A6-2　外部関数を含むXMLパス表現に対する問合せ処理手法

永井 孝明† 石川 佳治†† 北川 博之††
†筑波大学システム情報工学研究科 ††筑波大学電子・情報工学系
tnagai@kde.is.tsukuba.ac.jp {ishikawa,kitagawa}@is.tsukuba.ac.jp

1 概要

近年，インターネット上における情報交換のためのデータ記述言語として， XML (Extensible Markup Language) が注目されている．XML の記述能力の高さから様々な分野でデータ記述を XML で行ない，インターネットを介して情報交換をするための技術開発が行なわれている．その 1 つに地理情報を XML で記述するための手法がある．一方，大量の XML データの効率的な管理のために XML サーバが用いられており，一般に XML データの検索には XPath 言語 [1] のサブセット等がサポートされている．

今後インターネットにおける情報交換はますます盛んになると考えられ，それに伴って，地理情報を XML で記述したデータのような，さまざまな分野におけるデータが増大すると考えられる．そして，それらのデータを XML サーバに格納して効率的に管理する場合，通常の構造化文書に対する検索だけではなく，そのデータ固有の意味に応じた処理が必要になってくる．しかし，XPath 言語の サブセットなどではそのようなデータ固有の意味に応じた問合せを記述し，処理することは困難である．よって，本研究では XPath に汎用プログラミング言語により記述可能な外部関数を導入して拡張し，その外部関数を含んだ問合せを効率的に処理するための手法を提案する．

2 研究の背景と目的

2.1 XML データに対する問合せ

以下では XML 形式で記述された様々なドメインのデータに対する問合せ処理について述べる．対象とするデータの例として，G-XML[2] のような空間的なオブジェクトを表現した地図データを用いる．図 1 にその問合せ対象のデータ例を示す．最上位要素である GeoSpace 要素は，地図データに記述されている各地物データを表す GeoFeature 要素の並びから構成され，GeoFeature は category 属性，Property 要素，Geometry 要素を持つ．category 属性は各地物の種類 (領域，建物など) を表し，Property 要素は，地物オブジェクトの名前を表す name 要素，地物のタイプ (学校，店などの各地物の種類よりも細かい分類) を表す type 要素，URL を表す url 要素などから構成される．Geometry 要素は，点や多角形などの幾何形状を表す Point, Polygon 要素などから構成され，地物データの幾何形状を表現する．各幾何形状の頂点の位置は，Coordinates 要素に記述された 2 次元の座標値によって決定される．図 1 にはつくば市の領域と筑波大学のみが明示されているが，以降の説明の際にはつくば市内の地図データが記述されているものとする．

```
<GeoSpace>
  <GeoFeature category="area">
    <Property>
      <name>Tsukuba City</name>
      <type>City</type>
      <url>http://...</url>
    </Property>
    <Geometry>
      <Polygon>
        <Coordinates>16800000,14800000,...</Coordinates>
      </Polygon>
    </Geometry>
  </GeoFeature>
  ...
  <GeoFeature category="building">
    <Property>
      <name>University of Tsukuba</name>
      <type>University</type>
      <url>http://...</url>
    </Property>
    <Geometry>
      <Polygon>
        <Coordinates>17400000,19400000,...</Coordinates>
      </Polygon>
    </Geometry>
  </GeoFeature>
  ...
</GeoSpace>
```

図 1: 地図データの XML 表現の例

図 1 の XML 文書に対して，以下の **Q1** の問合せをする場合を考える．

Q1: つくば市の領域を表す多角形オブジェクトの情報を抽出せよ．

この問合せの場合，“領域を表す地物であり，かつ名前が‘つくば市’であるような多角形” という条件を指定すれば良いので，文書の構造やテキストに基づいた条件を指定することによって記述できる．つまり，XML 文書中の地物の種類を表す属性 category の値が，領域を表す値 “ area ” であり，かつその地物の名前を表す要素 name の値がつくば市を表す値 “Tsukuba City” であるような多角形 Polygon 要素を指定すれば良いことになる．

前述の XPath を用いてこの問合せ **Q1** を記述すると，以下のようになる．

```
/GeoSpace/GeoFeature[@category = "area"]
  [Property/name = "Tsukuba City"]/Geometry/Polygon
```

Query Processing Scheme for XML Path Expressions Including User-defined External Functions
Takaaki Nagai†, Yoshiharu Ishikawa††, and Hiroyuki Kitagawa††
† Graduate School of Sys. and Info. Eng., Univ. of Tsukuba
†† Inst. of Inf. Sci. and Elec., Univ. of Tsukuba

このようにXPathなどを用いた既存のXML問合せ言語では，要素内容や要素の親子関係などの，テキストや構造に基づいた条件を指定することによって **Q1** のような問合せを記述することが出来る．

一方，以下の問合せを考える．

Q2: 与えられた多角形 (Polygon) 領域内に含まれる学校の名前をすべて列挙せよ．

この問合せの検索条件として与えられる領域としては，つくば市のある地区を表すような多角形領域が例として考えられる．例では学校に関する情報も多角形領域として表されていることから，この問合せは空間問合せの一種である範囲問合せ (多角形間の包含関係に基づく) を含んでおり，XML 文書を単なる構造を持ったテキストと捉える従来の XML 問合せ言語ではその記述は困難である．すなわち，問合せ **Q2** を処理するためには，XML データ中の `Point` や `Polygon` といった空間データを表す文書要素を空間データとして認識し，範囲問合せの条件を判定する述語を構造化文書の検索において統合利用出来ることが必要となる．

2.2 本研究のアプローチ

XMLが今後データ記述言語として広く利用されるに従い，様々なドメインでデータ記述形式として利用されると考えられ，単なるテキストを記述したデータではなく各ドメインでの固有のセマンティクスを持ったデータが増大してくると考えられる．よってそれらのデータを XML サーバなどを用いて効率良く管理する際には，それらのドメイン固有のセマンティクスに基づいた問合せが必要になってくる．しかし，現在 XML サーバの問合せ言語として用いられている XPath のサブセットなどでは，前述のようにテキストと構造に基づいた問合せしか行えず，固有のセマンティクスに基づいた問合せは困難である．よって本研究では，XML データに対する問合せ言語として広く用いられている XPath に対して外部関数を導入することにより，それらの問題を解決する．XML 問合せ言語への外部関数処理機能の導入に関して，これまで我々の研究グループでは，外部関数による拡張機能を導入した X^2QL 言語の開発とその G-XML 問合せ処理についての研究を行なってきた [3, 4]．本稿で提案するアプローチは，これらの研究と関連しているが，XPath のサブセットをベースにユーザ定義の外部関数の支援機能の実現を図るという点が特徴となっている．

