2003－DD－39 (7)
2003／5／23

変換結果スキーマ指向の XML 変換

鬼塚 真[†] 小西一也[††]

本稿では 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。XTL は拡張した DTD で変換結果構造を指定し、入力 node を XPath 式で出力 node にマップする。XTL の特徴は、変換規則群の呼び出し構造と変換結果の構造を一元化し、且つこの構造を文脈自由言語に制限することによって、記述を簡略化している点にある。こうして XTL は XML 変換プログラムの生産性を XSLT の 5〜10 倍に向上している。一方記述能力に関して、結果構造の制限を除いて XTL は XSLT の形式的モデル $XSLT_0$ とほぼ同等の能力を有することを簡単に述べる。そして W3C の query use cases に XTL を適用し、69 の問い合わせの 97.10% が表現可能であることを確認したことを報告する。

Output Schema Driven XML Transformation

MAKOTO ONIZUKA[†] and KAZUYA KONISHI[††]

This paper describes an XML transformation model and language called XTL (XML transformation language). XTL is declarative and is based on the output schema driven approach, which gives it the following capabilities: 1) the XML schema language part (we are currently using DTD) specifies the output structure of transformations, 2) The XPath expression part maps the resulting node-set in the input XML documents to the output structure. Users need to understand only DTD and XPath specifications, along with a few XTL extensions and semantics. We applied XTL to the W3C query use cases, and the results showed that XTL can express 97.10% of the 69 queries. This paper also describes an algorithm to translate an XTL program into an XSLT program with hash join optimization rewrite option.

