推薦論文

変換結果スキーマ指向のXML変換

鬼塚 真[†] 小西一也[†,☆]

本論文ではXMLからXMLへの変換モデルと変換言語XTLについて提案する．本変換モデルの特徴は，1) スキーマ表現を用いて変換結果の構造を表現すること，2) 入力XMLの解析構文と変換結果スキーマを一元的に表現すること，3) データ操作の基本操作（重複排除，部分木抽出，変数の自動割当てなど）を導入することで，簡略なXML変換の記述を可能にしていることにある．実用問題を用いた評価実験の結果，XTLはXML変換プログラムの生産性をXSLTの5〜10倍に向上させたことを確認した．さらに記述能力に関して，結果構造が文脈自由文法に従わなければならないという制約があることを除いて，XTLはXSLTの形式的モデル $XSLT_0$ と同等の能力を有することを確認した．またW3Cのquery use casesにXTLを適用し，69の問合せの67がXTLによって表現可能であることを確認した．

Output Schema Driven XML Transformation

Makoto Onizuka[†] and Kazuya Konishi[†,☆]

This paper describes an XML transformation model and language called XTL (XML transformation language). The transformation model enables us to write a concise transformation programs by the following features; 1) the output schema is specified by an XML schema language (we are currently using DTD), 2) the grammar rules for input data and for output schema are integrated, 3) fundamental data operations (eliminating duplicated values, sort, implicit variable declaration) are supported. Experiments show that XTL improves the productivity of XML transformation program from five to ten of XSLT's. We prove the expressiveness of XTL is as powerful as $XSLT_0$, a formal model of XSLT, except XTL limits the output schema to follow a context free grammar. In addition, we applied XTL to the W3C query use cases, and the results showed that XTL can express 67 of the 69 queries.

1. はじめに

B2BやB2Cに代表されるデータ交換は普及しつつあり，このためXMLボキャブラリの標準化などがなされている．たとえば放送コンテンツを見るとメタデータの形式としてTVAnyTime[15]，MPEG7[9]，P/meta[12]などが整理されている．しかし放送コンテンツのメタデータの例に示されるように，類似した複数の標準XMLボキャブラリが普及しており，放送業界の中でのコンテンツの売買を促進するためには，複数のXMLボキャブラリの間でのXML変換の機能が必須である．

あるXML形式から異なるXML形式への変換を実現する方法としてXSLT[13]を利用する方法が広く普及しているが，通常のプログラマにはXSLTの仕様は複雑であり，XSL FAQ[13]にあるような特別な実装テクニックをしばしばを使う必要がある．XML変換の典型的な最初の例として，重複データの排除変換を考えてみよう．これを実現するXSLTプログラムは次に示すように手続き的で複雑なものである．

```
...
<xsl:template match="bib">
  <xsl:element name="bib">
    <xsl:apply-templates select="(//author)"
                         mode="distinct">
      <xsl:with-param name="nodes" select="//author"/>
    </xsl:apply-templates>
  </xsl:element>
</xsl:template>

<xsl:template match="author" mode="distinct">
  <xsl:param name="nodes"/>
  <xsl:variable name="pos" select="position()"/>
  <xsl:if test="count($nodes[$pos>position() and
                             .=current()/.])=0">
    <xsl:apply-templates select="." mode="author"/>
  </xsl:if>
</xsl:template>
...
```

† 日本電信電話株式会社NTTサイバースペース研究所
NTT Cyber Space Laboratories, NTT Corporation
☆ 現在，株式会社NTTデータ技術開発本部に所属
Presently with Research and Development Headquarters, NTT Data Corporation

本論文の内容は2003年5月のデジタル・ドキュメント研究会にて報告され，DD研究会主査により情報処理学会論文誌への掲載が推薦された論文である．

この例では 2 つの XSL のテンプレートを利用しており，最初のテンプレートで author リストを取得し，2 つ目のテンプレートで author リストに含まれる author 要素について重複値を排除している．具体的には，author リストに含まれる個々の author 要素について，author リストにおけるその位置より前に同じ値の author 要素があるか否かを判断し，そのような要素がなければ author 要素を出力するという処理になっている．これと等価な変換を XTL で記述した例を以下に示す．

```
<!ELEMENT bib AS {/bib} (auhtor * AS {//author}
                              DISTINCT BY {.})>
<!ELEMENT author          ANY>
```

このように XTL では，1) 変換後の構造を DTD で指定し，2) XPath 式により入力 XML を解析して，指定した DTD 構造を入力 XML の解析構文として利用し，3) DISTINCT（重複排除操作）や ANY（サブツリーのコピー）などのデータ操作の基本操作を利用することで，簡略な XML 変換の記述を可能としている．

次に 2 つ目の例として，複数の XML を結合するような XML 変換の例を考えてみよう．XSLT プロセッサ（xalan，saxon，xt など）は，複数の XML の結合処理を単純な 2 つのループを利用して処理するため，大規模な XML の処理ではスケーラビリティが良くない．この問題に対して，XSLT では xsl:key という命令を提供していて，プログラマは xsl:key を使うことでスケーラブルなハッシュ結合を利用することができるが，xsl:key の使い方は記述が煩雑であるという問題がある．

