法政大学学術機関リポジトリ
HOSEI UNIVERSITY REPOSITORY

PDF issue: 2025-05-12

XML ストリームの複合イベント処理向け問い合わせ言語の設計と実装

Matsuda, Tatsuki / 松田, 達希

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
11
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2016-03-24
(URL)
https://doi.org/10.15002/00012903

XML ストリームの複合イベント処理向け問い合わせ言語の設計と実装

Design and Implementation of Query Language for Complex Event Processing over XML Streams

松田　達希
Tatsuki Matsuda
法政大学大学院情報科学研究科情報科学専攻
E-mail: 14t0011@cis.k.hosei.ac.jp

Abstract

This paper describes a query language and a complex event processing (CEP) system over multiple continuous XML data streams. First, multiple stream processing is formulated in algebraic expressions for stream filtering, union, activation, decomposition and partition. Next, a query language, called QLMXS, over XML streams is defined on all functions of the algebraic models in an SQL-like form. Finally, a CEP system is proposed for processing QLMXS queries. The performance of the system is dominated by parsing XML documents and finding specific sequences of events. The former is resolved by adopting VTD parser, which is a modern XML parser and faster than the traditional technologies of DOM, SAX and StAX. The latter is achieved by a finite state automaton (FSA) that determines the acceptance of the event sequences represented in QLMXS queries. Experimental results demonstrate that the performance of the proposed QLMXS system becomes 1.7 times faster than the previous implementation which used the StAX parser and the visibly pushdown automaton (VPA) as cores of the analytical engines.

1. はじめに

近年, ストリームデータを高速に処理・分析するための手法として複合イベント処理(CEP)の研究が盛んに行われている．ストリームデータの形式は多種多様であり，高速な処理を実現するためにはその形式に合わせたシステムを構築し，効率的に処理を行う必要がある. この中でも本研究では XML 形式のストリームデータに注目し，XML ストリームデータ向けの CEP システムの構築を目指す.

XML は構造化されたデータ形式であり，データの検索・取得が比較的容易に行える．しかしながら，従来の XML 向け問い合わせ言語である XPath や XQuery はストリームデータのような連続で発生し続けるデータの処理・分析には適していない．一方，XPath をストリーム処理向けに拡張した問い合わせ言語及び処理システムとして XSeq[1]が提案されている．しかし，処理対象は単一ストリームのみとなっており，複数のストリームに跨るような複雑な検索には対応していない.

本稿では, 複数の XML ストリームに対しても高速な処理・分析を行えるような CEP システムの構築を目標にする．まず，CEP システムにおいて必要とされる処理を計算量も考慮した上でモデル化する．そして，このモデルに沿ったマルチ XML ストリーム向けの問い合わせ言語：Query Language for Multiple XML Streams(QLMXS)の設計を行う．また，QLMXS で記述された検索要求を実現するための検索エンジンを実装し，性能実験，及び実際の株式市場のデータを用いたシミュレーション実験により，その性能を確かめる.

第 2 章では関連技術について述べ，第 3 章ではマルチストリーム処理のモデルについて述べる．第 4 章では QLMXS の仕様を，第 5 章では処理システムの概要をそれぞれ説明する．第 6 章では実験とその結果について述べ, 第 7 章で考察を行い, 最後にまとめを行う.

2. 関連技術

2.1. Complex Event Processing

CEP とは，時々刻々と連続して発生するストリームデータをリアルタイムに処理・分析し，価値ある情報を見つけ出すための処理手法である．CEP では各ストリームデータの発生を 1 つのイベントとみなし，それぞれのイベントの組み合わせや発生のパターン等から高次のイベントを発見し，発見したイベントを利用して更に高次のイベントを連鎖的に発見していく.

