社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＤＤ－51（５）
2005／7／22

XMLストリーム処理: XPath評価からSVGレンダリングまで

鬼塚　真[†]　兵藤　正樹[†]　内山　寛之[†]　西岡　秀一[†]　山室　雅司[†]

†日本電信電話株式会社NTTサイバースペース研究所

あらまし　本稿では，インターネット上のアラートサービスなどに代表されるようなSDIシステムにおいて，大量の利用者のフィルタ条件を高速に処理し，フィルタ結果をHTML/SVGに変換・レンダリングするアーキテクチャについて述べる．その構成要素である一連のXMLストリーム処理技術の概要は以下の通りである．1) 大量の利用者が指定するフィルタ条件を高速に処理するXPathフィルタ処理技術，2) XPathフィルタ処理を用いて利用者毎に選別されたXMLデータをXSLTを用いてHTMLやSVG形式に変換し，その後新たな情報が新たに提供されてフィルタが実行された時に，得られるフィルタ結果を用いて先の変換結果を差分で更新する技術，3) 変換されたSVG形式のデータを一旦レンダリングし，その後新たな入力情報により差分で更新した変換結果を再レンダリングする技術．

キーワード　XML，XPath，ストリーム処理，XSLT，ビューメンテナンス，SVG

Processing XML Streams: From XPath Evaluation to SVG Rendering

Makoto ONIZUKA[†], Masaki HYODO[†], Hiroyuki UCHIYAMA[†], Shuichi NISHIOKA[†], and Masashi YAMAMURO[†]

† CyberSpace Laboratories, NTT Corporation

Abstract　We propose an architecture of SDI system for alert services on the Internet, which evaluates a large number of user profiles on incoming XML streams and then transforms the filtered result to HTML/SVG data for rendering. The key techniques for the SDI system are as follows. 1) Efficient XPath evaluation algorithm with DFA for a large number of user profiles written in XPath expressions, 2) Incremental maintenance algorithm for materialized XSLT/XPath views, which inputs the filtered incoming XML data and maintains the existing HTML/SVG data transformed by XSLT programs, and 3) Incremental SVG rendering algorithm, which inputs the updated part of SVG data, incrementally maintains the internal data model of SVG, and then update the rendered result.

Key words　XML, XPath, stream processing, XSLT, view maintenance, SVG

1. はじめに

yahooアラート，価格.comのお知らせサービス，livedoorによる未来検索，pubsub.com [1] によるアラートサービスに代表されるように，株価・商品価格・空港の発着情報等の特定コンテンツに対する変化をアラート通知したり，またgoogle alertによるwebコンテンツや速報系のニュースに対する変化を通知するサービスが急速に普及してきている．例えばyahooのオークションアラートは，出品者や出品物をキーと指定することで指定した出品者が新たな出品物を出した際や，また指定した出品物の入札価格が変化した際に，利用者に対してメール通知するようなサービスである．このようなシステムはSDI (selective dissemination of information) システム [2] と呼ばれ，情報利用者は自分が欲しい情報に対する検索条件をあらかじめSDIシステムへ登録しておき，情報提供者から新たな情報が提供された段階でSDIシステムは大量の利用者の検索条件を処理して，検索条件に合致する情報を選別して利用者に情報を配信する．

このようなSDIシステムを実現する方法は，プル型 (擬似プッシュ型) とプッシュ型とがある．プル型では，サーバ側にデータベース管理システムを利用して，利用者のクライアントプログラムから定期的にサーバにポーリングして，データベースを検索することで，利用者に対して選別された情報を提供する．プッシュ型では，サーバ側にストリームデータ管理システムを利用して，情報提供者から提供されるデータに対してフィルタ処理を行い，データに適合する検索条件を判定して，該当する利用者に対して選別された情報を提供する．文献 [3] で述べられているように，即時性を重視したサービスに対しては，データベース管理システムではなくストリームデータ管理システムを利用する方がスケーラビリティが圧倒的に優れている．この理由は，データベースでは各利用者が登録した条件を一件づつ

実行しなければならないのに対して，ストリームデータ管理システムによるフィルタ処理[4]〜[6]では，あらかじめ検索条件の共通部分を共有化しておくことで処理量を削減できるためである．このように即時性を重視したサービスに対しては，ストリームデータ管理システムを利用することが不可欠である．

