筑波大学大学院博士課程

システム情報工学研究科特定課題研究報告書

関係データベースに基づく XPath2.0処理器の開発 -XPath式の構文解析および中間表現への変換-

田中 勇也

(コンピュータサイエンス専攻)

指導教員　田中 二郎

2010年3月

概要

本報告書は、「高度 IT 人材育成のための実践的ソフトウェア開発専修プログラム」の科目「研究開発プロジェクト」の成果をまとめたものである。本プロジェクトは、筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻で XML とデータベースに関連する研究を行っている天笠俊之准教授からの委託である。

近年、XML 文書の増加に伴い、XML 文書のデータベースへの格納技術に関する研究や格納された XML 文書の検索に関する研究が盛んに行われている。本プロジェクトは、これらの研究活動の基盤となるシステムを目指し、筆者を含む 4 名のチームメンバーで XPath2.0 処理器「XPathProcceingPlatform」の開発を行った。XPathProcceingPlatform は、関係データベースに XML 文書を格納するとともに、XPath2.0 サブセットによる問合せを行い、問合せ結果を XML 文書に再構築するシステムである。関係データベースに XML 文書を格納する部分、XPath2.0 サブセットによる問合せを行う部分、問合せ結果を XML 文書に再構築する部分ごとに担当を分担し、それぞれの担当ごとに開発を進めた。

開発において筆者は、XPath2.0 サブセットによる問合せを行う部分の中で XPath2.0 パーサを主に担当した。XPath2.0 パーサは、入力された XPath 式を XPath2.0 から定義した XPath2.0 サブセットの文法に沿って、構文解析を行う。その解析結果をもとに XPath 式から SQL 問合せを生成するための中間表現へ変換する。

本プロジェクトでは、XPath 式から中間表現を生成し、SQL トランスレータへの橋渡しをすることができた。

目 次

図 目 次

表 目 次

第1章　序論

本報告書は、筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻で採択された「高度 IT 人材育成のための実践的ソフトウェア開発専修プログラム」の科目「研究開発プロジェクト」の成果をまとめたものである。「研究開発プロジェクト」は、三つの形態から一つを選択する形となっており、その中の教員が顧客となり、教員から委託されたシステムをチーム単位で開発するカテゴリ 3 に属するテーマを選択した。

本報告書で言及するプロジェクトは、同研究科の天笠俊之准教授からの委託である。天笠准教授はデータベースの研究を行っており、その中でも XML とデータベースに関連する研究に力を入れている。

XML は柔軟なデータ構造を文字列で簡単に表現できることから、Web サービスでのシステム間のデータ交換、アプリケーションの設定ファイル、Web サイトの見出しや要約情報など様々な用途で利用されている。今後は XML 形式で記述された文書が増大し、データベースに蓄積されることが予想される。そのため、XML 文書のデータベースへの格納と、格納された XML 文書の検索処理に関する研究が盛んに行われている。[1]

ところが、このような研究の実験で使用されるシステムは、研究内容に応じて一から構築されることが多く、本来の研究に加えてシステム開発に時間を費やさなくてはならないという問題がある。オープンソースのデータベースシステムや先行研究を行っている研究者が作成したシステムを再利用するという方法も考えられるが、前者はシステムの規模が大きいため全体像を把握しづらく、後者は特定の研究用に作成されているため、汎用性に欠け、再利用することが難しい。

そこで、委託元からシステム全体を理解しやすく、特定の研究に依存しない汎用的な処理系の開発が要求された。我々は、研究活動の基盤となるシステムを目指し、XPath2.0 処理器「XPathProcceingPlatform」(以下、本システム) の開発を行った。本システムは、関係データベースに XML 文書を格納するとともに、XPath2.0 サブセットによる問合せを行い、問合せ結果を XML 文書に再構築するシステムである。XML 文書の関係データベースへの格納、XPath2.0 サブセットによる問合せ、問合せ結果の再構築という機能ごとに担当者を割振り、開発を進めた。

本システムの開発において筆者は、XPath2.0 サブセットによる問合せを行う部分の中の XPath2.0 パーサを主に担当した。XPath2.0 パーサは、入力された XPath 式を XPath2.0 から定義した XPath2.0 サブセットの文法に沿って、構文解析を行う。そして、その解析結果をもとに XPath 式から SQL 問合せを生成するための中間表現へ変換する。研究者は、本システムを再利用することで短時間で実験に使用するシステムを開発することが可能となり、従来より

効率的に研究活動が行えると考えられる。

本報告書の構成は以下のとおりである。まず、2 章で本プロジェクトの最終的な成果物である XPath2.0 処理器の要素技術について説明する。次に、3 章では、我々が開発した XPath2.0 処理器の概要と本システムで使用可能な機能について説明する。4 章では、XPath 式から SQL を生成する際の橋渡しとなる中間表現 LeviathanGraph について説明する。その後、筆者が主となり開発を担当した XPath 式の構文解析および中間表現への変換について述べる。5 章では、開発体制・開発環境や成果物といった本プロジェクトについて説明する。最後に 6 章で本報告書のまとめについて述べる。

第2章　要素技術

2.1　XML 関係

2.1.1　XML(Extesible Markup Language)

XML(Extesible Markup Language) は、HTML と同様にタグを用いることで文書内の内容に付加情報を記述できるマークアップ言語である。HTML は、見出しを付ける <h1> タグやテキストを太字にする <b> タグなど決まりきったタグしか使用できなかったが、XML では、タグを自由に記述することができる。そのため、文書の内容に様々な意味を付与することが可能となった。1998 年に W3C(World Wide Web Consortium) により勧告されて以来、柔軟なデータ構造を記述できることから急速に普及が広まった。