CEP システムでは，処理を行う前に問い合わせを登録し，その問い合わせに対応した解析エンジンを生成する．そしてこのエンジンが，ストリームデータを逐次処理していく．従来のデータ分析手法の 1 つであるデータベースを用いてデータを蓄積して処理を行う手法とは異なり，CEP システムの処理はオンメモリで行われるため，よりリアルタイムな処理が可能である.

2.2. XML ストリーム処理

Mozafari らによって提案された XSeq は，XML ストリーム向けの問い合わせ言語および処理システムである．問い合わせ言語に関しては，XPath を拡張し，クリーネ*と兄弟関係を明示的に表す ‘\’オペレータを追加することで，時系列処理に適した検索を可能にしている．XML の解析エンジンには VPA[2]と呼ばれるプッシュダウンオートマトンの制約を強めたオートマトンをベースに利用している．VPA は有限状態オートマトン(FSA)と同等の最適化が可能であり, XML や JSON など入れ子構造データの解釈にも適しているという特徴がある．XSeq ではクエリを VPA で表現し，様々な最適化を VPA に施すことによって処理性能の向上を実現している．この方式では時系列を踏まえた複雑な検索要求も高速に処理することが可能であるものの，処理

Supervisor: Prof．Satoru Fujita

対象は単一ストリームのみとなっている．

3. ストリーム処理のモデル

シングル及びマルチストリームに対する分析処理を，代数表現を用いてモデル化する．モデル化する処理は，ストリームデータに対するフィルタリング(Filtering)，複数ストリーム同士の合成(Union)，複数のストリームデータの合成(Activation)，ストリームデータの細分化(Decomposition)，単一ストリームから複数ストリームへの分割(Partitioning)の 5 つとする．ストリームを $\boldsymbol{s}$，フィルタを $\boldsymbol{f}$，ストリーム中のデータ数を $\boldsymbol{n}$，クエリ数を $\boldsymbol{q}$ として，それぞれのモデルを計算量を踏まえて定義する．

(1) Filtering

Filtering は最も基本となるモデルであり，ストリームデータに対し，特定の条件を満たすものだけをフィルタリングする処理のモデルである．フィルタリングされたデータは，新たなストリーム又は検索結果として直接出力される．ストリーム $\boldsymbol{s}$ にフィルタ $\boldsymbol{f}$ を適用して新たなストリーム $\boldsymbol{s'}$として出力する処理は，以下のようなモデルで表現できる．

$$\boldsymbol{s} \mid \boldsymbol{f} \gg \boldsymbol{s}'$$

また，計算量は式(1)で表される．

$$O(\boldsymbol{nq}) \tag{1}$$

(2) Union

Union は別々のストリーム中のデータを同じストリームに出力することで，ストリームを合流させる処理のモデルである．この処理のモデルは，結合演算子: + を用いて以下のように表現する．

$$\boldsymbol{s} + \boldsymbol{s}' \gg \boldsymbol{s}''$$

このモデルに関しては，結合法則及び交換法則が成り立つ．また，計算量を式(2)に示す．

$$O((\boldsymbol{n} + \boldsymbol{n}')\boldsymbol{q}) \tag{2}$$

(3) Activation

Activation はあるストリームで発生したイベントが，もう一方のストリームのイベント処理を呼び起こすような処理のモデルである．出力されるデータは 1 つであり，2 ストリームのデータを合成して出力することも可能である．モデルは合成演算子: * を用いて以下のように表現する．

$$\boldsymbol{s} * \boldsymbol{s}' \gg \boldsymbol{s}''$$

Activation に関しては，2 ストリーム間に順序関係が存在するため結合法則及び交換法則は成り立たない．また，計算量を式(3)に示す．

$$O((\boldsymbol{n} + \boldsymbol{n}')\boldsymbol{q}) \tag{3}$$

しかしながら，イベントに対する検索条件によっては一方のストリームのデータがもう一方のストリームの一定期間のデータと比較を行ってから，イベント処理の呼び起こしが発生する場合が考えられる．その場合，2 つのストリーム中のデータが 1 対多の関係になる可能性があり，計算量が大きく変わってくる．よって，1 対多の場合に関しては期間であることを表す $\boldsymbol{t}$ を用いて以下のようにモデルを別に定義する．