一方，XML データに対する問合せを効率良く処理する手法として，構造化文書に対する検索モデルが利用されている．その中でも特に後述する Proximal Nodes モデル [5] は XML データに対する問合せの多くを表現できるという表現能力と，索引などを用いて効率的に処理することができるという効率の面でバランスが良いと考えられる．特に地図データに対する問合せ要求の場合には，空間問合せ等が含まれるため，空間索引が利用出来れば高速に処理することができる．

以上のことから本研究では，XPath に外部関数を導入した問合せを効率良く処理するための処理系の開発を目的とする．具体的には，外部関数を含んだ XPath を Proximal Nodes モデルに変換して処理するが，Proximal Nodes モデルには外部関数を含んだ問合せを処理することができないため，外部関数の概念を導入する．本研究では主に，外部関数を含んだ XPath から Proximal Nodes モデルへの変換規則を提案する．

3 XPath への外部関数導入

3.1 XPath

ここでは XPath [1] について説明する．XPath は木構造の XML データ中の特定の部分をパス式によって指定するための言語であり，XML データをツリー構造を持ったノードの集合として表現する．各ノードには，それぞれ型，名前，値がある．ノードの型には，ルート，要素，属性，テキストなどがあり，名前には要素名や属性名など，値には属性値やテキストなどを持ちうる．また，ノード間の関係としては親子関係と従属関係があり，要素同士の場合は親子関係，要素と属性の場合は従属関係となる．

次に，XPath の構文について説明する．XPath の構文は “/” で区切られたロケーションステップの連なりから構成されており，各ロケーションステップは以下のように，軸，ノードテスト，0 以上の述語から構成されている．

軸::ノードテスト [述語]...[述語]

軸は，評価の起点となるノード（コンテクストノード）から XML のツリーを探索する方向を指定する．例えば，`child`, `parent`, `attribute` などの軸は，各々ツリーをコンテクストノードから子ノード，親ノード，属性ノードの方向へ探索する．

ノードテストは，軸で指定された方向の中で特定のノードを指定するために使用され，ノードの名前などを指定する．述語は軸とノードテストによってノード集合が選択された後，さらにそのノード集合に対して条件を指定するときに用い，要素もしくは属性の存在やその値による条件付けなどを記述する．

以下に図 1 のデータに対する XPath の記述例を示す．

```
/child::GeoSpace/child::GeoFeature
          [attribute::category ="area"]
```

この問合せは，ルートノードを起点ノードとして起点ノードの子のノードのうち名前が GeoSpace であるノードを選択し，次にそのノードの子のノードのうち名前が GeoFeature であるノードを選択し，そのノードのうち従属している属性ノードの値が “area” であるノードのみを結果として選択する．つまり，ルートの子要素の `GeoSpace` 要素の子要素の `GeoFeature`

要素のうち，属性値が “area” である GeoFeature 要素を抽出する．また XPath には省略記法があり，ロケーションパスにノードテストしか記述しない場合 “child::ノードテスト” を意味し，“@ノードテスト” は “attribute::ノードテスト” を意味する．よって，XPath の省略記法を用いるとこの問合せは以下のように略記できる．

```
/GeoSpace/GeoFeature[@category = "area"]
```

3.2 外部関数の導入方式

現在策定中の XPath 2.0 [6] は XQuery [7] に包含されることから，XQuery の実行環境における XPath は，XQuery で定義されたユーザ定義関数を利用することが可能となると考えられる．しかし，XQuery でユーザが定義可能な関数は XQuery の構文で記述可能な処理に限られているため，包含関係の判定のような複雑な処理を XQuery のユーザ定義関数で記述することは困難である．また，XQuery 以外の言語で記述された外部関数の利用に関しては，現在は議論の段階にとどまっている．よって，本研究ではユーザ定義関数として外部関数も導入可能にし，この外部関数をユーザが記述，定義することによって，ユーザ定義関数を与えるアプローチをとる．

ここでは XPath のユーザ定義関数として外部関数を導入可能にする手法について説明する．外部関数を利用するためには，外部関数の定義と実装を与える必要がある．外部関数の実装は，Java 等の汎用プログラミング言語によって外部プログラムとして与える．また，外部関数定義とは，外部プログラムとして与えられる外部関数実装を問合せ中で利用可能にするための情報記述を指し，図 2 にその定義構文を示す．

```
default namespace "名前空間を含む URI"
namespace 名前空間名 = "名前空間を含む URI"
schema "[スキーマを含む URI リスト]"
declare-func 型名 関数名 ([ 引数リスト ])
defined-by "関数実装を含む URI"

スキーマを含む URI リスト ::= スキーマを含む URI [, スキーマを含む URI リスト]*
引数リスト ::= 型名 引数名 [, 型名 引数名 ]*
```

図 2: 外部関数定義

namespace 節では，外部関数の利用する名前空間とその名前空間が記述されている URI を宣言する．default namespace 節は，名前空間の宣言が無い場合の名前空間の場所を宣言する．schema 節は，外部関数の引数として与えられる XML データのスキーマを宣言する．declare-func 節では，外部関数の返値のデータ型，関数名，引数名およびそのデータ型を宣言する．defined-by 節ではその関数の実装を含む URI を宣言する．XML データ中の XML フラグメントが表す多角形が文字列で与えられた多角形領域内に含まれるかどうかを判定する外部関数 geo:contained() の定義を以下に示す．

```
default namespace "http://www.w3.org/2001/XMLSchema"
namespace geo = "http://www.sample.co.jp/geo"
schema "http://www.sample.co.jp/geo
        http://www.sample.co.jp/schemas/geo.xsd"
declare-func boolean geo:contained(geo:polygonType $poly,
                                   string $region)
defined-by "http://www.sample.co.jp/geo#contained"
```

図 3: 外部関数の宣言の例

この外部関数定義は，名前空間 geo の外部関数 contained() が polygonType 型，および string 型の XML データを引数とし，boolean 型を返す関数であることなどを宣言している．