XML で記述された XML 文書は、要素、属性、テキストが複数集まって、構成されている。要素は、要素の開始を表す開始タグと終了を表す終了タグの対から成り、入れ子構造をとることができる。開始タグは、< 要素名 >、終了タグは、</要素名 > という形で表し、開始タグと終了タグの間にある文字列がテキストとなる。属性は、開始タグ内の要素名と > の間に属性名=”属性値”という形で記述することができる付加的な情報である。XML 文書の例を図 2.1 に示す。

```
<books>
    <book no=”1”>
        <title>title1</title>
        <author>author1</author>
        <price>1000</price>
    <book>
    <book no=”2”>
        <title>title2</title>
        <author>author2</author>
        <price>2000</price>
    <book>
</books>
```

図 2.1: XML 文書の例

また、XML 文書は、XML 文書を構成する要素、属性、テキストをノードとみなすことで、

木構造として表現することができる。木構造で表した図 2.1 の XML 文書を図 2.2 に示す。

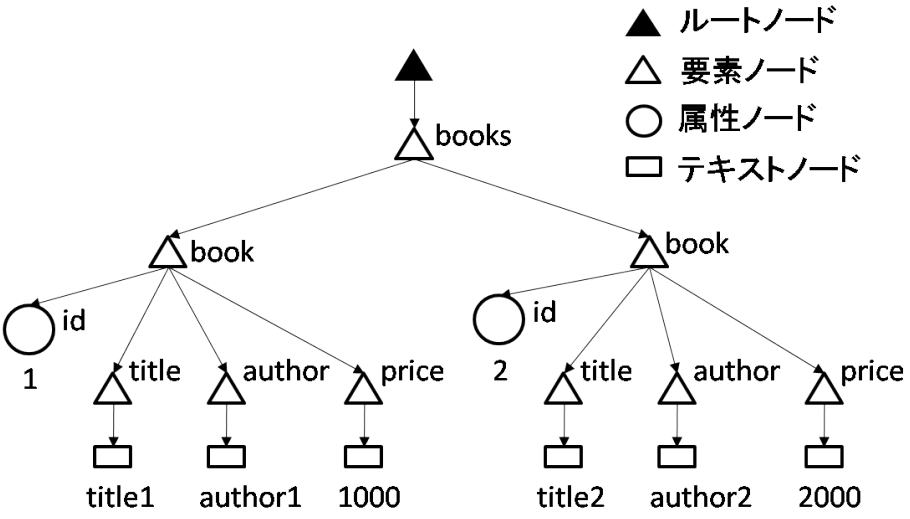

図 2.2: 木構造で表した XML 文書の例

2.1.2 XML 名前空間

2.1.1 節で述べたように XML では、ユーザがタグを自由に記述して文書の内容に様々な意味を付与することが可能である。そのため、複数の XML 文書を組み合わせた場合、異なる意味で用いていた同名の要素や属性が表れる可能性がある。

XML 名前空間とは、同名異義の要素や属性による衝突を避けるための仕組みであり、1999 年に W3C によって勧告された。一意に定まる URI(名前空間 URI) を要素、属性に付けて名前空間を指定し、同名の要素、属性を区別する。要素が名前空間に所属することを示すには、以下のように要素名の前に名前空間接頭辞:と記述する。名前空間接頭辞は、各要素と名前空間 URI を結びつける役割を担う文字列であり、xmlns という特別な属性を用いて名前空間の宣言を行う。

```
< 名前空間接頭辞:要素名 xmlns:名前空間接頭辞=”名前空間 URI”>
        ⋮
</名前空間接頭辞:要素名 >
```

名前空間にはスコープがあり、宣言を行った要素とその子孫要素で有効となり、子孫要素では要素名の前に名前空間接頭辞:を記述するだけでよい。

また、以下に示すように名前空間の宣言で xmlns 属性の後の名前空間接頭辞を省略すると、その名前空間 URI が、宣言を行った要素とその子孫要素で名前空間接頭辞を付けなくても有効となる。これをデフォルト名前空間と呼ぶ。

```
＜要素名 xmlns=”名前空間 URI”＞
        ⋮
</要素名＞
```

2.1.3 XML データベース

XML データベースとは、XML 文書をそのまま扱うための機能を持つデータベースである。XML 文書を格納の対象としており、格納された XML 文書に対して、XPath や XQuery などの問合せ言語による検索を行う。検索結果は、検索対象となった XML 文書の部分文書として出力され、ユーザは特定の部分のみを閲覧することが可能となる。

XML 文書をデータベースに格納する方法は大きく分けて、XML 文書に特化したネイティブ XML データベースに格納する方法と一般的な関係データベースに格納する方法の二つに分けられる。本システムは、タイトルにもあるように関係データベースに XML 文書を格納するシステムである。XML 文書を格納する際、次の技術が重要となる。

SAX(Simple API for XML)

SAX(Simple API for XML) は XML 文書を操作するための API の一つである。XML 文書を先頭から一行ずつ順に読み込んで解析を行い、その結果をイベントとしてアプリケーションへ渡し、イベントに対応する処理を実行する。そのため、ファイルサイズの大きい XML 文書を扱う際でもメモリの消費量が少なく、高速に処理することが可能である。一方、SAX はその性質上シーケンシャルアクセスしか行えないため、XML 文書中の特定箇所を入れ替えるようなランダムアクセスを必要とする処理は困難である。

ノードラベリング

XML 文書を関係データベースに格納する際には、木構造の XML 文書を二次元の関係表へ格納するために構造を変換する必要がある。XPath や XQuery による問合せは、元の木構造に対して行われるので、構造を変換した関係表内でも木構造を維持するための情報を与えなくてはならない。ノードラベリングとは、木構造を維持するためにノードにラベルを付与することである。