またSDIシステムでは検索条件を処理する他に，検索・フィルタ結果からHTMLやSVGなどのレンダリングフォーマットに変換して表示するという機能を有する．プル型のSDIシステムでは，利用者のクライアントプログラムからのポーリングの際にデータベースの検索結果をHTMLやSVG形式に変換して，その結果をレンダリングプログラムで表示すれば良い．しかしストリームデータ管理システムを用いる場合は，一旦フィルタ結果をHTMLやSVG形式に変換してレンダリング表示した後に，新たな情報が提供されてフィルタが実行された場合，得られたフィルタ結果をこれまでの変換結果及びレンダリング表示に反映させる必要がある．従来の技術では，このような変換結果やレンダリング表示の差分更新ができないため，変換とレンダリングの再実行が必要となり，これらの性能コストが大きいことから，サービスの即時性を維持することが難しいという問題があった．

以上の背景から，本稿ではストリームデータ処理方式を利用した上で，新たな情報が提供された際に差分でHTMLやSVG形式へと変換する方法と，差分でSVGレンダリング結果を更新する方法を組み合わせたSDIシステムを提案する．利用している要素技術の概要は以下の通りである．

XPath フィルタ処理

XMLストリームに対する大量のXPath式を処理するため，各XPath式を問い合わせ木に変換し，更にXPath式を構成するシングルパスに相当する処理をDFA[5]を用いて複数のパス処理を共通化した，高速XMLストリーム処理技術[7]．

XSLT 差分変換

XPathフィルタ処理を用いて利用者毎に選別されたXMLデータを，XSLT/XPathを用いてHTMLやSVG形式に変換し，その後新たな情報が新たに提供されてフィルタが実行された時に，得られるフィルタ結果を用いてこれまでの変換結果を差分で更新する技術[8]．

差分レンダリング

変換されたHTMLやSVG形式のデータを一旦レンダリングし，その後新たな入力情報により差分で更新した変換結果を再レンダリングする技術．

本論文の構成を示す．2.章では，株価データを用いたアラートサービスの例題を説明し，XPathフィルタ処理・差分変換・差分レンダリングの一連の技術の必要性について明らかにする．3.章では，これらの技術を連携させたアーキテクチャについて説明する．4.，5.，6.章では，それぞれXPathフィルタ処理，XSLT差分変換，差分レンダリング技術の詳細について説明する．最後に7.章で本稿のまとめを述べる．

2. 例　　題

株価データを用いた例題として，次のような特徴を持つサービスを考えてみよう．

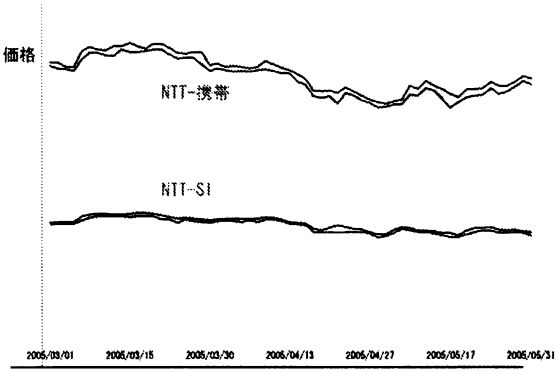

図1　SVGのレンダリング例(高値, 安値)
Fig. 1　An example of SVG rendering: highest and lowest prices

XPath フィルタ処理　上場されている株式の中から，個々の利用者毎に指定された銘柄の株価情報だけを選別して利用者へ提供する．

変換・レンダリング　選別されたXMLデータをSVG形式に変換してレンダリングすることで，株価の時系列的な変化を理解し，今後の株価の変動を予測する．

図1は，NTT-携帯とNTT-SIという二つの銘柄を選別し，その2005/3/1から2005/5/31までの高値と安値をSVGを用いてレンダリング表示したものである(注1)．縦軸が高値と安値の価格を示していて，横軸が時間の経緯を示している．株価のXMLデータからこのSVGデータを生成するXSLTプログラムを図2に示す．紙面の都合上，変数宣言や属性の一部を省略している．この基本的な処理は，選別された情報の中から銘柄ごとに(図2のB: データの図形の部分)，高値(C: 高値のグラフ図形)と安値(D: 安値のグラフ図形)の折れ線グラフを作成することである．折れ線を構成する個々の直線は，ある時点と次の時点の株価を接続している．

XPath処理からSVGレンダリングまでの一連の処理をプル型で実行する場合は，次のような動作となる．ある時点で(例えば2005/4/27の13:00)利用者が指定した銘柄の株価情報をデータベースから検索し，検索結果に対してtoSVG.xslを実行することでXMLデータをSVGデータに変換し，結果をレンダリングする．その後，定期的に(例えば30分おきに)，最新データが追加されたデータベースに対して利用者が指定した銘柄の株価情報をデータベースから検索し，得られた検索結果に対してtoSVG.xslを実行することでSVGデータを構築し直し，レンダリングする．この方法の問題点は，1) toSVG.xslによるXSLTの変換対象が常に検索結果全体であり，以前の変換結果を全く再利用していない．このため，入力XMLデータが大規模であったり，また複数の利用者によるXSLT変換を1台のwebサーバで実行するような場合は，XSLTによる変換が性能のボトル