このようにして外部関数を導入可能とすることにより，先に示した問合せ例 **Q2** は XPath の構文内から関数を呼び出すことによって以下のように記述される．

```
/GeoSpace/GeoFeature[Property/type = "school"]
 /Geometry[geo:contained(Polygon, "16300000,
                                   25500000,...")]
  /Property/name
```

4 Proximal Nodes モデルへの外部関数導入

本研究では，前節のように外部関数を導入可能にした XPath 記述を，Proximal Nodes モデルをベースとして拡張した検索モデルに変換することによって問合せ処理を実現する．XPath 記述をその検索モデルに変換し，Proximal Nodes モデルベースの問合せとして解釈することで，索引を使ったボトムアップの検索が可能になり効率的な検索が行える．また外部関数を用いた空間問合せに対しても，空間索引を利用することにより効率的な検索が可能である．

変換対象とする XPath については，提案する検索モデルに変換可能な XPath のサブセットとし，その具体的な内容については 5 節で述べる．

4.1 Proximal Nodes モデル

構造化文書の検索モデルに関しては，XML の出現以前から，SGML などを対象として多くの研究が行なわれてきた [8]．構造化文書に対する検索モデルは，文書の内容と構造に基づく検索条件を同時に指定して検索できる点を特徴としており，XML の分野における XPath などの言語と対応づけることができる．一般的にいって，既存の構造化文書検索モデルは，XQuery などの XML 問合せ言語に比べ表現能力の面で劣るが，より効率的な処理が可能であり，高レベルな XML 問合せ言語処理系を実現する際の検索サブシステムとしても利用可能と考えられる．

本研究では，外部関数を含んだ XPath 記述を Proximal Nodes モデルをベースとした検索モデルに変換して処理を行う．*Proximal Nodes* モデル [5] は，構造化文書に対する検索モデルの中でも，表現能力と効率の面でバランスが良いとされるモデルである．要素や属性，テキスト等の指定によって，ある条件を持つ

XML データ を抽出するといった，一般的な問合せを記述可能な表現能力を持ち，かつ索引の利用によって効率良く検索を行うことができる．

Proximal Nodes モデルの特徴は効率を意識した演算体系にあり，転置ファイルなどの索引に基づいて検索式の評価に必要な部分的な文書要素が特定できると，それをもとにボトムアップに検索結果が構成できる点にある．また，演算子の体系が整理されていることから，最適化などの効率化が行える余地が存在する．

Proximal Nodes モデルでは，構造化文書をノード (以降 **PN ノード**と呼ぶ) で表現する．PN ノードには構造化文書上のテキスト領域である**セグメント**が対応づけられる．上位の PN ノードのセグメントは，文書構造に基づいて，子孫の PN ノードのセグメントを包含し，テキスト階層を構成する．構造化文書の文書要素 (例: `Polygon`) については，その出現に応じて対応するセグメントが文書中に複数個存在し得ることになる．また，文書中に現れるテキストのパターン (例: "`Tsukuba`") に対しても，その出現箇所に応じてセグメントの集合が対応づけられる．

Proximal Nodes モデルには，以下の 3 種類の演算子が存在する．

1) テキストパターン照合演算子: 実際には，テキストパターン照合の言語は Proximal Nodes モデル自体では定義されておらず，想定されるパターンが与えられたときセグメントの集合を返すようなサブ言語を，実装環境に応じて与えるものとされている．ここでは，たとえば "`Tsukuba`" というパターンに対し対応するセグメントを返すような，基本的なサブ言語が存在すると想定する．

2) 文書構造に基づく検索のための演算子: たとえば `chapter` などの文書要素名の指定がこれに相当する．評価結果として，その文書要素名に対応するセグメントの集合が得られる．

3) 部分検索結果の統合: 表 1 に，この種の演算子の例を示す．[5] では他の演算子も定義されているが，ここでは，後で利用する演算子のみに限定して示している．図中の P, Q は，それぞれセグメントの集合を表している．

Proximal Nodes モデルの特徴は，上記 1), 2) の演算子集合により，まず検索対象となる部分文書 (セグメント) の集合を確定し，次いで 3) の演算子集合を用いてボトムアップに検索結果を構築する点にある．1), 2) の処理では通常，テキスト索引などが利用されるため，文書全体にアクセスせず必要な箇所の特定が可能となる．3) の統合処理を行う演算子は，セグメントが近隣に存在するノード間の処理のみを扱うように限定されており，評価の際の効率化が図られている．

以下に，簡単な Proximal Nodes モデルの例を示す．

演算子	意味 (例)
P **in** Q	Q 中のあるノードに含まれるような P 中のノードの集合 (`figure` **in** `section`)
P **with** Q	Q 中のノードを含むような P 中のノードの集合 (`section` **with** `"information"`)
$[s]$ P **child** Q	Q のあるノードの子であるような P のノードの集合．$[s]$ は結果に含めるべきノードの位置の範囲で，デフォルトは `[0..last-1]`．(`[3..5]` `title` **child** `chapter`)
P **parent** Q	テキスト階層中で Q のあるノードの親であるような P のノードの集合．
P **after** Q	P のセグメントで，その開始が Q のあるセグメントの終了より後であるものの集合．(`list` **after** `figure`)
P **before** Q	P のセグメントで，その終了が Q のあるセグメントの開始より前であるものの集合．
$P + Q$	P と Q の和集合 (`figure` + `list`)
$P - Q$	P と Q の差集合 (`chapter` − (`chapter` **with** `figure`))
P **same** Q	P 中のセグメントで，Q 中のあるセグメントと一致するものの集合．

表 1: Proximal Nodes モデルの演算子

```
Polygon child Geometry
```

`Polygon` と文書要素名を指定することにより，`Polygon` という文書要素名を持つセグメントの集合が得られる．`Geometry` も同様にして，`Geometry` という文書要素名のセグメントの集合が得られる．演算子 **child** はこれら 2 つの結果に対して演算を行い，結果としてセグメントの情報を基に `Geometry` を子ノードに持つ `Polygon` 要素のセグメントの集合を返す．

4.2 外部関数の導入による拡張

本研究で提案する手法では外部関数を含む XPath 記述を Proximal Nodes モデルをベースとした検索モデルに変換して処理を行うが，Proximal Nodes モデルには外部関数の概念がないため，外部関数を導入可能にする拡張を行い，外部関数を含んだ XPath 記述の変換を可能にする．

