Information Processing Society of Japan

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

2007－DBS－143（39）
2007／7／3

インクリメンタルに更新可能なXPushマシンの性能向上と応用に関する考察

武川 肇[†,††] 福田 直樹[†††] 石川 博[†††]

† 職業能力開発総合大学校情報システム工学科
†† 静岡大学創造科学技術大学院自然科学系教育部情報科学専攻
††† 静岡大学情報学部情報科学科

あらまし XMLフィルタリングにおける述部評価手法にGuptaらのオートマトン（XPushマシン）がある。これまでに我々は、オートマトンのマージに基づいて、XPushマシンが部分フィルタ交換を行えるように改良してきた。本稿では、このXPushマシンを既存のXMLフィルタリングエンジン（YFilter）に連携させる方法について述べる。
キーワード XMLフィルタリング, XPath, ストリーム処理

Performance Improvement and Application of An Incrementally Updatable XPush Machine

Hajime TAKEKAWA[†,††], Naoki FUKUTA[†††], and Hiroshi ISHIKAWA[†††]

† Department of Information System Engineering, Polytechnic University
†† Department of Information Science and Technology, Educational Division, Graduate School of Science and Technology
††† Department of Computer Science, Faculty of Informatics, Shizuoka University

Abstract In XML filtering, Gupta and Suciu proposed an automaton (called an XPush machine) for stream processing of XPath queries with predicates. Recently we improved the XPush machine based on merging automata to have a partial filter exchange function. In this paper, we will describe how to associate this XPush machine with ordinary XML filtering engine as YFilter.
Key words XML filtering, XPath, stream processing

1. はじめに

ニュース記事やセンサデータなどのXMLデータが逐次発生している。これらXMLデータを処理するシステム（例、SDIやnotificationシステム）において、XMLフィルタリングは主要な機能であり、その機能にはXMLデータごとに多くのフィルタを効率よくマッチングする能力が要求される[1], [2]。

XMLフィルタリングにおけるマッチングは構造マッチングと述部評価からなる。既存のXMLフィルタリングエンジンのYFilter[3]（現状ではこの研究分野の到達水準の1つ）は、構造マッチングには優れているが、単純な述部評価しかできない。一方、述部評価だけに特化したXPushマシンが提案されている[4]。

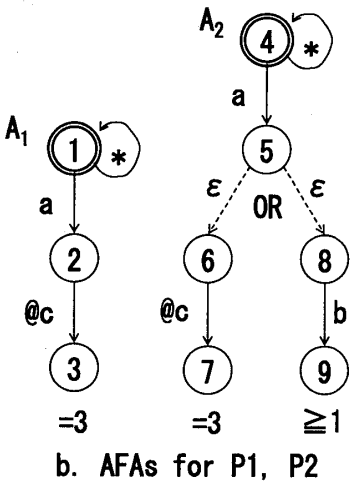
XPushマシンはXMLドキュメントのストリームに対して、複雑な述部を持ったフィルタ条件の巨大集合を評価できる。XPushマシンではフィルタ条件をXPath[5]で記述し、この条件をXPathフィルタという。XPathフィルタの述部には、フィルタごとに1つの述部しか許さず、またマッチした位置の特定もできない。しかしORなどの論理演算、≧などの比較演算が利用でき、子や子孫軸を許す。またXPushマシンは、述部評価のためのトラバース[6]によるバックトラックも、XMLデータへのランダムアクセスも不要であるため、XMLデータごとに1回のストリーム処理で全フィルタの述部評価を完了する。

XPushマシンはXPathフィルタごとにAFA（Alternating Finite Automaton[7]）を構築し、評価しているデータの構造をもとに、それらAFAから1つの決定性PDA（Push Down Automaton）へと姿を変える。それは「アクティブなAFA状態の全てをAFA上で遷移させる」という状態列挙処理からなり、さらにその結果はXPushマシンの内部状態（XPush状態）や状態遷移表としてキャッシュされる。そのため同じ（構造の）XMLデータに対するXPushマシンによる述部評価は効果的である。

