）**/
  [ $i//date , [ $i//title , [ $j/first , $j/last ]!
  ]! ]!
レイアウト式 3：/**（book データを処理）**/
  [ $i//date , [ $i//title , [ $j ]! ]!
```

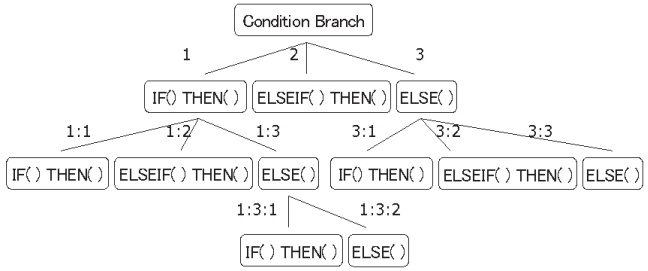


図 16　Dewey Order による目印付け
Fig. 16　Dewey Order label method.

その後，条件分岐木を目印を付けて結合し，(3) の変換スキーマ木に変換される．次の示すものが変換スキーマ木に対応する目印付き変換レイアウト式である．

```
[ $i//date , [
   ( ⟨ 目印 1⟩[ $i//title , [
      ( ⟨ 目印 1:1⟩[ $j ]! ) ! ( ⟨ 目印 1:2⟩[ $j/first , $j/last ]! )
   ]! ]! )
   !
   ( ⟨ 目印 2⟩[ $i//title , [ $j ]! ]! )
]! ],
```

変換スキーマ木に変換する際にネストを含む if-then-else 構文では then 句から生成するレイアウト式や else 句から生成するレイアウト式を識別するために目印付けを行う．目印は**図 16** で示した Dewey Order [12] のラベルを付ける．

(2) XML データアクセス式

XML データアクセス式は GENERATE 句と FOR 句で指定したパス式から得られるものであり，WHERE 句が存在する場合にはその条件も加える．この式に従ってソースとなる XML データを取り込む．たとえば，PPX 4 の例では以下で示した部分が XML データアクセス式となる．

```
$i in db('bib.xml')/biblio/*,
$j in $i//( author | editor )
```

4.2.2　XQuery 生成部

XQuery 生成部では XML データアクセス式とレイアウト式を参照して XQuery 文を生成する．その XQuery 文に基づいて探索された XML データからフラットなリスト構造を作り，中間的な出力が得られる．ここで PPX 4 からは次に示すような XQuery 文が自動生成される．

```
FOR
   $i in db2-fn:xmlcolumn('bib.xml')/biblio/*,
   $j in $i//( author | editor )
RETURN
    if ( $i//author )
    then (
      if ( $j/text() )
      then
        ⟨result⟩
        {$i//date/text()}, {$i//title/text()}, {$j/text()}
        ⟨/result⟩
      else
        ⟨result⟩
        {$i//date/text()}, {$i//title/text()}, {$j/first/text()}, {$j/last/text()}
        ⟨/result⟩
    )
    else
      ⟨result⟩
      {$i//date/text()}, {$i//title/text()}, {$j/text()}
      ⟨/result⟩
```

生成された XQuery 文によって次に示すフラットなリスト構造を持つ XML タグが外されたデータが探索される．ここでデータも図 16 の目印に基づいて目印付けを行う．なお，このデータでは簡潔性のために省略しているが実際には要素ノードの順序を示す NID [⋆1] がさらに追加される．

⋆1 ここで NID はデータ順に出力するため，目印は 4.2.1 項で述べるデータにレイアウト方法を付与するためである．

- フラットなリスト構造を持つデータ：

(("2006"(⟨ 目印 1⟩("情報管理システムの開発"(⟨ 目印 1:1⟩("山田二朗")))))
("2005"(⟨ 目印 1⟩("索引技術の高速化方法の提案"(⟨ 目印 1:1⟩("田中桃子")))))
("2006"(⟨ 目印 1⟩("情報管理システムの開発"(⟨ 目印 1:2⟩("吉沢" "貴子")))))
("2006"(⟨ 目印 2⟩("XML データベースハンドブック" "村田徹也")))
("2007"(⟨ 目印 2⟩("SQL クエリ言語ハンドブック" "天野一郎"))))

その後，これらのデータと変換スキーマ木が木構造生成部へ渡される．

4.2.3　木構造生成部