これら上述した XSLT の可読性に関する問題を解決するため，本論文では新たな XML 変換モデルを提案する．そして，そのモデルを実現する XML 変換言語 XTL と，XTL を XSL へ翻訳するための翻訳方法を提案する．XTL の特徴は，1) スキーマ表現を用いて変換結果の構造を表現すること，2) 入力 XML の解析構文と変換結果スキーマを一元的に表現すること，3) データ操作の基本操作（重複排除，部分木抽出，変数の自動割当てなど）を導入することで，簡略な XML 変換の記述を可能にしていることにある．評価実験の結果，XTL プログラムから自動的に翻訳される XSLT プログラムのサイズが平均的に XTL プログラムの 10 倍であることから，XTL は XML 変換プログラムの生産性を XSLT の約 10 倍向上させることができるといえるだろう．

また表現能力については，XTL では変換結果の構造の指定に DTD を用いることから，変換結果が文脈自由文法に従わなければならないという制約があるが，それを除けば XTL の表現能力は XSLT のサブセット $XSLT_0$ [1] と等価な能力を有することを 5 章で示す．また，W3C の問合せのユースケースに XTL を適用し，69 の問合せのうち 67 が XTL により記述可能であることを確認する．

本論文の構成を示す．2 章では XTL の変換モデルを説明し，続く 3 章ではこの変換モデルを表記する XTL の言語仕様を定義する．4 章では XTL プログラムを XSLT プログラムに翻訳する方法について述べる．5 章では提案モデルの特徴の効果を確認した実験結果を報告する．6 章では関連研究について整理し，7 章で本論文をまとめる．

2. XTL 変換モデル

XTL プログラムは変換規則 $r = (\Delta, M, V^*, \Sigma)$ の集合である．

- Δ は変換規則を一意に特定する変換シンボルである．変換規則を適用するための条件として XPath 式を付加的に指定することが可能である（XPath が指定された変換規則を，**ルート候補の変換規則**と呼ぶ）．ルート候補の変換規則の場合，指定された XPath 式が入力 XML データにおけるカレントノードにマッチするときにのみ，展開規則 Σ（後述）が適用される．カレントノードは，最初は入力 XML データのルートノードが設定され，変換規則が適用されたらその展開規則で指定された XPath 式の評価結果である node-set に属する個々のノードが順次カレントノードとして設定される．一方，ルート候補でない変換規則の場合は，他の変換規則の展開規則から呼び出される．直感的には，変換シンボルは生成規則における右辺であり，展開規則 Σ は生成規則における左辺である．
- M は出力モード指定であり，適用された変換規則のコンテキストノードに対して，どのように結果ノードを出力するかを指定する．このモード指定には 3 種類（出力なし，固定名称/固定値，入力ノードと同一の名称/値）ある．
- Σ は展開規則であり，以下の 4 つを指定をする．1) 呼び出す下位の変換規則を指定する変換シンボル，2) 下位の変換規則群の呼び出し順序や組合せの指定，3) 下位の変換規則を適用する際のコンテキストノードを指定するための XPath 式，4) XPath 式の結果の node-set のカージナリティに関する制約．

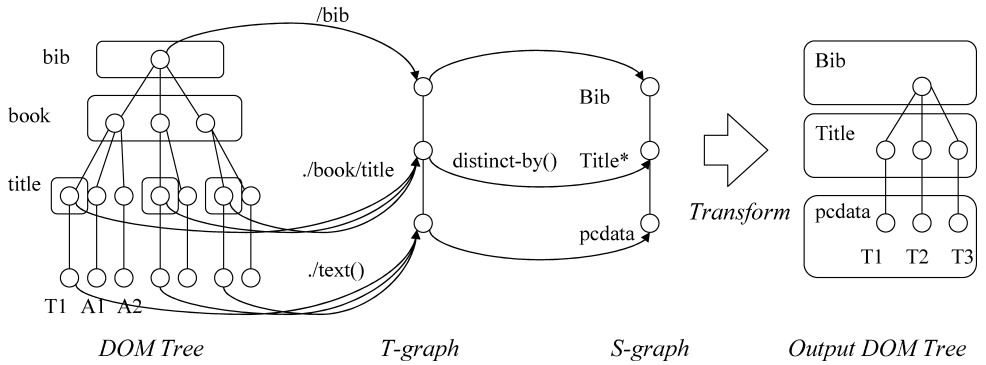


図 2 図 1 に示された XTL プログラムの変換モデル
Fig. 2 The transformation model of the XTL program in Fig. 1.

```
<!ELEMENT Bib AS {/bib} (Title * AS {./book/title}
                              DISTINCT BY {.})>
<!ELEMENT Title         (#PCDATA AS {./text()})
```

図 1 XTL プログラムの例
Fig. 1 An XTL program example.

- V は変換規則が適用されたコンテキストノードをバインドする変数であり，この変数を用いることで現在の変換規則の祖先変換規則のコンテキストノードを参照することができる．

XTL の変換モデルにおいて重要な特徴は，変換規則の集合が yacc プログラムと同様に入力 XML の解析構文を表すと同時に，変換結果スキーマも表していることにある．この結果，XTL のプログラムは XSLT や XQuery の表現よりも簡潔に XML 変換を表現することが可能である．また変換結果の静的な型チェックが可能となり，スキーマバリデーションによる動的な変換結果の型チェックが不要になるという利点も有する．

ここで，XTL の変換規則群が表現する入力 XML の解析構文を *T-graph*（transformation graph）と呼び，変換結果スキーマを *S-graph*（output schema graph）と呼ぶこととする．これらは XTL における変換の内部モデルである．