(注1)：株価データはhttp://www.rain-net.com/kabu/data.htmのデータをXML化したしたものを利用している．なお銘柄の名称は仮想の名称に変更している．

```
<!-- 変数宣言 xskip, xbase, ybase, xtop, ytop -->
....
<xsl:template match="/stock">
 <svg width="$xtop + 100" height="$ytop + 100"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
<!-- A: ベースの図形 -->
  <text x="0" y="74px">価格</text>
  <line x1="10" y1="$ytop - $ybase"
        x2="$xtop" y2="$ytop - $ybase"/>
  <line x1="$xbase" y1="$ybase"
        x2="$xbase" y2="$ytop"/>
<!-- B: データの図形 -->
  <text x="200" y="120">NTT-携帯</text>
  <xsl:apply-templates select="ログ [銘柄名='NTT 携帯']"/>
  <text x="200" y="240">NTT-SI</text>
  <xsl:apply-templates select="ログ [銘柄名='NTT-SI']"/>
 </svg>
</xsl:template>

<xsl:template match="ログ">
 <xsl:if test="not(position()=last())">
<!-- C: 高値のグラフ図形 -->
  <xsl:element name="line">
   <xsl:attribute name="x1">
    <xsl:value-of select="$xbase + position() * $xskip"/>
   </xsl:attribute>
   <xsl:attribute name="y1">
    <xsl:value-of
     select="$ytop - 高値 div 1000"/>
   </xsl:attribute>
   <xsl:attribute name="x2">
    <xsl:value-of
     select="$xbase +(position()+1) * $xskip"/>
   </xsl:attribute>
   <xsl:attribute name="y2">
    <xsl:value-of
     select="$ytop - (following-sibling::ログ [1]/高値)
                     div 1000"/>
   </xsl:attribute>
  </xsl:element>
<!-- D: 安値のグラフ図形 -->
...
<!-- E: X 軸の値表示 -->
 </xsl:if>
 <xsl:if test="(position() mod 10 = 1)">
  <text x="$xbase + position()*$xskip - 30"
        y="$ytop - $ybase - 10" writng-mode="tb-rl">
   <xsl:value-of select="年月日"/>
  </text>
 </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

図 2　toSVG.xsl (紙面の都合上，一部属性を省略)

Fig. 2　toSVG.xsl (several attributes are omitted due to the space limitation)

ネックになってしまう．2) 変換後の SVG データのレンダリング処理についても常に検索結果全体であり，以前の SVG データに関わる処理を全く再利用していない．これについても上記と同様に性能に影響してしまう．

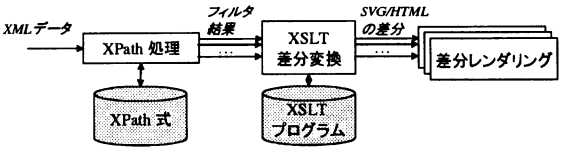

図 3　XML ストリーム処理アーキテクチャ

Fig. 3　An architecture of XML stream processing