木構造生成部ではフラットなリスト構造のデータに対して構造化を行う．

- 条件分岐木による予備グループ：先の中間的な出力として得られたリストに対して条件分岐木に基づいて括り出しの処理を行うことによって次のような階層的構造を持つリストに変換する．

(("2006"(⟨ 目印 1⟩("情報管理システムの開発"(⟨ 目印 1:1⟩("山田二朗"))(⟨ 目印 1:2⟩("吉沢" "貴子")))))
("2005"(⟨ 目印 1⟩("索引技術の高速化方法の提案"(⟨ 目印 1:1⟩("田中桃子")))))
("2007"(⟨ 目印 2⟩("SQL クエリ言語ハンドブック"(("天野一郎")))))
("2006"(⟨ 目印 2⟩("XML データベースハンドブック"(("村田徹也"))))))

- 変換スキーマ木による最終グループ：また，条件分岐木によって得られたリストを変換スキーマ木に基づいて先と同様の処理を行い，次のような出力媒体生成部へ渡す結果を得る．

(("2007"(⟨ 目印 2⟩("SQL クエリ言語ハンドブック"(("天野一郎")))))
("2006"(⟨ 目印 1⟩(("情報管理システムの開発"(⟨ 目印 1:1⟩("山田二朗"))(⟨ 目印 1:2⟩("吉沢" "貴子")))))(⟨ 目印 2⟩("XML データベースハンドブック"(("村田徹也")))))
("2005"(⟨ 目印 1⟩("索引技術の高速化方法の提案"(⟨ 目印 1:1⟩("田中桃子"))))))

このようなグルーピングは先の構文解析部において生成されたレイアウト式に基づいてそれと同様の階層的構造を持つように行われる．このような処理によってフラットなリスト構造は階層的構造を持つ 1 本のリストに変換され，出力媒体生成部に渡される．

4.2.4　レイアウト生成部

レイアウト生成部では構文解析部で分離されたレイアウト式から生成した目印付きの変換スキーマ木に基づいてタグ付与方法を生成し，出力媒体生成部に渡される．ここで 4.2.1 項の変換スキーマ木を表した目印付きの変換レイアウト式を例とすると，目印に基づいてそれに対応するデータに HTML タグを付与する方法は次のとおりである．

タグ付与方法 `= [ $i//date , [ <目印 1> | <目印 2> | <目印 1> + <目印 2> ]! ]!`

目印 1 = [$i//title , [⟨ 目印 1.1⟩ — ⟨ 目印 1.2⟩ — ⟨ 目印 1.1⟩ + ⟨ 目印 1.2⟩]!]!
目印 1.1 = [$j]!
目印 1.2 = [$j/first , $/last]!
目印 2 = [$i//title , [$j]!]!

ここで "|" はいくつかのレイアウト式の中でいずれか 1 つが出現する場合を表し，"+" はいくつのレイアウト式がともに出現する場合を表す．たとえば，最終グループ化されたリスト中にあるデータ

("2006"(⟨ 目印 1⟩(("情報管理システムの開発"(⟨ 目印 1:1⟩("山田二朗")⟨ 目印 1:2⟩("吉沢" "貴子")))))(⟨ 目印 2⟩("XML データベース書籍"(("村田徹也")))))

に対してはタグ付与方法の中でこれと同じ目印を持つ次のようなタグ付与方法を生成し，それに基づいてレイアウトを行う．

```
[ $i//date , [
    [ $i//title , [ [ $j ]! ! [ $j/first , $j/last ]! ]! ]!
  !
    [ $i//title , [ $j ]! ]!
]! ]!
```

4.2.5　出力媒体生成部

出力媒体生成部ではタグ付与方法に基づいて木構造生成部から生成された階層構造を持つリストにタグを付与し，指定された媒体への出力に変換する．GENERATE 句で指定した出力媒体が HTML の場合は HTML ソースファイルを生成し，出力する．

5.　実験・評価

提案した PPX の有用性を示すため，W3C の Query Use Cases [18] に適用し，表現能力を確認した．また，生産性について XQuery，XSLT と比較を行う．ここで生産性については XML データ（ワンソース）から用途に応じて見やすく加工し，複数の異なる表構造を持つ HTML の生成（マルチユーズ）を実現する XML データの再利用の効率に重点をおいた評価を行う．すなわち，同一の XML データに対して HTML 化する作業をある一定時間にそれぞれの言語（ここでは PPX，XSLT，XQuery）を用いて開発者が開発を行うという状況を仮定する．たとえば，DBLP [14] のような大規模な XML データを著者別，または年度

表 1　XML データセット
Table 1　XML data set.