Proximal Nodes モデルの演算子に加えて，新たに `ex-funcall()` という外部関数を呼出すための演算子を導入する．この演算子の定義を以下に示す．

```
ex-funcall("関数名", [引数リスト])
引数リスト ::= "引数名" [, "引数名" ]*
```

演算子の第 1 引数には外部関数名を指定し，第 2 引数以降に外部関数の引数を指定する．この外部関数の呼出しは外部関数モジュールに渡され，検索処理が行われる．外部関数評価モジュールの役割は，指定された外部関数に対する実装をもとに，指定された文書要素に対して指定された条件が成立するかを判定し，条件が成立した場合にはそのセグメントを返すことになる．最終的には `ex-funcall()` を呼出した結果としてセグメントの集合が得られるため，Proximal Nodes モ

デルのセマンティクスと同様にこの結果をもとにボトムアップに問合せ実行処理が行われる．

以下に ex-funcall() を含む Proximal Nodes モデルの記述例を示す．

```
type with ex-funcall("geo:contained",
      "Polygon", "16300000, 25500000,...")
```

この例は，先に示した geo:contained という外部関数を含む例であり，ex-funcall 演算子は第 1 引数に指定された geo:contained という名前の外部関数を呼出す．この外部関数は，第 2 引数に指定された文書要素のうち，第 3 引数に文字列として指定された多角形の領域に包含される文書要素のセグメントを返す関数であるため，結果としては第 3 引数に指定された多角形領域に空間的に包含される Polygon 要素のセグメントが得られる．一方，type によって type 要素のセグメントの集合が得られる．これらに対して with 演算子を適用することによって，指定された多角形領域に包含される Polygon 要素を子孫として含む type 要素のセグメントの集合が得られる．

5 問合せ変換方式

本節では，外部関数を含む XPath 式の問合せ処理方式について述べる．まず，5.1 節では，XPath 式を Proximal Nodes 式に変換する変換方式について述べる．

5.1 XPath 式から Proximal Nodes 式への変換

変換のアプローチ 本節では，与えられた XPath 式を Proximal Nodes 式に変換し，トップダウンの処理を要求する XPath 問合せをボトムアップな Proximal Nodes の問合せに変換する方式を示す．XPath の問合せは巡行 (navigation) に基づく処理によって実行するものと定義されており，XML 文書の木構造をたどりながら処理を適用することが実行のモデルとなっている．しかし，このようなアプローチでは，索引の利用や集合指向 (set-oriented) の効率的な処理を適用できない．そこで本研究では，Proximal Nodes モデルに基づくボトムアップな処理に変換することにより，効率的な問合せ処理の実現をはかる．

XPath と Proximal Nodes モデルにはセマンティクスの違いが存在するため，XPath のすべての構文を Proximal Nodes に変換することはできない．しかし，以下で示す変換ルールは，XPath における検索で用いられる主要な問合せパターンをカバーしている．以下で述べる変換ルールは Philip Wadler による XPath のセマンティクス記述 [9] を参考に Proximal Nodes モデルに対する独自の拡張を図ったものである．XPath については，その仕様は自然言語による文書として定義されており [1]，フォーマルな意味での定義としては十分でない．上記のセマンティクス記述は，XPath の一部の構文について，表示的意味論 [10] によりそのセマンティクスを与えている．本研究ではこのセマンティクス記述を拡張することにより，XPath 式の Proximal Nodes 式への変換ルールを記述する．これらのルールで変換を定義する構文が，本論文で支援を想定しているサブセットの XPath 言語である．

XPath パターンの変換 まず，図 4 に，XPath の与えられたパターンに対する変換ルールを示す．この変換ルールは本節における変換処理の中心となるものである．ルールの表記において，E を XPath 式としたとき，$[\![E]\!]$ はその変換結果である Proximal Nodes 式を示している．セマンティック関数 $\mathcal{S}$ は，軸と XPath のパターンを第 1，第 2 引数とする点は共通であるが，Wadler のセマンティクス関数と異なり，第 3 引数は構造ノード名 ($SNodeName$) である．構造ノード名は Proximal Nodes モデル上の概念であり，XML における要素名，属性名に対応しており，スキーマで定義された文書構造に応じて構造ノード名間に親子関係，先祖・子孫関係が定義される．変換ルールの出力は Proximal Nodes モデルの式であるが，Proximal Nodes 式を評価するとセグメントの集合が得られることから，出力の型は $Set(Segment)$ となる．

$$\mathcal{S} \;:\; Axis \to Pattern \to SNodeName \to Set(Segment)$$

$$\mathcal{S}^a[\![p_1|p_2]\!]x = \mathcal{S}^a[\![p_1]\!]x + \mathcal{S}^a[\![p_2]\!]x \quad (1)$$

$$\mathcal{S}^a[\![/p]\!]x = \mathcal{S}^a[\![p]\!]\texttt{Root} \quad (2)$$

$$\mathcal{S}^a[\![\texttt{text}()]\!]x = \texttt{Text}\ \mathcal{A}[\![a]\!]\ x \quad (3)$$

$$\mathcal{S}^a[\![p_1/p_2]\!]x = \mathcal{S}^a[\![p_2]\!]x_1\ \textbf{child}\ x_1,\ \text{ただし，}x_1 = \mathcal{S}^a[\![p_1]\!]x \quad (4)$$

$$\mathcal{S}^a[\![a_1 :: p_1]\!]x = \mathcal{S}^{a_1}[\![p_1]\!]x \quad (5)$$

$$\mathcal{S}^a[\![n]\!]x = \begin{cases} n\ \mathcal{A}[\![a]\!]\ x & x \text{ が } \mathcal{P}[\![a]\!] \text{ を } \mathcal{A}[\![a]\!] \text{ として} \\ & \text{名前 } n \text{ で有するとき} \\ error & \text{それ以外} \end{cases} \quad (6)$$

$$\mathcal{S}^a[\![*]\!]x = \mathcal{S}^a[\![n_1]\!]x + \cdots + \mathcal{S}^a[\![n_m]\!]x,\ n_1,\ldots,n_m \text{ は } x \text{ の } \mathcal{A}[\![a]\!] \text{ で，ノード型が } \mathcal{P}[\![a]\!] \text{ であるものすべて} \quad (7)$$