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 サブセットの標準化などがなされている。例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[16], MPEG7[9], P/meta[13] 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 サブセットが普及しており、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 サブセットの間での 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[3] 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、時にして独特の実装テクニック (XSL FAQ[14]) 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>
...
```

この例では 2 つの XSL のテンプレートを利用し、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することをしている。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 $author$ 要素を処理するという処理になっている。

† 日本電信電話株式会社 NTT サイバースペース研究所
†† 日本電信電話株式会社 NTT サイバースペース研究所
現在、株式会社 NTT データ 技術開発本部に所属

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 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 の処理に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* の使い方は分かりにくいという問題がある。

上述の最初の例に代表されるような XSLT の複雑さを解決するため、本稿では新たな 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 は、1) 変換結果が特定のスキーマ構造を満たすという制約を課すこと、2) データ操作の基本操作 (重複排除, 部分木抽出, 変数の自動割当など) を導入すること、により XSLT よりも簡略な XML 変換の記述を可能にしている。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、及び XTL の表現のベースである DTD は変換結果のサンプル XML から自動生成可能であることから、XTL は XML 変換プログラムの生産性を XSLT を約 10 倍向上することができると言えるだろう。

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ね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 $XSLT_0$'s[1] とほぼ等価な能力を有することを 5 章で示す。また、W3C の問い合わせのユースケースに XTL を適用し、69 の問い合わせのうち 97.10% が XTL により記述可能であることを確認する。

上述した 2 つ目の問題 (複数 XML の結合) に代表される XSL の最適化問題を解決するため、XTL から XSLT に翻訳する方法では、最適化された XSLT を出力するような工夫をいくつか施している。特に複数 XML の結合の処理については、翻訳オプションを用意することで自動的に xsl:key を用いたハッシュ結合する XSLT プログラムを出力する。こうして、XSLT プログラマは xsl:key の利用方法を知らなくても、ハッシュ結合によるスケーラブルな XML 結合プログラムを開発することができる。

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 式を選択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)。指定された XPath の結果の node-set に含まれる個々の node に対して、変換規則が適用される。ルート候補でない変換規則は、他の変換規則の下位の変換規則として陽に呼び出される。直感的には、変換シンボルは生成規則における右辺に相当する。
- M は出力モード指定であり、入力 XML の node 毎に適用された変換規則に対し、どのように結果 node 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/値割り当て, 入力 node と同一の名称/値の割り当て) ある。
- Σ は展開規則 (生成規則における左辺) であり、以下の 4 つの指定をする。1) 呼び出す下位の変換規則を表現する変換シンボルへの参照 2) 下位の変換規則の呼び出し順序 (これによって入力 XML における兄弟ノードの順序構造を変更するか保存するかを指定する) 3) 下位の変換規則を適用するための条件としての 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
- V はノード変数であり、変換規則の呼び出し階層における祖先変換規則が適用された入力ノードが設定される。

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 プログラムのような入力 XML の解析構文を表すと同時に変換結果の構造 (DTD 等) を表していることである。この結果、XTL のプログラムは XSLT プログラムよりも簡潔に XML 変換を表現可能である。入力 XML の解析構文を *T-graph* (transformation graph) と呼び、変換結果構造を *S-graph* (output schema graph) と呼ぶこととする。

図 1 は図 2 で示される XTL プログラムの変換例を示している。変換は 4 つのデータ構造からなる。入力 XML の DOM 木, T-graph, S-graph, そして変換結果の DOM 木である。最初に入力 XML がパースされて DOM 木が構成される。次に XTL プログラムがパースされて T-graph と S-graph が構成される。T-graph では、個々の node は変換シンボルから生成され、node 間の edge は変換規則の呼び出し関係から生成される。XPath 式は入力 XML の node を T-graph の node(変換規則) へ入力する。S-graph では、個々の node は T-graph の node であって出力モードが出力指定されたものから生成され、node 間の edge は T-graph での edge が継承さ

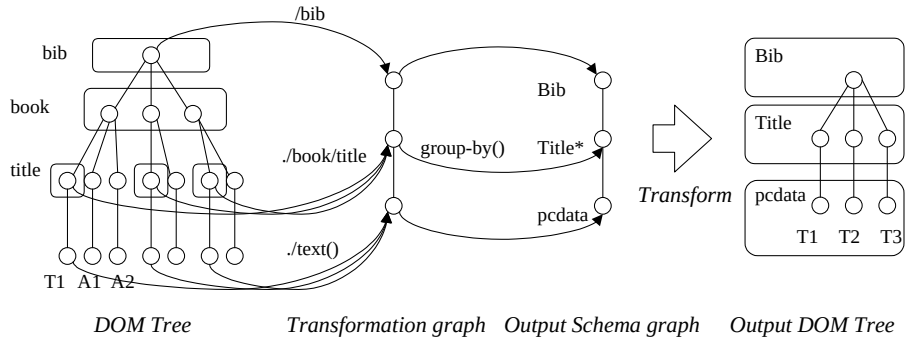

図 1 A Transformation Example

れる。T-graph から S-graph へは変換命令 (ソート, 重複排除等の node-set の集合演算) が指定される。最後に入力 XML に対して、T-graph と S-graph で表現される変換命令を適用した結果が、変換結果 DOM 木として生成される。

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

図 2 An XTL Program Example

3. XTL 仕 様

XTL プログラムは下記の 3 つの部分から構成される。1) T-graph, S-graph を表現するための拡張 DTD, 2) 入力 XML の node-set を T-graph の node に入力させる (マッピング) ための XPath 式, 3) T-graph から S-graph へ指定される変換命令 (ソート, 重複排除等の node-set の演算)。直感的には、Δ の変換シンボルは DTD において宣言された要素に相当し、Σ の展開規則は DTD の内容モデルに相当する。属性リスト型宣言については要素型宣言の考え方に順ずるため、本章では説明を省略する。

3.1 DTD (変換結果構造)

XTL の拡張 DTD では、入力 XML の解析構文 (T-graph) と変換結果構造 (S-graph) を表現する。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/値割り当て, 入力 node と同一の名称/値の割り当て) ある。最初に中間ノードの要素に関して 3 種類の出力モードの指定方法を説明した後、葉ノードの値に関して固定値の割り当てと入力 node により演算された値の割り当て方法を説明する。

適用された変換規則において、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る。適用された変換規則において、要素名を出力したくない場合は 変数型宣言 (XTL で DTD を拡張した) によって変換規則を記述する。また、入力 node 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。例えば、