ところがXPushマシンは更新できないために、1つのフィルタを削除することであっても、XPushマシン全体の再計算

NII-Electronic Library Service

P1: //a[@c=3]
P2: //a[@c or b/text()>=1]
a. XPath filters



b. AFAs for P1, P2

図 1 XPath フィルタとその内部表現
Fig. 1 XPath filters and AFAs

(構築) が必要となる。すなわちオートマトンの更新コストは AFA (フィルタ) の総数に依存してしまう。

そこで我々はこれまでに、XML フィルタリングにおけるオートマトンのマージに基づいたインクリメンタルな更新手法を提案してきた [8]。その手法は、Gupta らの XPush マシンと異なり、順序付きハッシュキーを用いることによって XPush マシンの更新コストを削減させている。そのためフィルタ (ユーザ嗜好、あるいは問合せ) がダイナミックである場合にとても役に立つ。

本稿では、インクリメンタルに更新可能な XPush マシンで、YFilter エンジンの述部評価を処理させることについて考えていく。

先に述べたように、従来方式の XPush マシンはマッチした位置が特定できない。そこで XML パーサに組込んだ XPush マシンを提案する。この XPush マシンは XML 要素ごとに述部評価結果が得られ、さらにその評価結果は SAX (Simple API for XML) [9] の startElement イベント処理から利用できる。このような XPush マシンを用いて YFilter の述部評価を強化する。

また XPush マシンのアクティブな AFA 状態を減らし、状態列挙処理を改善する試みも行う。これは、XPush マシンのキャッシュが不十分であっても、つまりヒット率が低い間でも、XPush マシンの述部評価速度が、改良前の YFilter の述部評価速度と同等以上になることを目的としている。具体的には XPush マシンの AFA 集合を決定性 PDA へ変換する前に、フィルタ間のサフィックスの共通性に基づいて 1 つの AFA に結合させる方法を提案する。

本稿の構成は以下のとおりである。2 章で、XPush マシンの概要について述べる。3 章で、XPush マシンを、XML フィルタリングの述部評価に用いる際の問題点を述べる。4 章で、

```
startDocument
  q ← φ
  s ← empty stack;
startElement(a)
  push(s, q);
  q←φ
endElement(a)
  q_aux ← t_pop(q, a)
  q_s ← pop(s);
  q ← t_add(q_s, q_aux);
text(str)
  q ← t_value(str);
endDocument
  return t_accept(q);
```

図 2 コールバック関数
Fig. 2 Call-back functions

XML パーサに組込んだ XPush マシンによる述部評価について述べる。なお提案方式の評価については今後の課題である。

2. 従来方式：XPush マシンの概要

この章では XPush マシンの概要について述べる。また YFilter の XPush マシンとの違いにも簡単にふれる。なお XPush マシンの決定性 PDA への変換と最適化 [4]、インクリメンタルな更新 [8] については省略する。

XPush マシンは、プッシュダウンオートマトンである。XPush マシンは、あらかじめ XPath フィルタごとにオートマトン (AFA) を構築し、その後 XML データごとにそれら構築済みの AFA 集合をボトムアップに状態遷移させる。状態遷移の際、バックトラックも、XML データへのランダムアクセスも必要としない。XPush マシンは、SAX ベースのイベント処理で全 XPath フィルタの評価を完了する。

例えば図 1a の XPath フィルタが与えられると、図 1b のように根が受理状態である AFA が構築される。その後 XML データごとに、図 2 のコールバック関数に従って、AFA 集合の状態遷移が発生し、図 3 のように、XPush マシンのカレント状態 (current state) q とスタック s が変化する。

XPath フィルタは、図 1a の式 P_1、P_2 のように XPath で記述される。XPath フィルタは構造ナビゲーション (structure navigation) と述部 (predicate) で構成され、"[ ]" の部分が述部である。XPath フィルタの述部は、XQuery [10] の Where 句のように、論理演算や軸を用いたパス表現を許し、パス上の葉の全てに値との比較演算が存在する。ただし YFilter などで扱われるフィルタとは異なり、1 つの XPath フィルタには複数の述部を持たせることができない。

XPush マシンは XPath フィルタの内部表現として、AFA を利用する。AFA は、XPath フィルタが論理演算や軸を許すように、それらに対応している。たとえば図 1b の A_1、A_2 では、AFA 状態の 5 が or に対応し、1 と 4 のように閉路によって子孫軸が表現される。一方 YFilter では、述部での論理演算や軸