$$\boldsymbol{s} * \boldsymbol{t}(\boldsymbol{s}') \gg \boldsymbol{s}''$$

また，この場合の計算量は式(4)になる．

$$O(\boldsymbol{nn}'\boldsymbol{q}) \tag{4}$$

(4) Decomposition

Decomposition はストリーム中のデータを指定されたキー単位で分割して出力する処理のモデルである．ストリームには複数のデータを内包した巨大なデータが流れてくる可能性もあり，データを細分化する必要が生じる場合がある．データを分割する際のキーは，XML におけるタグに相当するものである．キーを $\boldsymbol{k}$ として，分割演算子: / を用いて以下のようなモデルで表現する．

$$\boldsymbol{s} \,/\, \boldsymbol{k} \gg \boldsymbol{s}'$$

計算量は式(5)になる．

$$O(\boldsymbol{nq}) \tag{5}$$

(5) Partition

Partition はストリームデータを指定されたキーに対応する値毎にストリームを分割して出力する処理のモデルである．ストリームデータは発生元が同じでも様々な値を持つデータが混在しているため，前処理としてデータが持つ値毎にまとめる必要が生じる．分割されたストリームは配列として出力される．キーを $\boldsymbol{k}$ として，以下のようなモデルで表現できる．

$$\boldsymbol{s}[\boldsymbol{k}] \gg \boldsymbol{s}'[]$$

分割されたストリーム配列中から特定の値を持つストリームを指定する場合は，値を $\boldsymbol{v}$ とすると，以下のように指定する．

$$\boldsymbol{s}'[key = \boldsymbol{v}]$$

計算量を式(6)に示す．

$$O(\boldsymbol{nq}) \tag{6}$$

4. QLMXS

QLMXS は XPath をマルチストリーム処理向けに拡張し，SQL ライクな記述を可能にしつつ，3 章で述べたモデルに沿うように設計した言語である．

例として，株式売買システムより各時刻における各企業の株式売買データが XML として逐次流れてくるような状況を想定し，各 XML に対するクエリの例を示しながら記述方法を説明する．

```
Ex1.    return    StockStream4768
        select    *
        from      TestStream
        where     /StockRecord[StockCode=4768]
```

Ex1 は，各 XML から銘柄番号が 4768 であるものを抽出するクエリである．return 節には出力先ストリーム名，select 節には出力する XML のフォーマット，from 節には入力ストリーム名，where 節には XPath に準拠した記法でフィルタリング条件をそれぞれ記述する．QLMXS クエリは基本的にこの 4 節から成る．また，Ex1 は 3 章の Filtering の節で述べたモデルに相当する．

```
Ex2.1.  return    StockStreamOver1000
        select    *
        from      TestStream1
        where     $1N/StockRecord/
                          Quotes/Quote[Price>=1000]

Ex2.2.  return    StockStreamOver1000
        select    *
        from      TestStream2
        where     $1N/StockRecord/
                          Quotes/Quote[Price>=1000]
```

Ex2.1 及び Ex2.2 は別々のストリームから流れてきた XML に対して，株価の値が 1000 以上のものを同じストリーム: StockStreamOver1000 に出力している．この 2 つのクエリは，以下のモデルで表すことができる．

$$(\boldsymbol{s} \mid \boldsymbol{f}) + (\boldsymbol{s}' \mid \boldsymbol{f}') \gg \boldsymbol{s}''$$

```
Ex3.    return    StockStream4768_2
        select    *
        from      StockStream4768
        where     /StockRecord/Quotes/Quote/Price
                  < $N/StockRecord/Quotes/Quote/Price
        setting   $count = 0
        processing $count = $count + 1
        until     $count >= 3
```