T-graph *T-graph* はグラフであり，変換規則群の変換シンボルがノードに対応し，各変換規則の展開規則で表現される下位の変換規則群との関係が枝で表現される．変換規則で指定される XPath 式は，入力 XML のノードを *T-graph* のノードへ対応付ける．これは XPath 式の評価結果である node-set に含まれる各ノードを，順にコンテキストノードとして *T-graph* のノードで示される変換規則で使用することを指定している．

S-graph *S-graph* のノードは出力結果スキーマのノードを表現し，スキーマのノード間の親子関係は *S-graph* の枝によって表現される．*S-graph* のノードは，出力モード指定により出力指定された変換シンボル，つまり *T-graph* のノードに対応する．同様に *S-graph* の枝は，*T-graph* の枝と対応する．また，*T-graph* のノードから *S-graph* のノードへの対応に対して，node-set に対する集合演算（ソート，重複排除など）を指定することができる．これは，*T-graph* のノードに対応付けられた XPath 式の評価結果 node-set に対して，指定された集合演算を実行することを意味する．

具体例として，**図 1** で示される XTL プログラムの変換モデルを**図 2** に示す．*T-graph* では，DOM 木に対して XPath 式/bib を評価し，その結果 node-set に含まれるノードをコンテキストノードとして，最上位の変換規則が適用されることを示している．そしてこの変換規則は，コンテキストノードに対して XPath 式./book/title を評価し，その結果の node-set に含まれる各ノードごとに下位の変換規則を適用する．*S-graph* では，最上位の変換規則の適用ごとに Bib 要素を出力し，その下位の変換規則の適用ごとに Title 要素を出力することを示している．また DISTINCT が指定されているため，下位規則の適用の際に ./book/title の XPath 式の評価結果の node-set について，ノード全体の等価性によって重複が排除されることを示している．このようにして最終的に図 2 の右側に示される変換結果（Output DOM Tree）を得ることができる．

3. XTL 言語仕様

XTL プログラムを構成する変換規則 $r = (\Delta, M, V^*, \Sigma)$ は，以下の 3 つの要素により表現される．1) 解析構文（*T-graph*），変換結果スキーマ（*T-graph*）を一元的に表現する拡張 DTD，2) 変換規則を適用

するコンテキストノードを取得するための XPath 式，3) node-set に対する集合演算（ソート，重複排除などの node-set の演算）．直感的には，変換シンボル Δ は DTD で宣言された要素に対応し，展開規則 Σ は DTD の内容モデルに相当する．出力モード指定 M とノード変数 V の詳細は以降で述べる．また，DTD の属性の宣言については要素の宣言の考え方に準ずるため，本論文では説明を省略する．

3.1 拡張 DTD

2 章で述べたように XTL 変換モデルでは，出力モード指定には 3 種類（出力なし，固定名/値割当て，入力ノードと同一の名称/値の割当て）ある．まず中間ノード（要素）に関して 3 種類の出力モードの指定方法を説明した後，葉ノード（テキスト値）に関して固定値の割当てと入力ノードにより演算された値の割り当て方法を説明する．

固定名称の要素を出力したい場合は DTD の要素型宣言を用いて，要素名を出力しない場合は DTD を拡張した変数型宣言を用いる．要素型は *T-graph* と *S-graph* のノードを一元化したノードに対応し，変数型は *T-graph* に存在しないが *S-graph* に存在するノードに対応する．また，入力ノードと同一名称の要素名称を出力したい場合は，タグ変数を用いた要素型宣言を用いる．たとえば，

```
<!ELEMENT book (author*, title, publisher)>
```

という要素型宣言による変換規則では，本変換規則が適用される入力ノードに対して，変換結果として book 要素という固定名称を出力する（内容モデルの部分に関しては後述する）．一方，

```
<!VARIABLE book (author*, title, publisher)>
```

という変数型宣言による変換規則では，本変換規則が適用される入力ノードに対して，変換結果を出力しない．また，

```
<!ELEMENT $book (author*, title, publisher)>
```

というように $ から始まる名称の場合はタグ変数であり，入力ノードと同一名称の要素名称を出力する．

次に値を指定する葉ノードに関しては，XPath で用いられる関数（count，max などの集約関数や，concat などの文字列操作関数など）を用いて演算した結果を割り当てるか，もしくは固定値を割り当てることが可能である．たとえば，

```
<!ELEMENT author_count (#PCDATA VALUE {count(author)})>
<!ELEMENT version      (#PCDATA "Jan/3/2003")>
```

の最初の変換規則では，要素 author_count の値は count（author）の評価結果が割り当てられ，2 番目では要素 version のテキスト値として “Jan/3/2003” という固定値が割り当てられる．

要素内容

要素内容では DTD の定義と同様に *sequence* と *choice* という 2 つの指定方法がある．*sequence* 指定では，要素内容で指定された順で下位の変換規則が呼び出されるのに対し，*choice* 指定では，入力 XML のノードの文書順でマッチする XPath 式が指定された変換規則から順に呼び出される．いい換えると，*sequence* 指定では入力構造を変更する変換を実行し，*choice* 指定では入力構造を保存する変換を実行する．たとえば，

```
<!ELEMENT R AS {root} (A* AS {a}, B* AS {b}, C* AS {c})>
```