NII-Electronic Library Service

Information Processing Society of Japan

	data	q	s
0			
1	<x>	ϕ	{ϕ}
2	<a>	ϕ	{ϕ}{ϕ}
3	<@c>	ϕ	{ϕ}{ϕ}{ϕ}
4	3	ϕ	{ϕ}{ϕ}{ϕ}
5	</@c>	3, 7, 9	{ϕ}{ϕ}
6		2, 6	{ϕ}
7	<a>	1, 4	{ϕ}{1, 4}
8		ϕ	{ϕ}{1, 4}{ϕ}
9	1	ϕ	{ϕ}{1, 4}{ϕ}
10		9	{ϕ}{1, 4}
11		8	{ϕ}
12	</x>	1, 4	
13		1, 4	

図 3　XPush マシン実行時のトレース
Fig. 3　Trace of XPush machine execution

は扱えないため、P_2 の述部を評価できない。

XML データは、AFA 集合の状態を遷移させるための入力データとして扱われる。XML データは、データの先頭からタグや値ごとに SAX イベントとして分解され、SAX イベントごとにシーケンシャルにイベント処理される。なお XPush マシンのイベント処理は、標準の SAX とは少し違いがある。違とは、属性を要素と同じように扱うことと、1 つの要素のテキスト値はまとめて 1 つのイベントで扱う点である。また YFilter も、要素のテキスト値を属性と同じように要素の引数として扱う点で、標準の SAX と異なっている。

XPush マシンは、SAX イベントを図 2 のコールバック関数で処理している。コールバック関数は、カレント状態とスタックをグローバル変数として扱い、AFA 集合のボトムアップな状態遷移を実現している。カレント状態には、各 AFA のアクティブ状態の集合が格納され、アクティブ状態が AFA の受理状態となっている間、t_{accept} 関数による XPath フィルタの述部評価の結果は true となる。

たとえば図 3 では、q の値は最終的に 1 と 4 となっている。これらの値は図 1b の AFA のアクティブ状態の番号を意味している。図 1b において、AFA 状態 1 と 4 は、それぞれ A_1 と A_2 の受理状態である。したがって、入力データに対して XPath フィルタ P_1 と P_2 は、ともに true と見なされる。

イベント処理の作業は、コールバック関数の endElement プロシージャに集中している。endElement プロシージャでは、まず t_{pop} 関数によって、カレント状態 q を根方向に a（評価している要素名あるいは属性名）の遷移をさせ、その結果を q_{aux} に代入している。次に pop 関数によって、スタックから状態を取り出し、それを q_s に代入する。最後に t_{add} 関数によって、q_{aux} と q_s の和集合をもとめ、その結果を q に代入する。つまり、endElement プロシージャでは、q_{aux} は a を頂点とした枝に対する述部評価結果が代入され、そして q_s は a と兄弟であり、かつ a より前にでてくる枝の述部評価の合計が代入される。なお t_{pop} 関数と t_{add} 関数の結果は、XPush マシンの内部状態（XPush 状態）や状態遷移表としてキャッシュされる。

3. XPush マシンの問題点

この章では、XPush マシンを、XML フィルタリングエンジン（YFilter）の述部評価に用いる際の問題点を述べる。

状態遷移をキャッシュするまでの XPush マシンは、フィルタ間の共有を述部の葉にしか利用していない。そのため、例えば全ての述部が同じ式であっても、全ての式が異なっている場合と同様に、キャッシュするまでの時間はフィルタ数に依存する。アプリケーション（ここでは YFilter）側で、同じ述部式を取り除くことも考えられるが、部分的な一致を取り除くことは難しい。なおこのことに関する最適化は、従来方式 [4] には含まれていない。

また XPush マシンは、endDocument イベントの t_{accept} 関数によって受理を確認するため、マッチした位置が特定できない。たとえ終了要素の順序（つまりポストオーダ）で受理を確認できるようになったとしても、イベント処理のタイミングの違いが問題点として残る。なぜなら YFilter や lazyDFA [2] がそうであるように、XML フィルタリングの構造マッチングは開始要素の順序（つまりプリオーダ）で処理するためである。もしポストオーダの述部評価を、startElement イベントで利用しようとすると、述部評価のためのトラバースや XML データへのランダムアクセスが必要となり、XPush マシンを用いる利点がなくなる。

4. 提案方式：XML パーサに組込んだ XPush マシンによる述部評価

この章では本提案方式の XML パーサに組込んだ XPush マシンの概要を述べてから、フィルタ間のサフィックスの共通性に基づき XPush マシンの性能を向上させる方法、そして XPush マシンの述部評価結果をフィルタリングエンジンに渡す方法について述べる。