ノードラベリングはいくつかの手法が提案されているが、本システムでは、二つの基本的なノードラベリング手法を利用できる。一つは、木構造を走査し、Pre オーダーと Post オーダーの対をラベルとして付与する手法である。この手法を利用し、図 2.2 の木構造にノードラベリングを行うと図 2.3 のようになる。

ここで $x = (pre_i, post_i), y = (pre_j, post_j)$ とすると、$pre_i < pre_j$かつ $post_j < post_i$ が成り立つときのみ、x は y の先祖となり、これを利用することでノード間の先祖子孫関係が判定できる。

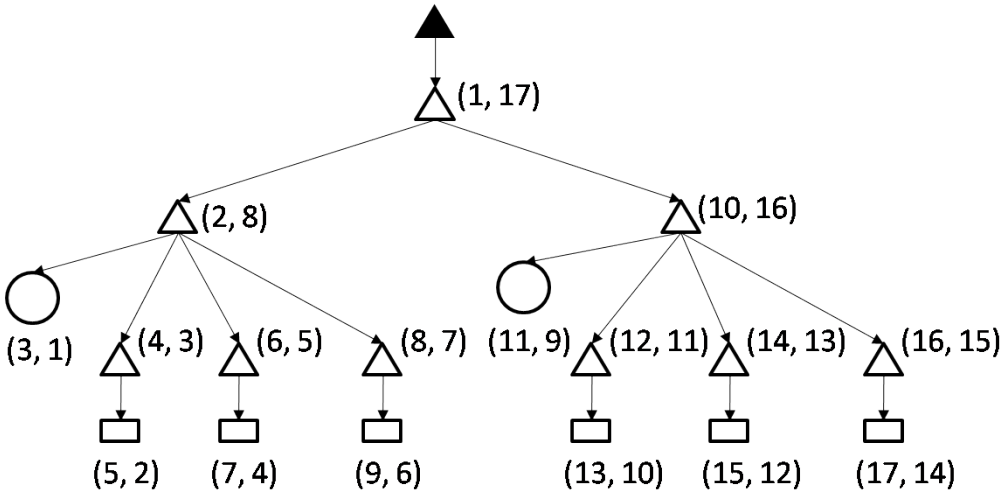

図 2.3: Pre-Post オーダー

もう一つのノードラベリング手法は、同じ親を持つノードに順番を付け、親のラベルと結合する Dewey オーダーという手法である。この手法を利用し、図 2.2 の木構造にノードラベリングを行うと図 2.4 のようになる。

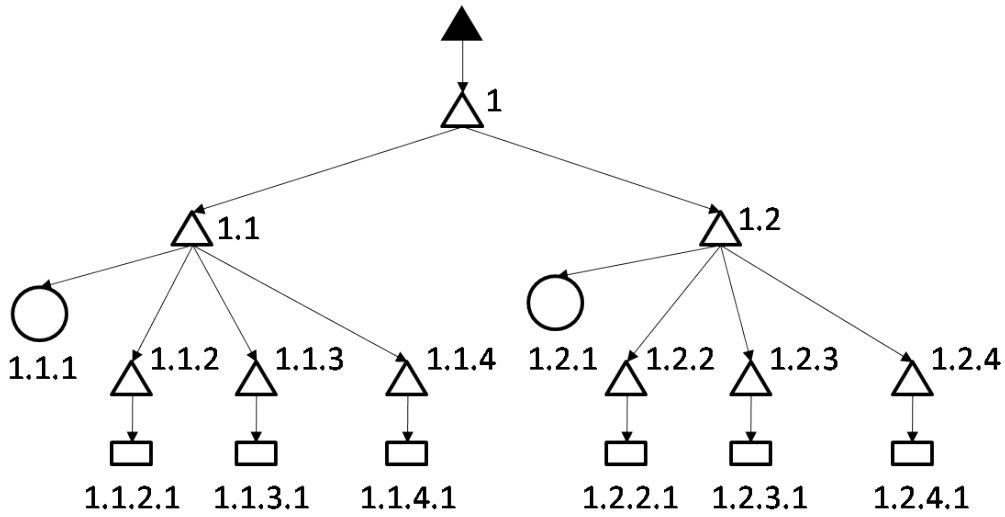

図 2.4: Dewey オーダー

Dewey オーダーでは、あるノード x のラベルと x より長いラベルを持つノード y のラベルを前方から比較して、x のラベルが最後まで y のラベルと一致するとき x は y の先祖となる。これを利用することで先祖子孫関係が判定できる。

2.2 XML 問合せ関係

2.2.1 XPath(XML Path Language)

XPath(XML Path Language) は、XML 文書内の要素や属性の特定の部分を指し示す言語である。XPath2.0[2] は、XPath1.0[3] の次期バージョンとして策定され、2007 年 1 月に W3C によって勧告された。

XPath による問合せの一般的な式は、パス式であり、パスによってコンテキストノード (現在選択されているノード) を起点として、目的のノードが指し示される。パスは、一つ以上のステップの並びから成り、Unix のディレクトリ構造のように / で区切られる。ステップは、次のように記述する。

軸::ノードテスト [述語]

軸 コンテキストノードからみた次のノードを探す方向

ノードテスト ノードの種類や名前

述語 ノードを絞り込むための条件

軸は、XML 文書の木構造における方向であり、表 2.1 に示す種類がある。

表 2.1: 軸

self	コンテキストノード
attribute	コンテキストノードの属性
namespace	コンテキストノードの名前空間
child	コンテキストノードの子
descendant	コンテキストノードの子孫
descendant-or-self	コンテキストノードとコンテキストノードの子孫
parent	コンテキストノードの親
ancestor	コンテキストノードの先祖
ancestor-or-self	コンテキストノードとコンテキストノードの子孫
following	コンテキストノードより後ろに出現するすべてのノード
following-sibling	コンテキストノードの後ろに続く兄弟
preceding	コンテキストノードより前に出現するすべてのノード
following-sibling	コンテキストノードの前に続く兄弟