$$\mathcal{S}^a[\![p[q]]\!]x = [q]\,\mathcal{S}^a[\![p]\!]x,\quad \text{ただし，}q \text{ は数値} \quad (8)$$

$$\mathcal{S}^a[\![p[q]]\!]x = x_1\ \textbf{with}\ \mathcal{Q}^a[\![q]\!]x_1,\ \text{ただし，}q \text{ が数値以外を返す式のとき．} x_1 = \mathcal{S}^a[\![p]\!]x \text{ とする．} \quad (9)$$

$$\mathcal{S}^a[\![@n]\!]x = \mathcal{S}^a[\![\texttt{attribute} :: n]\!]x \quad (10)$$

$$\mathcal{S}^a[\![@*]\!]x = \mathcal{S}^a[\![@n_1]\!]x + \cdots + \mathcal{S}^a[\![@n_m]\!]x,\ @n_1,\ldots,@n_m \text{ は } x \text{ の } \mathcal{A}[\![a]\!] \text{ で，属性であるものすべて} \quad (11)$$

図 4: XPath パターンに対する変換

図 4 の変換ルールについて説明を行う．ルール (1) は論理和表現に関する変換ルールであり，単純に分解した結果を評価した 2 つのセグメント集合の和集合をとる操作として実現される．ルール (2) はルート直下

のパターンに関する変換ルールを示している．ここで現れる Root は，ルート要素を表す特殊な構造ノード名である．ルール (3) における Text は XML のテキストノードに対して仮想的に存在する構造ノード名であるとする．

ルール (4) は連接するパスに関する変換ルールであり，Proximal Nodes モデルの **child** 演算子を用いた問合せに変換される．パスの長さが 3 個以上の場合，たとえば A/B/C というパスに対しては，$p_1 = \texttt{A/B}$, $p_2 = \texttt{C}$ というパターン照合と $p_1 = \texttt{A}$, $p_2 = \texttt{B/C}$ というパターン照合が可能であるが，この場合にはどちらも正しいマッチングであるため，それぞれに対する変換が可能である．このため，変換規則を可能な限り適用していくと，枝分かれして多くの変換結果が得られることになる．なお，$\mathcal{S}^a[\![p_2]\!]x_1$ の評価結果であるそれぞれのセグメント集合が $\mathcal{S}^a[\![p_1]\!]x$ の評価結果であるセグメント集合のいずれかに含まれることがスキーマ上から明らかな場合，

$$\mathcal{S}^a[\![p_1/p_2]\!]x = \mathcal{S}^a[\![p_2]\!]x_1 \quad \text{ただし } x_1 = \mathcal{S}^a[\![p_1]\!]x$$

と簡略化してよい．これはたとえば，$p_1 = \texttt{GeoSpace}$, $p_2 = \texttt{GeoFeature}$ という文書要素に対し GeoFeature/GeoSpace というパターンが与えられたときに，GeoFeature が GeoSpace の子要素としてしか現れないことがスキーマ上保障されている場合に相当する．

次にルール (6) について説明する．詳細は図 5 に示すが，このルールに現れる $\mathcal{A}$ とは，軸をとり，Proximal Nodes モデルの文書構造に関する演算子名を返す補助関数であり，一方，$\mathcal{P}$ は，軸をとり，ノードの型を返す補助関数である．このルールの意味は，たとえば $\mathcal{S}^{\texttt{child}}[\![\texttt{Person}]\!]\texttt{Root}$ が変換対象として与えられた際，

> スキーマ上で，Root が $\mathcal{P}[\![\texttt{child}]\!] = \texttt{Element}$ を $\mathcal{A}[\![\texttt{child}]\!] = \texttt{child}$ として，名前 Person で有しているならば，Person $\mathcal{A}[\![\texttt{child}]\!]$ Root を Person **child** Root に変換する．

すなわち，

> スキーマ上で，Root が Person という名の子要素を有するならば，Person $\mathcal{A}[\![\texttt{child}]\!]$ Root を Person **child** Root に変換する．

と解釈する．なお，ルール (6) において，n に対応するセグメント集合の各要素が x に対応するセグメント集合のいずれかの要素に対し関係 $n\ \mathcal{A}[\![a]\!]\ x$ を有することがスキーマ上から明らかな場合，

$$\mathcal{S}^a[\![n]\!]x = n$$

と簡略化してよい．たとえば，上の例で，Person が Root の下にしか現れないことがスキーマ上保障されている場合，$\mathcal{S}^{\texttt{child}}[\![\texttt{Person}]\!]\texttt{Root} = \texttt{Person}$ と変形できる．

$$\begin{aligned}
\mathcal{A} &: \mathit{Axis} \rightarrow \mathit{Operator} \\
\mathcal{A}[\![\texttt{child}]\!] &= \textbf{child} \\
\mathcal{A}[\![\texttt{parent}]\!] &= \textbf{parent} \\
\mathcal{A}[\![\texttt{descendant}]\!] &= \textbf{in} \\
\mathcal{A}[\![\texttt{ancestor}]\!] &= \textbf{with} \\
\mathcal{A}[\![\texttt{attribute}]\!] &= \textbf{child}
\end{aligned}$$

$$\begin{aligned}
\mathcal{P} &: \mathit{Axis} \rightarrow \mathit{Nodetype} \\
\mathcal{P}[\![\texttt{child}]\!] &= \texttt{Element} \\
\mathcal{P}[\![\texttt{parent}]\!] &= \texttt{Element} \\
\mathcal{P}[\![\texttt{descedant}]\!] &= \texttt{Element} \\
\mathcal{P}[\![\texttt{ancestor}]\!] &= \texttt{Element} \\
\mathcal{P}[\![\texttt{attribute}]\!] &= \texttt{Attribute}
\end{aligned}$$

図 5: 変換ルールの補助関数

ルール (8) とルール (9) は同じ XPath パターンを対象としているが，条件節中に現れる q が数値をとるか，それ以外の値を取るかで分岐することになり，XPath のセマンティクスの不透明な部分が影響している部分である．まず前者については，数値 q を指定された場合には，q 番目の子ノードを選択することを指定していることから，Proximal Nodes モデルの "[]" 演算子に変換する．なお，XPath では q として単なる数値ではなく数値を返す任意の式を記述可能であるが，これについては本研究では今後の課題としている．一方，後者については，後述する変換ルール $\mathcal{Q}$ を用いる．$\mathcal{Q}$ は，軸，条件，構造ノード名をとり，構造ノード名に対し条件を適用した際に得られるセグメントの集合を返す関数である．Proximal Nodes モデルの **with** 演算子を用いることで，この結果のセグメント集合によるフィルタリングが可能となる．