という *sequence* を用いた変換規則は，root 配下のノードについて最初に XPath 式 a にマッチしたノードをコンテキストノードとして A で指定される変換規則を呼び出し，次に変換規則 B，C という順で呼び出す．もし入力 XML が `<root><c/><a/><b/><a/></root>` ならば，出力結果は`<R><A/><A/><B/><C/></R>`となる．また，

```
<ELEMENT R AS {root} (A AS {a}| B AS {b} |C AS {c})*>
```

という *choice* を用いた変換規則は，root 配下のノードについてノードの文書順で，XPath 式 a，b，c にマッチするノードをコンテキストノードとして変換規則 A，B，C を呼び出す．もし入力 XML が `<root><c/><a/><b/><a/></root>` ならば，出力結果は`<R><C/><A/><B/><A/></R>`となる．

また DTD では，内容モデルの中の要素もしくは括弧によってまとめられた部分に対して*，+，?というように出現回数を指定することが可能であるが，XTL ではそれらは XPath 式の結果 node-set の要素数に対する制約であると定義する．たとえば，

```
<!ELEMENT book (author+ AS {author}, title AS {title})>
```

という変換規則は，入力 XML の book 配下の構造に対して author なる XPath 式にマッチするノードが 1 以上あり，かつ XPath 式 title にマッチするノードが存在しなければならないという制約を意味する☆．

混在内容

XTL では混在内容を拡張し，#PCDATA と要素が同様に扱われるようにしている．より正確には #PCDATA はタグのない要素という位置付けである．このため XTL では正規の DTD で許されていない次のよ

☆ 利用者の利便性を考慮し，後者の制約は XPath 式の結果 node-set のエントリ件数が 1 件であることは制約せず，node-set の先頭の要素を出力するものとして扱う．つまり，title は title[1] として解釈される．

うな表現が可能であり，この結果多様な変換を記述することができる．

```
<!ELEMENT paper (#PCDATA, title)>
<!ELEMENT paper (title | #PCDATA)>
```

EMPTY

EMPTY は XPath 式の評価結果である node-set が空集合であるという制約を意味する．

ANY

ANY は XPath 式の評価結果に含まれる（マッチする）ノード配下のサブツリーをコピーする．たとえば，

```
<!ELEMENT bib AS {bib} (book* AS {book})>
<!ELEMENT book ANY>
```

という変換規則は，XPath 式 book にマッチするノードごとに 2 行目の変換規則が適用され，そのノード配下のサブツリーをコピーして出力する．ただし，ルート候補変換規則があって，その変換シンボルに指定された XPath 式が ANY にマッチしたサブツリーのノードにマッチする場合は，サブツリーのコピー処理よりもそのルート候補の変換規則が優先されて，変換が実行される．たとえば，

```
<!ELEMENT bib AS {bib} ANY>
<!ELEMENT given AS {firstname} ANY>
```

という変換規則は，firstname の要素名を given に変更し，それ以外は入力 XML のデータ構造を保存する変換を示している．

3.2 XPath

XPath 式は変換規則を適用するコンテキストノードを取得するために用いられる．具体的には，XPath 式を評価した結果である node-set に含まれる各ノードをコンテキストノードとして変換規則が呼び出される．XPath 式の指定方法は，宣言される要素/変数に指定される場合と（ルート候補の変換規則），要素内容において下位の変換規則に指定される場合の 2 通りがある．

ルート候補の変換規則は次のいずれかのケースに呼び出される．1) 入力 XML のドキュメントルートをコンテキストノードとして評価した際に，XPath 式にマッチするノードがある場合，2) ANY が指定された変換規則が呼び出されていて，ANY にマッチしたサブツリーのノードに，ルート候補の変換規則で指定された XPath 式がマッチする場合．たとえば，ANY で説明した 2 番目の XTL プログラムの 1 行目は 1) の例であり，2 行目は 2) の例である．

ノード変数

XPath 式の記述では，現在もしくは祖先の変換規則（*T-graph* における祖先ノード）が適用されたコンテキストノードを，ノード変数 V を用いて参照することができる．変数名は宣言された要素/変数名を用いる．たとえば，

```
<!ELEMENT author (firstname?, lastname,
                  title* AS {//title[../author=$author]})>
```

という変換規則では，ノード変数 $author が利用されており，これによって現在の変換規則のコンテキストノードを参照することができる☆,☆☆．

3.3 変換基本操作

XTL では node-set に対する基本演算として，ソート（ORDER BY），重複排除（DISTINCT BY），他の集合操作を有する．

3.3.1 ソート（node-set, XPath+, order）→ node-set

ソート操作は 3 つの引数がありそれぞれ，1) ソート対象となる XPath 式の結果 node-set，2) ソートする際のキー値列を指定するための 1 以上の XPath 式，3) ソートの方向（昇順/降順），である．そしてソートされた node-set が返却される．XTL における表現方法は，mapping_XPath ORDER BY key_XPath+ [ASC|DESC] である．たとえば，

```
<!ELEMENT bib (author* AS {book/author}
                   ORDER BY {firstname} ASC)>
```

という変換規則は，XPath 式 book/author により返却される著者リストをその名前で昇順にソートすることを指定する．

3.3.2 重複排除（node-set, XPath+）→ node-set

重複排除操作は 2 つの引数があり，1) 重複排除対象となる XPath 式の結果 node-set，2) 重複排除する際のキー値列を指定するための 1 以上の XPath 式，である．そして重複排除された node-set が返却される．XTL における表現方法は，mapping_XPath DISTINCT BY key_XPath+である．たとえば，

```
<!ELEMENT bib (author* AS {book/author}
                   DISTINCT BY {e-mail})>
```

という変換規則は，book/author により返却される著者リストをその e-mail アドレスで重複排除することを指定する．

3.3.3 集合操作（node-set, node-set）→ node-set

和・差・積の集合操作が node-set に対して指定可

☆ 変換規則が再帰的に適用される場合，ノード変数は最も最近に呼び出された変換規則のコンテキストノードを参照する．

☆☆ 内容モデルや属性リスト型宣言において XPath 式が省略された場合，デフォルトの振舞いとして要素名/属性名を XPath 式として補完する．

表 1 XSLT 生成パターン
Table 1 XSLT generation patterns.