述語は、記述しなくても良いし、複数記述することも可能である。述語内では、演算子や関数、パス式などを用いることができる。

また、使用頻度の高い軸とノードテストの組み合わせは、省略した構文を記述することができる。これにより、XPath 式の可読性の向上が望める。表 2.2 に非省略構文と省略構文の対応を示す。

表 2.2: 非省略構文と省略構文の対応

非省略構文	省略構文
child::	何も記述しない
attribute::	@
self::node()	.
parent::node()	..
descendant-or-self::node()	//

次の XPath 式は、「ルートノードの子である books 要素の、さらに子である book 要素の中で no 属性の属性値が”1”という条件を満たし、なおかつ book 要素の子である title 要素を指し示せ」という意味になる。

```
/ books / book[@no=”1”] / title
```

図 2.1 に対して、この XPath 式で問合せたときの結果を図 2.5 に示す。赤色の文字が、この XPath 式によって示された部分である。

```
<books>
    <book no=”1”>
        <title>title1</title>
        <author>author1</author>
        <price>1000</price>
    <book>
    <book no=”2”>
        <title>title2</title>
        <author>author2</author>
        <price>2000</price>
    <book>
</books>
```

図 2.5: XPath 式による問合せ結果

2.2.2 XQuery

XPath が、特定の部分を指し示す言語であるのに対して、XQuery は、関係データベースにおける SQL に該当するような XML 文書を照会する言語である。2007 年 1 月に XPath2.0 の勧告と同時に XQuery1.0[4] が勧告された。XQuery1.0 は、XPath2.0 の上位言語であり、XPath2.0

と XQuery1.0 で構文的に正しく、正しく実行される式は、同じ問合せ結果を返す。

本システムでは、XQuery による問合せは行えないが、XQuery の文法で定義されている名前空間宣言の部分を XPath2.0 サブセットとして定義してある。

2.2.3 JavaCC

JavaCC(Java Compiler Compiler) は、yacc(Yet Another Compiler Compiler) に代表されるパーサを生成するパーサジェネレータの一つである。また、スキャナを生成するスキャナジェネレータも兼ねており、その両方を一つのファイルに記述することが可能である。JavaCC では、このファイルを JJ ファイルと呼び、EBNF(Extended Backus―Naur Form) という記法を用いて構文の規則を記述する。