条件部の変換　次いで，条件部に関する変換ルールを図 6 に示す．ルール (12)，ルール (13) は，それぞれ論理和，論理積に関する変換ルールである．ルール (14), (15), (16) は，同じパターンを共有しているが，p として与えられた内容がビルトイン関数であるか，外部関数であるか，それ以外か (たとえば要素名) に応じて分岐する．ルール (14) の p がビルトイン関数の場合には，builtin-funcall 関数を実行時に呼び出すように設定する．その引数は関数名 p とセグメント集合 x である．ルール (15) の p が外部関数名であるとき，まず *get_datatype* 関数により，セグメント集合 x がスキーマ上でとるデータ型を取得するものとする．データ型の情報を入手する理由は，外部関数 p との型の整合性を調べるためである．p が関数名でない場合，ルール (16) を適用する．これは，セグメント集合 x の子に p という名の子があるかどうかを調べるというルールとなる．

$\mathcal{Q}$: $Axis \rightarrow Qualifier \rightarrow SNodeName \rightarrow Set(Segment)$

$\mathcal{Q}^a[\![q_1 \texttt{ or } q_2]\!]x = \mathcal{Q}^a[\![q_1]\!]x + \mathcal{Q}^a[\![q_2]\!]x$ (12)

$\mathcal{Q}^a[\![q_1 \texttt{ and } q_2]\!]x = \mathcal{Q}^a[\![q_1]\!]x \textbf{ is } \mathcal{Q}^a[\![q_2]\!]x$ (13)

$\mathcal{Q}^a[\![p]\!]x =$
if p は XPath のビルトイン関数の呼出し **then**
  `builtin-funcall`(p, x) (14)
else if p は外部関数の呼出し **then**
  $type = get\_datatype(x)$
  // p と $type$ の整合性をチェック
  $args = exfun\_parse(p)$
  // 外部関数呼出し p から引数リスト作成
  `ex-funcall`$(x, args)$ (15)
else // p が要素名などの場合
  $\mathcal{S}^{\texttt{child}}[\![p]\!]x$ (16)

図 6: 条件部に関する変換ルール

条件部が比較演算子を含む場合，すなわち，$\mathcal{Q}^a[\![q_1\ op\ q_2]\!]x$ ($op \in \{=, !=, <, >, \leq, \geq\}$) については，XPath の仕様により照合条件が複雑になる．よって，図 6 とは別に詳しく述べる．以下のように場合分けして説明する．

1. q_1 がノード集合に対応し，q_2 が文字列の場合:

   (a) $op =$ '=' のとき:

   $$x \textbf{ with } (\mathcal{S}^a[\![q_1]\!]x \textbf{ same } q_2) \quad (17)$$

   と変換する．たとえば，

   $$\begin{aligned} & \mathcal{Q}^a[\![\texttt{author/lname = "Tanaka"}]\!]\texttt{Book} \\ \rightarrow\ & \texttt{Book with } (\mathcal{S}^a[\![\texttt{author/name}]\!]\texttt{Book} \\ & \textbf{same } \texttt{"Tanaka"}) \end{aligned}$$

   となる．このルールは，一つの `Book` 要素の下に `author` 要素が複数現れる場合にも対応している．

   (b) $op = \{!=, <, >, \leq, \geq\}$ のとき:

   $$x \textbf{ with comp\_as\_str}(\mathcal{S}^a[\![q_1]\!]x, q_2, op) \quad (18)$$

   という関数呼出しを含む文に変換する．この関数 **comp_as_str**(S, str, op) は，実行時に呼び出される．与えられたセグメント集合 S の各要素セグメント $seg \in S$ に対応する文字列を索引などを利用して入手し，与えられた文字列 str と op の関係が成立するかを調べる．成立した場合には，seg を結果に含める．最終的に，条件を満たすセグメントの集合を返す．

   なお，x のあるセグメントに対する q_1 が表すセグメントがただひとつに限定できる場合には，

   $$x - \mathcal{Q}^a[\![q_1 = q_2]\!]x \quad (19)$$

   と変形できる．たとえばこれは，上の例で `Book/author/lname` というパスが一つの `Book` 要素に対しただ一つに確定できるような場合に相当し，

   $$\begin{aligned} & \mathcal{Q}^a[\![\texttt{author/lname != "Tanaka"}]\!]\texttt{Book} \\ \rightarrow\ & \texttt{Book} - \mathcal{Q}^a[\![\texttt{author/lname = "Tanaka"}]\!]\texttt{Book} \end{aligned}$$

   と変形できる．

   q_1, q_2 の立場を入れ替え，$\mathcal{S}^a[\![q_1]\!]x$ が文字列で $\mathcal{S}^a[\![q_2]\!]x$ がノード集合の場合も同様である．

2. $\mathcal{S}^a[\![q_1]\!]x$ がノード集合に対応し，$\mathcal{S}^a[\![q_2]\!]x$ が数値の場合:

   $$x \textbf{ with comp\_as\_num}(\mathcal{S}^a[\![q_1]\!]x, q_2, op) \quad (20)$$

   という関数呼出しを含む文に変換する．**comp_as_num**(S, num, op) は，実行時に呼び出される．与えられたセグメント集合 S の各要素セグメント $seg \in S$ に対応する文字列を索引などを利用して入手し，与えられた数値 num と op の関係が成立するかを調べる．成立した場合には，seg を結果に含める．最終的に，条件を満たすセグメントの集合を返す．

   $\mathcal{S}^a[\![q_1]\!]x$ が数値で $\mathcal{S}^a[\![q_2]\!]x$ がノード集合の場合も同様である．

3. $\mathcal{S}^a[\![q_1]\!]x$ がノード集合に対応し，$\mathcal{S}^a[\![q_2]\!]x$ がブール値の場合:

   $$x \textbf{ with comp\_as\_bool}(\mathcal{S}^a[\![q_1]\!]x, q_2, op) \quad (21)$$

