学位論文　博士（工学）

XML整形出力言語PPXに関する研究

２０１０年度

慶應義塾大学大学院理工学研究科

金　哲

謝 辞

本研究をまとめるにあたり，御教授頂きました遠山元道准教授に深く感謝致します．また，本論文の審査委員を御承諾して頂き，懇切丁寧な御指導を頂いた増田靖教授，藤代一成教授，高田眞吾准教授に深く感謝致します．

本研究について様々な貴重なコメントを下さった日本電信電話株式会社サイバースペース研究所の鬼塚真氏に感謝いたします．

また，日本アイ・ビー・エム株式会社東京基礎研究所の小坂一也氏をはじめ，本研究について様々な貴重なコメントを下さった方々に感謝いたします．

最後に，留学生活や研究活動を応援してくれた常に明るく前向きな家族に心から感謝します．

目次

表 目 次

図 目 次

第 1 章

序論

1.1 研究背景と提案方法

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．データ，および文書の表現形式である XML データを Web ブラウザで表示する場合はそれを HTML 文書に変換する必要がある．

例えば，The University of Washington of XML repository [19] が DBLP Computer Science Bibliography のデータ（図 1.1 は DBLP の一部のデータを示している．）を提供している．コンテンツ情報だけなる XML データを著者別，年度別に見易くレイアウトできるように HTML に変換し，それを異なる媒体（ブラウザ，携帯電話，印刷出力）で表示することが考えられる．このような処理を本研究では XML データ変換・出版，または XML 整形出力と呼ぶ．

そこで 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ない，HTML 化することを考えてみる．これを実現する XQuery を次に示す．但し，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>
     <paper> { $a/title/text() } </paper>
     { for $c in distinct-values ( $b )
            sortby (author ascending)
       return <name> { $c/text() } </name>
     }
   </result>
} </results>
```

このクエリでは 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なう 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>
...
```

この例では二つの 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．

2 つの例で示すように XQuery や XSLT は複雑であり，一般ユーザがこれらのプログラミングを行なうことは難しい．

また，XML の研究分野で XML データの変換，および出版は格納，検索とともに重要な研究課題であり，手軽に HTML 化できる言語のニーズも高い．

本研究では XQuery や XSLT とは異なる方法で XML データから HTML への変換，出版を行なう XML 整形出力言語 PPX（Pretty Printer for XML） [9] を提案した．例えば，上述した XSLT，XQuery などと同等の変換を行なう PPX クエリを次に示す．

例 1：

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

また，次の PPX クエリは book 要素ノードを根ノードとする部分 XML を&自動整形演算子によって自動変換を行なう．

例 2：

```
GENERATE html
   [ & ( $a ) ]!
```

```
<dblp>
  <article>
  <authors>
  <author><last>Frank</last><first>Manola</first></author>
  </authors>
  <title>Object Data Model Facilities for Multimedia Data Types.</title>
  <journal>GTE Laboratories Incorporated</journal>
  <volume>TM-0332-11-90-165</volume>
  <year>1990</year>
  <cdrom>GTE/MANO90b.pdf</cdrom>
  </article>
  <phd>
  <authors>
  <author><last>Dimitrios</last><first>Georgakopoulos</first></author>
  <author>M. Tamer zsu</author>
  </authors>
  <title>PRPL: A Database Workload Specification Language, v1.3.</title>
  <year>1992</year>
  <school>Univ. of Wisconsin-Madison</school>
  </phd>
  <article>
  <authors>
  <author><first>Alejandro</first><last>Buchmann</last>
  <affiliation>Univ. of Wisconsin-Madison</affiliation></author>
  <editor>Frank Manola</editor>
  </authors>
  <title>SQL/Data System, General Information Manual.</title>
  <journal>IBM Publication</journal>
  <volume>GH24-5012</volume>
  <publisher>IBM Corporation, White Plains, NY</publisher>
  <year>1981</year>
  <cdrom>ibmTR/gh24-5012.pdf</cdrom>
  </article>
</dblp>
```

図 1.1: XML データインスタンス

FOR
$a in db2('bib.xml')/dblp

これら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．これはPPXでは縦横の接続やグルーピングを専用の演算子で表現するためクエリが簡潔になる反面，生成されるHTMLタグは既定のものとなる．

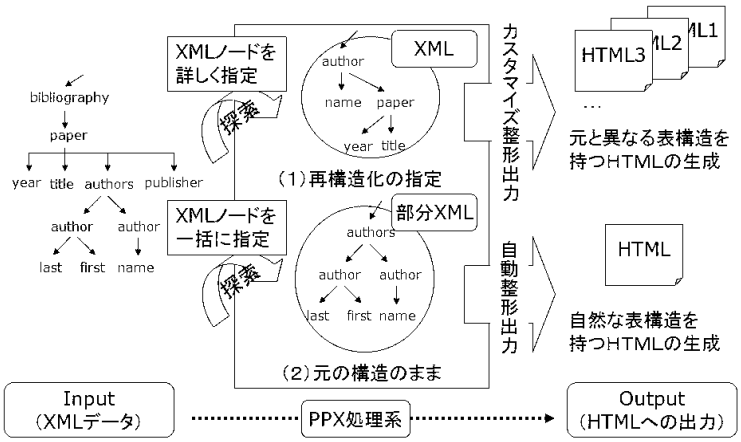


図 1.2: XML 整形出力

PPXによるXML整形出力は次の２種類のことを行なう．一つは例1で示すようにユーザがレイアウト式でXMLノードを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ない，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1.2(1)）である．もう一つはXMLデータのデータ構造やそれに対するレイアウト出力などを直接意識することなく，システムが自動的にHTML化して出力を行なうことを目的とする自動整形出力（図1.2(2)）である．即ち，例2で示すようにユーザがレイアウト式でXMLノードを一括指定によってシステムがXMLデータをオリジナルのデータ構造に基いて自然な階層構造[1]を持つHTML表形式やインデント形式[2]などで出力する．この二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．これらの変換方法について3章と4章でそれぞれ詳しく述べる．

1.2 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

[1] 4章の2.2項の初めでの部分で後述する．
[2] 2章の2.1.2項の自動整形演算子で後述する．

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ない，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) クエリ言語・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上手く用いることにより変換を行なう．SQL の影響を強く受けている XQuery はクエリ関数の作成や 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なう．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 [11] は正規表現のパターンマッチを行な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なう．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なう必要がある．提案する PPX は XML データの探索部分において 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 の一つの機能である XSL-FO は XSLT によって XML データの構造変換を行ない，その際に Web プ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．

1.3 論文構成

本論文の構成は以下の通りである．まず，次章で基本となる概念と PPX の言語構文について述べる．次に，3 章では XML データのカスタマイズ整形出力について述べる．また，4 章で PPX の自動整形出力について述べる．最後に，5 章で本論文の結論を述べる．

第 2 章

PPX（Pretty Printer for XML）クエリ言語

2.1 基本概念

2.1.1 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 [14] に基いており，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 B で示している．

完全なパス表現式は，テキストノードまでを表示したパス表現式である．例えば，1 章 1 項の例 1 の PPX の一番目の FOR 句で利用したパス表現式（//article）と GENERATE 句で利用したパス表現式（/title/text()）を連結した次のパス表現式

//article/title/text()

は title 要素ノードのテキストノードの値を探索している完全なパス表現式である．これをレイアウト式 [1] 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，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パス表現式である．例えば，1 章 1 項の例 2 の PPX の FOR 句で利用したパス表現式

/dblp

は dblp 要素ノード以下にある部分 XML を探索している不完全なパス表現式である．ここで言う部分 XML とは XML データの全部，または一部である．即ち，不完全なパス表現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，その根ノードの全ての子孫ノードを含む部分木である．

2.1.2 データモデル

XPath 2.0 [14] 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，新たに NID（Node identifier）ノードを導入し，それを含めた 4 種類のノードを使用する．例を図 2.1 に示す．ここで NID ノードは各要素ノードにユニークな番号を付与するための特別な属性ノードである．番号は深さ優先順序のラベリング方法 [15] を用いている．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．ここで NID はテキストノードの値（データ）を要素ノードの現れる順に出力するために利用されており，本章の 2.2.1.2 項の反復演算子でその説明がある．図 2.2 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．

[1] 本章の 2.2.1 項で後述する．

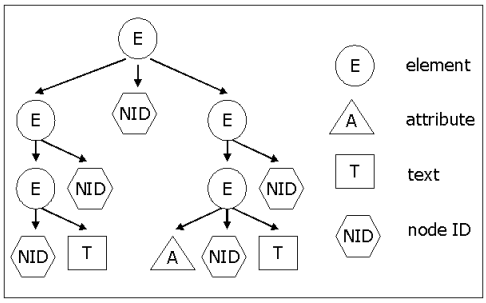


図 2.1: 利用した XML ノードの種類

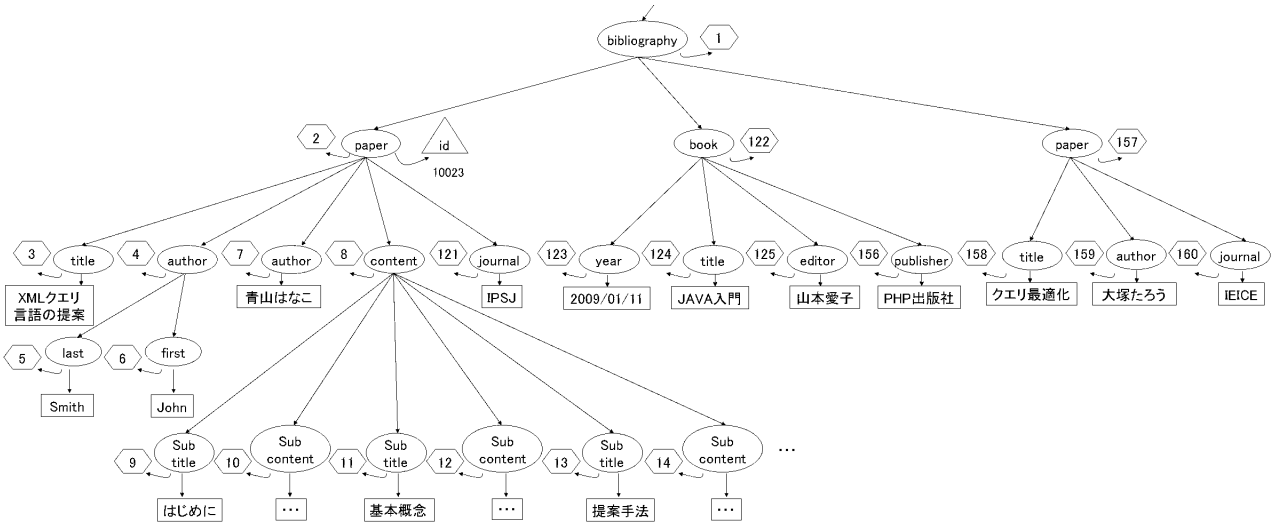


図 2.2: NID を付与したデータモデル

2.1.3 イレギュラー XML データ

イレギュラー XML データとは XML データで規則的に出現する任意の要素ノードを親ノードとする部分木と異なるデータ構造を持つ非定型木構造である．例えば，図 2.2 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ーと言い，逆に book は paper に対してイレギュラーと言う．

2.2 言語構文

PPX の構文は以下のように表す．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（拡張 TFE）>
FOR <”$” + 変数名 in db [2] (’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．

2.2.1 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レイアウト式である．

2.2.1.1 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 を持つ要素ノードである．以下に示すように変数名とパス表現式の組み合わせからなる．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 に対して探索をしていることを表す．

オペランド ::= ”$” + 変数名/パス表現式

2.2.1.2 演算子

演算子にはレイアウト指定演算子と自動整形演算子がある．

(1) レイアウト指定演算子

[2] これは DB2 データベースシステムに依存することを表す．

- 結合演算子

結合演算子は両辺の整形結果を何れかの方向（次元）に結合する二項演算子である．図 2.3 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）を表わす深度結合（%）演算子を示す．

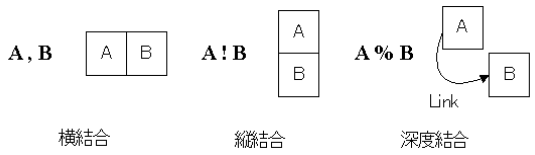


図 2.3: 結合演算子

- 反復演算子

反復演算子には二種類の単項演算子がある．一つは重複（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．もう一つは本研究で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．二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）を行ない，反復的に連結を行なう．

（a）データの反復連結

二種類の反復演算子はデータがある限り，指定する方向に繰り返し結合する．図 2.4 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 < >%）を示している．

（b）データのソート・グルーピング・重複処理

要素ノードの現れる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，要素ノードの現れる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．図 2.5 では探索されたフラットなリスト構造のデータを二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．

即ち，前者はデータによるソート，グルーピング，重複データの処理などを行ない，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なう．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（例えば，[(desc) $i/author]!）すると降順でソートを行ない，ASC と指定するか，または何も指定しない（例えば，[

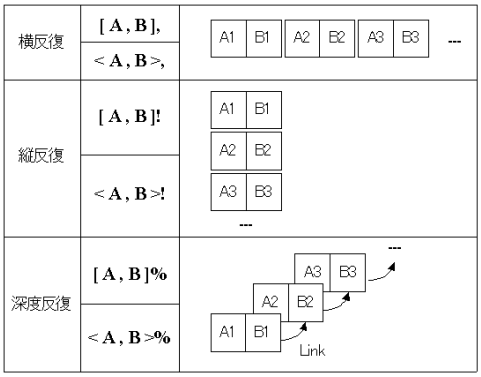


図 2.4: 反復演算子による反復連結

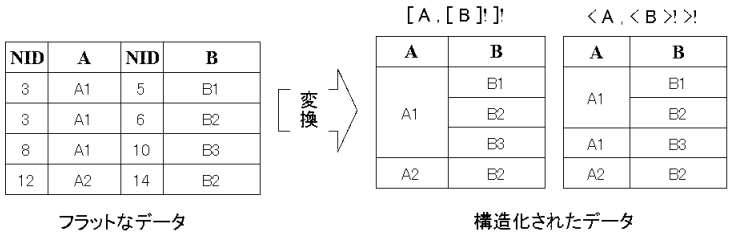


図 2.5: 反復演算子によるグルーピング

$i/author]!）場合は昇順でソートを行なう．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．

- 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．例えば，$A@\{width = 300\}$ という装飾子を用いるとオペランド A のセル幅を 300px にする．

(2) 自動整形演算子

自動整形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 に対して出力媒体への自動変換を指示する単項演算子である．

- &-，&+演算子

&-演算子は XML データからタグを除いてインデント形式で表示する単項演算子であり，&+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．インデント形式とはある階層に属する一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．4 章の図 4.2 ではその一例を示す．

- &演算子

&演算子は XML データインスタンスの統計的な性質に基いて出力する表構造を決定し，HTML への自動変換を行なう単項演算子である．

2.2.1.3 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．

2.2.1.4 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 句の条件式の指定をパターンと呼び，then 句や else 句のレイアウト式の指定を変換方法と呼ぶ．

(1) 条件分岐構文の記述規則

イレギュラー 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 (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

- 基本記述

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．

if ($j/name/text()) then ([$j/name]!)