本システムでは、XPath2.0 サブセットの文法を JavaCC で記述し、XPath 式の構文解析を行う。

第3章　XPath2.0処理器の概要

本システムの開発にあたり、委託元から以下の機能が求められた。

- XML 文書を関係データベースへ格納する
- XPath 式による問合せを行い、その結果を XML 文書に再構築する

要求をもとに開発した本システムの概要を図 3.1 に示す。

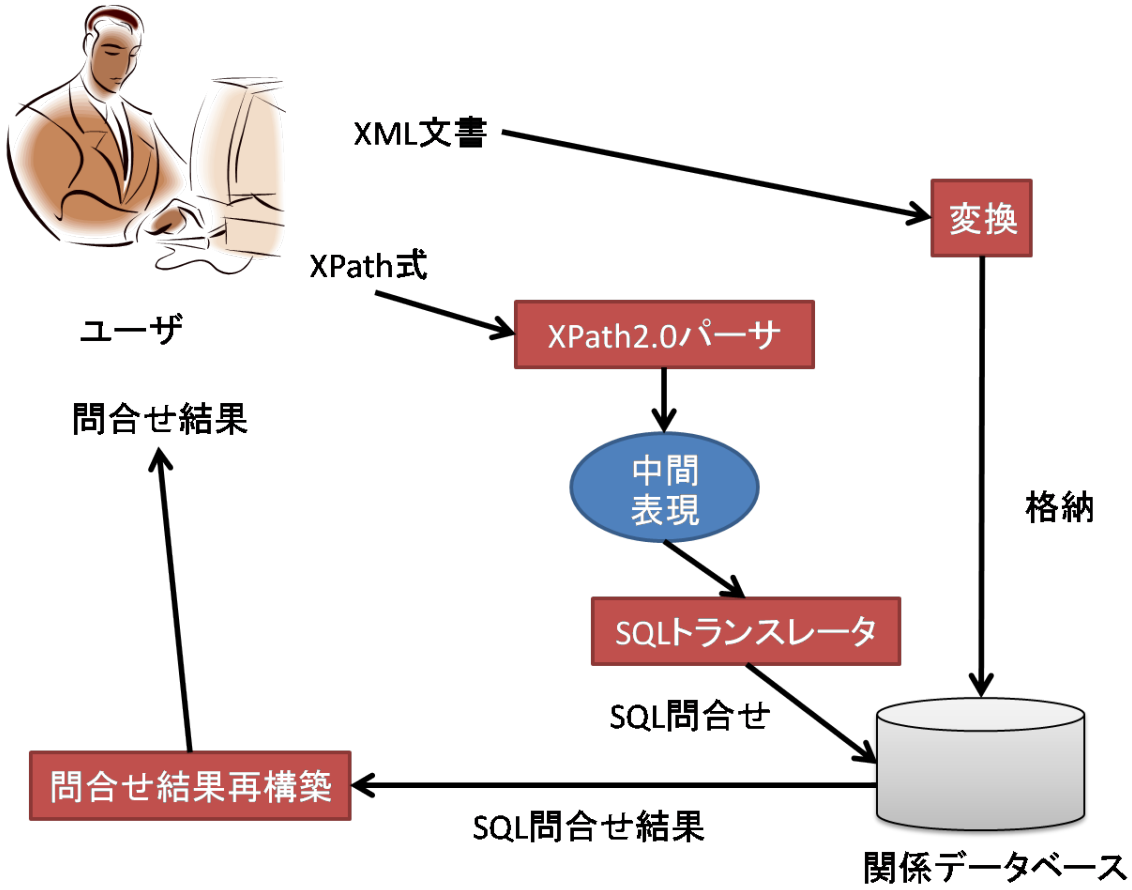

図 3.1: システム概要

3.1 関係データベースへの XML 文書の格納

関係データベースへ XML 文書を格納する際、まず SAX によって XML 文書が解析される。XML 文書は木構造であり、そのままでは二次元の表である関係データベースへ格納できない。そこで、XML 文書を関係データベースに格納できる形へ変換する必要がある。

SAX によって解析された XML 文書は、要素ノード・属性ノード・名前空間ノード・テキストノードに分解される。それぞれのノードには、設定ファイルに記述されているノードラベリングのアルゴリズムに基づき、XML 文書の木構造が保たれるようにノードにラベルが付与される。その後、ノードごとに SQL によって関係データベースに格納される。

本システムでは、XML 文書をコレクションという単位で管理するため、一つのコレクションに複数の XML 文書を格納することが可能である。一つのコレクションは document テーブル、namespace テーブル、node テーブルの三つのテーブルから構成される。document テーブルは、XML 文書のファイル名や XML 文書を格納した日時といった XML 文書の情報を格納するためのテーブル、namespace テーブルは、XML 文書内で宣言されている名前空間を格納するためのテーブル、node テーブルは、ノードラベル付けされた要素ノード・属性ノード・名前空間ノード・テキストノードを格納するためのテーブルである。各テーブルの関係スキーマは、以下のような構造となっている。

- document(docID, docName, insertDate)
- namespace(nsID, nsURI)
- node(nodeID, docID, nsID, type, name, value, start, end, label, path)

docID、nsID、nodeID はそれぞれ文書、名前空間 URI、ノードを識別するための ID であり、各テーブルの主キーとなっている。document テーブルの docName には、格納された XML 文書のファイル名、insertDate には、XML 文書を格納した日付が格納され、namespace テーブルの nsURI には、名前空間 URI が格納される。node テーブルの type には、各ノードの種類を識別するため値が格納され、name と value には、ノードの種類に応じて表 3.1 に示すような値が格納される。

表 3.1: name と value に格納される値

ノードの種類	name	value
要素	要素名	null
属性	属性名	属性値
名前空間	名前空間接頭辞	名前空間 URI
テキスト	null	解析されたテキスト

start、end、label には、ノードラベリングのアルゴリズムに基づいた値が格納される。Pre-Post オーダーの場合、start に Pre オーダーのラベル、end に Post オーダーのラベルが格納され

る。Dewey オーダーの場合、label に Dewey オーダーのラベルが格納される。path には、ルートノードからの経路が格納される。

3.2 XPath 式による問合せ

入力された XPath 式は、まず、XPath2.0 パーサによって構文解析が行われ、中間表現へと変換される。その中間表現をもとに SQL トランスレータによって、XPath 式が指し示す部分を求める SQL が生成され、問合せが行われる。SQL 問合せ結果は、そのままでは XPath 式が指し示す部分の要素しか得られないので、問合せ結果再構築によって、その要素の子要素以下を取得し、格納した XML 文書の部分文書として出力される。

3.3 XPathProcceingPlatform の起動

本システムは、ユーザが CUI 画面上からコマンドを入力し、シェルがそのコマンドの内容を解釈して指示を与える。図 3.2 で示すようにコマンドプロンプト上から java コマンドを用いて、シェルを起動するとプロンプトが表示され、コマンドの入力を受けつけるようになる。その際、本システム、関係データベースへ接続するための MySQL コネクタ、XML 文書を解析するための Xerces、XML 文書を出力するための XMLWriter のクラスパスを設定し、main メソッドのコマンドライン引数に設定ファイルを渡す必要がある。