   という関数呼出しを含む文に変換する．**comp_as_bool**(S, b, op) は，実行時に呼び出される．与えられたセグメント集合 S の各要素セグメント $seg \in S$ に対応する文字列を索引などを利用して入手し，与えられたブール値 b と op の関係が成立するかを調べる．成立した場合には，seg を結果に含める．最終的に，条件を満たすセグメントの集合を返す．

   $\mathcal{S}^a[\![q_1]\!]x$ がブール値で $\mathcal{S}^a[\![q_2]\!]x$ がノード集合の場合も同様である．

この他，XPath では，ノード集合とノード集合の比較によるフィルタリング条件が指定可能であるが，これは Proximal Nodes モデルへの翻訳が容易でないため，今後の課題とする．また，算術演算 '+', '−', `div`, `mod` などの支援に関しては省略する．

5.2 問合せ例の変換と最適化

問合せ Q1 の変換と最適化 ここでは，まず，先に示した問合せ **Q1**

Q1: /GeoSpace/GeoFeature[@category = "area"]
[Property/name = "Tsukuba City"]/Geometry/Polygon

に対し変換ルールを適用した一例を図 7 に示す．この結果は，与えられた XPath 式の先頭部分をできるだけ優先して変換するように変換ルールを適用した場合に得られるものであるが，その変換の過程の詳細は，ここでは省略する．$t1, \ldots, t_4$ は，中間結果のセグメント集合をを保持する一時変数であり，複数箇所で共通に現れる共通部分式への参照を表す．最終的な問合せ結果は ans で与えられる．これらの Proximal Nodes 文を上から順番に実行することで最終的な結果が得られることになる．

$$
\begin{aligned}
t_1 &= \texttt{GeoFeature with (GeoFeature with} \\
&\qquad \texttt{(category same "area"))} \\
t_2 &= \texttt{Property child } t_1 \\
t_3 &= t_1 \textbf{ with } (t_1 \textbf{ with } (((\texttt{name child } t_2) \textbf{ child } t_2) \\
&\qquad \textbf{same } \texttt{"TsukubaCity"})) \\
t_4 &= \texttt{Geometry child } t_3 \\
ans &= (((\texttt{Polygon child } t_4) \textbf{ child } t_4) \textbf{ child } t_3) \\
&\qquad \textbf{child } \texttt{GeoSpace}
\end{aligned}
$$

図 7: 問合せ **Q1** の変換例

この変換結果には冗長な部分が存在する．特に，同じ演算が同じ対象に複数回適用される場合が非常に多く見られる．同じ演算の重なる適用は冗長であることから，容易に削除することができる．図 7 の例で説明すると，たとえば，

$$
\begin{aligned}
t_1 &= \texttt{GeoFeature with (GeoFeature with} \\
&\qquad \texttt{(category same "area"))} \\
&\rightarrow \texttt{GeoFeature with (category same "area")} \\
t_3 &= t_1 \textbf{ with } (t_1 \textbf{ with } (((\texttt{name child } t_2) \textbf{ child } t_2) \\
&\qquad \textbf{same } \texttt{"TsukubaCity"})) \\
&\rightarrow t_1 \textbf{ with } (((\texttt{name child } t_2) \textbf{ child } t_2) \\
&\qquad \textbf{same } \texttt{"TsukubaCity"}) \\
&\rightarrow t_1 \textbf{ with } ((\texttt{name child } t_2) \\
&\qquad \textbf{same } \texttt{"TsukubaCity"}) \\
ans &= (((\texttt{Polygon child } t_4) \textbf{ child } t_4) \textbf{ child } t_3) \\
&\qquad \textbf{child } \texttt{GeoSpace} \\
&\rightarrow ((\texttt{Polygon child } t_4) \textbf{ child } t_3) \\
&\qquad \textbf{child } \texttt{GeoSpace}
\end{aligned}
$$

という直接的な変換が可能である．さらに，スキーマ情報からノード間の関係を考慮することにより式を簡略化できる．これらの簡略化を行なってまとめると，

$$
\begin{aligned}
t_1 &= \texttt{GeoFeature with (category same "area")} \\
ans &= \texttt{Polygon in } (t_1 \textbf{ with } ((\texttt{name in } t_1) \\
&\qquad \textbf{same } \texttt{"TsukubaCity"}))
\end{aligned}
$$

という 2 ステップの簡潔な実行処理となる．

問合せ Q2 の変換と最適化　次に問合せ **Q2** の変換について述べる． この問合せは外部関数を含んでいるため，既存の Proximal Nodes モデルでは処理できないことから，外部関数呼出しのための `ex-funcall` 関数を呼び出すような形に変換する．図 8 にその変換結果を示す．

$$
\begin{aligned}
t_1 &= \texttt{GeoFeature with} \\
&\qquad \texttt{(GeoFeature with (type same "school"))} \\
t_2 &= (\texttt{Geometry child } t_1) \textbf{ child } t_1 \\
t_3 &= t_2 \textbf{ with } \texttt{ex-funcall("geo : contained",} \\
&\qquad \texttt{"Polygon", "16300000, 25500000, ...", } t_2) \\
t_4 &= (\texttt{Geometry child } t_3) \textbf{ child } t_3 \\
ans &= (\texttt{Polygon child } t_4) \textbf{ child } t_4
\end{aligned}
$$

図 8: 問合せ **Q2** の変換例

ここで問合せ **Q1** と同様に簡略化すると，以下のようになる．

$$
\begin{aligned}
t_1 &= \texttt{Geometry child} \\
&\qquad \texttt{(GeoFeature with (type same "school"))} \\
ans &= \texttt{Polygon child } (t_1 \textbf{ with } \\
&\qquad \texttt{ex-funcall("geo : contained",} \\
&\qquad \texttt{"Polygon", "16300000, 25500000, ...", } t_1))
\end{aligned}
$$

6　問合せの前処理と実行処理の方式

XML データがデータベースに新たに登録された際には，後の問合せ処理のために索引付けなどの前処理を行う必要がある（図 9）．索引付けの過程で作成する索引は，構造索引，テキスト索引，空間索引の 3 つの種類であり，索引作成時に XML データとともにスキーマ情報も読み込むものとする．構造索引は，文書構造に基づく検索のための演算子において，構造ノード名が指定されたときに，それに対応するセグメントの集合を返すための索引であり，通常の B^+-木などを用いて実現できる．