	ACMSIGMOD	DBLP	SHAKES
サイズ	467 KB	127 MB	7.5 MB
ノード数	15,263	3,736,406	179,690
最大深度	6	6	7

別，または論文誌別などに様々な用途に応じて見やすく加工，および複数の異なる表構造を持つ HTML に出力する変換能力の比較を通じて PPX の生産性を評価する．

5.1 実験環境

実験は CPU：Pentium III 1.4 GHz Dual，メモリ：2 GB，OS：Windows XP のマシンで行う．また，PPX と XQuery の処理系として DB2 Version 9 を使用し，XSLT 1.0 と XSLT 2.0 の処理系として XMLSpy [15] を使用した．

5.1.1 XML データセット

実験で主に利用した 3 種類のデータセット（DBLP，ACMSIGMOD，SHAKES）を**表 1** に示している．そこで 1 番目と 2 番目のデータセットは The University of Washington of XML repository [16] で提供する DBLP Computer Science Bibliography の dblp データと SIGMOD Record の SigmodRecord データである．3 番目の SHAKES [17] はボサックシェイクスピアコレクションの Shakespeare データである．表における XML ノード数は属性ノードの数も含んでいる．最大深度は XML データにおける最も長い経路の長さである．

5.2 表現能力

ここでは PPX の実際の応用例でテスト XML データに適用し，XQuery や XSLT よりシンプルな表現で変換を表現できること，および W3C 問合せのユースケース[18] を用いてその表現能力を確認する．

5.2.1 実データを用いた評価

PPX 問合せを表 1 のデータに適用し，変換能力を評価した．用いる PPX 問合せのリストの一部を**表 2** に示す．問合せに付与されている “QXY” という名前の “X” は “S”（Shakespeare データ），“D”（dblp データ），“A”（SigmodRecord データ）のうちのいずれかを表す．“Y” は問合せタイプの番号である．

(1) Shakespeare データの HTML 化

Shakespeare データでは要素ノードの現れる順序が重要である．たとえば，SPEECH 要素ノードの子要素ノードである LINE 要素ノードはスピーカがスピーチした内容順に現れ

表 2　PPX 問合せ
Table 2　PPX queries.

No.	PPX
QS1	GENERATE html [$i/SPEAKER]! FOR $i in db('Shakespeare.xml')/PLAY/ACT/SCENE/SPEECH
QS2	GENERATE html < $i/TITLE , < $j/SPEAKER , < $l >! >! >! FOR $i in db('Shakespeare.xml')/PLAY, $j in $i/ACT/SCENE/SPEECH/*, $l in $j/LINE
QD1	GENERATE html < $i/title , if ($i/mastersthesis) then (< $i/school , < $j >! >!) else if ($i/article) then (< $i/journal , < $j >! >!) else (< $i/year , < $j >! >!) >! FOR $i in db('dblp.xml')/dblp/(mastersthesis | article | phdthesis)/*, $j in $i/(author | editor)
QD2	GENERATE html [$i/titlel , < if ($i/text()) then ($i) else if ($i/last/@nid < $i/first/@nid) then ($i/last , $i/first) else ($i/first , $i/last) >!]! FOR $i in db('dblp.xml')/dblp/(mastersthesis | article | phdthesis)/*, $j in $i/(author | editor)
QA1	GENERATE html [$i/title , $i/initPage , $i/endPage , < $j >!]! FOR $i in db('SigmodRecord.xml')//article, $j in $i//author
QA2	GENERATE html < $j ! [$i/title , $i/initPage , $i/endPage]! >! FOR $i in db('SigmodRecord.xml')//article, $j in $i//author