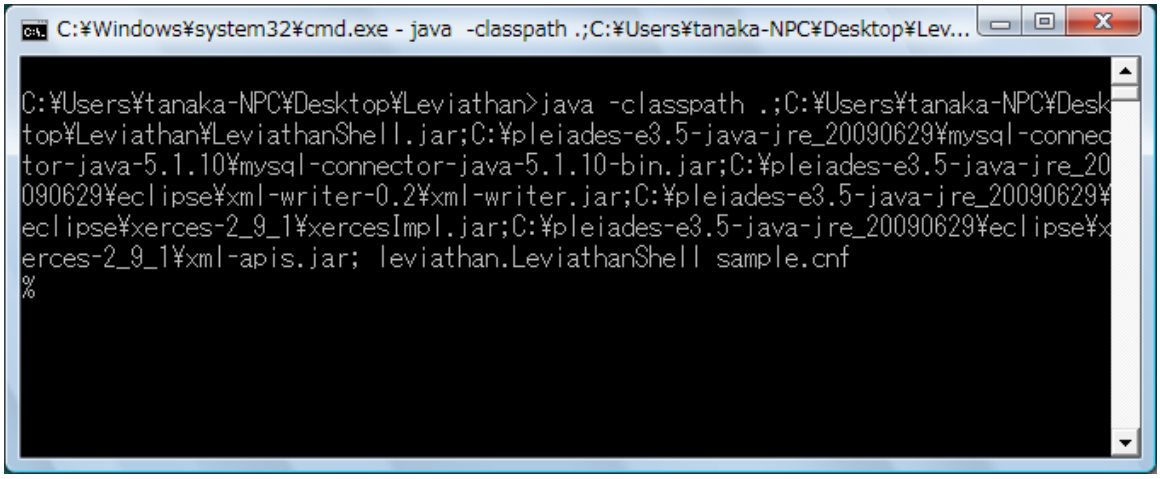

図 3.2: シェル起動画面

図 3.3 に示す設定ファイルには、起動時に必要なパラメータを記述する。設定ファイルに記述される各パラメータは以下の通りである。

prompt

シェルに表示されるプロンプトを記述する。

db_url
接続するデータベース名を記述する。

db_user
データベースへ接続するためのユーザ名を記述する。

db_password
データベースへ接続するためのパスワードを記述する。

db_options
データベースへどのように接続するか指定する。以下のパラメータがあり、true か false を記述する。

- createDatabaseIfNotExist
  true の場合、db_url で示されるデータベースがまだ存在しなければ、データベースを作成する。false の場合、データベースを作成しない。
- useSSL
  true の場合、データベースとの接続に SSL を使用する。false の場合、SSL を使用しない。

algorithm_id
本システムで使用するノードラベリングのアルゴリズムを選択する。デフォルトで使用可能なアルゴリズムは、Pre オーダー、Post オーダーに基づきラベル付けを行う prepost と Dewey オーダーに基づきラベル付けを行う dewey の 2 種類がある。なお、シェル上でノードラベリングのアルゴリズムは変更できないので、変更する場合は一度シェルを終了し、本パラメータを書き換える必要がある。

3.4 XPathProcceingPlatform で使用可能なコマンドと機能

3.1 節、3.2 節で説明した関係データベースへの XML 文書の格納、XPath 式による問合せが本システムにおけるコアとなる機能である。これらの機能以外にも、本システムはデータベースを操作するための一般的な機能を持つ。本節では、本システムで使用可能なコマンドとそれらの機能について説明する。なお、本システムのコマンドは、オープンソースの XML データベースである eXist と Xindice のコマンドを参考にした。

3.4.1 データ定義コマンド

データ定義コマンドとは、コレクションを作成、削除するためのコマンドで、表 3.2 に示すコマンドがある。

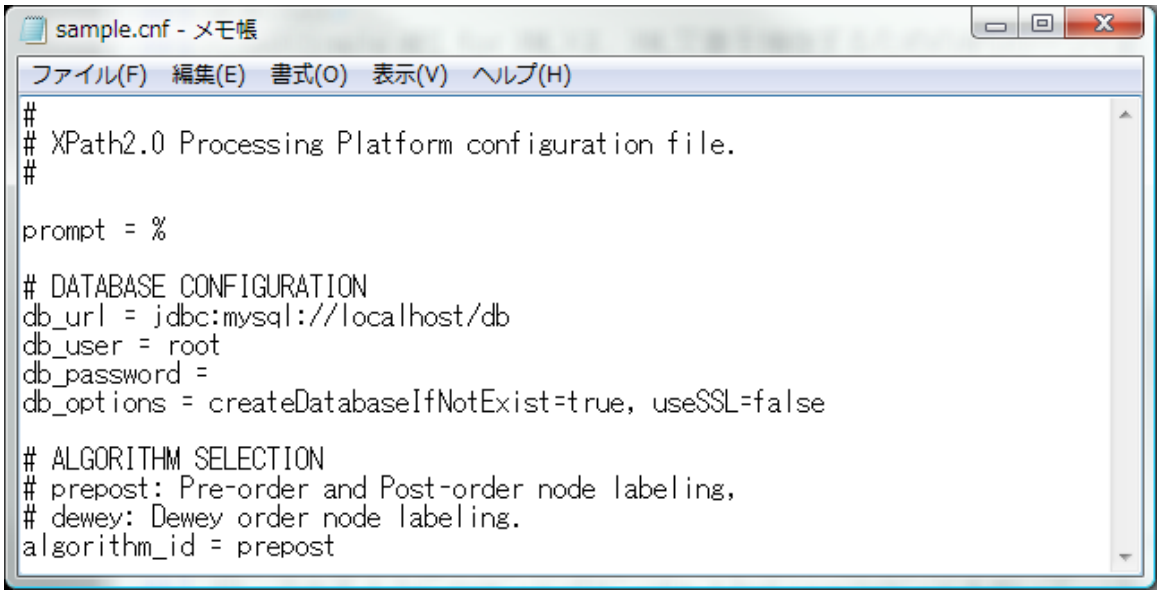

図 3.3: 設定ファイル

表 3.2: データ定義コマンド

コマンド名	説明
mkcol	コレクションを作成する
rmcol	コレクションを削除する

mkcol コマンド

mkcol コマンドは、データベースに新しいコレクションを作成するコマンドである。mkcol コマンドの引数として、コレクション名を指定することでそのコレクション名のコレクションが作成される。test コレクションを作成した結果を図 3.4 に示す。

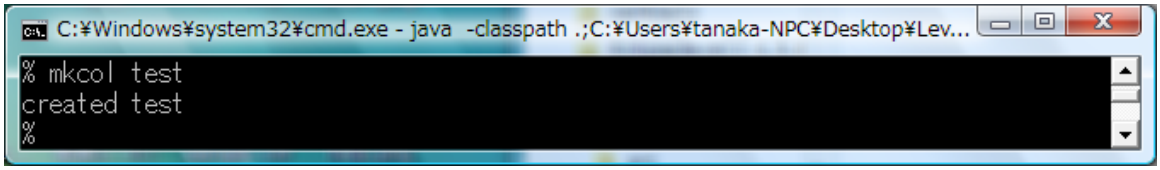

図 3.4: mkcol コマンド実行結果

mkcol コマンド実行後、MySQL プロンプトでデータベースのテーブルを確認してみると、ドキュメントテーブル、名前空間テーブル、ノードテーブルが作成されていることがわかる (図 3.5)。