一方，XPath 処理から SVG レンダリングまでの一連の処理をプッシュ型で実行する場合は，次のような動作となる．ある時点で (同様に 2005/4/27 の 13:00) 利用者が指定した銘柄の株価情報を XPath 式によるフィルタ処理を実行し，フィルタ結果に対して toSVG.xsl を実行することで XML データを SVG データに変換し，結果をレンダリングする．その後，定期的に追加された最新データだけ対してフィルタ処理を実行することで，利用者が指定した銘柄の株価情報を抽出し，最新データに関するフィルタ結果に対して toSVG.xsl を差分実行して SVG データに対する変更差分を作成し，その変更差分をレンダリングされている SVG の内部データに反映して再レンダリングする．この方法の利点は，利用者が指定した銘柄の株価情報の抽出 (XPath 処理)，抽出結果を toSVG.xsl を用いて SVG データに変換する処理 (XSLT 処理)，SVG データをレンダリングする処理 (SVG 処理) の全ての処理において，最新データに関わる差分だけを処理することで処理量を削減していることである．

3. アーキテクチャ

プッシュ型の SDI システムのアーキテクチャを図 3 に示す．最初に入力 XML データをフィルタ処理する **XPath 処理**部があり，個々の利用者毎に指定された XPath 式を評価してフィルタ結果を **XSLT 差分変換**部に渡す．**XPath 処理**部では大量の XPath 式をまとめて高速に処理する手法 [5], [7] を用いている．

XSLT 差分変換部では，フィルタ結果に対して個々の利用者毎に用意された XSLT プログラムを差分で処理を行い [8]，SVG/HTML の変換後の差分情報を生成し，**差分レンダリング**部に渡す．この図では **XPath 処理**部と **XSLT 差分変換**部は共にサーバ側にあり，全利用者のフィルタ条件と XSLT プログラムを格納しているが，**XSLT 差分変換**部をクライアント側に置くことも可能である．なお **XSLT 差分変換**部において，大量の XSLT プログラムをまとめて高速に処理する手法については今後の課題である．

差分レンダリング部では，SVG の変換後の差分情報を入力することで，SVG をメモリ上へ展開したデータ構造を差分で更新することで，レンダリング結果を更新する．

4. XPath 処理

大量の XPath 式を高速に処理する問題に対する我々のアプローチは，XPath 式を述語の部分が分岐になるように 1 つの問い合わせ木に変換して，分岐部分はボトムアップに評価を行う [7]．また問い合わせ木を構成する全ての (シングル) パスを

まとめて一つの DFA [5] とすることで，複数 XPath 式におけるパスの処理を共有化して処理コストを削減する．以降，4.1 章で DFA を用いたシングルパス群の処理について述べ，4.2 章で分岐部分を評価する処理について述べる．

4.1 結合問い合わせ木からの DFA 生成

結合問い合わせ木は大量の XPath 式から構成され，全ての XPath 式から 1 つの DFA を導出する．この処理は 2 つのステップからなり，1) 結合問い合わせ木から非決定性有限オートマトン (NFA) を導出した後，2) NFA から DFA を得る．ここでは各ステップについて簡単に述べるが，NFA から DFA の導出方法の詳細については文献 [9] などを参照されたい．結合問い合わせ木の例として，図 4(a) に示した P を用いる．図 4(b) は第一ステップの結果得られる NFA A_n を表している．このステップでは一般的に良く知られている XPath 式から NFA に変換する手法 (Tukwila [10] や YFilter [4]) を用いている．正規表現を NFA に変換するサーベイについては，文献 [11] を参照されたい．図 4(b) における $*$ でラベル付けされた推移は，結合問い合わせ木 P の $*$ もしくは // を表している．またこの NFA を構成する主な要素として，1 つの初期状態と，各変数 ($X, $Y, ...) 毎の一つの終了状態と，シングルパス XPath 式を連結するための ϵ 推移がある．ここで注目すべきことは，一般的に NFA A_n の状態数は P の長さに比例することである．

結合問い合わせ木 P に含まれるタグ，属性，文字列定数の集合を Σ とし，XPath 式の $*$ もしくは // にマッチするその他のシンボルを ω で表すこととする．また Σ^* に属する w に対して，w を入力したときに到達可能な A_n の状態群を $A_n(w)$ と表すこととする．図 4 の例では，$\Sigma = \{a, b, d, \omega\}$, $A_n(\epsilon) = \{1\}$, $A_n(ab) = \{3, 4, 7\}$, $A_n(a\omega) = \{3, 4\}$, $A_n(b) = \emptyset$ となる．

P から得られる DFA A_d は，以下の状態と推移から構成される．

$$states(A_d) = \{A_n(w) \mid w \in \Sigma^*\} \quad (1)$$