Ex3 は，Ex1 で抽出した XML に対し，株価が 3 回上昇した瞬間を検出し，その瞬間の XML を抽出するクエリである．where 節では，$N という接頭辞を用いることで，同ストリーム中で次に来る XML を参照している．setting 節では変数宣言を行うことができ，宣言された変数はクエリ中で利用可能となる．processing 節では where 節の条件成立時に実行される処理を記述することができ，Ex3 中では変数$count を 1 加算している．until 節にはループの終了条件を記述することが可能である．until 節の条件文は where 節の条件判定及び processing 節の処理が行われた後に条件判定が行われ，until 節の終了条件を満たすまで where 節及び processing 節の処理が繰り返される．また，Ex3 のクエリは以下のようなモデルで表現することができる．

$$(\boldsymbol{s} \mid \boldsymbol{f}) * (\boldsymbol{s} \mid \boldsymbol{f}') \gg \boldsymbol{s}'$$

```
Ex4.    return    ActivateStockStream
        select    $2/*
        chaining  TestStream1，TestStream2
        where     $1/StockRecord/Quotes/Quote/Price
                  < $2/StockRecord/Quotes/Quote/Price
        while     10 sec
```

Ex4 は TestStream1 及び TestStream2 のストリームデータの株価を比較し，TestStream2 の方の株価が高い場合に，TestStream2 のデータを別ストリームへ出力するクエリである．TestStream1 におけるイベントの検出が TestStream2 でのイベント検出を呼び起こす形になっている．chaining 節は from 節とは異なり，2 つのストリームを入力ストリームとして指定することが可能になっている．chaining 節で指定しているストリームには順序関係があり，Ex4 では TestStream1 が先，TestStream2 が後となる．where 節の条件文では，接頭辞にそれぞれ $1，$2 を付けることでどのストリームに対する問い合わせ文なのかを示しており，$1 は TestStream1 に，$2 は TestStream2 にそれぞれ対応している．while 節にはストリームデータの到着時間の最大間隔を日単位，時間単位，分単位，秒単位，ミリ秒単位で指定することが可能である．即ち，Ex4 の場合 TestStream1 のデータが到着してから TestStream2 のデータが到着するまでに 10 秒以上経過した場合，条件不成立となる．また，Ex4 は以下のモデルに相当する．

$$(\boldsymbol{s} \mid \boldsymbol{f}) * (\boldsymbol{s}' \mid \boldsymbol{f}') \gg \boldsymbol{s}''$$

```
Ex5.    return    UpperStream
        select    *
        from      ActivateStockStream
        where     $T/StockRecord/Quotes/Quote/Price
                  < $N/StockRecord/Quotes/Quote/Price
        setting   $ts = _$timestamp
        unless    _$timestamp - $ts > 10000
```

Ex5 は，Ex4 で抽出した株データの中から，10 秒間以内に 1 度上昇した瞬間を検出して出力するクエリである．where 節において$T という接頭辞を用いること

で, where 節の条件が不成立だった場合にも終了せず次に来るデータに対する検索結果を待つことが可能となる．また，setting 節及び unless 節において，_$timestamp を利用しており，これによりその処理が行われている時刻のタイムスタンプをミリ秒単位で取得することが可能である．unless 節では，破棄条件を記述することができ，破棄条件を満たした場合には出力は行わず処理が終了する．Ex5 では，setting 節で宣言した$ts と，破棄条件の判定が実行される瞬間の時刻の差が 10000 ミリ秒以上の場合に処理を終了するという意味になる．ここで指定している時間は，処理が始まってから計測される時間のことであり，while 節で指定できる最大到着間隔時間とは異なる．破棄条件が判定されるタイミングは, where, processing, until 節の実行判定後である．また，Ex5 は以下の代数モデルで表現できる．

$$(\boldsymbol{t}(\boldsymbol{s}) \mid \boldsymbol{f}) * (\boldsymbol{s} \mid \boldsymbol{f}') \gg \boldsymbol{s}'$$

```
Ex6.    return      StockStreams[]
        select      *
        from        TestStream
        partition_by /StockStream/StockCode
```