る．このような XML データに対して PPX の問合せ（QS1，QS2）を適用し，次のことを評価した．1）重複を排除し，要素ノードの現れる順を並べ替えて変換を行うこと．2）重複を許し，要素ノードの現れる順に変換を行うことでユーザの要求に対応できるのかを確認する．ここで QS1 は元の順序を無視する反復演算子を用いて探索された SPEAKER 要素ノードのテキストノードの値に対して重複を排除し，アルファベットの昇順に並べ替えて出力した．また，QS2 は順序を保存する反復演算子を用いてそれぞれの TITLE 要素ノードのテキストノードの値，および個々の TITLE 要素ノードが持つそれぞれの SPEAKER 要素ノードのテキストノードの値，さらに個々の SPEAKER 要素ノードが持つ LINE 要素ノードのテキストノードの値に対して重複を許し，元の順序のとおりに出力した．

(2) DBLP データの HTML 化

DBLP データの特徴は非定型木構造を持つ．たとえば，article に関するデータや phdthesis に関するデータなど異なる種類のデータが混じっており，それらのデータ構造も異なる．また，元の dblp データで一部の author 要素ノードに子要素ノードを異なる順序（たとえば，last，first の順や first，last の順）で表現している．このような XML データに対して PPX の問合せ（QD1，QD2）を適用し，次のことを評価した．1）異なる非定型木構造に対してそれぞれの異なるレイアウト方法で変換を行うこと．2）同じ非定型木構造内にある異なる順序で出現する共通の要素ノードに対してそれぞれの異なるレイアウト方法で変換を行うことができるのか確認する．ここで QD1 は条件分岐を用いて出現する要素ノード（mastersthesis，article，phdthesis）をルートノードとする部分 XML の非定型木構造に対して異なるレイアウト方法の適用によって出力を行った．また，QD2 も同様に条件分岐を用いて同じ非定型木構造でありながら異なる順序で出現する author 要素ノードの子要素ノード（last，first）に対してそれらの要素ノードが持つ NID によって元の順序のとおりに出力を行う．すなわち，この問合せでは last 要素ノードの NID 値が first 要素ノードの NID 値より小さいならば last，first の順に出力し，大きいならば first，last の順に出力した．しかし，author 要素ノードがさらに子要素ノードを持つ場合（たとえば，last，first，middle）は多数の if 文を指定しなければならない．

(3) SigmodRecord データの HTML 化

SigmodRecord データに対して曖昧なパス式を利用した PPX の問合せ（QA1，QA2）を適用し，次のことを評価した．1）探索されたフラットなリスト構造のデータに対してレイアウト式の変更によってユーザの意図に応じて自由に再構造化された HTML 表に変換できるのか確認する．ここで QA1 はそれぞれの article 要素ノード以下にある 3 つの（title，initPage，endPage）要素ノードのテキストノードの値と author 要素ノードのテキストノードの値を 1 組にしたフラットなリスト構造を作り出し，それに対して author を title，initPage，endPage でグルーピングを行い，再構造化された HTML 表を出力した．また，QA2 は author 要素ノードと 3 つの（title，initPage，endPage）要素ノードのテキストノードの値を 1 組にしたフラットなリスト構造を作り出し，それに対して title，initPage，endPage を author でグルーピングを行い，再構造化された HTML 表を出力した．

表 3　表現能力の確認結果
Table 3　Expression abilities.

ユースケース	ケース数	表現可能	表現不可能
XMP	12	12	0
TREE	6	4	2
SEQ	5	4	1
R	18	17	1
SGML	10	10	0
STRING	4	2	2
NS	8	8	0
PARTS	1	0	1
STRONG	12	4	8
合計	76	61	15

5.2.2　W3C 問合せのユースケース

W3C の Query WG で XML の問合せのユースケース[18] が整理されており，それと同等な PPX を表 1 で示した XML データに適用し，XQuery と同じ結果データを得られるか確認する．W3C のユースケースは XQuery によって XML から XML への変換を行うものであり，PPX の行う XML から HTML への変換ではないが構造変換能力のベンチマークとしてこれを採用した．

たとえば，次の XQuery は**表 3** の左で示す代表例（“XMP”）の問合せ Q2 である．

Use Case “XMP”: Q2 の記述例：
⟨results⟩ {
for $a in db2('bib.xml')//book,
$b in $a/title, $c in $a/author
return ⟨result⟩{ $b } { $c }⟨/result⟩
} ⟨/results⟩

この問合せでは表題（title）と著者（author）を 1 組にしたフラットなリストを作成する

が，それと同じ結果データを得られる PPX は次に示す．

PPX の記述例：

```
GENERATE html
  [ $a/title , $b ]!