$$\delta(A_n(w), a) = A_n(wa), a \in \Sigma$$

図 4 の例では，DFA A_d は (c) のようになる DFA の各状態は，重複のない明示された推移集合と付加的な [other] 推移を持つ．[other] が示す推移は，Σ に属する任意のシンボルによる推移であり，且つその状態における明示された推移集合に含まれないものである．[other] は，任意のシンボルによる推移を示す $*$ とは異なる．例えば，状態 $\{3, 4, 8, 9\}$ での [other] は，明示された推移である b, d 以外，つまり a もしくは ω である．また，状態 $\{2, 3, 6\}$ での [other] は，a, d, もしくは ω である．DFA の 1 つの状態は複数の NFA の状態を表すため，DFA における終了状態は複数の変数によりラベル付けされることがある．例えば，状態 $\{3, 4, 5, 8, 9\}$ は $Y と $Z がラベル付けされている．DFA の状態において，sax_f フラグは次のように定義される．DFA の状態を構成する NFA の状態が 1 つでも sax_f = true と指定されている場合は，その DFA の状態の sax_f フラグは真 (true) であり，そうでない場合は偽 (false) である．

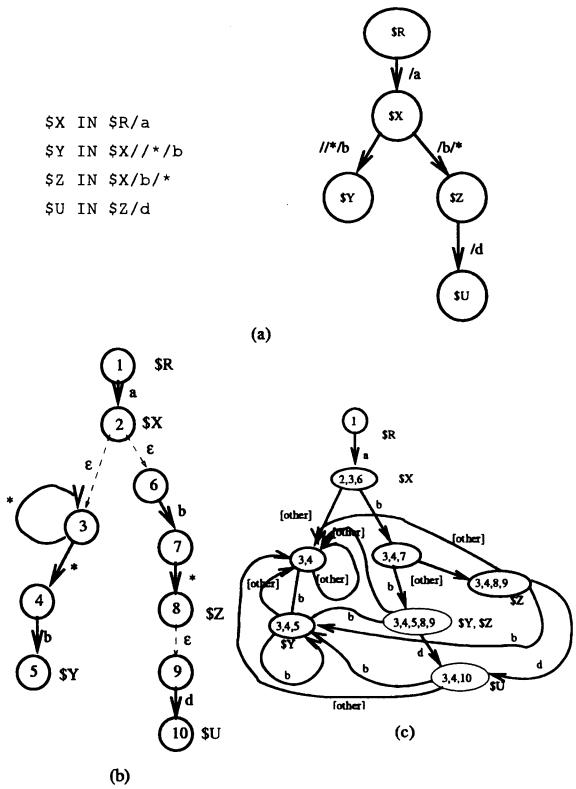

図 4 (a) 結合問い合わせ木 P, (b) P の NFA A_n, (c) P の DFA A_d

Fig. 4 (a) A query tree P, (b) its NFA, A_n, and (c) its DFA, A_d.

4.1.1 DFA の動作

DFA を用いることで XML ストリームは非常に高速に処理することができる．XML の木構造を処理するために，現在の DFA 状態を示すポインタと，DFA 状態のスタックを用いる．SAX イベントは以下のように処理される．startElement(e) イベントの際には，現在の DFA 状態をスタックにプッシュし，e 推移(注2)により到達できる状態を現在の DFA 状態として設定する．endtElement(e) イベントの際には，スタックから状態をポップしその状態を現在の DFA 状態として設定する．このようなスタック操作によって，現在の SAX イベントに対応する XML ノードの先祖ノード群があって，そのノード群を処理した状態群を常にスタックに保持することができる．現在の状態に状態ラベルが付与されている場合は，各状態$V 毎に，startElement イベントの際には startVariable($V) イベントを応用プログラムに対して発行し，endElement イベントの際には endVariable($V) イベントを応用プログラムに対して発行する．更に現在の状態の sax_f フラグが真である場合は，SAX イベントを応用プログラムに対してフォワードする．

上記のような DFA の処理アルゴリズムでは，DFA 構築後のメモリの確保・開放操作は不要であり，また各 SAX イベント

(注2)：各 DFA 状態において，推移群はハッシュテーブルにより管理される．

毎に適切な推移を選択する方法はハッシュテーブルを用いて実装されるため，SAX イベントに対するアルゴリズムのコストは $O(1)$ となる．つまり，この DFA を用いるアルゴリズムの性能は，XPath 式の件数に依存しないという重要な特徴を有している．

4.1.2 DFA の状態数の分析

一般的な正規表現の場合，それを処理する DFA の状態数は指数的になる可能性があることが知られている [9]．我々の問題のケースは制限された正規表現であるため，一般的な DFA の状態数の下限が当てはまるとは限らない．本節では，*eager DFA* の状態数と *Lazy DFA* の状態数について分析する．*eager DFA* とは，式 1 で示されるパワーセットコンストラクションにより得られる通常の DFA のことを指す．*Lazy DFA* については，入力ストリームの入力にあわせて DFA 処理に必要となる状態と推移を遅延的に構築して得られる DFA のことを指す．

4.1.3 Lazy DFA

Lazy DFA では，入力データに応じて必要な DFA の推移のみを評価することで，DFA の状態を実行時に生成する方法である．この方法によって，生成しなければならない DFA の状態を減らすことが期待できる．この DFA の遅延評価の考え方は，テキスト処理 [12] で既に適用されてきているが，我々の応用のように大量の条件式に対して適用されたことはなかった．形式的に *Lazy DFA* を A_l とすることとする．*Lazy DFA* の状態は下記の式で与えられるが，既に述べた *eager DFA* の状態を表す式 (1) と比較されたい．