Ex6 はストリーム中のデータを銘柄ごとにストリーム配列へ出力するクエリである．partition_by 節ではストリームを分割するためのキーを指定することができる．partition_by 節でキーを指定した場合，return 節で指定する出力ストリームは配列形式でなければならない．Ex6 における出力ストリームの末尾の[]は配列であることを示す．また，このクエリは 3 章の Partition の節で述べたモデルに相当する．

5. システム概要

処理システムの構成を図 1 に示す．各ストリームから到着した XML はキューに格納され, 到着時のタイムスタンプを保持する. キューは XML が持つタイムスタンプを基にソートを行う. キュー内の XML は先頭から逐次処理されていく．XML のパーサには VTD パーサを用いた．VTD は DOM 同様に XML 全体を読み込んで処理を行うタイプのパーサであり，DOM や SAX よりも高速な処理が可能となっている. QLMXS で記述されたクエリは，クエリ中の XPath の評価結果を遷移シンボルとするオートマトンへと解釈される. XPath の評価は VTD で行い，その結果を基に processing 節の処理や until 節の条件判定などを行う. 即ち, XML に関する処理は VTD が一任しており，オートマトンは VTD から得られた結果・イベントを処理していく. 例として，4 章で示した Ex1 と Ex2 から生成されるオートマトンを図 2 に示す．図 2 に示す通り，システムに登録されたクエリは共通の初期状態を持ち，初期状態から派生する形の 1 つのオートマトンへと変換される．オートマトンでは, 各 XML の処理が開始される度に初期状態

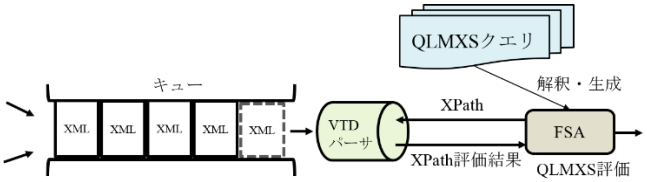

図 1　システム構成

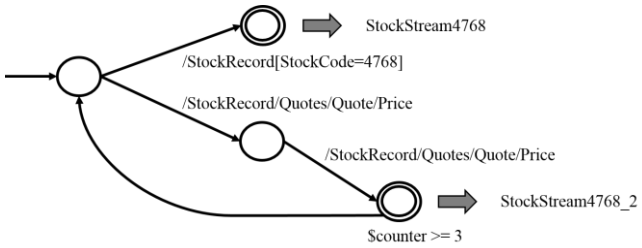

図 2　オートマトン生成例

表 1　実験環境

CPU	Intel® Core™ i7-4790 CPU @ 3.60GHz
Memory	32.0GB
XML	東証株式市場 2010 年 1 月 4 日分

にトークンが生成され，XML の解析が行われていく．トークンは遷移時に，遷移に必要となる XPath を VTD で評価し，その評価結果が返って来た場合，遷移シンボルとして利用する．評価結果が返ってこない場合や，検索対象が存在しない場合にはトークンを破棄する．評価結果が得られた場合は，遷移先での条件判定などに結果を利用し，条件が成立した場合には遷移が確定する．トークンが最終状態に到達した時点で解析終了となり，XML はそれぞれの出力先へ出力される．他のストリームへと出力された場合には，XML の発生元となるストリーム名を変更し，再びキューに挿入される．キューに再び挿入された XML は, 保持するタイムスタンプによってソートされるため，基本的にはキューの先頭に挿入され，すぐに再び処理される．

6. 実験

本システム(以後，VTD+FSA とする)の性能を実験により検証する．実験環境を表 1 に示す．また，比較対象として StAX パーサと VPA を利用したシステム(以後，StAX+VPA とする)[3][4][5]を利用する．