4.1 XML パーサに組込んだ XPush マシンの概要

ここでは XML パーサに組込んだ XPush マシンの概要について述べる。

図 4 に、XPush マシンと YFilter による XML フィルタリングシステムのアーキテクチャを示す。システムは XPath パーサ、XML パーサ、フィルタリングエンジン、データ配信の 4 つのモジュールで構成されている。各モジュールとも YFilter のモジュールをベースに拡張している。ただしデータ配信モジュールについては、YFilter からの変更はない。

本システムの XPath パーサは、YFilter の XPath パーサにクエリ書換え機能を追加したモジュールである。この XPath パーサにクエリ集合が渡されると、各クエリ Q_i に含まれる 0 個以上の全ての述部が XPath フィルタ P_j として抽出され、各クエリ Q_i はその抽出箇所が [@P=j] の形式で書き換えられ、クエリ Q_i' となる。抽出された P_j 集合は XML パーサに組込まれた XPush マシンに渡され、Q_i' はさらに構文解析されてから、フィルタリングエンジンに渡される。

NII-Electronic Library Service

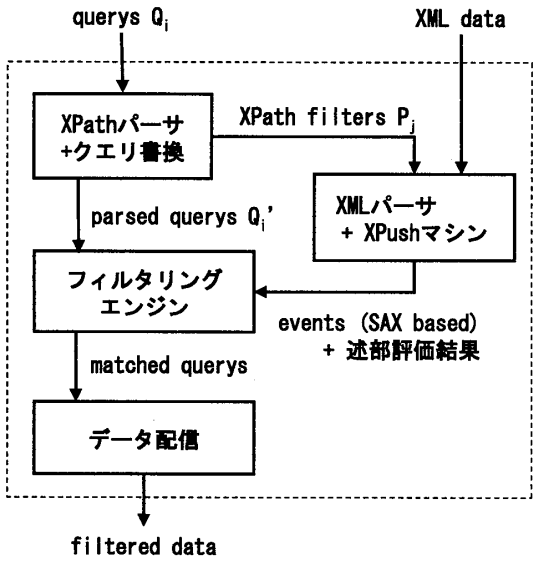


図 4 XPush マシンと YFilter による XML フィルタリングシステム
Fig. 4 Architecture of a filtering system using XPush machine and YFilter

Q_1://a[@c=3]/text()
Q_2://a[@c=3 and b/text()>=1]

a. querys

P_1:a[@c=3]
P_2:a[@c=3 and b/text()>=1]

b. XPath filters for Q_1, Q_2

Q_1' ://a[@P=1]/text()
Q_2' ://a[@P=2]

c. rewrited Querys of Q_1, Q_2

図 5 クエリの書き換え
Fig. 5 Rewriting queries

たとえば、図 5a のように XPath パーサに 2 つのクエリ Q_1 と Q_2 が渡されるとする。まず各クエリから図 5b の XPath フィルタ P_1 と P_2 が抽出され、各クエリは図 5c の Q_1' と Q_2' に書き換えられる。P_1 と P_2 は XML パーサに組込まれた XPush マシンに渡され、Q_1' と Q_2' は構文解析されてから、フィルタリングエンジンに渡される。

なお、抽出される XPath フィルタの構造ナビゲーションに子孫軸を持たせていない。そのため AFA の受理状態は閉路ではない。この変更点は XPath フィルタが true になる位置を特定させるために必要となる。もし従来方式のように受理状態が閉路のままであると、XPath フィルタがある要素を true と判定すると、その要素だけでなく、その先祖の要素も true となってしまう。

次に、本システムの XML パーサについて説明する。本システムの XML パーサは、YFilter の XML パーサモジュールに XPush マシンを組込んでいる。ただし組込まれる XPush マシンは次の特徴をもっている。

(A) XPath フィルタ集合からサフィックスが共有された AFA を 1 つだけ構築する。
(B) XML 要素ごとに述部評価結果が得られ、その評価結果は startElement イベントで利用できる。

なお変更点 A については 4.2 節で、変更点 B については 4.3 節で述べる。

最後に、本システムのフィルタリングエンジンについて説明する。フィルタリングエンジンは述部評価結果を扱えるように YFilter を少しだけ拡張している。そこでは述部評価の実際の処理は行わず、必要時に XML パーサから送られてくる述部評価結果を利用する。[@P=j] は、XPath フィルタ P_j の評価結果を利用する。たとえば [@P=1] は、r[*eid*] に P_1 が含まれていれば true であり、そうでなければ false となる。ただし *eid* は評価中の XML データの処理済み要素数+1 である。