ここで 1 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

イレギュラー 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．例えば，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，或いは要素ノードか，或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．

if ($j/name/text()) then ($j/name)
else if ($j/@name) then ($j/@name)
else ($j/name/first , $j/name/last)

ここで 1 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．

第 3 章

XMLデータのカスタマイズ整形出力

3.1 本アプローチの目的

本章では XML データを HTML 化する PPX のカスタマイズ整形出力を述べる．PPX の基礎技術である SuperSQL [12] 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．文献 [6] では以前版の PPX に関して述べているが，XML データ構造の不規則性について処理できない．

そこで PPX は XML データの性質に基づき，特に 3 点について SuperSQL の TFE を拡張する．PPX のカスタマイズ整形出力では次の 2 点を述べ，残りの 1 点である XML データの自動整形を指示する演算子は 4 章の PPX の自動整形出力で述べる．

(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．例えば，図 2.2 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なわれ，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．下図では論文のコンテンツ部分で実験・評価が提案手法より前に出力されていることが分かる．

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良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（2 章の 2.1.2 項）を提供した．

(2) 条件分岐の導入

2.1.3 項で前述したイレギュラー 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（2 章の 2.1.4 項）の指定を導入した．例えば，図 2.2 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．

本章で述べるカスタマイズ整形出力の貢献の一つは元の 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，およびイレギュラー XML データの処理を指定するための条件分岐構文を PPX に導入したことである．もう一つの貢献は PPX 処理系の実装を行ない，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なったことである．

3.2 PPX の変換モデル

PPX の構文規則（拡張 TFE）を付録 A 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．
- 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．

- 二項式は二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 構文に論理式と二つのレイアウト式を組み合わせたものであり，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わ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．

ソース木: ソース木は PPX クエリ文のソースを構文解析して得られる木構造表現である．ソース木から変換スキーマ木を直接作成することも可能である．ここでは以降で述べる技術的なキーポイントの一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，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．

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．

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．

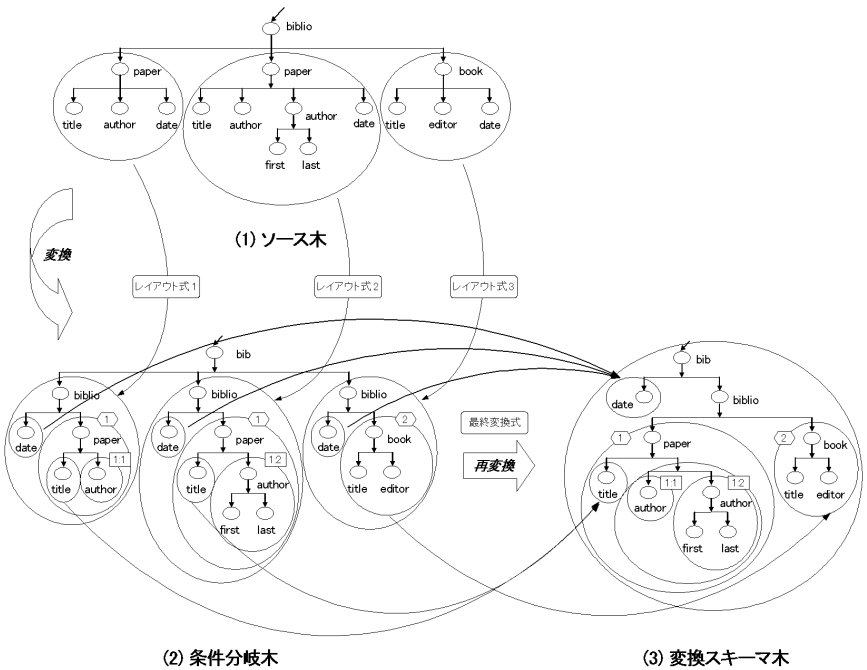


図 3.1: PPX 4 の変換モデル

具体的な例として次項で説明する PPX 4 の変換モデルを図 3.1 に示す．図の条件分岐木と変換スキーマ木に付与されている番号は目印である．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なうことを図 3.1 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なった 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なうことを図 3.1 の右側の下の (3) で示している．

3.3 応用例

図 3.2 は人工の XML データインスタンスの一部であり，それを利用して PPX が整形出力を行なう応用例を示す．

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合わせて指定することによって整形出力を行なうものである．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．例えば，以下に示す PPX 1 は図 3.2 の XML データから図 3.3 のような HTML 表への変換を行う．

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

この PPX では GENERATE 句のレイアウト式でオペランドに演算子を組合わせて記述によって 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先ず著者の所属で著者名をソート・グルーピングし，さらに論文タイトルをソート・グルーピングするなどそれらの要素ノードのテキストの値であるデータに対して再構造化を行う．その後，レイアウト指定演算子によってデータに対して結合を行う HTML タグを付与する．

- 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 表を生成することができる．例えば，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なっている．

```
PPX 2:
GENERATE html
   [ $i/title ! [ $j/affiliation , [ $j ]! ]! ]!
FOR
   $i in db('bib.xml')/biblio/*,
   $j in $i//( author | editor )
```

この結果，図 3.3 と異なる表構造を持つ図 3.4 のような HTML 表を生成する．

```
<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>
  <affiliation>名古屋情報大学</affiliation>
  </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>
  <affiliation>四国電気電子大学</affiliation>
  <author><last>羽多野</last><first>貴子</first></author>
  <affiliation>四国電気電子大学</affiliation>
  </paper>
   ...
</biblio>
```

図 3.2: XML データインスタンス

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 3.3: PPX 1 によるフォーマット結果

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 3.4: PPX 2 によるフォーマット結果

- 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通りにレイアウトを行なうことが可能である．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なう．

```
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 3.5 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 表を生成する．

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 3.5: PPX 3 によるフォーマット結果

- 非定型木構造の変換:

PPX はイレギュラー 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いた if-then-else 構文にオペランドを演算子と組合わせて指定したものである．例え

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ー XML データを処理している．その結果，図 3.6 のような HTML 表を生成している．

```
PPX 4:
GENERATE html
  [ (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ない，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なう．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なう．

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 3.6: PPX 4 によるフォーマット結果

- 多重ハイパーリンクの変換:

PPX は複数の HTML 表をリンクで表示される．例えば，次の PPX 5 は深度結合演算子（%）の左辺である$j/univ 要素と$i/title 要素がアンカとなり，右辺で生成されるサブページ群がハイパーリンクで結合される表現をしている．この結果，ハイパーリンクで結合される図 3.7 のような HTML 表が生成される．

PPX 5:

```
GENERATE html
  [ $j/univ %
    [ $j/name ! [ $i/year , [ $i/title %
      [ $k/conference , $k/volume , $k/pages ! $k/pdf ],
  ]! ]! ]! ]!
FOR
  $i in db('bib.xml')//( author | editor ),
  $j in $i//paper,
  $k in db('bib.xml')//( paper | book )
WHERE
  $k/title = $j/title
```

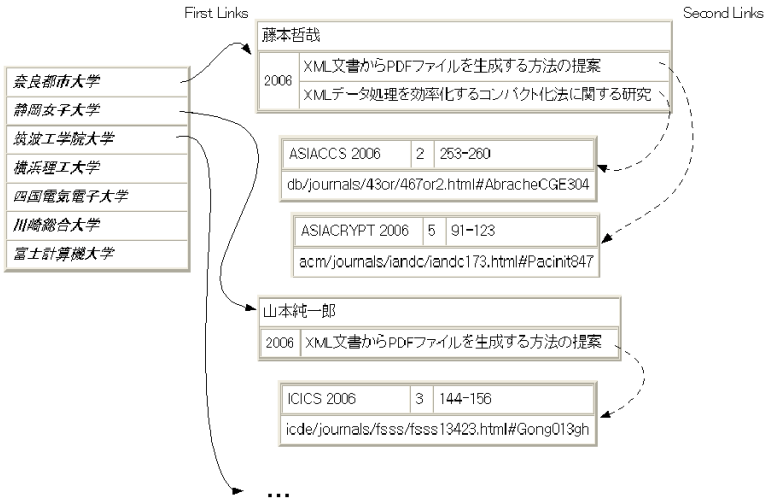


図 3.7: PPX 5 によるフォーマット結果

- 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 6

はそれに関して文字列定数で見出しをつけて対処する一つの例である．その結果，図 3.8 のような HTML 表を生成している．

```
PPX 6:
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 )
```

【著者】	藤本哲哉
【論文】	XML文書からPDFファイルを生成する方法の提案
【内容】	
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	・・・
【著者】	山田達朗
【論文】	関係データベースを利用したXML文書の格納
【内容】	
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 3.8: PPX 6 によるフォーマット結果

3.4 実装

3.4.1 カスタマイズ整形出力の流れ

カスタマイズ整形出力の流れは図 3.9 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 のクエリは構文解析部によってレイアウト式（GENERATE 句）と XML データアクセス式（FOR 句と WHERE 句）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 [1] 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．出力媒体生成部では変換スキーマ木に基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ない，カスタマイズ整形された結果を出力する．

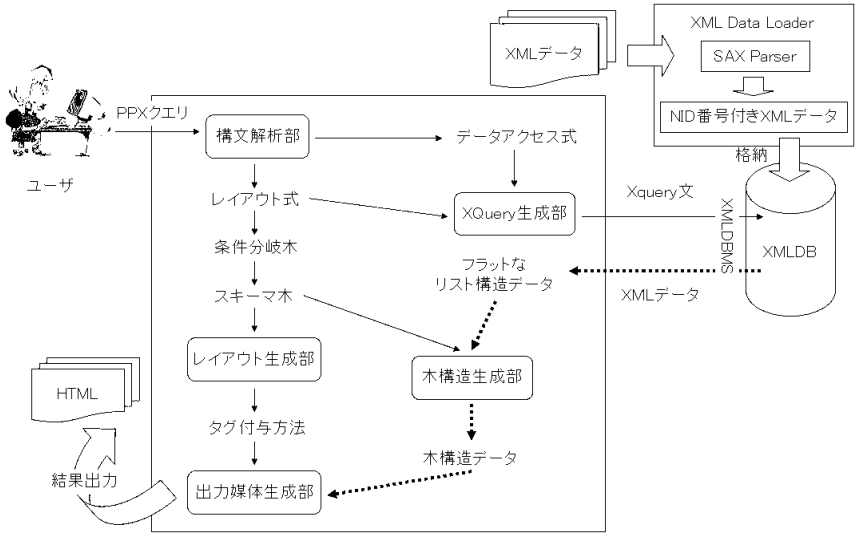


図 3.9: カスタマイズ整形出力の流れ

3.4.2 処理アルゴリズム

本項では PPX のカスタマイズ整形出力が処理を行なうアルゴリズムについて詳しく述べる．

3.4.2.1 構文解析部

PPX のクエリ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二つに変換される．即ち，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

[1] 不完全なパス式によって探索された部分 XML の自動整形は 4 章の 4.2.5 項で説明する．

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

レイアウト式は先ず if 句による条件分岐のケース毎に分けた 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上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．例えば，図 3.1 で示す変換モデルは 3.3 項で説明した 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図 3.1(2) の条件分岐木に変換される．条件分岐木に対応する 3 つのレイアウト式を次に示す．

レイアウト式 1: /**（paper データを処理）**/

[$ı//date , [$ı//title , [$j]!]!]!

レイアウト式 2: /**（paper データを処理）**/

[$ı//date , [$ı//title , [$ȷ/first , $ȷ/last]!]!]!

レイアウト式 3: /**（book データを処理）**/

[$ı//date , [$ı//title , [$j]!]!

そこで最初の 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．その後，条件分岐木に目印を付けて結合し，図 3.1(3) の変換スキーマ木に変換される．

次に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なう．目印は図 3.10 で示した Dewey Order [15] 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．例えば，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．

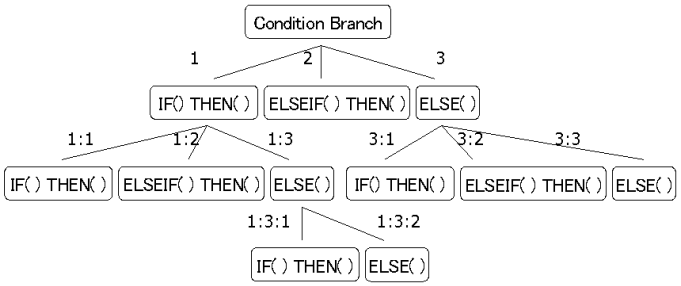


図 3.10: Dewey Order による目印付け

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

3.4.2.2 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 文に基づいて得られた 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得られる．ここでデータも図 3.10 の目印付け方法に基づいて目印付けを行なう．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 [2] が更に追加される．

- 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．

3.4.2.3 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なう．

- 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な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 ”)))))
)
```

[2] ここで NID はデータ順に出力するため，目印はデータにレイアウト方法を付与するためである．

- 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ない，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なわれる．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一本のリストに変換され，出力媒体生成部に渡される．

3.4.2.4 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わ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わすし，“+”はいくつのレイアウト式がともに出現する場合を表わす．例えば，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，それに基いてレイアウトを行なう．

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

3.4.2.5 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．

3.5 実験と評価

提案した PPX の有用性を示すため，W3C の Query Use Cases [21] に適用し，表現能力を確認した．また，生産性について XQuery，XSLT と比較を行なった．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易く加工し，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なった．即ち，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なうという状況を仮定した．例えば，DBLP [17] のような大規模な XML データを著者別，または年度別，または論文誌別などに様々な用途に応じて見易く加工，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 の生産性を評価した．

3.5.1 実験環境

実験は CPU: Pentium III 1.4GHz Dual，メモリ: 2GB，OS: Windows XP のマシンで行なう．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 [18] を使用した．

表 3.1: XML データセット

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

3.5.1.1 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 3.1 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 [19] 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 [20] 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．

3.5.2 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 クエリのユースケース [21] を用いてその表現能力を確認する．

3.5.2.1 実データを用いた評価

PPX クエリを表 3.1 のデータに適用し，変換能力を評価した．用いる PPX クエリのリストを表 3.2 で一部を示す．クエリに付与されている”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”はクエリ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．例えば，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ーがスピーチした内容順に現れる．このような XML データに対して PPX のクエリ（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なうこと．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

表 3.2: PPX クエリ

N0.	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

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．例えば，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（例えば，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 のクエリ（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なうこと．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な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なった．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なう．即ち，このクエリでは 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（例えば，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 のクエリ（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三つの（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一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ない，再構造化された HTML 表を出力した．また，QA2 は author 要素ノードと三つの（title，initPage，endPage）要素ノードのテキストノードの値を一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ない，再構造化された HTML 表を出力した．

表 3.3: 表現能力の確認結果

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

3.5.2.2 W3C クエリのユースケース

W3C の Query WG で XML のクエリのユースケース [21] が整理されており，それと同等な PPX を表 3.1 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 への変換を行なうものであり，PPX の行なう 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．

例えば，次の XQuery は表 3.3 の左で示す代表例（”XMP”）のクエリ 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>
```

このクエリでは表題 (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 3.3 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 [22]．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.