テキスト索引は，テキストパターン照合演算子において，パターンに該当する各テキストのセグメントからなるセグメント集合を返す索引である．これに関しては，接尾辞配列 (suffix array) [11] などの文字列索引により，効率的な支援が実現可能である．その他の索引としては，アプリケーションに応じて導入される外部の索引が存在する．たとえば，空間情報を扱う XML データに対しては空間索引の支援が必要となることが多いことから，空間索引を新たに導入することが考えられる．空間索引は，先に述べたように，空間的な述語を指定して問合せを行ったときに，その問合せを満たすようなセグメントの集合を返すような索引である．空間索引を，たとえば `Polygon`, `Point` などの要素それぞれに対し一つ作成するか，それとも XML 文書中の空間データすべてに対して一つ作成するかなどの実装は，実装ポリシーに依存して決められることになる．

問合せの実行は，XPath の構文から Proximal Nodes モデルの構文への変換・簡略化の過程と，Proximal Nodes モデルの問合せの実行という 2 つの過程に分けられる（図 10）．変換の過程では，5.1 節で述べた変換ルールに基づいて変換を行なう．変換された Proximal Nodes モデルの記述を実行するステップで

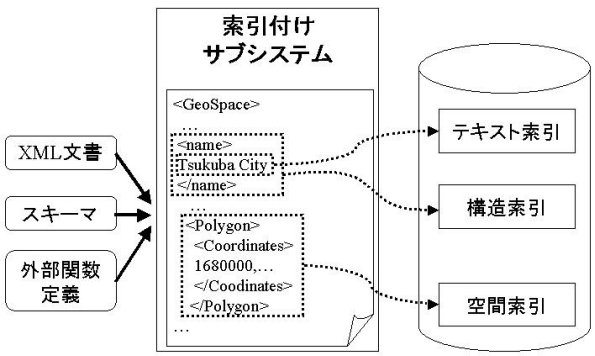

図 9: 問合せの前処理

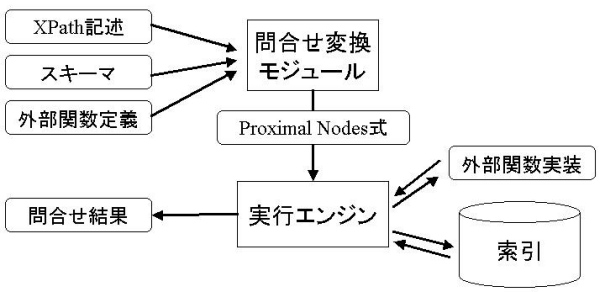

図 10: 問合せの実行処理

は，索引から対応するセグメントを取得し，そのセグメントの情報をもとに各演算子を実行する．また，外部関数記述を含む場合には外部関数定義を参照し，その情報から外部関数を呼出し，必要に応じて空間索引も用いて結果となるセグメントの集合を返す．

7 まとめと今後の課題

本研究では，XML に対する問合せ言語 XPath においてユーザ定義の外部関数の処理を実現するための枠組みについて提案した．具体的には，外部関数を含む XPath 問合せを，構造化文書検索のためのモデルとして提案された Proximal Nodes モデルを拡張した検索モデルに変換し，外部関数の処理も考慮した効率的な検索処理の実現を図った．表示的意味論に基づいて XPath サブセットの式を Proximal Nodes 式に変換する変換ルールをフォーマルに定義し，変換ルールの適用結果の Proximal Nodes 式をさらに簡略化するための方式について述べた．

今後の課題としてはまず，変換ルールをより詳細化し，XPath の構文で支援可能なものをより増やすことを検討する必要がある．また，現在の Proximal Nodes モデルに変換するアプローチでは，XPath とのセマンティクスの違いにより，XPath のフル仕様を支援することは困難である．そのため，Proximal Nodes モデルに完全に変換できない問合せについては，問合せを 1) Proximal Nodes モデルにより部分文書を抽出する処理（できるだけ小さい部分文書を効率的に得る），2) 部分文書に対し XPath の仕様に基づいて問合せを行う実行処理部分，と分けて実行する必要があるとも考えられる．このような分割処理方式に対する検討も必要である．また，今後主流となるであろう XPath 2.0 や XML Schema [12] の仕様に対し，より準拠したような方向での手法の開発も必要であると考えられる．

謝辞

本研究の一部は，日本学術振興会科学研究費基盤研究 (B)(12480067)，奨励研究 (A)(12780183)，および文部科学省科学研究費特定領域研究 (C)(13224008) による．

参考文献

[1] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

[2] G-XML ホームページ
http://gisclh.dpc.or.jp/gxml/

[3] N. Shinagawa, H. Kitagawa, and Y. Ishikawa. X^2QL: An eXtensible XML Query Language Supporting User-defined Foreign Functions, in *Proc. 2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems*, Prague, Czech, Sept. 2000, pp. 251–264.

[4] N. Shinagawa, T. Nagai, H. Kitagawa, and Y. Ishikawa. Querying Geographic Data in XML via eXtensible XML Query Language X^2QL. in *Proc. International Symposium on Asia GIS 2001*, Tokyo, Jun. 2001.

[5] G. Navarro and R. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM TOIS*, 15(4):400–435, 1997.

[6] XML Path Language (XPath) 2.0
http://www.w3.org/TR/xpath20

[7] XQuery 1.0: An XML Query Language
http://www.w3c.org/TR/xquery/

[8] R.A. Baeza-Yates and G. Navarro. Integrating Contents and Structure in Text Retrieval. *ACM SIGMOD Record*, 25(1):67–79, 1996.

[9] Philip Wadler. Two semantics for XPath.
http://www.research.avayalabs.com/user/wadler/topics/xml.html

[10] 井田 哲雄 ,「プログラミング言語の新潮流」計算機科学 / ソフトウェア技術講座 2，共立出版，1988.

[11] 石川 佳治，定兼 邦彦，北川 博之,「文書データを対象とした索引技術」情報処理 Vol.42 No.10, pp. 980-987, 2001.

[12] W3C XML Schema.
http://www.w3c.org/XML/Schema

[13] R. Baeza-Yates and G. Navarro. XQL and Proximal Nodes. *Journal of the American Society for Information Science and Technology (JASIST)*, (to appear).