FOR
  $a in db2('bib.xml')//book,
  $b in $a//author
```

このような方法で実験した結果は表 3 で示したとおり，15 件について表現ができかなった．この 15 件はすべてユーザ定義関数を用いる FNPARM ユースケースと呼ばれるもので，その以外のすべてが PPX によって表現可能であることを確認した[19]．その中で条件分岐を利用した 3 個の XQuery に対して PPX でも条件分岐指定によって表現することができた．

5.3 変換能力

ここでは変換を行うための指定方法や記述量などについて XQuery や XSLT と比較する．

5.3.1 指定方法

PPX と同等の変換を行う XSLT，XQuery などの指定方法を比較する．たとえば，5.2.2 項で示した PPX の記述例に基づき，レイアウト式の$b に 1 個の反復演算子（[ ]!）を追加することによって得られる表構造を生成する変換された PPX の記述例を 1 章のはじめにも示している．この PPX のレイアウト式で追加された反復演算子のネストによって author を title でソート，グルーピング，および重複排除する．これに相当することを行うために 1 章で既存技術の問題点としてあげた例に示した XQuery や XSLT は直感的とはいい難く複雑である．また，XML データに対するソート，グルーピング，および重複データの処理を行う際に PPX は XSLT，XQuery などと次の点が異なる．

(1) ソート：XQuery や XSLT などは SORT BY 関数を用いて昇順，または降順でソートを行う．たとえば，XSLT や XQuery は重複のないノード集合を基本として定義されている XPath 1.0[10] を用いて XML データを読み込むときにはデータの順序を無視する．一方，XPath 2.0[11] では XML データを読み込むときにデータの順序を持つ．この場合，データの順序を並べ替えるためにはソート関数を利用する．これに対して PPX は 2 種類の反復演算子を利用して値の昇順，または降順で並べ替えるソートや元の順序のままの配列を行う．

(2) グルーピング：XQuery は GROUP BY 関数を利用してグルーピングを行う．XSLT 1.0 はグループ化をサポートしていないので Muenchian 方法[13] を利用する．また，XSLT 2.0 はグループ化することを簡略化するために XSL:FOR-EACH-GROUP 要素を使用して 1 群の XML ノードをいくつかの基準に基づいてグループ化し，そのような選択処理によって形成されたグループごとに処理する．これに対して PPX は 2 種類の反復演算子のネスト指定によってグルーピングを行う．

(3) 重複データの処理：XQuery，XSLT は DISTINCT-VALUES 関数を使って XML データの重複排除を行う．ここで XSLT 1.0 の問題の 1 つは XML ノード・グループに対して直接 SELECT DISTINCT を実行できないことである．このような変換を行うために対象になる要素名のすべての XML ノードを選択してそれを要素名ごとにソートする．さらに XSL:IF ブロックを使用して処理される要素名がそれまでに処理した XML ノードと同じ要素名かどうかを判別する必要がある．これに対して PPX は 2 種類の反復演算子を用いて重複データの処理を行う．すなわち，元の順序を無視する反復演算子によって要素ノードの現れる順序を並べ替える出力は要素名の同一，かつテキストの同値なら自動的に重複排除を行う．それに対して元の順序を保存する反復演算子によって現れる要素ノードの順序のとおりの出力が得られる．

5.3.2 記述量

PPX と同等の変換を行う XQuery や XSLT の記述量を比較する．前項で例として示した XQuery は HTML タグを埋め込んでない状況を示している．これに対して問合せ式に HTML タグを埋め込むと記述量は増える．たとえば，PPX ならばレイアウト式：『 年度 , 題目 , 著者 』によって生成する表構造からレイアウト式：『 年度！題目 , 著者 』によって生成する表構造への出力は 1 つの結合演算子だけの変更で済む．しかし，XQuery の場合は問合せ式に埋め込んだ HTML タグを**表 4** で示したようにレイアウト 1 からレイアウト 2 のように変更しなければならない．また，レイアウト式：『 [年度！ [題目 , [著者]!]!]! 』のように反復演算子のネストによるグルーピングの変更の場合，XQuery ではさらにずっと大きな記述量が必要となる．すなわち，XQuery や XSLT などは処理を行うデータに対してソートを行う関数（SORT BY），グルーピングを操作する関数（GROUP BY や XSL:FOR-EACH-GROUP），および重複排除を行う関数（DISTINCT-VALUES）などを利用する以外，HTML タグの記述もする．それに対して PPX は出力する HTML 表構造のテンプレートが既定のため，HTML タグを記述せずに HTML への変換を行う．また，反復演算子を利用してデータに対するソート，グルーピング操作と重複出現するデータの排除を行うことが記述量の大幅削減に貢献している．この結果，XQuery や XSLT などは PPX より多くの記述が必要になる．**表 5** では図 10 の出力を行う PPX，XSLT，XQuery の記述に必要な文字数を比較した 1 例の結果を示している．

表 6　指定方法
Table 6　Specification methods.