3.5.3 変換能力

ここでは変換を行な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.

3.5.3.1 指定方法

PPX と同等の変換を行なう XSLT，XQuery などの指定方法を比較する．例えば，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 に一個の反復演算子（[ ]!）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なうために 1 章で既存技術の問題点として挙げた例に示した XQuery や XSLT は直感的とは言い難く複雑である．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なう際に PPX は XSLT，XQuery などと次の点が異なる．

（1）データのソート： 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なう．例えば，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 [13] 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 [14] 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なう．

（2）データのグルーピング： XQuery は GROUP BY 関数を利用してグルーピングを行なう．XSLT 1.0 はグループ化をサポートしてないので Muenchian 方法 [16] 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 要素を使用して一群の 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 は二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なう．

（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なう．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な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 は二種類の反復演算子

を用いて重複データの処理を行なう．即ち，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，且つテキストの同値なら自動的に重複排除を行なう．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．

3.5.3.2 記述量

PPX と同等の変換を行なう 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．これに対してクエリ式に HTML タグを埋め込むと記述量は増える．例えば，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一つの結合演算子だけの変更で済む．しかし，XQuery の場合はクエリ式に埋め込んだ HTML タグを表 3.4 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．即ち，XQuery や XSLT などは処理を行なうデータに対してソートを行なう関数（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なう関数（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 への変換を行なう．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な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 3.5 では図 3.4 の出力を行なう PPX，XSLT，XQuery の記述に必要な文字数を比較した 1 つの例を示している．

3.5.3.3 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わ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，一つの 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．例えば，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 への出力を行なうとする．PPX ならば図 3.11 でその一部を示しているとおり，オペランド$i/title と $j/author 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．即ち，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クエリ式やスタイルシー

表 3.4: レイアウト式の変更過程

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 3.5: クエリの文字数の比較

	PPX	XQuery	XSLT 1.0	XSLT 2.0
図 3.4 の HTML 生成	94	391	745	836

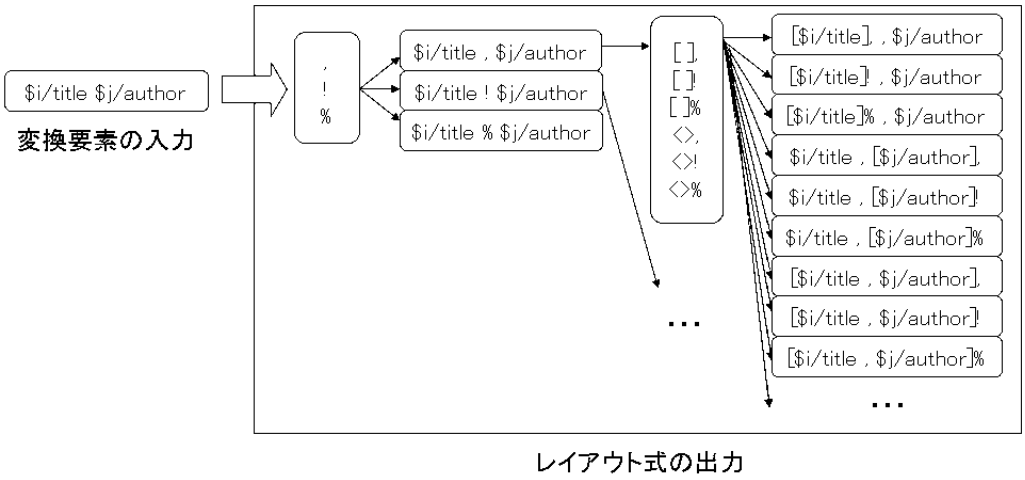


図 3.11: 異なるレイアウト式の生成

表 3.6: 指定方法

	データのソート	データのグルーピング	重複データの処理
PPX	反復演算子	反復演算子	反復演算子
XQuery	SORT BY	GROUP BY	DISTINCT-VALUES()
XSLT 1.0	SORT BY	MUENCHIAN GROUPING	DISTINCT-VALUES()
XSLT 2.0	SORT BY	XSL:FOR-EACH-GROUPY	DISTINCT-VALUES()

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 への変換を行なう際の異なる点を表 3.6 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な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．

第 4 章

XMLデータの自動整形出力

4.1 本アプローチの目的

本章では SuperSQL の TFE を拡張した 3 点のうちの 1 点である自動整形演算子によって XML データを自動的に HTML 化する PPX の自動整形出力について述べる．

先ず，PPX が XML データに対してカスタマイズ整形出力を行う 1 つの例を挙げる．例えば，次の PPX 7 は図 1.1 の XML データに対してカスタマイズ整形出力を行なう．

PPX 7:

```
1: GENERATE html
2:     [ $i//year , [
3:         if ( $i/phd )
4:         /** phd に関する XML データを指定 **/
5:         then (
6:             [ $i//title , <
7:                 if ( $j/author ) then ( $j/author ) else ( $j/editor )
8:             >! ]! )
9:         /** article に関する XML データを指定 **/
10:         else ( [ $i//title , < $j/author >! , $j/school ]! )
11:     ]! ]!
12: FOR
13:     $i in db('bib.xml')/dblp/*,
14:     $j in $i//authors
```

この PPX では条件分岐（if-then-else）構文を用いて DBLP の異なるデータ構造に対し，それぞれの異なるレイアウト方法で処理を行なう．即ち，3 行目の if 句の条件が TRUE の場合は phd 要素ノード以下の XML データに対して処理を行ない，そうではない場合に 10 行目の else 句で article 要素ノード以下の XML データに対して処理を行なう．また，著者の順序が重要であるため，7 行目のオペランド（$j/author や$j/editor）に元の順序を重視する反復演算子（< >!）を指定して変換を行なう．その後，phd と article に関するデータを 2 行目の日付（year）でグルーピングを行なっている．この結果，year と title 要素ノードが持つテキストノードの値に対して重複を排除し，アルファベット順に並べ替えるが著者が持つテキストノードの値に対しては重複（例えば，ここでは同姓同名な著者に対して）を許し，要素ノードの現れる順に出力することが可能である．

しかし，この例から見られるようにカスタマイズ整形出力はユーザが XML データの非定型木構造や要素ノードの現れる順序保存の必要性などを事前に把握し，条件分岐構文を用いて演算子やオペランドをレイアウト式で詳細に指定する作業が必要である．

そこで XML データを整形出力する際に木構造の再構成を含む詳細な指定によってプログラマの意思に基づいて整形を行う要求がある一方，元の木構造の論理的な構成を変えずに自然な出力をなるべく少ない手間で得たいという相反する要求が存在する．例えば，

XML データの全部，または一部を取得（文書中心の Shakespeare データ [20] の場合は節や段落など）してそれに対して文書順に HTML などに自動変換して閲覧したいことを考えられる．特に不規則性が予見できず，条件分岐で対処できない場合に必要である．

XML データの全部，または一部とは不完全なパス表現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，その根ノードの全ての子孫ノードを含む部分木であり，それが自動変換の対像となる XML データである．これを以降は部分 XML と呼ぶ．

文献 [7, 8] は DTD に基づいてレイアウト方法を決定し，レイアウト対象となる部分 XML を単純変換表示する機能を提供するにとどまっている．また，DTD が変更される頻度が高い妥当な XML データや DTD が不明な整形式 XML データに対しては有効ではない．

本章では XML データインスタンスに基づいた自動変換ルール（4.2 項）を用いて部分 XML に対して自動的に HTML 化する新たな自動整形出力を提案する．即ち，XML データから非定型木構造の出現パターンを解析し，それぞれの異なる木構造に対して異なるレイアウト方法で変換を行なう．

例えば，次の PPX 8 では PPX 7 が条件分岐を用いて詳しく指定した XML ノードを一括指定によって自動整形出力を行う．

PPX 8:

```
1: GENERATE html
2:     [ $i/year ! $i/title !
3:         < & ( $i/authors ) >!
4:     ]! ]!
5: FOR
6:     $i in db('bib.xml')//( phd | article )
```

この PPX では GENERATE 句のレイアウト式でテキストノードまで指定した 2 行目の $i/year と $i/title オペランドによってカスタマイズ整形出力を行なう際に，authors 要素ノードまで指定した 3 行目の $i//authors オペランドと & 自動整形演算子の組合わせは不完全なパス表現式（//(article | phd)/authors）によって authors 要素ノード以下にある部分 XML に対して自動整形出力を行なう．

また，次の PPX 9 では 2 行目のレイアウト式で XML ノードを指定せず，自動整形出力を行なう．

PPX 9:

```
1: GENERATE html
2: & ( $i )
3: FOR
4:     $i in db('bib.xml')/dblp
```

この PPX は dblp 要素ノード以下にある article 要素ノードを根ノードとする部分 XML や phd 要素ノードを根ノードとする部分 XML を現れる順序とおりに自動整形して出力を行

なった結果を図 4.1 に示す.

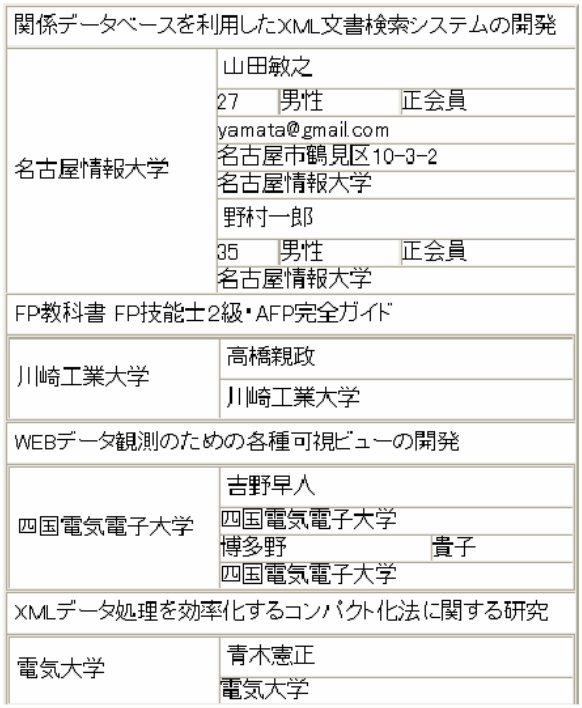


図 4.1: PPX 9 によるフォーマット結果

さらに，部分 XML に対して階層構造を単純に実現するインデント形式に自動変換することができる．例えば，以下に示す PPX 10 は GENERATE 句のレイアウト式で$i/author と&-自動整形演算子を組合わせて記述し，部分 XML（authors 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 4.2 に示す.

```
PPX 10:
GENERATE html
  < $i/title !
    < $i/year , < &- ( $i/authors ) >! >!
  >!
FOR
  $i in db('bib.xml')/dblp
```

自動整形出力の主な貢献は PPX のカスタマイズ整形出力による XML データの変換のための作成する作業にかかる労力を軽減することである．また，自動整形出力の実装はカスタマイズ整形出力と排他的ではなく，PPX 8 の例に見られるようにユーザのニーズに合わせて 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など二つの方法が両立できる.

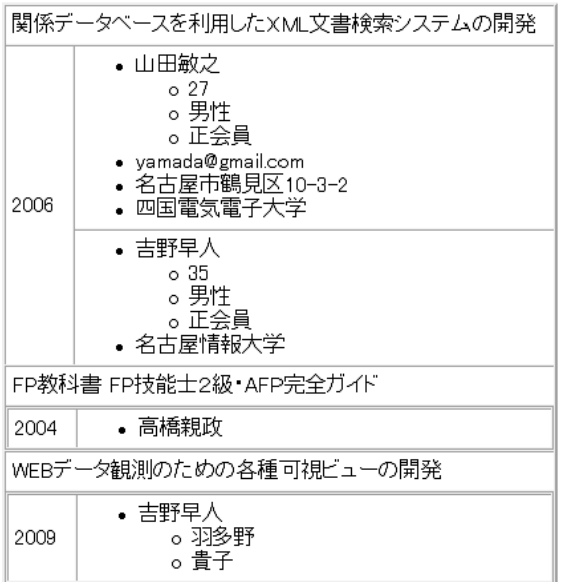


図 4.2: PPX 10 によるフォーマット結果

4.2 自動変換ルール

ここでは XML データに対して自動変換を指示する&演算子によって自動整形出力を行なう自動変換ルールを説明する.

4.2.1 自動変換ルールの概要

XML データインスタンスに基づいた自動変換ルールは大きく 3 つのステップからなる.

(1) 木パターンの抽出： XML データインスタンスから木パターンを抽出し，それに基づいてレイアウト対象となる XML データの統計情報を求める.