```
<!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 node 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 node 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 node と同一名称の要素名称を出力する。

次に値を有する葉ノードに関しては、XPath で用いられる関数 (count, max, concat 等) を用いて入力 XML の値に対して演算を施した結果を出力するか、固定値を割り当てることが可能である。例えば、

```
<!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 $count(author)$ の結果が割り当てられ、2 番目では要素 $version$ の値は ”Jan/3/2003” という固定値が割り当てられる。

要 素 内 容

要素内容には *sequence* と *choice* という 2 種類がある。*sequence* 指定では、本変換規則から呼び出される下位の変換規則が要素内容で指定された順で呼び出されるのに対し、*choice* 指定では、下位の変換規則は入力 XML の node の文書順にマッチする XPath が指定された変換規則が呼び出される。こうして *sequence* 指

定では入力構造を変更する変換を実現し、*choice* 指定では入力構造を保存する変換を実現している。例えば、

```
<!ELEMENT R AS {r} (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 a なる XPath 式にマッチした node 毎に A なる変換規則を呼び出し、次に b なる XPath 式にマッチした node 毎に B なる変換規則を呼び出す (C も同様)。もし入力 XML が `<r><c/><a/><b/><a/></r>` ならば、出力結果は `<R><A/><A/><B/><C/></R>`となる。

```
<ELEMENT R AS {r} (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 a, b, c なる XPath 式にマッチする node 毎にそれぞれ A, B, C なる変換規則を呼び出す。もし入力 XML が`<r><c/><a/><b/><a/></r>` 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 に対する制約であることを意味する。例えば、

```
<!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 node が 1 以上あり、且つ $title$ なる XPath 式にマッチする node 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 は名無しの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 式によってマップされた結果 node-set が空集合であるという制約を意味する。

ANY

ANY は XPath 式によってマップされた node 配下の入力 XML のサブツリーをコピーする。例えば、

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

という変換規則は、*book* なる XPath 式にマッチする node 毎に 2 行目の変換規則が適用され、その node 配下のサブツリーをコピーして出力する。

但し、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 式がサブツリーの node にマッチする場合は、入力 XML 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る。

3.2 XPath (マッピング)

XPath 式は入力 XML の node-set を T-graph の node (変換シンボルによって特定される変換規則に相当) に入力する。具体的には、XPath 式により返却される node-set の個々の node に対して、その node をカレント node として変換規則が呼び出される。個々の変換規則では、下位の変換規則を呼び出す条件をカレント node を基点とした相対 XPath 式で記述する。XPath 式は変換シンボルに指定されるか (ルート候補変換)、展開規則における下位の変換規則を呼び出す条件として指定される場合の 2 通りがある。ルート候補変換は次のいずれかのパターンによって呼び出される。1) 指定された XPath が入力 XML のルート要素にマッチする場合、2) 内容モデルが ANY である変換規則によって処理される入力 XML の node であって、その node 配下の node が指定された XPath がマッチする場合。例えば、

```
<!ELEMENT Bib AS {bib} (Book* AS {book})>
<!ELEMENT book         (author*, title, publisher)>
```

という変換における Bib 要素の変換規則はルート候補変換の例である。

また XPath 式の記述では、祖先の変換規則 (T-graph の祖先 node) が適用されたカレント node をノード変数 V を用いて参照することができる。例えば、

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

という変換規則では、ノード変数 $\$author$ が利用されており変換結果の *author* 要素にマッピングされた入力 node を参照する。このように要素/変数型宣言によって宣言された要素/変数にマッピングされた入力 node は、ノード変数に自動割り当てされる。☆☆ ☆☆☆

3.3 変換基本操作

XTL では変換基本操作として、ソート (ORDER BY) と重複排除 (DISTINCT BY) 操作、及び XPath 式の結果の node-set に対する集合操作を有する。