	PPX	XQuery	XSLT 1.0	XSLT 2.0
ソート	反復演算子	SORT BY	SORT BY	SORT BY
グルーピング	反復演算子	GROUP BY	MUENCHIAN GROUPING	XSL:FOR-EACH-GROUP
重複データの処理	反復演算子	DISTINCT-VALUES()	DISTINCT-VALUES()	DISTINCT-VALUES()
データ装飾	装飾演算子	XSL-FO や CSS	XSL-FO や CSS	XSL-FO や CSS
異なる表構造	レイアウト式の変更	問合せ式の変更	スタイルシートの変更	スタイルシートの変更

表 4　レイアウト 1 からレイアウト 2 への変更
Table 4　Change from layout 1 to layout 2.

レイアウト 1	レイアウト 2
年度，題目，著者	年度！題目，著者
<table>	<table><tr>
<tr>	<td>
<td> 年度 </td>	<table>
<td> 題目 </td>	<tr>
<td> 著者 </td>	<td> 年度 </td>
</tr>	</tr><tr>
</table>	<td> 題目 </td>
	</tr>
	</table>
	</td>
	<td> 著者 </td>
	</tr></table>

表 5　クエリの文字数の比較
Table 5　Comparison of character amount.

	PPX	XQuery	XSLT 1.0	XSLT 2.0
図 10 の HTML 生成	94	391	745	836

5.3.3　ワンソース・マルチユーズ

PPX は XML データに対して演算子の組合せによって容易に木構造を再構成したり，オペランドの順番や反復演算子の変更によってネストの構造を変えたり，1 つの HTML 表にハイパーリンクされた複数の HTML 表を生成したりなどレイアウト式の指定によって様々な表構造を持つ HTML を出力することができる．たとえば，2 つの要素ノード（title と author）が持つテキストノードの値に対して様々な表構造を持つ HTML への出力を行うとする．PPX ならば**図 17** でその一部を示しているとおり，オペランド$i/title と$j/author

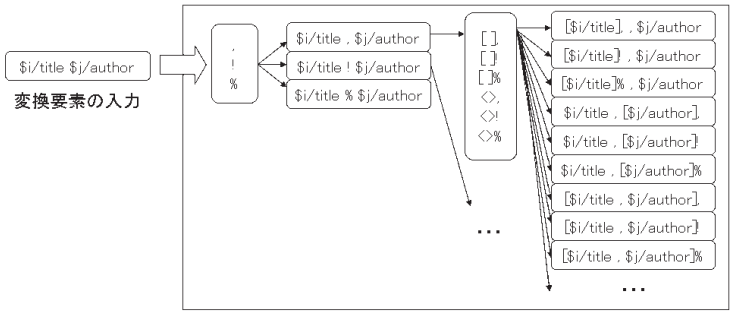


図 17　異なるレイアウト式の生成
Fig. 17　Generation of variety of layout expressions.