(2) レイアウト方法の決定： また，統計情報に基いて木パターン群の配置や各木パターンの連結方法を決定する.

(3) 結果出力： その後，レイアウト方法を用いてデータに対して元のデータ順にソート・グルーピングを行ない，加工されたデータに対して HTML タグの付与によって HTML に変換する.

4.2.2 木パターンの抽出

XQuery 生成部から生成した XQuery 文によって探索された XML データは木パターン生成部に渡され，木パターンの抽出を行なう．即ち，XML データに対する構造要約を行

ない，要約された非冗長木の分割によって木パターンが生成される．また，XML ノードの反復出現回数，木パターンの反復出現回数とテキストノードの値の文字列長の平均値などについて統計情報を求める．

4.2.2.1 構造要約

経路索引とも呼ばれる XML データに対する構造要約は枝分かれ経路に対する索引を生成する F&B [24] という索引方法を用いることによって非冗長木を生成する．次のような二つの基準に満たす．

(1) 正確性：

- データ中の全ての経路が木構造中にも出現する．
- また，木構造中の全ての経路がデータ中にも出現する．

(2) 非冗長性：

- 任意のノードを親とする子孫ノードが全て同じであり，かつ現れる順序も同じの場合はそのノードをルートノードとする部分木を一つにする．

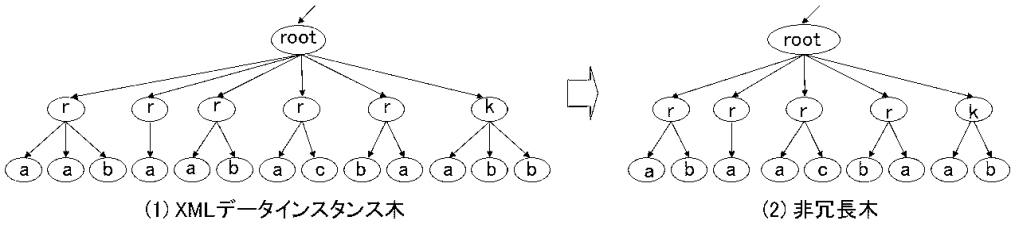


図 4.3: XML データの構造要約

図 4.3 では XML データを木構造で表現した XML データインスタンス木から非冗長木を生成する構造要約の例を示す．ここで図 4.3(1) の 1 番目の r 要素ノードが持つ二つの子要素ノード a をルートノードとする部分木は同じであり，それらを一つにする．その後，やはり 1 番目の r 要素ノードをルートノードとし，a と b の順に出現する子要素ノードを持つ部分木は 3 番目にある r 要素ノードをルートノードとし，a と b の順に出現する子要素ノードを持つ部分木と一つにする．しかし，5 番目の r 要素ノードをルートノードとし，b と a の順に出現する子要素ノードを持つ部分木は要素ノードの現れる順序が異なるので一つにしない．このように要約された結果，図 4.3(2) のような非冗長木が生成される．この方法を用いて図 4.4 で示したレイアウトとなる XML データインスタンスに対して構造要約を行い，生成された非冗長木を図 4.5 で示している．

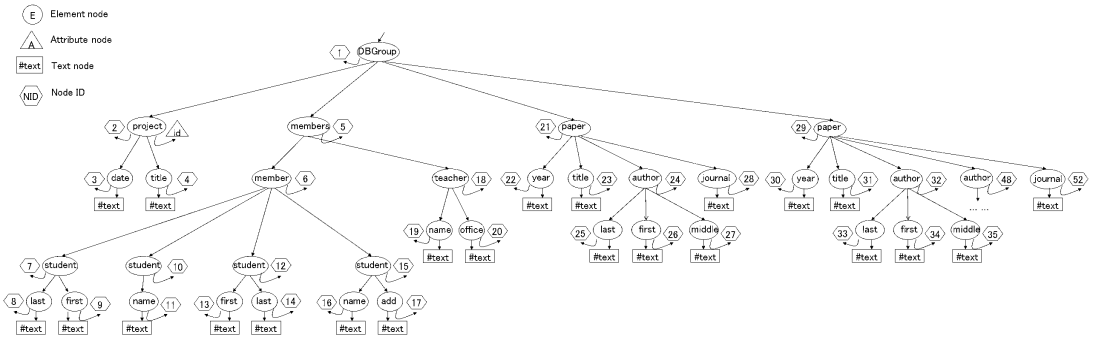


図 4.4: レイアウト対象となる XML データインスタンス

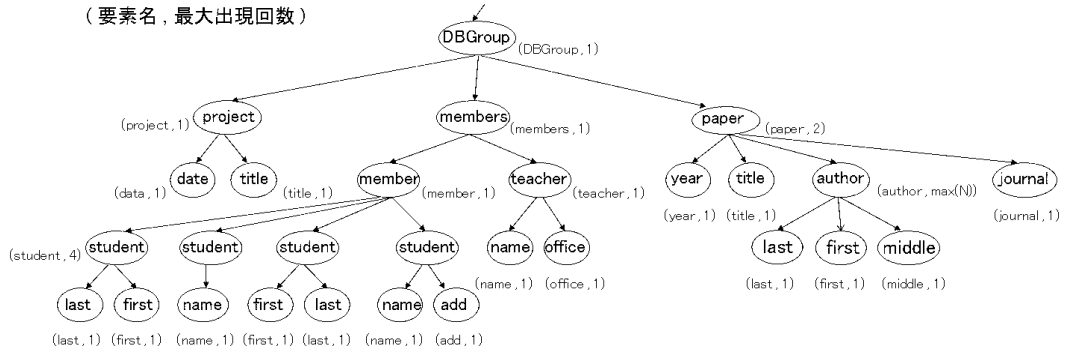


図 4.5: 要約された XML データインスタンス

4.2.2.2 分割

要約された非冗長木を複数の部分木に分割し，それに相応する木パターンを抽出するため，レイアウト対像となる XML データインスタンスから XML ノードの反復出現回数について統計を行なう．以下の統計 1 に従って算出する．

統計 1: XML ノードの反復出現回数

要約された XML データインスタンス木中のノード X，その親ノード P に対して XML データインスタンス木中には P を根とする部分木が複数存在し得る．それらにおけるルートノード直下の X に相当するノードの出現回数の最大値を X の反復出現回数と呼び，これを統計 1 とする．

図 4.5 で示す非冗長木には統計 1 の統計値が各要素ノードが持つ．例えば，author 要素ノードの統計値は反復出現するそれぞれの paper 要素ノードが持つ author 要素ノードの最大出現回数を author 要素ノードが持つ．また，student 要素ノードの統計値は member 要素ノードが持つ反復出現する student 要素ノードの出現回数である．ここで student 要素ノード以下の子要素ノードが異なっても計数する．

この統計によって非冗長木の分割を行なう XML ノードを発見する．非冗長木を分割する方法を一言で言えば反復出現する XML ノード，および兄弟ノードをそれぞれ根ノードとする部分木を収縮と展開によって分割を行なう．

(1) 収縮による木パターン生成：

非冗長木で反復出現する XML ノードと木の同じレベルにある兄弟 XML ノードをそれぞれの先祖ノードにすると，その以下の子孫 XML ノードはそれぞれの先祖 XML ノードへ収縮される．収縮によって生成した部分木は一つの木パターンになる．ここで収縮された先祖ノートを木パターンが仮想 XML ノードとして持つ．

(2) 展開による木パターン生成：

収縮によって生成された部分木に含まれているそれぞれの仮想 XML ノードは展開を行ない，仮想 XML ノードをルートノードとする部分木はそれぞれの木パターンになる．

(3) 展開され生成した部分木に (1) と (2) を繰り返しながら適用し，反復出現する XML ノードがなければ非冗長木の分割は終了する．

また，分割を行なう際に生成された木パターンを識別するために DEWEY ORDER [15] を用いて各木パターンに PID（PatternID）番号を付ける．具体的な分割方法，および PID 付き方法は図 4.5 の要約された XML データインスタンスを利用して説明する．

先ず，その前で説明した統計 1 の値が 2 以上となるノードを非冗長木のルートノードから子孫ノードまで探す．図 4.5 で paper 要素ノードが最初に反復出現する XML ノードであり，そのノードをルートとする部分木の XML ノードは paper 要素ノードへの収縮を行うことを図 4.6 で示している．同時に，paper 要素ノードの同じレベルにある兄弟要素ノード（project，members）についてもそれぞれをルートノードとする部分木はそれぞれの project，members 要素ノードへの収縮を行う．これによって最初の木パターンが生

成される．この木パターンに PID として P1 を付与する．図 4.6 の右上に示している木パターン P1 は仮想ノード（project’，members’，paper’）を持っていることが分かる．

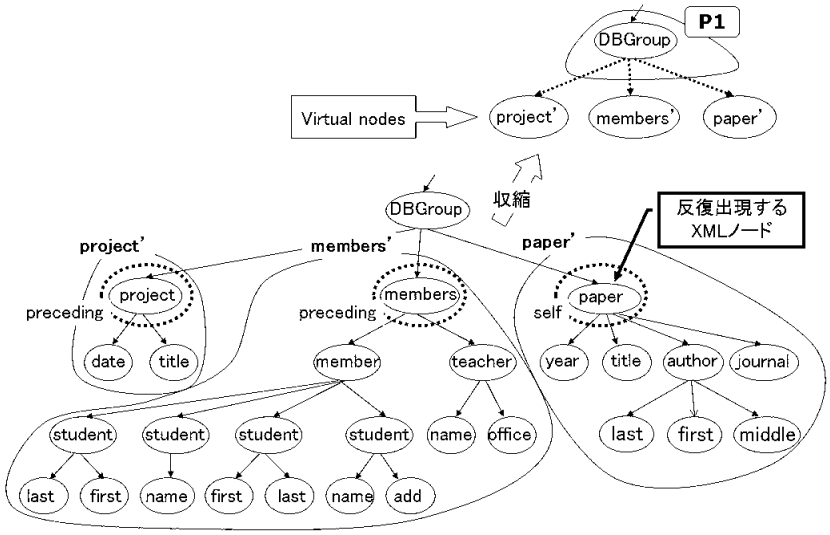


図 4.6: 収縮による木パターン生成

次は，木パターン P1 が持つ 3 つの仮想ノード（project’，members’，paper’）を展開・分割によって project，members，paper 要素ノードをそれぞれのルートノードとする部分木は木パターン P1 の 3 つの子木パターンとして生成することを図 4.7 で示している．これらの子木パターンにそれぞれ P1.1，P1.2，P1.3 とする PID を付与する．

また，図 4.8 では木パターン P1 から分割された 3 つの子木パターン（P1.1，P1.2，P1.3）でさらに反復出現する XML ノードを探す．木パターン P1.2 では何れか反復出現，かつ異なる要素ノードを持つ 3 つの student 要素ノードが見つかり，そのノードをルートノードとする部分木の XML ノードはそれぞれの student 要素ノードへの収縮を行なう．これによって元の木パターン P1.2 は反復出現する XML ノードを含んでない新たな木パターン P1.2 が生成する．同時に，木パターン P1.3 も元の木パターン P1.2 と同じ方法で新たな木パターン P1.3 を生成される．

その後，木パターン P1.2 が持つ 3 つの仮想ノード（student’）は展開・分割を行ない，生成した子木パターンにそれぞれ P1.2.1，P1.2.2，P1.2.3，P1.2.4 とする PID を付与する．また，同じ方法で木パターン P1.3 から生成した四つの子木パターンには P1.3.1，P1.3.2，P1.3.3，P1.3.4 とする PID を付与することを図 4.9 で示している．

図 4.10 では最終生成した PID 付きの木パターンを示しており，それぞれの木パターンは深さ分割（Depth Partition [25]）によって完全なパス表現式集合を持つようにする．表 4.1 では前例から生成した木パターンが持つ PID，パス表現式集合とテキストノード（の値）などの各要素をまとめている．表から分かるように XML データは一つ以上の木パターンを持ち，それぞれの木パターンは一組のパス表現式集合から構成されている．ま