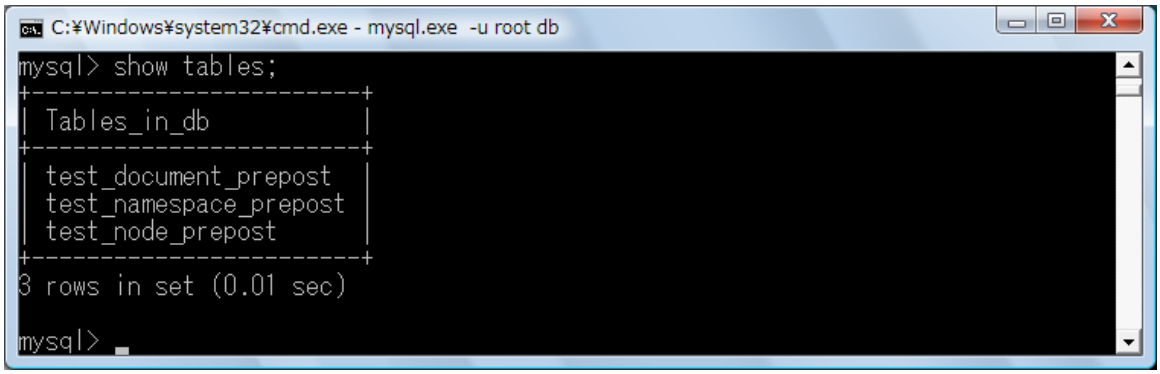

図 3.5: mkcol コマンド実行後のデータベース

rmcol コマンド

rmcol コマンドは、データベースからコレクションを削除するコマンドである。rmcol コマンドの引数として、コレクション名を指定することでそのコレクション名のコレクションが削除される。test コレクションを削除した結果を図 3.6 に示す。

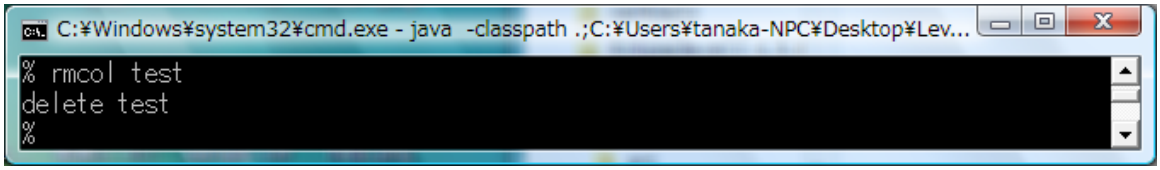

図 3.6: rmcol コマンド実行結果

rmcol コマンド実行後、MySQL プロンプトでデータベースを確認してみると、Empty set が返され、ドキュメントテーブル、名前空間テーブル、ノードテーブルが削除されていることがわかる (図 3.7)。

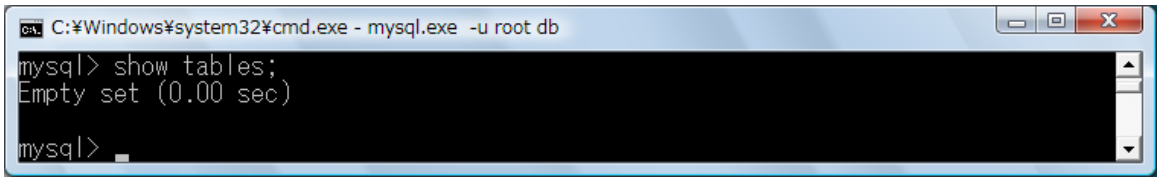

図 3.7: rmcol コマンド実行後のデータベース

また、コレクションに XML 文書が格納されている場合、rmcol コマンドを実行してもコレクションは削除されない。図 3.8 を見ると、エラーが出力されコレクションが削除されていないことがわかる。

XML 文書ごとコレクションを削除するには rmcol コマンドに-f オプションをつける必要がある。図 3.9 を見ると、-f オプションをつけることで XML 文書ごとコレクションが削除され

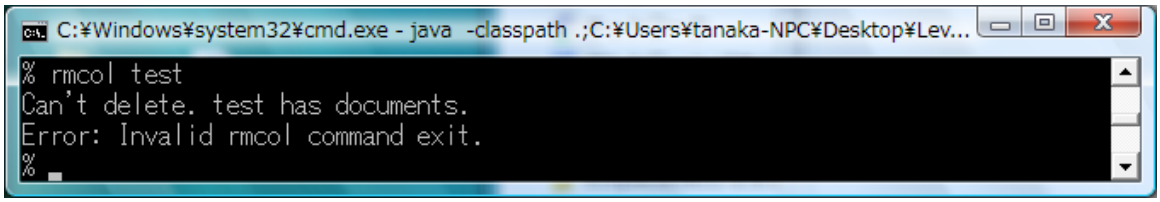

図 3.8: XML 文書格納後の rmcol コマンド実行結果 (-f オプションなし)