XTL	*XSLT*
変換規則	xsl:template
XPath	xsl:template の select 属性
変換シンボル	xsl:template の mode 属性
出力モード	xsl:element, xsl:copy
下位変換規則呼び出し	xsl:apply-templates, mode 属性, select 属性
sequence 指定	xsl:apply-templates の並び
choice 指定	xsl:choose, xsl:when
ANY, DISTINCT BY	専用の templates
EMPTY, 制約	xsl:if
ORDER BY	xsl:sort

能である．XTL における表現方法はそれぞれ+，-，* である．たとえば，

```
<!ELEMENT bib (author* AS {book/author}*{article/author})>
```

という変換規則は，積集合演算することで book と article の両方の著者になっている著者リストを作成することを指定する．

4. XSLT 翻訳方法

本論文では XTL 言語により記述されたプログラムを直接実行する方法の代わりに，XTL を XSLT に翻訳して，通常の XSLT プロセッサにより変換を実行する方法について説明する．

4.1 XSLT 翻訳の概要

XTL から XSLT に翻訳する基本的な考え方は，個々の変換規則を XSLT の template に対応させ，その際に変換シンボルを template の mode 属性として指定する，という方法である．XSLT 翻訳方法の概要を**表 1** にまとめた．この詳細に関しては，文献 11) を参照されたい．

4.2 XSLT の最適化

XTL から XSLT に翻訳する際に，最適化された XSLT を出力するいくつかの工夫を施している．4.1 節で述べたように変換シンボルを template の mode に指定することで，呼び出されるべきテンプレートの探索は高速に処理される．また XTL の変換結果構造において指定要素が 1 件であることが DTD によって明示されている場合は，XPath 式に [1] を付与することで不要な探索処理をスキップしている[7]．その他重要な最適化オプションとして，複数の XML の結合処理する際にハッシュ結合するよう xsl:key を用いた XSLT プログラムを出力することができる．

xsl:key による複数の XML の結合最適化

複数の XML 文書を結合する処理は XSLT の処理の中で最も高価な処理の 1 つである．XSLT プロセッサ（xalan，saxon，xt など）は，複数の XML の結合処理を単純な 2 つのループを利用した総当たりによって実装するため，大規模な XML の処理にはスケーラビリティが良くないという問題がある．しかしハッシュ結合を利用することには性能のトレードオフがあって，一概に xsl:key を使った方が良い性能が得られるとは限らない．場合によっては，xsl:key の指定によりハッシュ表を構築するコストが，ハッシュ表を使わない検索コストよりも大きくなってしまうことがあるからである．

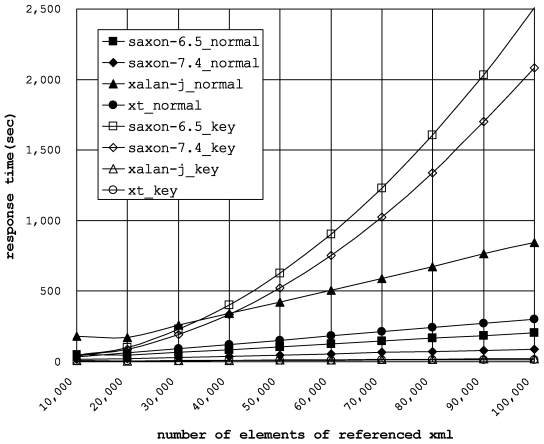


図 3 XML 結合操作の性能
Fig. 3 The performance of XML join.

図 3 に xalan 2.2D11，saxon 6.5/7.4，xt 20020426 の 4 種類の XSLT プロセッサを対象に，xsl:key を使用する/しない場合の性能の実験結果を示す．横軸は，ハッシュ表のエントリ数を表し，縦軸は XSLT の処理時間を示している．またハッシュ表による検索回数は 100 回である．この結果で興味深いことは，

- saxon は xsl:key を使う場合ハッシュ表のサイズが小さい場合は高速だが，2 万件を超えるような大規模なエントリ数になると逆に性能が劣化する．
- xalan は xsl:key を使う方が 20 倍〜40 倍高速である．
- xt は saxon と xalan より高性能であり，かつ xsl:key を使う方が 6 倍〜20 倍高速である．

この実験はハッシュ表の構築コストがハッシュ表を使わない検索コストよりも大きいというケースではないが，xsl:key の使う/使わないの選択ができることの重要性を示している．

しかし xsl:key を利用しないで結合処理を行う XSLT プログラムを，xsl:key を利用して結合処理を行う XSLT プログラムに変更するには実装の手間が生じ