6.1. 基本性能実験

異なるサイズの XML を 8 種類用意し，各 XML に対して単純なフィルタリングクエリを適用したときの処理時間を計測する．実験結果を図 3 に示す．図 3 からも分かる通り, 467KB の時で StAX+VPA が秒間 80 件, VTD+FSA が秒間 555 件程度の性能であった．

6.2. シミュレーション実験

実際の株式市場の過去のデータを用いてシミュレーション実験を行う．

6.2.1. 処理件数による性能の変化

単純な検索クエリ 1 件を適用したときの，本システムの処理速度を計測する．適用するクエリは 4 章で示した Ex1 とし，それぞれの実験結果を図 4 に示す．図 4 に示す通り，StAX+VPA に比べ，VTD+FSA は 10 万件処理時には 1.7 倍，50 万件処理時には 1.9 倍ほどの性能向上が確認でき，秒間約 37000~40000 件，1 件あたり約 0.02~0.03 ミリ秒で処理できた．また，1 日の全株データ約 500 万件を処理させたときは，通信時間を除いて約 133 秒で処理することができた．

6.2.2. クエリ数による性能の変化

システムにクエリを複数登録し，クエリ数によってシステムの性能がどう変化するのかを確かめる．登録するクエリは 6.1 と同じ Ex1 のクエリとし，Ex1 のクエリを複数回登録する．また，処理する XML の件数は 10 万件とする．StAX+VPA でも同様の実験を行い，結果を比較する．ただし，同じクエリを登録したことによる VPA エンジンの最適化は実行せず，単純にクエリを増加させたときの性能を計測する．それぞれの実験結果を図 5 に示す．結果から分かる通り，VTD+FSA の方が処理時間は短いものの，両者共にクエリ数の増加に伴い処理時間も増加し，クエリを 10 件登録したときでは，秒間約 5600 件，1 件あたりの処理時間が約 0.20 ミリ秒程度の性能となっている．

6.2.3. マルチストリーム処理時の性能評価

マルチストリームに跨るような複雑な検索を実行したときの処理時間を確認する．ある会社の株価が下落している一方，もう 1 つの会社の株価が高騰している時のデータを検出することを目的として，以下のクエリを登録し，その際の処理速度を観測する．処理する XML の件数は，1 日分の全データ約 500 万件とする．

```
Exp1.   return   StockStream6701
        select   *
        from     TestStream
        where    /StockRecord[StockCode=6701]

Exp2.   return   StockStream6702
        select   *
        from     TestStream
        where    /StockRecord[StockCode=6702]

Exp3.   return   ExpStockStream
        select   *
        from     StockStream6701
        where    /StockRecord/Quotes/Quote/Price
                 > $N/StockRecord/Quotes/Quote/Price

Exp4.   return   ExpStockStream
        select   *
        from     StockStream6702
        where    /StockRecord/Quotes/Quote/Price
                 < $N/StockRecord/Quotes/Quote/Price
```