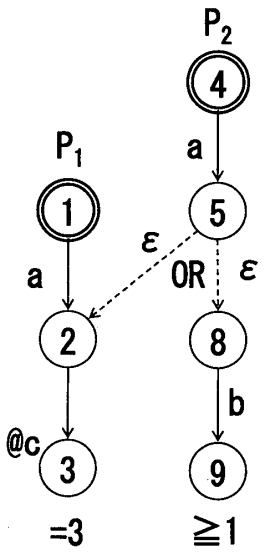


図 6 P_1 と P_2 を結合して構築した AFA
Fig. 6 A combined AFA for P_1, P_2

4.2 XPath フィルタのサフィックス共有

ここではフィルタ間のサフィックスの共通性に基づき XPush マシンの性能を向上させる方法を述べる。

3 章で述べたように、従来方式の XPush マシンは XPath フィルタが同じ式であっても、全ての式が異なっている場合と同様に、キャッシュするまでの時間は XPath フィルタの数に依存する。そこで本システムでは、XPath フィルタ間のサフィックスの部分的な一致を取り除くことによって、AFA に対する状態遷移の計算コストを削減させる。そのために XPath フィルタ集合からサフィックスが共有された AFA を 1 つだけ構築する方法を提案する。

たとえば図 5b の P_1 と P_2 が渡されると、図 6 の AFA が得られる。図 7 の q の 5 番目と 6 番目の値はそれぞれ 3,7 と 2 であり、従来方式の図 3 と比べて、アクティブな AFA 状態の数が削減されていることがわかる。

NII-Electronic Library Service

	data	q	s
0			
1	<x>	ϕ	{ϕ}
2	<a>	ϕ	{ϕ}{ϕ}
3	<@c>	ϕ	{ϕ}{ϕ}{ϕ}
4	3	ϕ	{ϕ}{ϕ}{ϕ}
5	</@c>	3,7	{ϕ}{ϕ}
6		2	{ϕ}
7	<a>	1,4	{ϕ}{1,4}
8		ϕ	{ϕ}{1,4}{ϕ}
9	1	ϕ	{ϕ}{1,4}{ϕ}
10		7	{ϕ}{1,4}
11		6	{ϕ}
12	</x>	1,4	
13		ϕ	

図 7 アクティブな AFA 状態の削減
Fig. 7 Reducing number of active AFA states

```
startDocument
  q ← φ
  eid ← 0
  s ← empty stack;
  e ← empty stack;
startElement(a)
  if (not a.startWith("@"))
    push(e, eid++);
  push(s, q);
  q←φ
endElement(a)
  q_aux ← t_pop(q, a)
  if (not a.startWith("@"))
    r[pop(e)]←t_accept(q_aux);
  q_s ← pop(s);
  q ← t_add(q_s, q_aux);
text(str)
  q ← t_value(str);
endDocument
  parse(xml, r);
```

図 8 提案方式のコールバック関数
Fig. 8 Call-back functions of a proposed method

なお XPath フィルタのサフィックス共有アルゴリズムの開発、そしてそれにともなう AFA のデータ構造の変更については今後の課題である。

4.3 XML 要素ごとの述部評価結果

ここでは XPush マシンによる述部評価結果を、フィルタリングエンジンに渡す方法について述べる。

XPush マシンの問題点については 3 章で述べたように、XPath フィルタのサフィックス共有を利用していないこと以外にもある。その問題点とは、XPath フィルタが true となる位置が特定できないことと、イベント処理のタイミングが違うことであった。そのための解決策として、4.1 節で述べた XPath

	data	q	s	q_{aux}	r[]
0					
1	<x>	ϕ	{ϕ}		
2	<a>	ϕ	{ϕ}{ϕ}		
3	<@c>	ϕ	{ϕ}{ϕ}{ϕ}		
4	3	3,7	{ϕ}{ϕ}{ϕ}		
5	</@c>	2	{ϕ}{ϕ}	2	
6		1,4	{ϕ}	1,4	P1,P2
7	<a>	ϕ	{ϕ}{1,4}		
8		ϕ	{ϕ}{1,4}{ϕ}		
9	1	7	{ϕ}{1,4}{ϕ}		
10		6	{ϕ}{1,4}	6	ϕ
11		1,4	{ϕ}	4	P2
12	</x>	ϕ		ϕ	ϕ
13					