るし，また xsl:key を利用するよう XSLT プログラムを自動的に書き換えるのは困難である．XTL から XSLT に翻訳する方法では，変換の記述が簡略化されているという XTL の特性を利用することで，自動的に xsl:key を利用する XSLT を出力することを実現している．具体的には，1) 複数文書の結合を指定する XPath 式を見つける．2) 1) で見つけた XPath 式から，結合対象を指定する XPath 式と結合キーを指定する XPath 式を生成し，xsl:key 命令を構築する．3) 結合演算を行うため，結合対象の XML データのドキュメントルートを xsl:for-each により読み込みハッシュ表を構築し，key 関数を用いてハッシュ表を検索して結合処理を行うよう XSLT プログラムを生成する．

たとえば，

```
<!ELEMENT Bib AS {BookCatalogue} (Book*)>
<!ELEMENT Book (
   Title, Author+,
   Cost* AS {document('c.xml')//Book[Title=$Book/Title]/
             Cost})>
...
```

というように，Title をキーとして c.xml 内の Book を結合し，その Cost を取り出すような XTL 変換プログラムからは，以下のような XSLT プログラムを生成する．

```
<xsl:key name="h" match="//Book" use="Title"/>
...
<xsl:template match="*|@*|text()" mode="Book">
 ...
 <xsl:for-each select="document('c.xml')">
  <xsl:apply-templates select="(key('h',$Book/Title)/Cost)"
                       mode="Cost">
  ...
```

5. 評　　価

5.1 XTL と $XSLT_0$ の表現能力

$XSLT_0$ [1] は XSLT のサブセットを形式化したものであり，XML-QL [4] に代表される一階述語に基づく XML 問合せ言語よりも（結合操作を除いて）高い表現能力を有する．$XSLT_0$ での template 定義は変数宣言部と変換構築部からなっており，それぞれについて XTL と比較する．

5.1.1 変数宣言部

$XSLT_0$ の template での変数宣言は，1) コンテキストノードの子テキストノードの値，もしくは 2) XSLT の template（群）を呼び出して得られる単一の値，を設定可能である．またこのように設定された $XSLT_0$ の変数は，呼び出される子孫の template にパラメータとして渡されるため，子孫の template から参照することが可能である．

まず現在の template（XTL では変換規則）において，変数を利用せずに XPath 式で同等の値が参照可能であることを示す．上記の 1) の場合，子テキストノードの値は明らかに XPath 式で参照可能である．2) の XSLT の template 呼び出しの場合，呼び出された template（郡）において値の変換操作がなくまた最終的に 1 つの結果値を出力することから，その値の取得は XPath 式で代用することが可能である．以上より，XTL において XPath 式の表現を工夫することで $XSLT_0$ の変数に相当する情報を，XTL の変換規則で参照することが可能となる．

次に XTL の子孫の変換規則において $XSLT_0$ の変数に相当する情報を参照可能であることを示す．XTL の変数は変換規則が適用されるコンテキストノードが設定され，子孫の変換規則の XPath 式より参照することが可能である．このことと先の現在の template に関する議論から，XTL の変数を用いて template が適用されたコンテキストノードを取得し，その template で定義される変数を参照する XPath 式を用いることで，XTL の子孫の変換規則において $XSLT_0$ の変数に相当する情報を参照可能となる．

5.1.2 結果構築部

$XSLT_0$ の template での結果構築は，変数もしくは子テキストノードに対する条件によって結果の出力条件を判定する．先の変数宣言部に関する議論より，$XSLT_0$ の変数は XTL の変数と XPath 式によって表現可能である．このことから XTL では XPath 式の述語を用いることによって，$XSLT_0$ の出力条件と等価な表現が可能である．また $XSLT_0$ では，*constructing* と *selecting* という 2 種類の template がある．この区別は XTL では出力モードの切替えにより実現できる．

しかし XTL の結果構築には制約があり，結果構造が文脈自由文法に従わなければならない．これは変換シンボルと変換結果の要素名が分離されていないためであり，同一の要素名は異なる文脈で利用することができない☆．

5.2 実　　験

5.2.1 excel データの XML 化

総務省実験のプロジェクトとして，複数の放送コンテンツのメタデータスキーマを統合した J/Meta [16] というスキーマを策定する活動がある．J/Meta のスキーマの仕様は excel で文書化されていたが，これを XML Schema 化する作業に対し，XTL を適用した．J/Meta のスキーマ仕様は 2 種類あって，カタログ系

☆ これを解決する方法としては，XTL を拡張して文脈を判断させるようにするか，もしくは変換シンボルと変換結果の要素名を分離する方法が考えられる．

表 2 XTL vs. XSLT（J/Meta の XML Schema 化）
Table 2 XTL vs. XSLT (generating the XML Schema for J/Meta).

	中間スキーマ生成	*XML Schema* 生成
XTL	24	20
自動翻訳による XSLT	286	340

（excel で 505 行）・権利系（1,756 行）という膨大な量であり，手作業での XML Schema 化はとうてい困難であった．この XML Schema 化の手順は以下のとおりである．1) excel から得られる CSV を単純な形式の XML（CSU-ML と呼ぶこととする）に変換し，2) RELAX などの他の XML スキーマ表現にも変換することを考慮し，CSV-ML を中間スキーマ表現に XTL で変換し，3) 中間スキーマ表現を XTL で XML Schema 化した．

2 つの XTL プログラムの開発は 1 人日（人数 × 日数），中間スキーマ生成 XTL の変換規則数（要素/変数型宣言と属性リスト形宣言の和）は 10 個，XML Schema 生成 XTL の変換規則数は 12 個であった．XTL プログラムと自動生成された XSLT プログラムの行数（空白とコメント行を除く）の比較を**表 2** にまとめた．