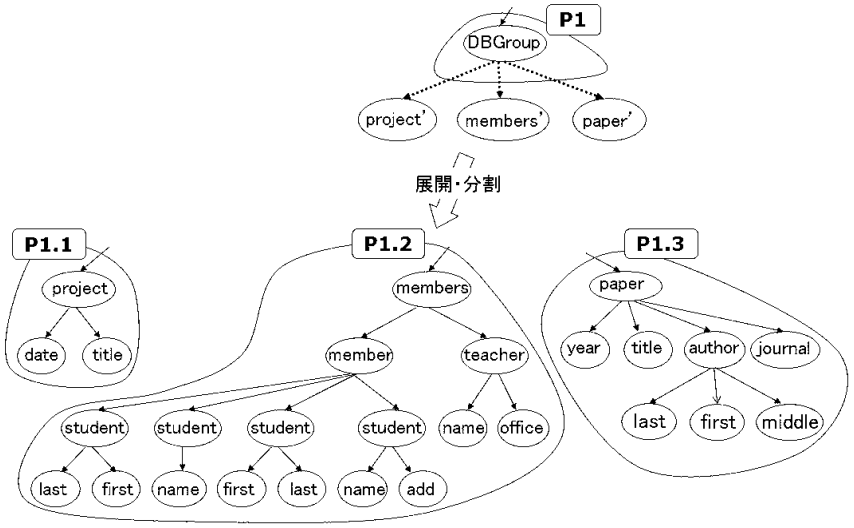


図 4.7: 展開・分割による木パターン生成

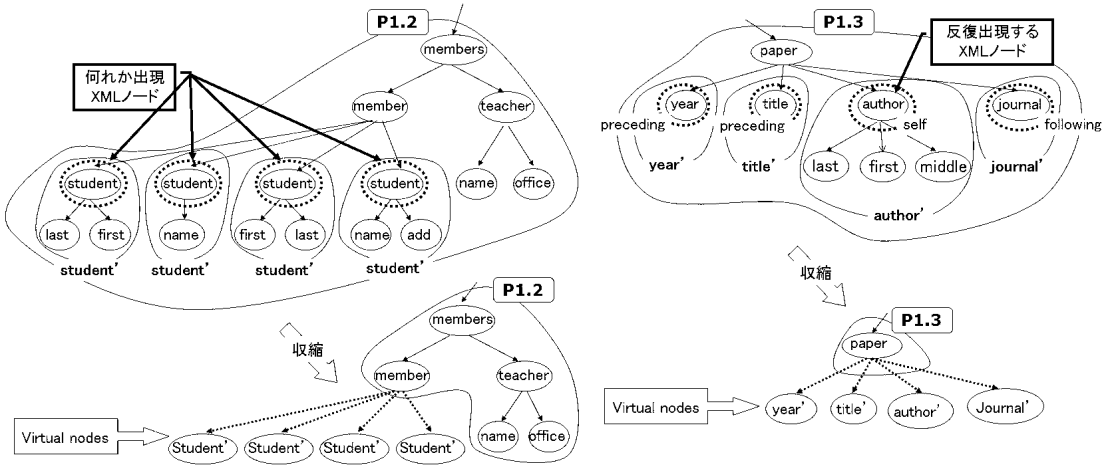


図 4.8: 収縮による木パターン生成

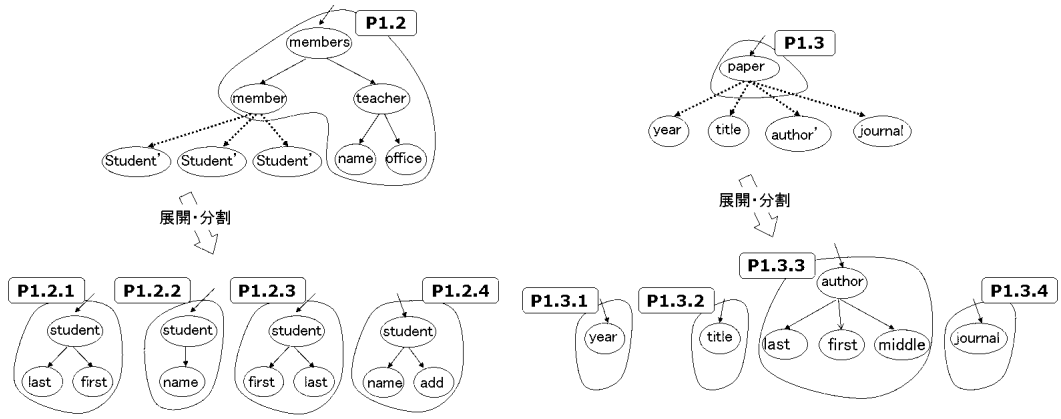


図 4.9: 展開・分割による木パターン生成

た，個々のパス表現式集合は一組のテキストノード（の値）に対応している．例えば，木パターン P1 のパス表現式集合はテキストノードを持っていないが，木パターン P1.2 のパス表現式集合は一組のテキストノードを持つことが分かる．

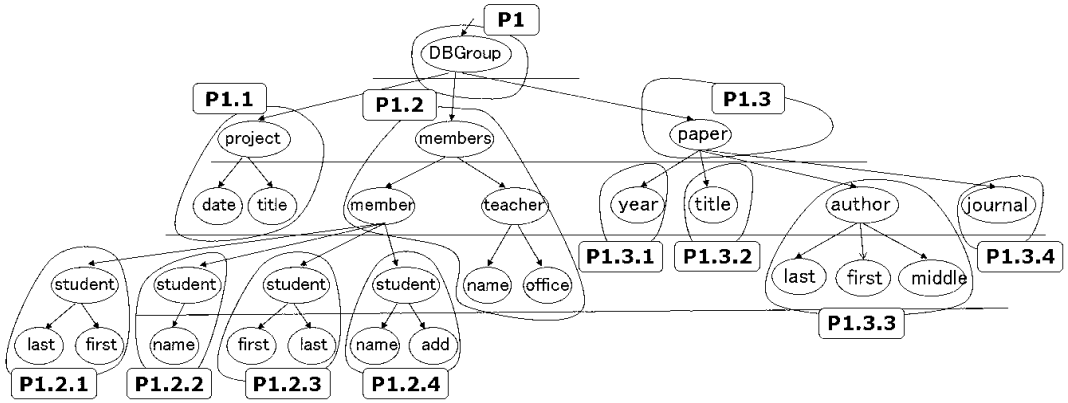


図 4.10: PID 付き木パターンの深さ分割

4.2.2.3 木パターン

分割された木パターンの種類を判断し，対応するレイアウト方法を決定するためにレイアウト対像となる XML データインスタンスで木パターンの出現回数について統計を行なう．以下の統計 2 に従って算出する．

統計 2: 木パターンの出現回数

木パターンを構成した一組のパス表現式集合が XML データインスタンス中で完全にマッチし，かつ M 回出現すれば，その木パターンの出現回数は M 回として計数する．

表 4.1: 木パターンが持つ要素

木パターンの PID	パス表現式集合	テキストノード
P1	/DBGroup	-
P1.1	/DBGroup/project	-
	/DBGroup/project/date	#pcdata
	/DBGroup/project/title	#pcdata
P1.2	/DBGroup/members	-
	/DBGroup/members/member	-
	/DBGroup/members/teacher	-
	/DBGroup/members/teacher/name	#pcdata
	/DBGroup/members/teacher/office	#pcdata
P1.2.1	/DBGroup/members/member/student	-
	/DBGroup/members/member/student/last	#pcdata
	/DBGroup/members/member/student/first	#pcdata
P1.2.2	/DBGroup/members/member/student	-
	/DBGroup/members/member/student/name	#pcdata
P1.2.3	/DBGroup/members/member/student	-
	/DBGroup/members/member/student/first	#pcdata
	/DBGroup/members/member/student/last	#pcdata
P1.2.4	/DBGroup/members/member/student	-
	/DBGroup/members/member/student/name	#pcdata
	/DBGroup/members/member/student/add	#pcdata
P1.3	/DBGroup/paper	-
P1.3.1	/DBGroup/paper/year	#pcdata
P1.3.2	/DBGroup/paper/title	#pcdata
P1.3.3	/DBGroup/paper/author/last	#pcdata
	/DBGroup/paper/author/first	#pcdata
	/DBGroup/paper/author/middle	#pcdata
P1.3.4	/DBGroup/paper/journal	#pcdata

この統計に基づいて木パターンを主に非頻出木パターンと頻出木パターンに分けられる．

(1) 非頻出木パターン: 木パターンを構成する一組のパス表現式集合がレイアウト対象となる XML データインスタンスで 1 回だけ出現するものを非頻出木パターン (Non-Frequent Tree Pattern) と言う．例えば，表 4.1 で示した木パターン P1.2.1 のパス表現式集合は図 4.4 の XML データインスタンスで 1 回出現するので非頻出木パターンである．

(2) 頻出木パターン: 木パターンを構成する一組のパス表現式集合がレイアウト対像となる XML データインスタンスで 2 回以上出現するものを頻出木パターン (Frequent Tree Pattern) と言う．例えば，表 4.1 で示した木パターン P1.3.3 のパス表現式集合は図 4.4 の XML データインスタンスで 2 回以上出現するので頻出木パターンである．

4.2.3 レイアウト方法の決定

木パターン生成部で得られた木パターンと統計情報はレイアウト生成部に渡され，レイアウト方法を決定する．これは PPX のカスタマイズ整形出力でユーザが指定する GENERATE 句のレイアウト式に相当するものであり，自動整形出力ではそれを自動的に生成する．本研究でレイアウト方法を生成する基準は XML データから出力を行なう HTML を全体的に自然な表構造にして WEB ブラウザで表示することである．自然な表構造とは任意の XML データは木構造で表現できることから木パターンク群は階層構造を持つように配置し，それぞれの木パターンに連結方法の付与によって木パターンに属するデータ[1] を入れ子構造を持つように加工してから連結を行う．

4.2.3.1 木パターンの配置方法

非冗長木から分割された木パターン群は親子関係を持ち，それらの木パターン群をディレクトリ型階層構造を持つように配置する．ディレクトリ型階層構造（Hierarchical Structure）は任意の親木パターンから下位階層に位置する複数の子木パターンが枝分かれした状態で配置することである．

図 4.11 では左側の分割された木パターン群を右側にディレクトリ型階層構造を持つように配置した結果を示している．ここで親木パターン（P1）は子木パターン（P1.1，P1.2，P1.3）と左向き T 字型に配置されていることの他に，子孫木パターンである P1.2 の子木パターン（P1.2.1，P1.2.2，P1.2.3，P1.2.4）や P1.3 の子木パターン（P1.3.1，P1.3.2，P1.3.3，P1.3.4）と左向き T 字型に配置されおり，全体的に左向き T 字型のようなディレクトリ型階層構造を形成されている．

以下のルールを満たすようにする．

ルール 1:（先祖子孫・親子関係木パターンの配置）任意の親木パターンはルール 2 を満たす複数の子木パターンと上（下）向き，或いは左（右）向き T（L）字型のように並べる．また，先祖子孫関係を持つ木パターンも親子関係木パターンの配置と同じ向きの T（L）字型になるように並べる．

ルール 2:（兄弟関係木パターンの配置）複数の子木パターン，すなわち演算子の優先順位を変更する中括弧（{ }）で括られた兄弟木パターン同士は同じ方向へ反復配置する．

上述したルール 1 とルール 2 を用いて図 4.11 で示している左側の木パターン群を右側にディレクトリ型階層構造を持つように配置するため，レイアウト方法の決定手順を次の例を挙げて説明する．

前項で説明した T（L）字型のように配置された最初の親子関係を持つ木パターン群で木パターン P1.1，P1.2 と P1.3 はテキストノードの値を持ち，これらの木パターンに対して連結方法を付与する．即ち，図 4.12 で示す非頻出木パターンである P1.1 や P1.2 に対し

[1] 木パターンのパス表現式集合にマッチする要素ノードのテキストノードの値である．

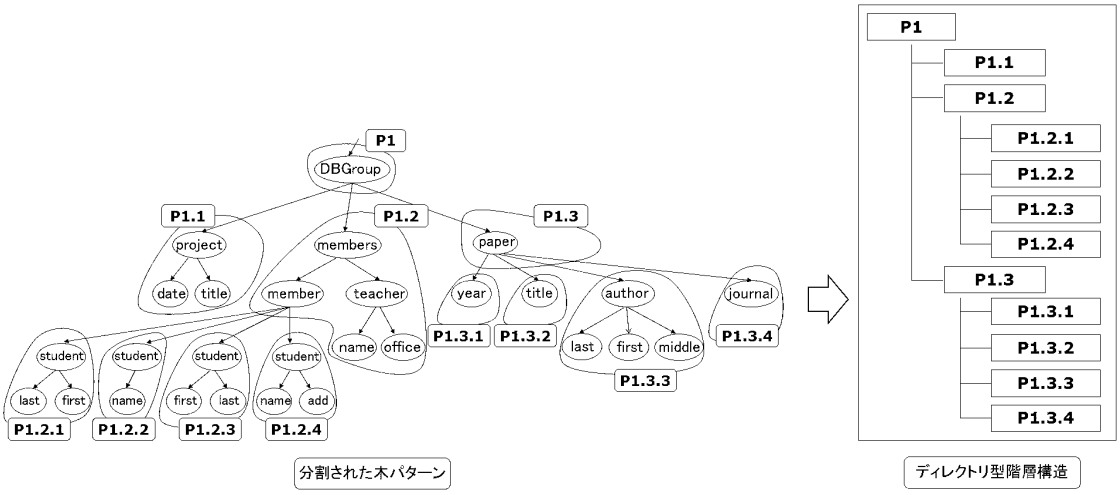


図 4.11: 木パターンの配置方法

てはそれぞれ { } で括れてからテキストノードを持つ要素ノード（例えば，teacher 要素ノードが持つ name と office 要素ノード）は二項結合演算子を付与する．一方，頻出木パターンである P1.3 は反復出現するため，それに対しては要素ノードの現れる順序を保存する反復連結演算子を付与する．これによって生成した連結方法は木パターン P1.1，P1.2 と P1.3 がそれぞれ持つ．同様な方法でそれぞれの各木パターンにも連結方法を付与する．

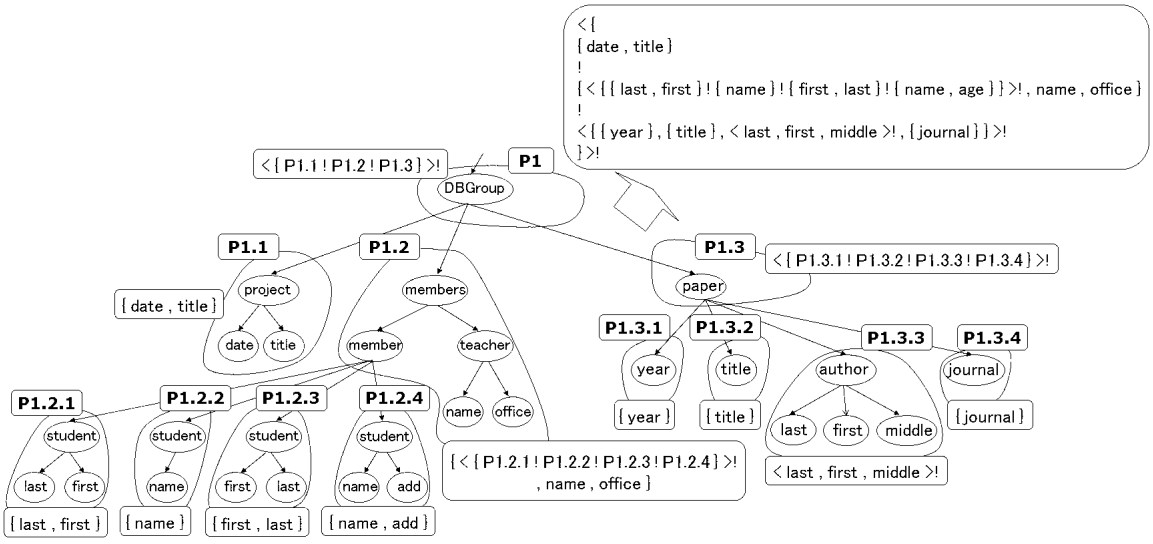


図 4.12: 連結方法の生成例

4.2.3.2 木パターンの連結方法

前項で説明したとおりに木パターン群に対して配置が完了したら，それぞれの異なる木パターンに応じて連結方法の付与は木パターンに属するデータに対して加工・連結を行なう方法を決定する．ここで XML データの要素ノードを含んでいる木パターン群は親子関係になっていることから木パターンに属するデータに対して入れ子構造を持つようにする．入れ子構造（Nested Structure）とはある親木パターンに属するデータを分類，また子木パターンに属するデータを再分類，さらに子孫木パターンに属するデータを再々分類することである．また，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るため，要素ノードの現れる順序のとおりに出力を行う．