☆ 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、$title$ を $title[1]$ として扱う。

☆☆ 同一の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カレント node 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、デフォルトの振る舞いとして要素名/属性名を XPath 式として補完する。

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。そしてソートした結果を node-set として返却する。XTL の言語における表現方法は、mapping_XPath ORDER BY key_XPath+ [ASC|DESC] である。例えば、

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

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を指定するための 1 以上の XPath 式、である。そして重複排除した結果を node-set として返却する。XTL の言語における表現方法は、mapping_XPath DISTINCT BY key_XPath+ である。例えば、

```
<!ELEMENT bib (author* AS {book/author} GROUP 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 対して指定可能である。XTL の言語における表現方法はそれぞれ +,-,* である。例えば、

```
<!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

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[12] を参照されたい。

4.2 XSLT の最適化

XTL から XSLT を翻訳する際に、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 章で述べたように、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することができる[7]。その他、重要な最適化オプションとしてハッシュ結合処理するように xsl:key を用いた XSLT プログラムを出力することができる。

表 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

xsl:key による複数 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ひとつである。XSLT プロセッサ (xalan, saxon, xt など) は、複数 XML の結合処理を単純な 2 つのループを利用した総当りによって処理するため、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 xsl:key 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 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。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、且つ 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ケースに当たらないが、ハッシュ表に登録したけれどもそのデータをほと

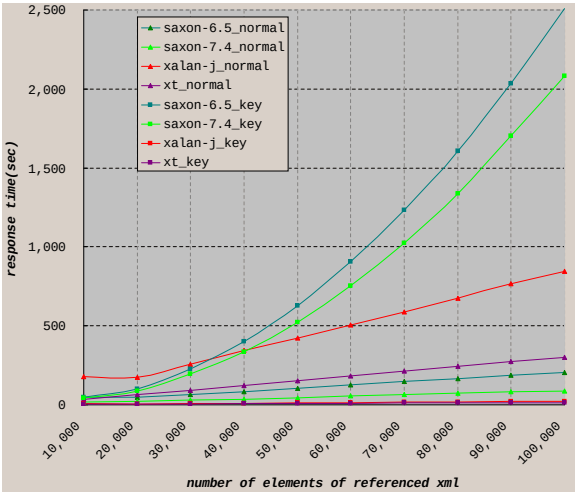

図 3 The Performance of XML join

んど参照しないようなケースではハッシュ表を構築しない方が高速になるため、xsl:key の使う/使わないの選択が自由にできることは重要である。

通常の XSLT プログラムを xsl:key を利用する XSLT プログラムに変更するには実装の手間が生じるし、xsl:key を利用するよう XSLT プログラムをに自動的に書き換えるのは困難である。XTL から XSLT を翻訳する方法では、出力される XSLT は形式が整形されているため、自動的に xsl:key を利用する XSLT を出力するオプションが可能となった。

5. 議　　論

5.1 XTL と $XSLT_0$ の比較

$XSLT_0$[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 問い合わせ言語よりも (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

$XSLT_0$ の template での変数宣言は 1) カレント node の子テキスト node の値、もしくは 2) 値を返却するような XSLT の template を呼び出した結果値、を設定可能である。

証　明　1

それぞれのケースについて、まず現在の template (XTL では変換規則) から、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。1) の場合は、明らかに XPath 式で変数を代用することができる。2) の XSLT の template 呼び出しでは、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である。

証　明　2

次に $XSLT_0$ 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、XTL の下位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カレント node を設定し、下位の変換規則の XPath 式より参照することが可能である。このことと証明 1 の結果を組み合わせることで、XTL の変換規則は祖先の変換規則で定義される $XSLT_0$ の変数に相当する全ての変数を参照することが可能となる。