このケースでは XTL は自動翻訳された XSLT プログラムと比較して約 1/14 のプログラム量で済んでいる．人手による XSLT プログラムは作成していないが，自動翻訳された XSLT プログラムの内容から判断すると，人手による XSLT のプログラム量が自動翻訳された XSLT の半分以下になることはないと考えられる．

しかし，開発に要した稼動量はプログラム量の小ささと比較してかなり多い．この原因は，XPath 式の記述とそのデバッグに時間を要したためである．CSV-ML 形式の XML は表層的な 2〜3 段の深さしかないが，それが意味する放送コンテンツのメタデータは複雑な構造であり，XPath 式を用いて excel で視覚的にインデントされた部分構造を抽出するには XPath の following-sibling，preceding-sibling 軸を駆使する必要があった．このような処理は XPath が不得意であることが知られていて[10]，その結果 XPath の記述が複雑になりデバッグも時間を要することになってしまった．これは XTL の問題ではなく XPath の問題であり，XSLT を用いて開発した場合も，同様の問題が生じると考えられる．

5.2.2 放送メタデータの MPEG7 への変換

次の実験は，XTL を使って放送コンテンツのメタデータを別のメタデータスキーマ MPEG7 に変換する例である．この実験では XML 変換に詳しいプログラマ 2 名を被験者として，1 名が XTL プログラムを開発し，もう 1 名が XSLT プログラムを開発した．XTL プログラム開発は，1) サンプルの出力 XML を元に XMLSpy を用いて DTD を自動生成し，2) 次にその DTD に対し XPath 式の追加や DTD の修正をすることで XTL プログラムを作成し，3) XTL より XSLT を自動生成した後，JavaScript や namespace の指定などの XSLT への必要な修正を行い，それら必要な変更を RCS で版管理することで，XTL の変更が生じても自動的に変更がマージされるようなアプローチを採った．

表 3 XTL vs. XSLT（放送メタデータ間の変換）
Table 3 XTL vs. XSLT (transformation of broadcast metadata).

	ライン数	変換規則/*template* 数
DTD	138	102
XTL	218	114
自動翻訳による XSLT	2,433	85
人手による XSLT	897	28

XTL プログラムの開発は 1 人日であり，XSLT の開発は 5 人日であった．このことからこのケースでは XTL の XML 変換プログラムの生産性は XSLT の 5 倍であったといえる．また，XMLSpy により自動生成された DTD のサイズ，XTL プログラムと自動生成された XSLT プログラムのライン数（空白，コメント行，JavaScript 部分を除く）などの比較を**表 3** にまとめた．なお，XTL における変換規則数は要素/変数型宣言と属性リスト形宣言の総和であり，XSLT における template 数は xsl:template の宣言数である．

この実験結果で重要な点は，DTD と XTL のサイズの差が少ないことと，人手による XSLT のライン数が自動翻訳による XSLT と比較してかなり小さいことである．前者は変換結果の DTD があれば XTL の開発が容易であることを意味し，後者からは変換が複雑なケースでは人手による XSLT はコンパクトになると考えられる．

5.2.3 W3C 問合せのユースケース

XML 変換という領域とは若干異なるが，W3C の query WG で XML の問合せのユースケースが整理されている[2]．XTL をこのユースケースに適用することで，XTL の表現能力について評価した．ここでは namespace の処理のための NS ユースケースとユーザ定義関数の処理のための FNPARM ユースケースは扱わなかった．その理由は，DTD が namespace に対応していないことと，ユーザ定義関数が必要となる

ような複雑な変換は XSLT では他のスクリプト言語に任せているためである．

実験の結果，69 の問合せのうち 2 つを除いた 67 が XTL で表現できることが確認できた．XTL で表現できなかった 2 つの問合せ（TEXT ユースケースの Q3, Q6）は，部分文字列を抽出するための再帰的なユーザ定義関数を必要とするためであった．

6. 関連研究

文献 11) は，以前の版の XTL 言語に関するチュートリアルと，XTL から XSLT への翻訳方法の詳細について述べている．本論文の 3 章で述べた XTL の言語仕様の部分は文献 11) の内容と共通点が多いが，本論文では文献 11) の内容を拡張して，1) XTL の変換モデルを形式化し，2) *T-graph* を導入して変換能力を拡張し，3) xsl:key による複数の XML の結合最適化を導入し，4) 詳細な評価実験，を行っている．

XSLT 生成という観点では，SynTree[14] や GUI ツール XSlerator[6] などがある．SynTree は DTD を形式化した森正規文法に従う言語間での変換モデルであり，文献 14) では SynTree から XSLT を生成する方法も提案している．しかし，変換の表現能力が限られていて比較的簡単な変換しか表現することができないという問題がある．XSlerator は XSLT を作成するための簡易 GUI ツールであり，XSLT を作成する作業が若干簡略化されているにすぎないため，生産性はあまり向上しないと考えられる．

XML の変換という観点では，XQuery[8] や XDuce[5] などがある．変換能力という観点でこれらと XTL とを比較することは重要かもしれないが，XML 変換という分野で，コミュニティも大きく処理系が普及している XSLT 以外の言語が普及するとは難しいと考えられる．XTL は XSLT に翻訳可能であり XSLT の処理系を利用できるという点において，これらの言語より優れるといえる．