に演算子を組み合わせて指定できるレイアウト式の数は合計で 698 種類である．すなわち，698 個の異なる表構造を持つ HTML を作れる．また，オペランドが追加されると指定できるレイアウト式の数は急激に増える．

これに対して XSLT や XQuery などが同様なことをすると元の問合せ式やスタイルシートは簡単に再利用できず，埋め込んだ HTML タグなどを中心に何度も書き直す必要がある．この際に XQuery は FOR 句，LET 句などを RETURN 句の中で何回もネストして指定し，XML データのどの要素の何番目の子要素という具合に細かく指定して検索・抽出しなければならない．XSLT はパターンマッチングで指定した変換元をどのように変換するというテンプレートルール間の競合解消を基本とする変形プロセスの理解が要求される．PPX，XQuery，XSLT などが XML データから HTML への変換を行う際の異なる点を**表 6** で一部をまとめて示している．

これによってワンソース・マルチユーズを実現するための指定方法の複雑性や同等の変換

を行う記述量の比較などから XQuery や XSLT より PPX の生産性が高いと考えられる．

6. まとめと今後の課題

本論文では XML データから HTML への変換を行うため，拡張 TFE による簡潔な記述方法を提供する XML 整形出力言語 PPX を提案した．本言語は重複を許し，要素ノードの現れる順に変換・出力を行うこと，条件分岐を用いてイレギュラ XML データから構成された非定型木構造に対して変換を行うこと，不完全なパス表現式によって探索された部分 XML に対して自動変換を行うことの 3 点が RDB を対象とした先行研究の SuperSQL と最も異なる点である．実験では W3C の Query Use Cases に PPX を適用し，ユーザ定義関数の処理のための FNPARM ユースケースを除いたすべてについて PPX によって等価な結果データを得られる表現が可能であることを確認した．また，大規模な XML データを用途に応じて見やすく加工し，複数の異なる表構造を持つ HTML をユーザに提供することについて XSLT や XQuery よりも生産性が高いことを示した．

今後の研究課題としては XML データの非定型木構造をそれぞれの異なる木パターンとして抽出し，木パターンごとのインスタンスの統計情報などに基づいて自動変換の最適化を行うレイアウト自動決定アルゴリズムの提案が必要となる．また，PPX をこれと等価な XSLT に自動変換する方法の開発により定型化した変換を高速に実行するシステムの開発に取り組みたいと考えている．

謝辞　本研究について様々な貴重なコメントをくださったメタレビューアの鬼塚真氏（日本電信電話（株）サイバースペース研究所），および査読者の方々に感謝いたします．

参 考 文 献

1) Clark, J.: XSL Transformations (XSLT), Version 1.0. W3C Recommendation 16 November 1999. W3C (1999).
2) Kay, M.: XSL Transformations (XSLT), Version 2.0. W3C Recommendation 27 January 2007. W3C (2007).
3) XQuery 1.0: An XML Query Language. http://www.w3c.org/TR/xquery/
4) Pawson, D.: XSL-FO: *Making XML Look Good in Print*, O'Reilly, United States (2002).
5) Lie, H., Bos, B., Lilley, C. and Jacobs, I.: Cascading Style Sheets, Level 2. http://www.w3.org
6) Jin, Z. and Toyama, M.: A prototype implementation of PPX: Pretty Printer for XML, *The 4th International Conference on Internet and Web Applications and Services* (*ICIW*), pp.149–156 (2009).
7) Seto, T., Nagafuji, T. and Toyama, M.: Generating HTML Sources with TFE Enhanced SQL, *ACM Symposium on Applied Computing* (*SAC'97*), ACM, pp.96–105 (1997).
8) Hosoya, H. and Pierce, B.C.: XDuce: A Statically Typed XML Processing Language, *ACM Trans. Internet Technology*, pp.117–148 (2003).
9) Toyama, M.: SuperSQL: An Extended SQL for Database Publishing and Presentation, *Proc. ACM SIGMOD*, pp.584–586 (1998).
10) XPath 1.0: XML Path Language (XPath) 1.0. http://www.w3.org/TR/xpath/
11) XPath 2.0: XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20
12) Tatarinov, I., Viglas, S.D., Beyer, K., Shanmigasundaram, J., Shekita, E. and Zhang, C.: Storing and Querying Ordered XML Using a Relational Database System, *Proc. ACM SIGMOD 2002*, pp.204–215 (2002).