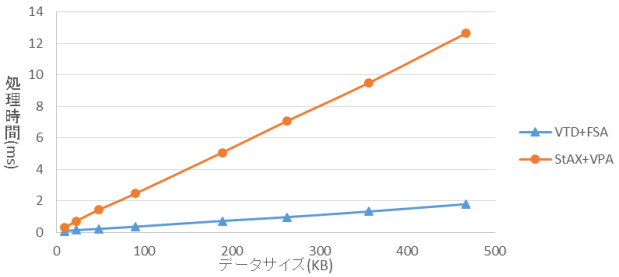

図 3　データサイズの増加に伴う処理時間の推移

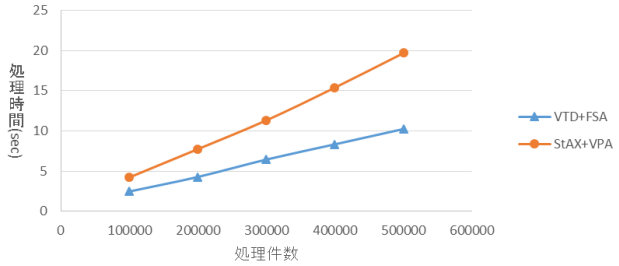

図 4　処理件数の増加に伴う処理時間の推移

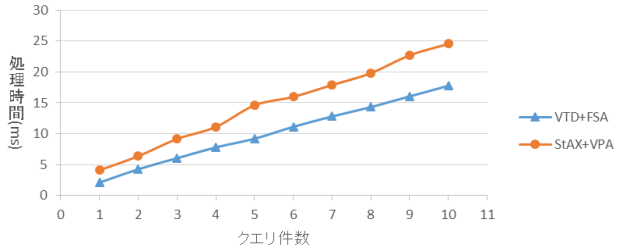

図 5　クエリ件数の増加に伴う処理時間の推移

```
Exp5.   return   ResultStream
        select   *
        from     ExpStockStream
        where    /StockRecord[StockCode=6701]
                 and $N/StockRecord[StockCode=6702]
```

Exp1~5 を実行したところ，通信時間を除いた処理時間は，約 230 秒であり，秒間約 22000 件，1 件あたり約 0.05 ミリ秒，ヒット件数は 1792 件であった．

7. 考察

今回設計した QLMXS では，時系列処理に加え，マルチストリーム処理にも対応できるようにモデルを定義し，それに沿った設計を行った．Mozafari らが提案した XSeq では，時系列処理に対応してはいるものの，多数のデータが内包されている一つの巨大な XML を繰り返し処理することを想定した構成になっている．これに対し，QLMXS では細かい単位の XML を処理することを想定しているが，decomposition や partition モデルの採用により同様の処理の記述は可能になっている．マルチストリーム処理に関しては，異なるストリーム間のイベントの連鎖という形でマルチストリーム処理

を表現した Activation モデルの採用によって, より複雑な検索要求の記述も可能になっている．しかしながら，時系列順序を含んだ表現であるため, 例えば 6.3 の Exp3, 4，5 のクエリを以下の Exp6 のように記述しようとすると，意味が異なってしまう．

```
Exp6.   return     ResultStream
        select     *
        chaining   StockStream6701，StockStream6702
        where      $1/StockRecord/Quotes/Quote/Price
                 > $1N/StockRecord/Quotes/Quote/Price
                   and
                   $2/StockRecord/Quotes/Quote/Price
                 < $2N/StockRecord/Quotes/Quote/Price
```

Exp3，4，5 と Exp6 の検索対象は同じに見えるが，評価順の違いから検索結果が異なる．それぞれが捉えようとしているイベントを，図 6 に示す．Exp3，4，5 で検出できるのが図 6 の左側，Exp6 で検出できるのが右側のグラフで示されるような下落と高騰である．マルチストリーム処理においては，順序が先のストリームに対する処理が優先されるため，クエリ中の XPath の評価順序は，$1，$1N，$2，$2N の順となっている．そのため，Exp6 では StockStream6701 の下落を検出した後に StockStream6702 の高騰を検出し始めるようなオートマトンが生成される．即ち，Exp6 は「下落直後の高騰」というイベントを検出することになる．これに対し Exp3，4，5 では，それぞれの下落と高騰を検出した後に 1 つのストリームにまとめ，銘柄番号が 6701 のものと 6702 が連続しているかどうかを調べる条件判定を行っているため，図 6 の左のグラフに示されるようなイベントを検出することが可能になっている．