7. おわりに

本論文では，変換結果スキーマ指向の XML 変換モデルとその言語 XTL，および XTL を XSLT に翻訳する方法について提案した．そして XTL を XSLT のサブセット $XSLT_0$ と比較して変換能力がほぼ同等であることを示した．また，XTL を実用的な問題，excel データの XML 化・放送メタデータの MPEG7 への変換に適用し，このケースでは XML 変換の生産性が XSLT の 5 倍であったことが分かった．さらに，XTL を W3C の問合せのユースケースに適用することで，その 97.10%が XTL で表現可能であることが分かり，実用上表現能力が高いことも確認した．

実験などを通して分かった XML 変換における今後の課題は以下のとおりである．

(1) XTL は XSLT の記述の簡略化を実現したが，excel データの XML 化の例で顕著であったように XPath 式の記述の支援技術が必要であると考えられる（関連研究，文献 17)）．

(2) 放送コンテンツのメタデータの変換に限らず，XML 間の変換を考える場合，双方向の変換が必要なケースが多いと考えられる．現状ではそれぞれの方向に関して XSLT を記述する必要があるが，一元的に対応関係を記述することで，双方向の変換を実現する XTL/XSLT プログラムを生成する技術が重要であると考えられる．

(3) 本論文の XSLT プロセッサの性能評価実験では大規模な XML を対象に実験を行ったが，ファイルサイズが数 MB 程度でエントリ数が 3,000 程度の XML でもその XSLT 変換はかなり重く，利用者が web でページをアクセスする時点で XSLT を用いて HTML を生成することは現実的ではない．キャッシュされた HTML を適切に管理する技術が必要であると考えられる．

謝辞 XSLT プロセッサの性能計測に協力していただいた NTT ソフトウェア（株）の中野健司氏に感謝いたします．

参考文献

1) Bex, G.J., Maneth, S. and Neven, F.: A formal model for an expressive fragment of XSLT, *Information Systems*, Vol.27, No.1, pp.21–39 (2002).
2) Chamberlin, D., Fankhauser, P., Florescu, D., Marchiori, M. and Robie, J.: XML Query Use Cases. http://www.w3.org/TR/xmlquery-use-cases/
3) Clark, J.: XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt
4) Deutsch, A., Fernandez, M.F., Florescu, D., Levy, A.Y. and Suciu, D.: XML-QL: A Query Language for XML, *WWW The Query Language Workshop* (*QL*) (1998).
5) Hosoya, H. and Pierce, B.C.: XDuce: A Typed XML Processing Language, *WebDB* (2000).
6) IBM alphaworks: X-IT. http://www.alphaworks.ibm.com/tech/xit
7) Kay, M.H.: Saxon: XSLT プロセッサーの解体新書. http://www-6.ibm.com/jp/developerworks/

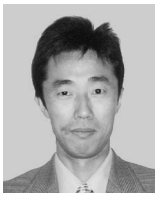
xml/010914/j_x-xslt2.html
8) Marchiori, M.: XML Query. http://www.w3.org/XML/Query/
9) MPEG-7 Japan 情報処理学会情報規格調査会 SC29/WG11/MPEG-7 小委員会（編）：MPEG-7. http://www.itscj.ipsj.or.jp/mpeg7/
10) Murata, M.: Extended Path Expressions for XML, *ACM PODS* (2001).
11) Onizuka, M.: XTL: An XML transformation Language, *Markup Languages*, Vol.3, No.3, pp.251–284 (2001).
12) EBU: EBU metadata specifications. http://www.ebu.ch/en/technical/metadata/
13) Pawson, D.: XSL Frequently Asked Questions. http://www.dpawson.co.uk/xsl/xslfaq.html
14) Tang, X. and Tompa, F.W.: Specifying Transformations for Structured Documents, *WebDB* (2001).
15) TV-Anytime Forum: Welcome to the TV-Anytime Forum Website. http://www.tv-anytime.org/
16) 財団法人マルチメディア振興センター：「共通メタデータ体系 J/Meta」仕様の公開. http://www.fmmc.or.jp/fmmc-html/jmeta/detail.html
17) 森嶋厚行，松本 明，北川博之：例示操作に基づく XQuery の問い合わせ，日本データベース学会 Letters，Vol.1, No.1, pp.15–18 (2002).

(平成 16 年 12 月 1 日受付)
(平成 17 年 9 月 2 日採録)

推 薦 文

新規性があり，かつ実用的にも有効と思われるので，推薦する．

（デジタル・ドキュメント研究会主査 大野邦夫）



鬼塚 真（正会員）

1968 年生．1991 年東京工業大学工学部情報工学科卒業．同年日本電信電話株式会社（株）入社．2000 年，2001 年ワシントン州立大学客員研究員．現在日本電信電話（株）サイバースペース研究所主任研究員．データベース管理システム，XML データ処理の研究に従事．2004 年情報処理学会山下記念賞，2005 年電子情報通信学会 DEWS2005 優秀論文賞．ACM，電子情報通信学会，日本データベース学会各会員．

小西 一也（正会員）

1971 年生．1995 年茨城大学工学部情報工学科卒業．1997 年同大学院理工学研究科情報工学専攻修士課程修了．同年，NTT データ通信（株）（現（株）NTT データ）入社．2000〜2003 年，日本電信電話（株）サイバースペース研究所にて XML メディエータシステムの研究に従事した後，現在，（株）NTT データ所属．