5.1.2 結 果 構 築

$XSLT_0$ の template での結果構築は、変数もしくは子テキスト node に対する条件によって結果の出力条件を判定する。証明 1,2 の結果より $XSLT_0$ の変数は XTL の変数と XPath によって表現可能である。このことから $XSLT_0$ の出力条件は XTL では XPath 式の述語により表現することが可能となる。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り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ねばならない。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 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 (excel で 505 行)・権利系 (1756 行) という膨大な量であり、到底手作業での XML Schema 化は困難であった。この XML Schema 化では、1)excel から得られる CSV を単純な形式の XML である CSV-ML (comma separated

☆ これを解決する方法しては、XPath 式を用いて文脈を判断させるようにするか、もしくは XTL を拡張して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。

markup language[11]) に変換、2)RELAX 等の他の 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、3) 中間スキーマ表現を XTL で XML Schema 化するアプローチを採った。

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。

表 2 XTL vs XSLT(J/meta の XML Schema 化)

	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

このケースでは XTL は自動翻訳された XSLT と比較して約 1/14 のプログラム量で済んでいる。このケースでは人手による XSLT を作成していないが、自動翻訳された XSLT の内容から判断するに、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 は構造が 2〜3 段の深さしかないが、そこに記述されている放送コンテンツのメタデータスキーマ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 が不得意である処理であり[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。

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 倍であるといえる。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、java script 部分を除く) 等の比較を表 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。

表 3 XTL vs XSLT(放送メタデータ間の変換)

	ライン数	変換規則/*template* 数
DTD	138	102
XTL	218	114
自動翻訳による XSLT	2433	85
人手による XSLT	897	28

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。前者からは変換結果の 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 問い合わせのユースケース

XML 変換という領域とは若干異なるが、W3C の query WG で XML の問い合わ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、namespace は DTD と両立しないことと、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 の問い合わせのうち 2 つを除いた 97.10% 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 つの問い合わせ (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

XSLT 生成という観点では、SynTree[15] 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[15] 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過ぎないため、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] 等がある。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 XSLT が普及している現状で XSLT とは異なる言語の処理系が普及するとは考えにくい。XTL は XSLT に翻訳可能という点でこれらの言語より優れるといえる。

7. お わ り に

本稿では、変換結果スキーマ指向の XML 変換モデルとその言語 XTL、及び 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、XML 変換の生産性が XSLT の 5 倍になるケースがあることが分かった。一方、XTL を W3C の問い合わ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。

実験等を通して分かった XML 変換における今後の課題は以下の通りである。

(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[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) 本稿の 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 3000 程度の XML でもその XSLT 変換はかなり重たく、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 適切に管理する技術が必要であると考えられる。

参 考 文 献

1) G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002.
2) D. Chamberlin, P. Fankhauser, D. Florescu, M. Marchiori, and J. Robie. XML query use cases. http://www.w3.org/TR/xmlquery-use-cases/.
3) J. Clark. XSL transformations (XSLT) version 1.0. http://www.w3.org/TR/xslt.
4) A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. "XML-QL: A Query Language for XML". In *WWW The Query Language Workshop (QL)*, 1998.
5) H. Hosoya and B. C. Pierce. "XDuce: A Typed XML Processing Language". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, 2000.
6) IBM alphaworks. X-IT. http://www.alphaworks.ibm.com/tech/xit.
7) M.H. Kay. 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) M.Marchiori. 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) M. Murata. Extended path expressions for XML. In *ACM PODS*, 2001.
11) B. O. Nolan and J. Divney. Comma separated markup language (CSV-ML) specification. http://www.amadan.net/spec/csvml.html.
12) M. Onizuka. XTL: An XML transformation language. *Markup Languages*, 3(3):251–284, 2001.
13) C. Owens. PMC project P/META (metadata exchange standards). http://www.ebu.ch/departments/technical/pmc/pmc_meta.html.
14) D. Pawson. XSL frequently asked questions. http://www.dpawson.co.uk/xsl/xslfaq.html.
15) X. Tang and F. W. Tompa. Specifying transformations for structured documents. In *WebDB*, 2001.
16) TV-Anytime Forum. http://www.tv-anytime.org/.
17) 森嶋厚行, 松本明, and 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*, 1(1):15–18, 2002.