以下のルールを満たすようにする．

ルール 3:（頻出木パターンの連結方法）頻出木パターンはデータ集合を囲んだ二つ以上のタプル [2] の集まりに対応し，それらのタプルに対しては要素ノードの現れる順序を保存する反復連結演算子を付与する．また，それぞれのタプル内にあるデータ集合に対しては二項結合演算子を付与する．

ルール 4:（非頻出木パターンの連結方法）頻出木パターンはデータ集合を囲んだ一つのタプルに対応し，それらのタプルに対して中括弧（{ }）を付与する．また，タプル内にあるデータ集合に対しては二項結合演算子を付与する．

上述したルール 3 とルール 4 を用いてそれぞれの木パターンが持つパス式集合にマッチするデータに対して加工と連結を行なうため，レイアウト方法の決定手順を次の例を挙げて説明する．

前項で説明した T（L）字型のように配置された最初の親子関係を持つ木パターン群で木パターン P1.1，P1.2 と P1.3 はテキストノードの値を持ち，これらの木パターンに対して連結方法を付与する．即ち，非頻出木パターンである P1.1 や P1.2 に対してはそれぞれ { } で括れてからテキストノードを持つ要素ノード（例えば，teacher 要素ノードが持つ name と office 要素ノード）は二項結合演算子を付与する．また，木パターン P1.2 では member 要素ノードが木パターン P1.2 の子木パターンを持つため，teacher 要素ノードが持つ name 要素ノードや office 要素ノードと同類の要素ノードとして扱う．一方，頻出木パターンである P1.3 は反復出現するため，それに対しては要素ノードの現れる順序を保存する反復連結演算子を付与する．これによって生成した連結方法は木パターン P1.1，P1.2 と P1.3 がそれぞれ持ち，図 4.12 ではそれを示している．同様な方法でそれぞれの各木パターンにも連結方法を付与する．

このように対象となる XML データインスタンス（図 4.4）に対して最終的に生成したレイアウト方法を図 4.12 の上側にある四角形で示している．

[2] 本研究でタプルとは同じ木パターンにある一連のテキストノードの組である．

4.2.3.3 書式情報

木パターンに付与された連結方法はさらに文字の幅，スタイルや色など様々な書式情報を付加することも可能である．今回は HTML 表のセルの延び過ぎを制限し，セルの無駄な空間を無くすために文字列の平均長によってセルの幅を自動的に設定する．このため，木パターンに含まれているテキストノードの値である文字列長の平均値について統計を行なう．以下の統計 3 に従って算出する．なお，この統計は木パターン生成部で行なう．

統計 3: 文字列長の平均値

同じ木パターンであり，かつ同じ要素ノードのテキストノードの値である文字列の長さについて平均値を求める．

この統計に基いてレイアウト生成部で生成されたレイアウト方法にセルの幅が付加される．例えば，木パターン P1.1 に付与されたレイアウト方法にセルの幅情報を与えると次のようになる．

{ date@{width=100} , title@{width=600} }

4.2.4 結果出力

木パターン生成部の XML データはパス表現式集合のマッチングによって木パターンが持つ PID をマッチする XML データに付与する．その後，木構造生成部に渡されて XML タグが外されたデータに対して重複を許し，現れる順にソート・グルーピングを行なう．

最後，加工されたデータは出力媒体生成部に渡され，レイアウト生成部から渡された連結方法を持つ木パターンと PID のマッチングを行なう．PID にマッチするデータに対して木パターンが持つ連結方法を用いて HTML タグを付与し，変換する．

4.2.5 自動整形出力の流れ

自動整形出力の流れは図 4.13 で示したように構文解析部，XQuery 生成部，レイアウト生成部，木パターン生成部，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 のクエリ文は構文解析部によってレイアウト式とデータアクセス式に分ける．XQuery 生成部でデータアクセス式から XQuery 文を生成し，それに基いて XML データが探索される．ここで XQuery 文の不完全なパス式によって探索された部分 XML は自動変換する対象となり，その部分に含まれているデータは一つのタプルとして構造化は行なわず，元の構造のまま木パターン生成部に渡される．その後，XML データの非定型木構造の解析によって木パターンを抽出する．また，木パターンに基づいてレイアウト対象となる XML データの統計情報を求める．さらに，統計情報と木パターンはレイアウト生成部に渡され，レイアウト方法を決定する．一方，部分 XML データは木構造生成部にも渡され XML タグの除去，およびデータに対

して重複を許し現れる順にソート・グルーピングを行なう．最後に，加工されたデータは出力媒体生成部に渡されレイアウト生成部から渡されたレイアウト方法によって HTML タグ付けが行ない，自動整形された結果を出力する．

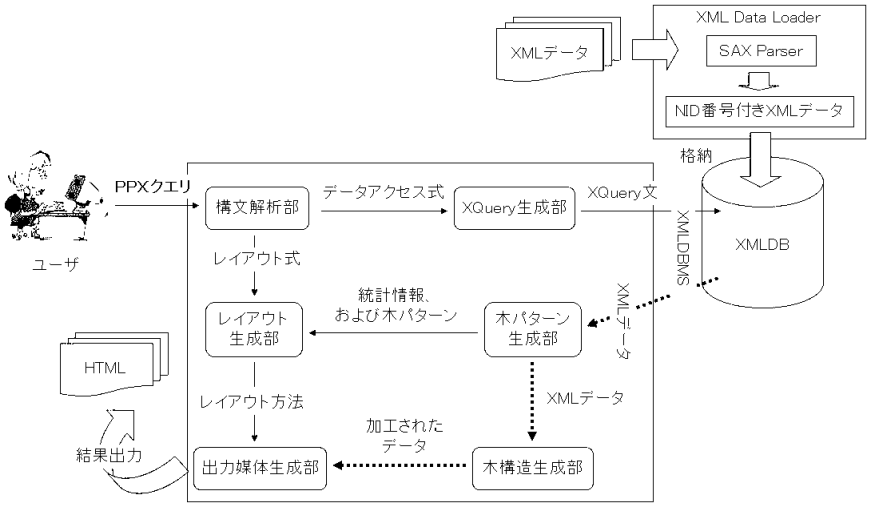


図 4.13: 自動整形出力の流れ

4.3 実験と評価

PPX の自動整形出力を様々な XML データに適用して自動変換能力について評価を行ない，その有用性を確認する．

4.3.1 実験環境

実験は CPU: Pentium III 1.4GHz Dual，メモリ: 2GB，OS: Windows XP のマシンで行なった．

4.3.1.1 データセット

実験で用いたデータは The University of Washington of XML repository [19] から提供している XML データを利用した．データの内容については 3 章の 5.1.1 項で説明があるため，ここでは省略する．その以外，W3C の Query WG で XML のクエリのユースケース [21] で提供する様々なデータ構造を持つサンプル XML データも利用した．

表 4.2: 不完全なパス表現式

パス表現式特徴	不完全なパス表現式
単純パス表現式	QD1: /dblp QD2: /dblp/article/authors QS1: /PLAY QA1: /SigmodRecord QB1: /book
曖昧なパス表現式	QD3: //author QD4: //editor
枝分かれパス表現式	QD5: //(author | editor) QD6: //(article | mastersthesis | phdthesis)/editor
制約条件付き	QD7: /dblp/article/authors[1] QD8: /dblp/mastersthesis/authors[1] QD9: /dblp/article/authors[2]

4.3.1.2 XML データの格納

NID が付けられた XML データを XML データベース [23] に格納する．格納方法は（a）CLOB（Character Large OBjects）などの列にそのまま格納する方法と（b）XML データを分解してデータベーススキーマに写像する方法がある．本研究では前者を採用した．理由は不完全なパス表現式によって探索された生の XML データの全部，または一部に対して自動変換を行ないたいことである．

4.3.1.3 クエリセット

PPX クエリのサブセットとして利用した不完全なパス表現式を表 4.2 で一部を示している．表での QD1，QD2，QS1，QA1 と QB1 は単純パス表現式である．ここで QD1 と QD2 の”D”は dblp データ，QS1 の”S”は Shakespeare データ，QA1 の”A”は SigmodRecord データ，QB1 の”B”は XML のクエリのユースケースで提供する book データのうちのいずれかを表す．また，QD3，QD4 は曖昧なパス表現式であり，QD5 と QD6 は枝分かれパス表現式であり，QD7，QD8 と QD9 は制約条件付きパス表現式など 4 つのタイプの不完全なパス表現式を用意した．

4.3.2 木パターン抽出の評価

ここでは PPX の自動整形出力を用いて XML データから木パターンを抽出する能力を評価する．即ち，イレギュラー XML データの非定型木構造に応じてそれに対応する異なるレイアウト方法を生成することができるか確認する．

4.3.2.1 データサイズの変換

- 異なるデータのサイズの XML データから木パターン抽出

XML データサイズを約 10MB ずつ 100MB まで変化させ，その部分を単純なパス表現式（QD1，QS1，QA1）によって探索された部分 XML から木パターンの抽出を行なった．ただし，属性ノードは現段階で処理できないので除いた．表 4.2 の QD1 は dblp データ，QS1 は Shakespeare データ，QA1 は SigmodRecord データに対してそれぞれ探索を行ない，自動変換ルールを適用した結果，データサイズの変換にはあまり関係なくそれに相応する木パターンを抽出することを確認した．

4.3.2.2 要素ノードの変換

- 異種の要素ノードを含む XML データから木パターン抽出

このような XML データに対して木パターンを抽出した結果，反復出現する XML ノード，および兄弟ノードをそれぞれの根ノードとする部分木は分割されることを確認した．例えば，表 4.2 の単純パス表現式である QD2 によって図 3.2 の XML データから探索された部分 XML は図 4.14(A) に示している．図で”+”は部分木が反復出現することを表し，”//”は木パターンを生成する端点を表す．この部分 XML では反復出現する author 要素ノードや editor 要素ノード以降にある子要素ノード名とテキストノードの値がまったく同じでも親要素ノードが異なるので author 要素ノードを根ノードとする部分木と editor 要素ノードを根ノードとする部分木に分割され，それに相応する木パターンを抽出した．また，曖昧なパス表現式である QD3，または枝分かれパス表現式である QD5 によって探索された部分 XML は図 4.14(B) に示している．この部分 XML では反復出現する author 要素ノードの子要素ノードが異なるので affiliation 要素ノードを含んでいる author 要素ノードを根ノードとする部分木と school 要素ノード含んでいる author 要素ノードを根ノードとする部分木に分割され，それに相応する木パターンを抽出した．更に，図 4.15 の部分 XML はやはり QD3，または QD5 によって探索された XML データであり，図 4.14(B) に示している部分 XML と異なるデータ構造を持つ．この部分 XML では同様な著者の情報を表現しているが反復出現する author 要素ノードの子要素ノードである school 要素ノードがある部分木，school 要素ノードがない部分木と middle 要素ノードが増えた部分木は分割され，それに相応する木パターンを抽出した．

制約条件付きパス表現式を用いた場合は次のことを確認した．図 4.16(A) の部分 XML は QD8 によって探索された XML データであり，authors 要素ノード以降にある 2 つの author 要素ノードを根ノードとする部分木の木構造は異なり，かつそれぞれ一回出現する．これは前例で示した author 要素ノードを根ノードとし，同じ木構造を持つ部分 XML が反復出現することとは異なる．このような場合も last と first 要素ノードを持つ author

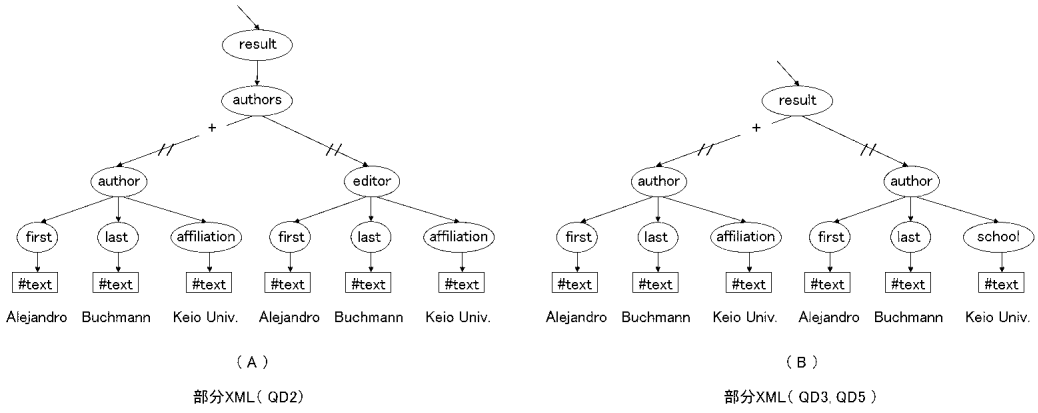


図 4.14: 異種の要素ノードを含む XML データその 1

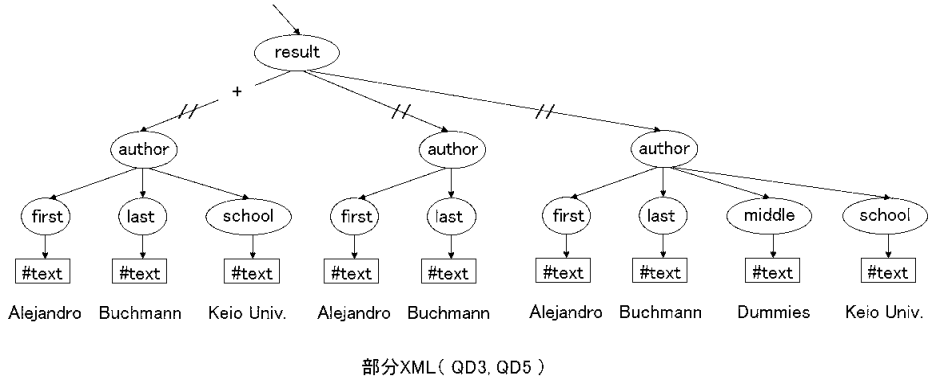


図 4.15: 異種の要素ノードを含む XML データその 2

要素ノードと直接テキストノードを持つ author 要素ノードはそれぞれ部分木として分割され，それに相応する木パターンを抽出することが可能である．

しかし，XML データから木パターンを抽出することは可能が，適切ではなかった場合もあった．例えば，図 4.16(B) の部分 XML は QD7 によって探索された XML データであり，author 要素ノードが一回だけ出現するので author 要素ノードを根とする部分木が分割されず，authors 要素ノードを根ノードとする部分木が 1 個の木パターンとして抽出された．また，図 4.16(C) の部分 XML は QD9 によって探索された XML データであり，異なるデータ種類である author と editor の情報を含んでいる XML データに対して木パターンを抽出した結果，authors 要素ノードを根ノードとする部分木が 1 個の木パターンとして抽出された．それは author 要素ノードや editor 要素ノードがどれでも 1 回しか出現していなかったためである．