また，システムの性能に関しては，各実験の結果より，StAX+VPA に比べ性能が向上していることが確認できた．主な要因としては，ボトルネックの 1 つであった XML のパース処理の高速化であると考えられる．それぞれのパース時間に関しては，10 万件の XML を処理したときでは XML の総パース時間が VTD で 400~500 ミリ秒，StAX で 2100~2200 ミリ秒となっており，VTD の方が速いことが分かる．また，XPath の評価時間に関しては, VTD の方が StAX+VPA よりも速いものの，両者共にクエリの増加に伴い評価時間も増大しており，全体の処理時間の 8 割~9 割を占めている．即ち，システムの処理時間は評価される XPath の数に大きく左右される．マルチストリーム処理の確認実験に関しては，5 つのクエリを登録しているが，6.2 の実験における 5 クエリ登録時に比べ処理時間が短い．これは，6.3 の実験では 1 つの XML に対して最大で 5 つの XPath が適用されるが，全ての XML が各条件を満たすわけではないため，条件を満たすデータの数によって XPath の評価回数が変動することが原因である．実際に 6.3 の実験を 6.2 の実験と同じ 10 万件のデータに対して適用したところ，6.2 の 5 クエリ登録時では XPath の評価回数が 500000 回，処理時間が約 9 秒であったのに対し，6.3 のクエリでは XPath の評価回数が 201398 回，処理時間は約 4.5 秒であった．

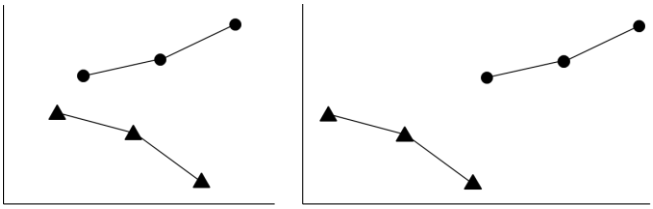

図 6　検索対象の違い

8. まとめ

本論では, マルチストリーム処理のモデル化, モデルを踏まえた XML ストリーム向けの問い合わせ言語の設計及び実装を行った．マルチストリーム処理のモデルに関しては，Activation モデルにおける期間データとの比較が発生する場合を考慮してモデルを別に定義するなど，網羅的なモデルを定義した．また，モデルに沿いつつ，SQL ライクな文法や XPath に準拠した記法を採用することで, よりマルチ XML ストリーム向けの問い合わせ言語: QLMXS を設計することができた．処理システムの実装に関しては，XML パーサに VTD を用いたことにより，より簡潔なシステム構成にすることができ，実験により以前までのシステムに比べて性能の向上を確認した．

今後の課題の一つとして，クエリから生成されるオートマトンの最適化が挙げられる．現状では図 2 で示したように，クエリが追加される毎に初期状態から派生していく形でオートマトンが大きくなっていく．7 章でも述べたように，処理時間の大部分は XPath の評価処理が占めているため，同じ XPath がある場合にはまとめて評価するように最適化することができれば，さらなる性能向上が期待できる．また，パーサと問い合わせ言語の変更による他のデータフォーマットへの対応も今後の課題として行っていきたい．

参考文献

[1] Mozafari B., Zeng K., Zaniolo C., “High-performance complex event processing over xml streams”, Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp.253-264 (2012).

[2] Alur R., Madhusudan P., “Visibly pushdown languages”, Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pp.202-211 (2004).

[3] 松田達希, 内田友樹, 藤田悟, “XMLStream 処理のモデル化と検索言語の設計”, 情報処理学会 第 77 回全国大会, (2015).

[4] 内田友樹, 松田達希, 藤田悟, “XMLStream の時系列イベント処理の性能評価”, 情報処理学会 第 77 回全国大会, (2015).

[5] Tatsuki M., Yuki U., Satoru F., “Method of Complex Event Processing over XML Streams”, ExploreDB '15 Proceedings of the Second International Workshop on Exploratory Search in Databases and the Web, pp.21-26(2015).

[6] 松田達希, 藤田悟, “マルチ XML ストリーム向けの複合イベント処理システム”, 情報処理学会 第 78 回全国大会(2016).