図 9 XML 要素ごとの述部評価
Fig. 9 Predicate evaluation every XML element

フィルタの構造ナビゲーションの変更点について述べたが、それだけでは不十分である。本提案の XML パーサに組込まれた XPush マシンには、さらに図 8 のコールバック関数を用いる必要がある。

このコールバック関数には次の 3 つの改良点がある。

(A) t_{accept} 関数を endElement イベントの t_{pop} 関数の結果 q_{aux} に対して利用している。ただしこの処理は要素に対して行い、属性にはその必要はない。

(B) A によって XPath フィルタの評価結果が XML 要素ごとに得られる。評価中の XML データの要素数だけ一時記憶する必要があるため、配列 r を増やしている。

(C) プリオーダの処理で評価結果が利用できるように、r には開始要素の順序で格納している。変数 eid とスタック e を利用して、r に格納する際の添え字を計算している。

まず改良点 A と B について考える。図 9 の q_{aux} のように、t_{pop} 関数によって q_{aux} が得られる。さらにこの q_{aux} に対して、t_{accept} 関数によって r が得られる。q_{aux} は、3 章で説明したとおり、a を頂点とした枝に対する述部評価結果である。そのため r には、XML 要素ごとに XPath フィルタの評価結果が代入される。

次に改良点 C について考える。図 9 の r は、ポストオーダの処理で得られた終了要素の出現順であり、開始要素の出現順ではない。一方、図 10 のように、e から取り出される pop(e) を r 上の格納位置とすれば、r に格納された述部評価結果は、開始要素順となることがわかる。なお図 10 では r の 1 要素ごとに *event* に埋め込んでいるが、実際にはその必要はなく、配列全体がインデックスとしてフィルタリングエンジンに渡されればよい。

5. ま と め

XPush マシンを、XML フィルタリングの述部評価に用いる際の問題点を明らかにし、その問題点を解決するために、「XML

NII-Electronic Library Service

	data	e	pop(e)	events
0				
1	<x>	{1}		1:x, r[1]
2	<a>	{1} {2}		2:a, r[2]
3	<@c>			, @{c=3}
4	3			
5	</@c>			
6		{1}	2	-1:$
7	<a>	{1} {3}		3:a, r[3]
8		{1} {3} {4}		4:b, r[4]
9	1			, text=1
10		{1} {3}	4	-1:$
11		{1}	3	-1:$
12	</x>		1	-1:$
13				

図 10　開始要素順での述部評価結果の格納

Fig. 10　Storage of a predicate evaluation result by start element order

パーサに組込んだ XPush マシン」を提案した。

今後の課題として、XPath フィルタのサフィックス共有アルゴリズムの開発、提案方式の評価を行う予定である。

謝辞

本研究の一部は科学研究費補助金基盤研究（B）（課題番号 19300026）の助成による。

文　献

[1] Mehmet Altinel, Michael J. Franklin: Efficient Filtering of XML Documents for Selective Dissemination of Information, Proceedings of VLDB, pages 53-64, 2000.

[2] Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, Dan Suciu: Processing XML streams with deterministic automata and stream indexes, ACM TODS, 29(4):752-788, 2004.

[3] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, Peter Fischer: Path sharing and predicate evaluation for high-performance XML filtering, ACM TODS, 28(4):467-516, 2003.

[4] Ashish Kumar Gupta, Dan Suciu: Stream processing of XPath queries with predicates, Proceedings of SIGMOD, pages 419-430, 2003.

[5] J. Clark: XML path language(XPath), http://www.w3.org/Tr/xpath.

[6] K. Selcuk Candan, Wang-Pin Hsiung, Songting Chen, Junichi Tatemura, Divyakant Agrawal, AFilter: Adaptable XML Filtering with PrefixCaching and SuffixClustering, Proceedings of VLDB, 2006.

[7] Ashok K. Chandra, Dexter C. Kozen, Larry J. Stockmeyer: Alternation, Journal of the ACM, pages 114-133, 1981.

[8] 武川肇, 片山薫, 石川博: インクリメンタルに更新可能な XPush マシン, 情報処理学会:データベース, Vol.46, No.SIG18(TOD28) pp.116-128, 2005.

[9] http://www.saxproject.org/.

[10] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery.

NII-Electronic Library Service