$$states(A_l) = \{A_n(w) \mid w \in Ł_{data}\} \quad (2)$$
$$\delta(A_n(w), a) = A_n(wa), wa \in Ł_{data}$$

我々は理論的且つ実験的に Lazy DFA の状態数が XPath 式の件数に依存せず，DataGuide で表現される入力 XML ストリームの複雑さに依存することを導いた．この詳細については文献 [5] を参照されたい．

4.2 分岐 XPath 処理

一般的に XPath 式は複数の述語を含み，この述語の部分が問い合わせ木における分岐を構成する．本章では，*Lazy DFA* のような分岐を含まない XPath 式処理系を利用して，分岐 XPath を処理する手法 *XTree* [7] について説明する．分岐 XPath 式は，図 4(a) で示したように，相互に連結した分岐を含まない XPath 式 (シングルパス) に分解することができる．

シングルパス XPath 処理プロセッサは，入力 XML の SAX イベントを読み込むことで相互に連結した分岐のない XPath 式を評価し，XPath 式のマッチの開始/終了に応じて *Variable* の開始/終了イベントを，分岐 XPath 処理モジュールに対して通知する．通知された *Variable* イベントは，XPath における分岐がマッチしたことを意味するため，分岐 XPath 処理モジュールでは *Variable* の終了イベントをボトムアップに集約することによって，分岐点配下の全ての枝がマッチしたかどうかを評価することができる．また // の処理では複数の評価コンテキストが生じるため，*Variable* の開始イベントの際に複数の評価コンテキストを保持できるよう，分岐点ごとに評価コンテキストのスタックを使用する．

5. XSLT 差分変換

$XP^{\{[],*,//,vars\}}$ で定義された XPath/XSLT による実体化ビューの漸進的更新アルゴリズム *XTim* [8] の概要について述べる．*XTim* は XSLT プログラムの動的実行フロー (XT 木と呼ぶ) を利用可能なデータとして用いる．XT 木では，1) XSLT 式 (XSL 名前空間で定義される要素) への参照，2) 実体化ビュー，3) XSLT テンプレートがコンテキストとして利用したノード，を保持する．XT 木では，ロケーションステップの評価結果である中間ノード集合を全て保持しないことに注意して欲しい．XSLT 式は複数の XPath 式を利用するため，XT 木に保持された XSLT 式の木は XPath 式の木を構成する．ここで，入力 XML データのルートノードから更新された部分木のルートノードまでのパスを *update-path* と定義する．実体化ビューの漸進的更新処理は 3 つの部分処理から構成される．1) XSLT プログラムを再評価する際に使用するコンテキストノードと，XT 木で変更影響を受ける箇所 (impacted parts) とを特定する．この処理は，XPath の式の木から導出されたオートマトンに *update-path* を入力することで実現される．2) 特定したコンテキストノードを利用して，XSLT プログラムを部分評価し，XT 木の変更影響を受けた箇所を更新する．3) XT 木に保持された更新後の実体化ビューを出力する．

$XP^{\{[],*,//,vars\}}$ は，述語，要素や属性のワイルドカード，child と descendant 軸，変数参照が許された XPath サブセットである．$XP^{\{[],*,//,vars\}}$ ではノード間の順序関係を指定する軸の使用が許されていないが，一般的なデータ指向の XML には実用的なサブセットとなっている．テレビ番組ガイドのメタデータ (MPEG7, TVAnyTime, P/META) や電子カルテなどのデータ指向の XML データはノード間の順序関係に意味を持たない．このため，このような XML データではノード間の順序関係を指定する軸を利用しない．

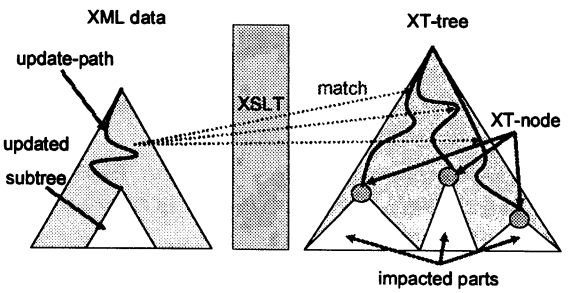

図 5 *XTim* によるビューの更新
Fig. 5 Overview of incremental view maintenance

XSLT の差分更新の技術の中で，最も重要である XPath の差分更新に関わる特徴を次の章で説明する．定理の証明については文献 [8] を参照されたい．

5.1 XPath の差分更新

ルートノードから入力 XML データのノードへの全てのパス集合を P と定義する．最初に，1 つのパスに対して XPath を評価する関数 *eval* を定義する．