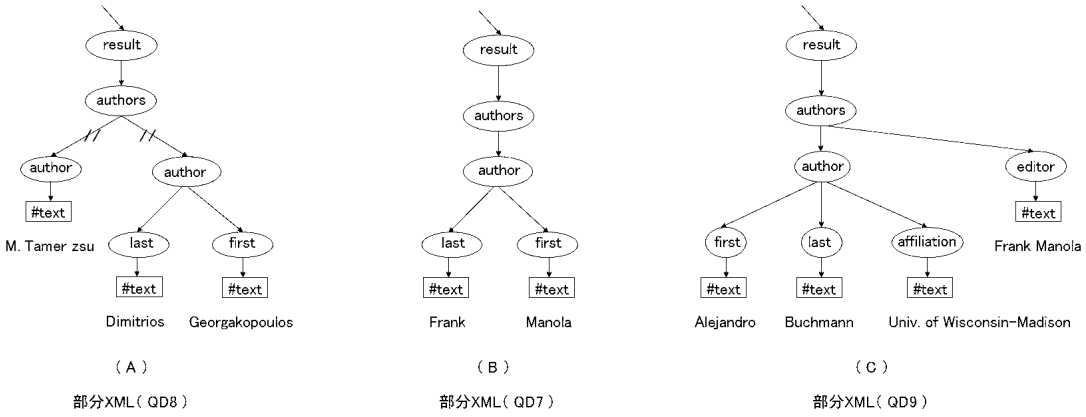


図 4.16: 異種の要素ノードを含む XML データその 3

- 異なる順序で現れる要素ノードを含む XML データからの木パターン抽出

このような XML データから木パターンを抽出した結果，同じデータ種類，かつ同じデータ構造を持つが現れる要素ノードの順序が異なると異なる部分木に分割されることを確認した．例えば，表 4.2 の QD4，または QD5，または Q6 によって探索された部分 XML は図 4.17 で示したとおり 3 つの editor 要素ノードが持つ first，last と affiliation 要素ノードは現れる順序が異なるのでそれぞれ異なる部分木に分割され，それに相応する木パターンを抽出した．

- 再帰構造を持つ XML データから木パターン抽出

このような XML データに対して木パターンを抽出した結果，再帰する部分に対して部分木も再帰的に分割されることを確認した．例えば，表 4.2 の QB1 によって探索された部分 XML は図 4.18 で示したとおりであり，再帰的に出現する section 要素ノードを根ノードとする部分木が出現回数まで分割され，それに相応する木パターン数を抽出した．

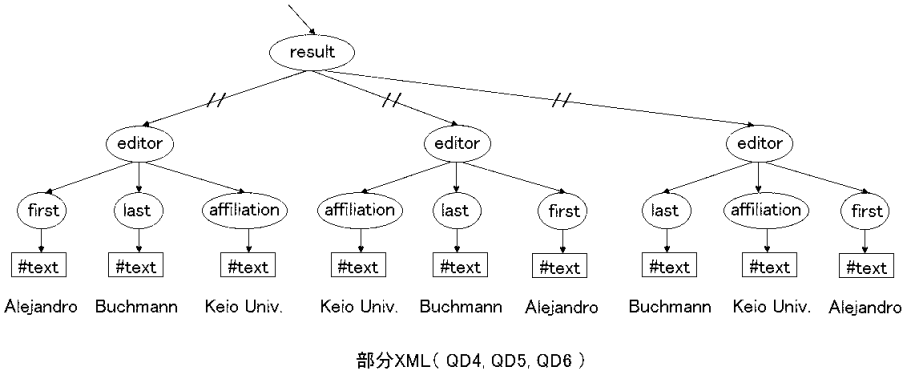


図 4.17: 異なる順序で現れる要素ノードを含む XML データ

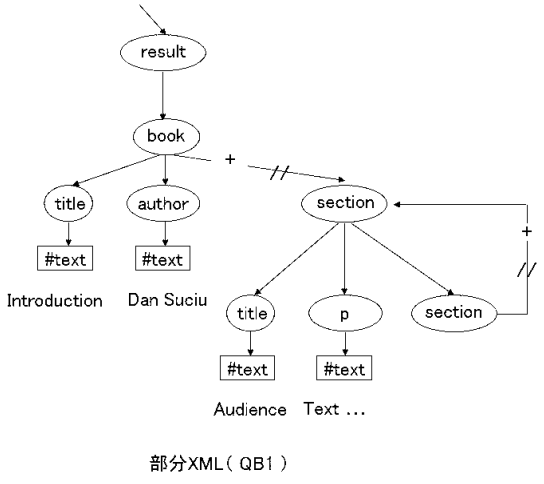


図 4.18: 再帰構造を持つ XML データ

4.3.3 レイアウト効果の評価

ここではイレギュラー XML データのような想定が難しい XML データの不規則性や要素ノードの現れる順序保存などに対して自動変換を行なう PPX の自動整形出力のレイアウト効果を評価する．そのために本論文で提案した自動整形出力ができることを述べた後，PPX のカスタマイズ整形出力と比較した．

4.3.3.1 PPX の自動整形出力

PPX の自動整形出力ができることを表 4.3 でまとめて示している．

表 4.3: PPX の自動整形出力

	自動変換能力
非定型木構造	解析による木パターンの抽出
レイアウト方法	異なる木パターンに異なる変換方法を付与
要素ノードの順序	テキストノードを持つ要素ノードの現れる順に出力
書式情報	HTML の幅のセルなどに設定が可能

- 非定型木構造

自動整形出力は非定型木構造に対して解析を行ない，異なる木パターンを抽出することが可能である．しかし，前項で述べたように図 4.16(C) の部分 XML のような非定型木構造に対して解析することが不可能になる場合もあった．

- レイアウト方法

自動整形出力は異なる木パターンが持つ様々なデータ構造に対して木パターンごとのインスタンスの統計情報に基いて対応するレイアウト方法を生成することが可能である．例えば，The University of Washington of XML repository [19] から提供している XML データを用いて確認した．また，図 4.16(C) の非定型木構造を持つ部分 XML に対して解析はできないがレイアウト方法の生成は可能であった．更に，図 4.19 で (1) は多数のタグに囲まれたテキストノードの値を持ち，(2) はさらに再帰的に出現するタグを持つ．(3) は多数のタグに囲まれた 2 個以上のテキストノードの値を持ち，(4) は異なるレベルにテキストノードの値を持つなど様々な形で深い階層構造を持つ XML データを示す．このような XML データに対してもタグの深さに関係なく正しくレイアウト方法が生成できることを確認した．

- 要素ノードの順序

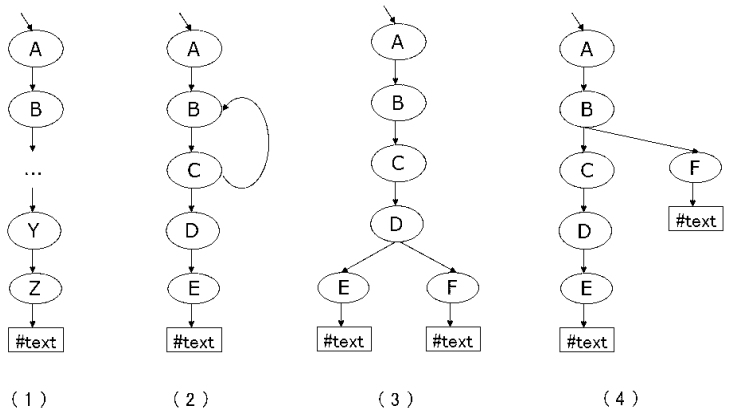


図 4.19: 深い階層構造を持つ XML データ

文書中心の XML データに対して重複データを許し，要素ノードの現れる順に出力することができる．例えば，図 4.16(C) の部分 XML に対して author と editor 要素ノードがそれぞれ根ノードとする部分木に分割できなかったがそれに対して要素ノードの現れる順（先に author のデータ，次は editor のデータの順）に出力することができた．また，図 4.17 で示したデータの種類やデータ構造，かつテキストノードの値が同じである部分 XML に対して重複データを許し，要素ノードの出現順序（first，last，affiliation の次は affiliation，last，first，そして last，affiliation，first の順）のとおりに出力することができた．更に，図 4.18 の部分 XML は階層構造を持ち，section の部分が再帰的に出現する XML データに対しても兄弟要素ノードの現れる順（title，author の順）はもちろん，親子要素ノードの現れる順（まずは section 要素ノードの親ノードである book 要素ノードが持つ title，author の順，その後は book 要素ノードの子ノードである section 要素ノードが持つ title，P の順）に出力することができた．

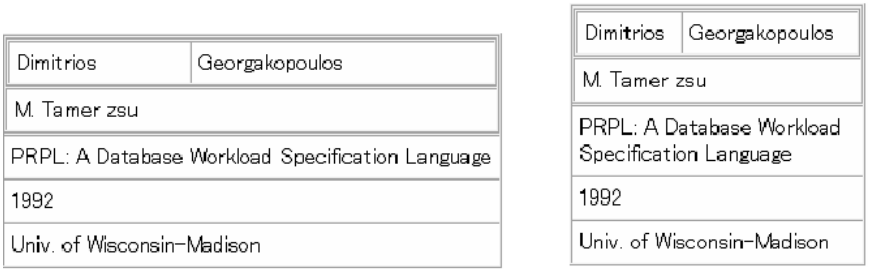
- 書式情報

その他，要素ノードのテキストノードの値である文字列長の平均値からセルの幅を自動的に設定できるようにしたのでセルの幅を適切に制御することが可能であった．その結果を示す 1 つの例は図 4.20 のとおりである．

4.3.3.2 PPX のカスタマイズ整形出力との比較

XML データに対するレイアウト式の生成結果，およびそれによって変換されたレイアウト満足度についてカスタマイズ整形出力と比較を行なう．

- レイアウト式の生成結果



（1）セルの幅に設定なし　　（2）セルの幅に設定あり

図 4.20: PPX によるフォーマット結果 1

主にイレギュラー XML データの非定型木構造を持つ XML データ，異なる順序で現れる要素ノードを持つ XML データに対してレイアウト式の生成結果について比較を行なった．

(1) 非定型木構造に対するレイアウト式の生成結果の比較

ここでは QD2 によって探索された図 4.14 の部分 XML に対してカスタマイズ整形出力を行なうためのユーザ指定によるレイアウト式，並び自動整形出力を行なうためのシステムによって決定されたレイアウト式を表 4.4 で示す．この例から分かるように，カスタマイズ整形出力を行なう場合は条件分岐構文を用いて authors 要素ノード以下に author のデータが現れているか，或いは editor のデータが現れているか判断しながらそれに対応する処理を行なう．また，ファーストネームとラストネームを { } によって先に横連結してから所属を縦連結を行なう．これに対して自動整形出力を行なう場合は条件分岐構文を自動生成することができないし，ファーストネーム，ラストネームと所属に対して演算子の優先順位を変換することもできない．ただ，author のデータをすべて横に連結したものと editor のデータをすべて横に連結したものをさらに縦連結を行なう．また，要素ノードの現れる順序を保存する演算子を自動整形出力では三回も使っているがカスタマイズ整形出力では一回で済む．なお，表 4.4 と次の表 4.5 の自動整形出力のレイアウト式では簡潔性のために省略しているが実際には木パターンが持つ PID が更に追加される．

(2) 異なる順序を持つ要素ノードに対するレイアウト式の生成結果の比較

ここでは同じデータ種類，かつ同じデータ構造を持つ図 4.17 の部分 XML で first，last と affiliation 要素ノードの現れる順序が異なる場合に対してカスタマイズ整形出力を行なうためのユーザ指定によるレイアウト式，並び自動整形出力を行なうためのシステムによって決定されたレイアウト式を表 4.5 で示す．この例から分かるように，カスタマイズ整形出力の場合は同じ木構造であり，異なる順序で出現する last，first と affiliation 要素ノードに対して NID を用いて多数の if 文を指定する．それに対して自動整形出力の場合は抽出した木パターン数と等しいサブレイアウト式が自動的に生成される．

- レイアウト満足度

表 4.4: レイアウト式の生成結果の比較例その 1

QD2 によって探索された部分 XML（図 4.14）	
変換種類	レイアウト式の生成結果

カスタマイズ整形
(ユーザ指定)
	<
	if ($j/author)
	then
	({ $j/author/first , $j/author/last } ! $j/author/affiliation)
	else
	({ $j/editor/first , $j/editor/first } ! $j/editor/affiliation)
	>!

自動整形 (自動生成)	< < $j/author/first , $j/author/last , $j/author/affiliation >! ! < $j/editor/first , $j/editor/last , $j/editor/affiliation >! >!

表 4.5: レイアウト式の生成結果の比較例その 2

QD4 によって探索された部分 XML（図 4.17）	
変換種類	レイアウト方法
カスタマイズ整形 (ユーザ指定)	< if ($i/last/@nid < $i/first/@nid < $i/affiliation/@nid) then ($i/last , $i/first , $i/affiliation) else if ($i/last/@nid > $i/first/@nid < $i/affiliation/@nid) then ($i/first , $i/last , $i/affiliation) ... >!
自動整形 (自動生成)	< < $j/author/first , $j/author/last , $j/author/affiliation >! ! < $j/author/last , $j/author/first , $j/author/affiliation >! ... >!