13) Tennison, J.: Grouping Using the Muenchian Method (2007). Available from http://www.jenitennison.com/xslt/grouping/muenchian.html
14) DBLP data set. ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz
15) XMLSpy. http://www.Altova.com/XMLSpy
16) The University of Washington XML repository. http://www.cs.washington.edu/research/xmldatasets
17) http://metalab.unc.edu/bosak/xml/eg/shaks200.zip
18) Chamberlin, D., Fankhauser, P., Florescu, D., Marchiori, M. and Robie, J.: XML Query Use Cases. http://www.w3.org/TR/xquery-use-cases/
19) 実験結果．http://www.db.ics.keio.ac.jp/PPX/

付　　録

A.1 拡張 TFE の構文規則

```
<レイアウト式> ::= <単項式> | <二項式> | <条件分岐式>
<オペランド> ::= <変数/パス表現式> | {<レイアウト式>}
<単項式> ::= <オペランド> | <自動式> | <反復式>
<自動式> ::= <自動演算子> <単項式>
<反復式> ::= [<レイアウト式>]<結合演算子>
<二項式> ::= <オペランド> <二項演算子> <レイアウト式>
<条件分岐式> ::= if <条件式> then <レイアウト式> [else <レイアウト式>]
<条件式> ::= <論理式>
<論理式> ::= <論理単項式> | <論理演算子> | <論理式>
```

```
<論理単項式> ::= <論理定数> | <論理関数> | <比較式>
<演算子> ::= <単項演算子> | <二項演算子> | <条件演算子>
<単項演算子> ::= <自動演算子>
<自動演算子> ::= &+ | &- | &
<条件演算子> ::= <論理演算子> | <比較演算子>
<論理演算子> ::= AND | OR | XOR | NOT
<比較演算子> ::= > | >= | < | <= | == | !=
<二項演算子> ::= , | ! | %
```

A.1.1　演算子の優先順位

高い

```
<>, <>! <>%
[], []! []%
, ! %
&+ &- &
> >= < <= == !=
NOT AND OR
```

低い

A.2　本言語がサブセットとして利用する XPath 式

```
[1] LocationPath ::= RelativeLocationPath
                   | AbsoluteLocationPath
[2] AbsoluteLocationPath ::= '/' RelativeLocationPath?
                           | AbbreviatedAbsoluteLocationPath
[3] RelativeLocationPath ::= Step
                           | RelativeLocationPath '/' Step
                           | AbbreviatedRelativeLocationPath
[4] Step ::= AxisSpecifier NodeTest Predicate*
           | AbbreviatedStep
[5] AxisSpecifier ::= AxisName '::'
                    | AbbreviatedAxisSpecifier
[6] AxisName ::= 'ancestor'
               | 'ancestor-or-self'
               | 'attribute'
               | 'child'
               | 'descendant'
               | 'descendant-or-self'
               | 'following'
               | 'following-sibling'
               | 'namespace'
               | 'parent'
               | 'preceding'
               | 'preceding-sibling'
               | 'self'
[7] NodeTest ::= NameTest
               | NodeType '(' ')'
               | 'processing-instruction' '(' Literal ')'
[8] Predicate ::= '[' PredicateExpr ']'
[9] PredicateExpr ::= Expr
[10] AbbreviatedAbsoluteLocationPath ::= '//' RelativeLocationPath
[11] AbbreviatedRelativeLocationPath ::= RelativeLocationPath '//' Step
[12] AbbreviatedStep ::= '.'
                       | '..'
[13] AbbreviatedAxisSpecifier ::= '@'?
```

- NodeTest は名前空間接頭辞以外，すべて可能である．
- Predicate は関数呼び出しが不可能である．

(平成 22 年 6 月 20 日受付)
(平成 22 年 10 月 4 日採録)

(担当編集委員　鬼塚 真)

金　　哲（学生会員）

平成 16 年慶應義塾大学大学院理工学研究科修士課程修了．現在，慶應義塾大学大学院理工学研究科博士課程在学中．データベースシステムの研究に従事．日本データベース学会，電子情報通信学会各学生会員．

遠山　元道（正会員）

慶應義塾大学理工学部情報工学科准教授．昭和 54 年慶應義塾大学工学部管理工学科卒業．昭和 59 年同大学大学院博士課程修了後，管理工学科助手，専任講師を経て現職．博士（工学）．平成 8 年 Oregon Graduate Institute 客員研究員．平成 10〜13 年科学技術振興事業団さきがけ研究 21「情報と知」領域研究員．主にデータベースの研究に従事．電子情報通信学会，日本データベース学会，ACM，IEEE 各会員．