[定義 5.1] $eval(P, xp, p)$ はブール関数であり，XPath 式 xp がパス p にマッチするか否かを評価する．xp の評価の際に *eval* 関数が参照しなければならないスコープは，第一パラメータで指定される (この場合は P である)． ■

続けて，複数のパス群に対する XPath 評価関数 $eval_c$ を定義する．

[定義 5.2] P' を P の部分集合とし，xp を XPath 式とする．P' に対する XPath 評価関数は以下のように定義される．

$$eval_c(P, xp, P') = \{p \mid p \in P', eval(P, xp, p)\}$$ ■

よって，入力 XML データ内の全てのパス P に対する xp の評価は $eval_c(P, xp, P)$ と表現される．

5.1.1 $XP^{\{*,//\}}$

まず述語を含まない XPath 式について検討する．述語の対処についてはその後に説明する．xp_s が述語を含まない $XP^{\{*,//\}}$ に属する場合，$eval(P, xp_s, p)$ は P に属する他のパスを参照することなく有限オートマトン [4], [5] により実装可能である．この理由は，xp_s が正規表現であり p がノード名称の順序列だからである．よって，$eval(P, xp_s, p) = eval(\{p\}, xp_s, p)$ が成り立ち，以下の命題を得る．

[命題 5.3] xp_s が $XP^{\{*,//\}}$ に属し，P' は P の部分集合とする．P' に対する関数 $eval_c$ は，P に属する他のパスを参照することなく評価される．

$$eval_c(P, xp, P') = eval_c(P', xp, P')$$ ■

これは単純な命題であるが，以降に述べる 2 つの定理はこれに基づいている．

5.1.2 $XP^{\{[],*,//\}}$

次に述語を含む XPath 式について検討する．入力 XML データのルートノードから更新された部分木のルートノードまでのパス *update-path* は，更新部分木に含まれるノード群への最長の共通パスであることに注意して欲しい．

まず実体化されたノードに変更影響を与える更新操作を考えてみよう．定理 5.4 はビュー更新の際の参照が必要となる入力 XML データ上のスコープを表している．

[定理 5.4] (locality) xp が $XP^{\{[],*,//\}}$ に属し，更新部分木に含まれるノード群へのパス群を ΔP とし，xp のステップで述語を含む最も先頭のステップを $step_{xp}$ とし，$step_{xp}$ がマッチする *update-path* のステップで最も先頭のステップを $step_{update}$ とする．もし，ΔP に属するあるパスが xp の最後のステップもしくはある述語にマッチするならば，ΔP に対する関数 *eval* は $step_{update}$ で示されるノードを含む全てのパス群 (P") を参照することで評価が可能である．

$$eval(P, xp, \Delta P) = eval(P'', xp, \Delta P)$$ ■

例えば，nodeID が 5 であるような *paper* が図 6 に示す更新操作によって，//paper[author/country = "Japan"] の XPath 式にマッチするようになる場合を想定しよう．この場合，$step_{xp}$ は //paper であり $step_{update}$ は (5,paper) であるから，P" は nodeID が 5 であるノードをルートとするような部分木になる．よって，XPath 式 //paper[author/country = "Japan"] は，その部分木に閉じて再評価することができる．

次に，実体化されたノードに変更影響を与えない更新操作を考えてみよう．*XTim* はこの定理 5.5 を利用して，変更影響を受けない XT 木の箇所を識別している．

[定理 5.5] (unaffected) xp と ΔP を上記と同様とし，更新操作が実施される前に xp の評価により実体化されたノードを n_{mat} とすることとする．もし，*update-path* のあるステップ $step_{update}$ についてその nodeID が n_{mat} の nodeID と同一であり，且つ ΔP に属するどのパスも xp の述語にマッチしないならば，更新操作は n_{mat} に変更影響を与えない． ■

例えば，nodeID が 5 であるようなノード n が //paper[year>2002] によって実体化されていて，図 6 に示す更新操作が n に実行されたと想定しよう．この更新操作は述語にマッチしないため，XPath 式の再評価は不要である．