XML データの自動 HTML 化に対して変換されたレイアウト満足度について比較を行なった．カスタマイズ整形出力は XML データに対してユーザの用途に応じて様々な表構造の HTML を出力するワンソース・マルチユーズを実現することが可能である．例えば，図 4.14 の部分 XML で author の情報に対してカスタマイズ整形出力の場合は何十種類のレイアウト式が指定できる．また，これらのレイアウト式によって変換された HTML 表を一般ユーザの評価によってレイアウト満足度の良い順にランキングされた一部を表 4.6

で示している．これに対して自動整形出力を行なった場合，ユーザ指定によってカスタマイズ整形出力を行なうレイアウト式のどの部分に当たるかを確認した．その一例を表 4.6 で示したように自動生成したレイアウト満足度は 3 位にランクされた．このような方法を用いて実験した結果，異なるデータ種類や異なるデータ構造に対して異なる木パターンを抽出した場合はレイアウト効果が上位にランキングされていることが分かった．そうではない場合はレイアウト効果が落ちる．図 4.16(C) の部分 XML で author 要素ノードと editor 要素ノードが分割されなかった場合と分割された場合に対して自動変換した結果を図 4.21 で示す．図 4.21 の (1) は 2 人の著者のデータが横に連結されてしまう．また，図 4.22 ではもう 1 つの出力結果を示す．この図から分かるように，カスタマイズ整形出力ではテーブルヘッダの指定，および著者名は縦に連結された個人情報と横に連結するなどをユーザのニーズに合わせて変換を行う．それに対して自動整形出力ではテーブルヘッダの指定ができないのでメタ情報が失われる．また，自由にレイアウトを設定できないので著者名と個人情報は単純に結合連結を行う．

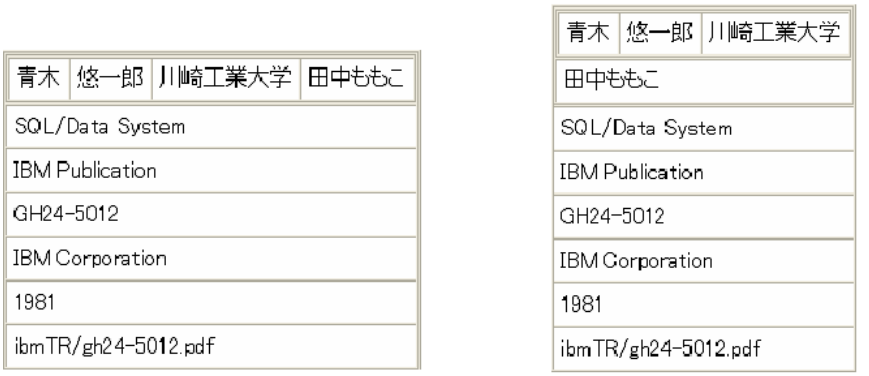
表 4.6: レイアウト満足度の比較例

利用した XML データ: DBLP		
利用した不完全なパス表現式: /dblp/article/authors/author		
変換種類	満足度	レイアウト方法
カスタマイズ整形 (ユーザ指定)	1	< { $j/author/first , $j/author/last } ! $j/author/affiliation >!
	2	< { $j/author/first , $j/author/last } , $j/author/affiliation >!
	3	< $j/author/first , $j/author/last , $j/author/affiliation >!
	4	[{ $j/author/first , $j/author/last } ! $j/author/affiliation]!
	...	...
自動整形 (自動生成)	3	< $j/author/first , $j/author/last , $j/author/affiliation >!

4.3.3.3 検討

実験を通じて PPX の自動整形出力はカスタマイズ整形出力よりユーザにとって記述量を減らすこと，レイアウト対象となる XML データの構造，要素ノードの現れる順序保存，およびそれに対するレイアウト出力などを直接意識することなく XML データを HTML 化することができる．しかし，次のような欠点もある．

(1) 部分 XML 全体に対してそのまま自動変換を行なうため，必要ではない要素ノードのテキストの値も出力されてしまう．即ち，自動整形出力は不完全なパス表現式によって XML データの一部分が探索され，その部分に含まれているすべての XML ノードが変換・出力する対像となる．図 4.23 で示すように部分 XML に対して自動整形出力する場合，その部分に含まれている著者名や個人情報が自動変換される対象となって出力される．これに対してカスタマイズ整形出力はユーザが必要とする要素ノード，たとえば著者の個人情



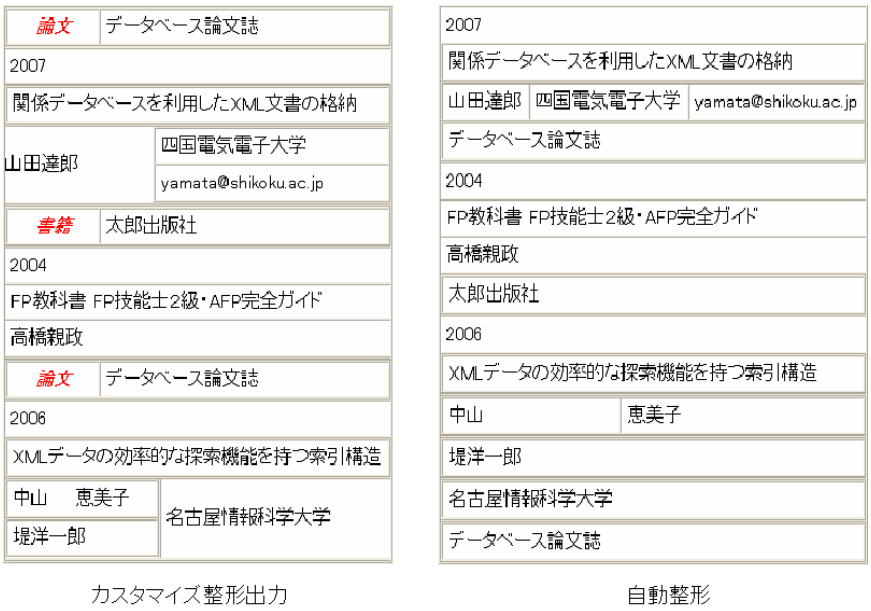
青木	悠一郎	川崎工業大学	田中ももこ
SQL/Data System			
IBM Publication			
GH24-5012			
IBM Corporation			
1981			
ibmTR/gh24-5012.pdf			

(1) 部分木に分割されてない場合

青木	悠一郎	川崎工業大学
田中ももこ		
SQL/Data System		
IBM Publication		
GH24-5012		
IBM Corporation		
1981		
ibmTR/gh24-5012.pdf		

(2) 部分木に分割された場合

図 4.21: PPX によるフォーマット結果 2



論文	データベース論文誌
2007	
関係データベースを利用したXML文書の格納	
山田達郎	四国電気電子大学
	yamata@shikoku.ac.jp
書籍	太郎出版社
2004	
FP教科書 FP技能士2級・AFP完全ガイド	
高橋親政	
論文	データベース論文誌
2006	
XMLデータの効率的な探索機能を持つ索引構造	
中山　恵美子	名古屋情報科学大学
堤洋一郎	

カスタマイズ整形出力

2007		
関係データベースを利用したXML文書の格納		
山田達郎	四国電気電子大学	yamata@shikoku.ac.jp
データベース論文誌		
2004		
FP教科書 FP技能士2級・AFP完全ガイド		
高橋親政		
太郎出版社		
2006		
XMLデータの効率的な探索機能を持つ索引構造		
中山	恵美子	
堤洋一郎		
名古屋情報科学大学		
データベース論文誌		

自動整形

図 4.22: PPX によるフォーマット結果 3

報は除いて名前だけ出力する．

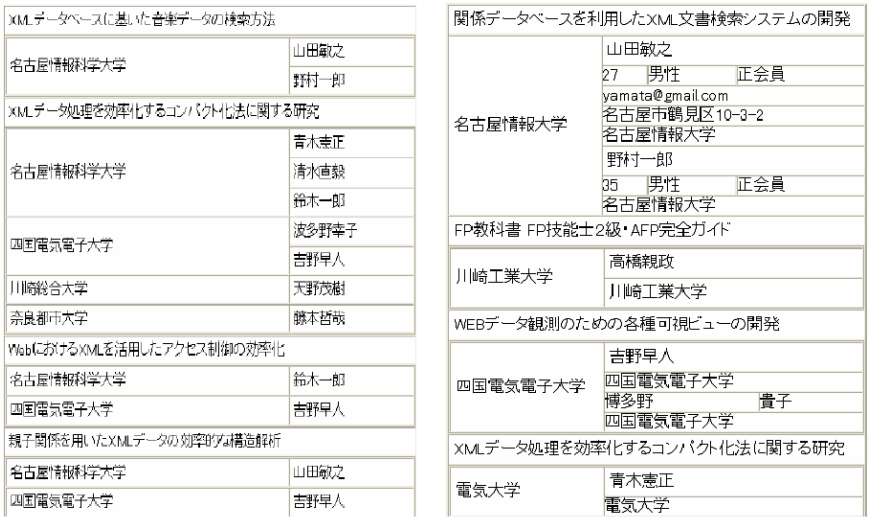


図 4.23: PPX によるフォーマット結果 4

(1) 混在内容を持つ XML データに対して整形出力を行なう場合，自動整形出力はもちろんカスタマイズ整形出力も同じ問題点が発生する．例えば，次のような XML データに対して整形出力を行なうとする．

<person> 田中さんは <age>23</age> です．</person>

この場合，age 要素ノードは person 要素ノードの子要素なので出力された HTML 表では 23 が別のセルでレイアウトされてしまう．これによって HTML 化された文書の一貫性がなくなる．

(3) 再帰的部分を持つ XML データに対して整形出力を行なう場合，自動生成したレイアウト式が冗長になる．例えば，表 4.2 の QB1 によって探索された図 4.18 の部分 XML で section 要素ノード以下の部分が再帰的に出現する場合，それに対するレイアウト式は再帰的に生成してしまう．これは XQuery のように再帰的に所定の処理をするような関数を作成することができないことである．

第 5 章

結論

本論文では広く普及しているXQueryやXSLTとは異なる方法でXMLデータからHTMLへの変換，出版を行なうXML整形出力言語PPX（Pretty Printer for XML）を提案し，実験に基づいてその有用性を示した．

2章では本研究で関連する基本概念とXML整形出力言語PPXの言語構文を述べた．そして，3章ではPPXによるXMLデータのカスタマイズ整形出力を述べた．

本方法はRDBに対して同様のことを行なう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，大規模な人工XMLデータと実XMLデータを用途に応じて見易く加工し，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．

また，XMLデータを整形出力する際に木構造の再構成を含む詳細な指定によってプログラマの意思に基づいて整形を行う要求がある一方，元の木構造の論理的な構成を変えずに自然な出力をなるべく少ない手間で得たいという相反する要求が存在する．4章ではこれを実現するPPXによるXMLデータの自動整形出力を述べた．

本方法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ことを目的とする．レイアウト対象となるXMLデータから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する．そして木パターンごとの統計情報などに基づいて自然な階層構造を持つHTMLへの自動変換を行なう．実験によって提案した自動整形出力を様々なXMLデータに適用し，その有効性を確認した．

以上に述べた，本論文で提案する二つの手法は排他的ではなく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が可能である．

今後の研究課題は次のとおりである．

(1) 本研究で提案した自動変換ルールによって自動的に生成したレイアウト方法が不適合と判断された場合に連結方法の再調節などレイアウト方法の最適化を行う必要があると考えられる．

(2) 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．

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., Toyama M., A Formatting Method for Transforming XML Data into HTML, International Journal of Computer and Information Science and Engineering, vol.2，no.1，pp.52-58，2008.

[7] Jin Z., Toyama M., A Proposal of an Automatic Formatting Method for Transforming XML Data, International Journal of Computer Science, vol.3，no.3，pp.182-189，2008.

[8] Jin Z., Toyama M., A prototype implementation of PPX: Pretty Printer for XML, The Fourth International Conference on Internet and Web Applications and Services(ICIW), pp.149-156, 2009.

[9] 金哲，遠山元道: PPX：XML 整形出力のための出版言語, 情報処理学会論文誌データベース, Vol.3, No.4, pp.13-33, 2010.12.

[10] Seto, T., Nagafuji, T. and Toyama, M., Generating HTML Sources with TFE Enhanced SQL, ACM Symposium on Applied Computing(SAC'97), ACM(1997), pp.96-105, 1997.

[11] Hosoya, H., and Pierce, B. C., XDuce: A Statically Typed XML Processing Language, ACM Transactions on Internet Technology, pp.117-148, 2003.

[12] Toyama, M., SuperSQL: An Extended SQL for Database Publishing and Presentation, Proc. ACM SIGMOD, pp.584-586, 1998.

[13] XPath 1.0: XML Path Language (XPath) 1.0, http://www.w3.org/TR/xpath/.

[14] XPath 2.0: XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20.

[15] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmigasundaram, Eugene Shekita and Chun Zhang, Storing and Querying Ordered XML Using a Relational Database System, In Proc. ACM SIGMOD 2002, pp.204-215, 2002.

[16] Tennison, Jenni (2007), Grouping Using the Muenchian Method. Available http://www.jenitennison.com/xslt/grouping/muenchian.html.

[17] DBLP data set: ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz

[18] XMLSpy: http://www.Altova.com/XMLSpy.

[19] The University of Washington XML repository: http://www.cs.washington.edu/research/xmldatasets.

[20] http://metalab.unc.edu/bosak/xml/eg/shaks200.zip

[21] Chamberlin, D., Fankhauser, P., Florescu, D., Marchiori, M. and Robie, J., XML Query Use Cases. http://www.w3.org/TR/xquery-use-cases/.

[22] 実験結果: http://www.db.ics.keio.ac.jp/PPX/

[23] 天笠俊之, 吉川正俊: XML データベース技術概説, オペレーションズ・リサーチ, vol.50, no.6, pp.365-372, 2005.

[24] S. Abiteboul, P. Buneman and D Suciu, Data on the Web: From Relations to Semistructured Data and XML, Morgan Kauffman Publishers, Los Altos, 1999.

[25] John N. Wilson, Richard Gourlay, Robert Japp, Mathias Neumuller, Extracting Partition Statistics from Semistructured Data. DEXA Workshops, pp. 497-501, 2006.

付 録 A

拡張 TFE

A.1 構文規則（拡張 TFE）

<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.2 演算子の優先順位

降順に示す

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

付 録B

本言語が利用するXPath式のサブセット

```
[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．