```
insert(/(1,bib)/(5,paper)/(8,author),
       "<author role="3rd">
          <name>makoto onizuka</name>
          <country>Japan</country>
        </author>"
```

図 6 SUpdate 式の例

Fig. 6 An example of SUpdate expression

6. SVG 差分レンダリング

SVG データに対する変更差分を入力し，レンダリングされている SVG の内部データに反映して再レンダリングする方法について述べる．我々は SVG のレンダリングエンジン batik [13] を拡張して SVG の差分レンダリングを実現した．その概要を図 7 を用いて説明する．

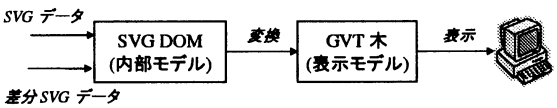

図 7 SVG 差分レンダリングの概要

Fig. 7 Overview of SVG incremental rendering

最初に SVG データがパースされてメモリ上に SVG DOM として保持される．以降，新しい入力 XML に対して差分変換された結果，差分の SVG が生成され，それを SAX パーサで解析して既存の SVG DOM に追加・反映する．続いて，SVG DOM はレンダリングのための表示モデルである GVT 木に変換され，GVT 木がレンダリングされて表示される．

現状の実装では SVG DOM だけを差分で更新し，GVT 木の生成とレンダリング操作については差分での更新を実現していない．GVT 木の生成とレンダリング操作の差分更新については今後の課題である．

7. まとめ

本稿では，インターネット上のアラートサービスなどに代表されるようなSDIシステムにおいて，大量の利用者のフィルタ条件を高速に処理し，フィルタ結果をHTML/SVGに変換・レンダリングするアーキテクチャと，必要となる一連のXMLストリーム技術 – XPathフィルタ処理, XSLT差分変換, 差分レンダリング – について説明した．今後は，アラートサービスで最も重要な機能であるフィルタ条件を中心に実サービスへの適用を検討する予定である．また残されている研究課題は以下の通りである．

XPathフィルタ

現状の分岐XPath処理については，XPush [14] のようにDFAにより分岐を共有化する方法か，本稿で述べた *XTree* に代表されるようにシングルパスのXPathの処理を共有化した上で，分岐処理は個々のXPath式毎に処理する方法のいずれかである．前者の方法は空間コストの問題が大きく実用的ではないし，また後者の方法はXPath式の件数に比例して性能が劣化するため，インターネット規模での情報提供を考えると，更なる性能向上が不可欠である．今後の課題としては，分岐XPathの処理に関して，分岐処理を共有化するDFA的な側面と状態数の爆発を招かないNFA的な側面を有するハイブリッドな手法の研究が挙げられる．

XSLT差分変換

提案した手法は個々のXSLT変換毎に実体化ビューをメンテナンスする手法であったが，これを拡張して複数のXSLT変換をまとめてメンテナンスする方法は今後の課題である．また，入力XMLのタグの上下関係を逆転させるような構造変換においては更なる高速化の余地があることが分かっているし [8]，$XP^{\{[],*,//,vars\}}$ よりも汎用なXPathのクラスにおける効率的な差分変換手法についても今後の課題である．

SVGレンダリング

SVGレンダリングについては，現状のHTML/SVGブラウザが標準的な機能としてデータの差分更新を具備することが重要である．またグラフ表示に関しては集計的な操作が多いことから，差分での処理の余地が多く，更なる高速化が可能であると考えられる．

文　献

[1] PubSub Concepts, Inc: "PubSub". http://www.pubsub.com.

[2] M. Altinel and M. J. Franklin: "Efficient filtering of XML documents for selective dissemination of information", Proceedings of VLDB, pp. 53–64 (2000).

[3] S. Nishioka, Y. Yaguchi, T. Hamada, M. Onizuka and M. Yamamuro: "XML-based e-barter system for circular supply exchange", Proceedings of DEXA (to appear, 2005).

[4] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang and P. Fischer: "Path sharing and predicate evaluation for high-performance XML filtering", ACM Trans. Database Syst., **28**, 4, pp. 467–516 (2003).

[5] T. Green, A. Gupta, G. Miklau, M. Onizuka and D. Suciu: "Processing XML streams with deterministic automata and stream indexes", ACM Trans. Database Syst., **29**, 4, pp. 752–788 (2004).

[6] C. Y. Chan, P. Felber, M. N. Garofalakis and R. Rastogi: "Efficient filtering of XML documents with XPath expressions", Proceedings of ICDE (2002).

[7] M. Onizuka: "Light-weight XPath processing of XML stream with deterministic automata", Proceedings of CIKM, pp. 342–349 (2003).

[8] M. Onizuka, F. Y. Chan, R. Michigami and T. Honishi: "Incremental maintenance for materialized XPath/XSLT views", Proceedings of WWW, pp. 671–681 (2004).

[9] J. E. Hopcroft, R. Motwani and J. D. Ullman: "Introduction to automata theory, languages, and computation (Second Edition)", Addison-Wesley (2001).

[10] Z. G. Ives, A. Y. Halevy and D. S. Weld: "An XML query engine for network-bound data", The VLDB Journal, **11**, 4, pp. 380–402 (2002).

[11] B. W. Watson: "A taxonomy of finite automata construction algorithms", Computing Science Report 93/43, University of Technology Eindhoven (1993).

[12] V. Laurikari: "Nfas with tagged transitions, their conversion to deterministic automata and application to regular expressions", Proceedings of SPIRE, pp. 181–187 (2000).

[13] The Apache Software Foundation: "Batik SVG toolkit". http://xml.apache.org/batik/.

[14] A. K. Gupta and D. Suciu: "Stream processing of XPath queries with predicates", Proceedings of SIGMOD, pp. 419–430 (2003).