ていることがわかる。

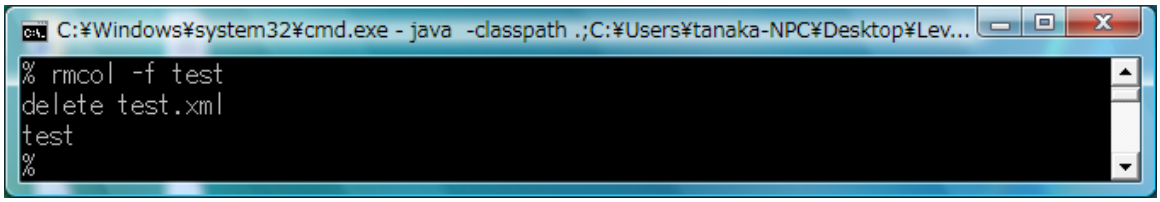

図 3.9: XML 文書格納後の rmcol コマンド実行結果 (-f オプションあり)

3.4.2 データ操作コマンド

データ操作コマンドとは、コレクションに XML 文書を格納、削除したり、コレクションに格納されているデータに対して問合せを行うコマンドで、表 3.3 に示すコマンドがある。

表 3.3: データ定義コマンド

コマンド名	説明
ls	コレクション内の XML 文書の情報またはコレクション名を表示する
lsns	XML 文書内で使用されている名前空間を表示する
put	XML 文書を格納する
rm	XML 文書を削除する
find	XPath 式による問合せを行い、その結果を画面に出力する

ls コマンド

ls コマンドは、コレクションに関する情報を表示するコマンドである。図 3.10 に示すように、引数にコレクション名を指定して実行すると、そのコレクションに格納されている XML 文書の情報 (XML 文書のファイル名、XML 文書の ID、コレクション名、XML 文書の格納日) が表示される。

図 3.11 に示すように、引数を指定せず、ls コマンドを実行した場合、データベースに存在するコレクション名が表示される。

図 3.10: ls コマンド実行結果 (引数あり)

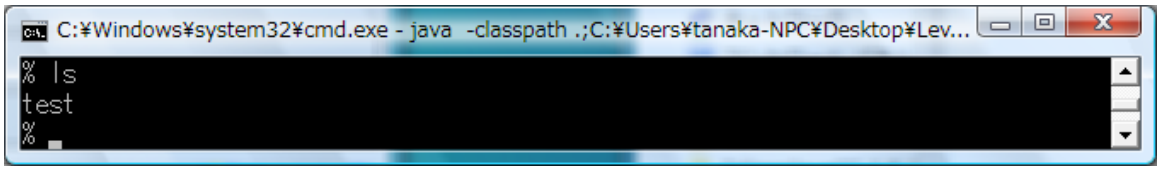

図 3.11: ls コマンド実行結果 (引数なし)

lsns コマンド

lsns コマンドは、XML 文書内で使用されている名前空間を表示するコマンドである。図 3.12 に示すように、引数にコレクション名を指定して実行すると、名前空間の ID とコレクションに格納されている XML 文書内の名前空間が表示される。表示された ID1 と 2 の名前空間は、それぞれ xml という接頭辞の名前空間と xmlns という接頭辞の名前空間であり、特殊な名前空間として扱われ、mkcol コマンドでコレクション作成時、自動的に名前空間テーブルに格納される。

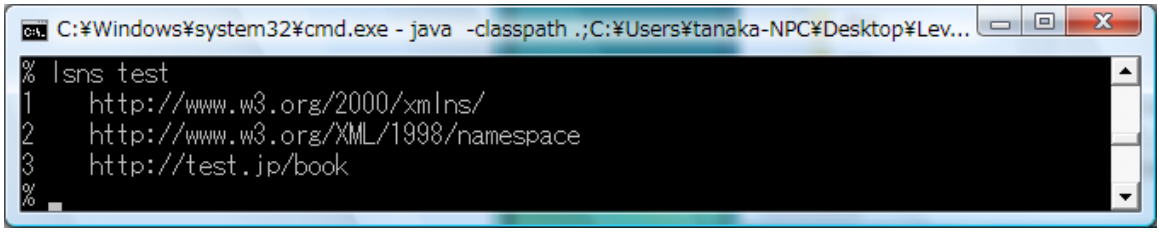

図 3.12: lsns コマンド実行結果

put コマンド

put コマンドは、XML 文書をコレクションに格納するコマンドである。図 3.13 に示すように、引数としてファイル名を指定し、-c の後にコレクション名を指定すると、そのコレクションに指定したファイルが格納される。格納に成功すると、要素ノード数、属性ノード数、テキストノード数、格納時間が表示される。

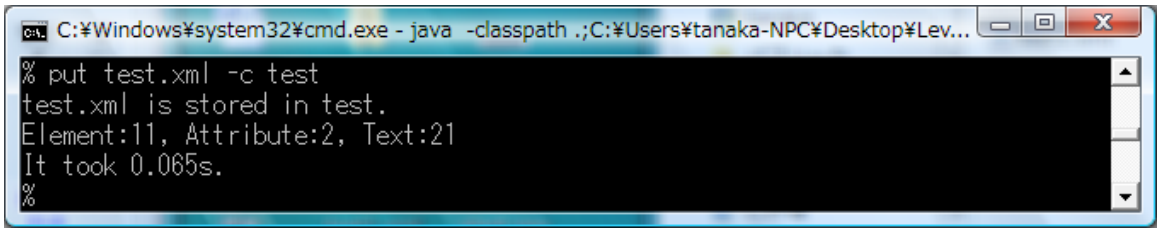

図 3.13: put コマンド実行結果

rm コマンド

rm コマンドは、コレクションに格納されている XML 文書を削除するコマンドである。図 3.14 に示すように、引数としてファイル名を指定し、-c の後にコレクション名を指定すると、そのコレクションに指定したファイルが削除される。削除に成功すると、削除時間が表示される。

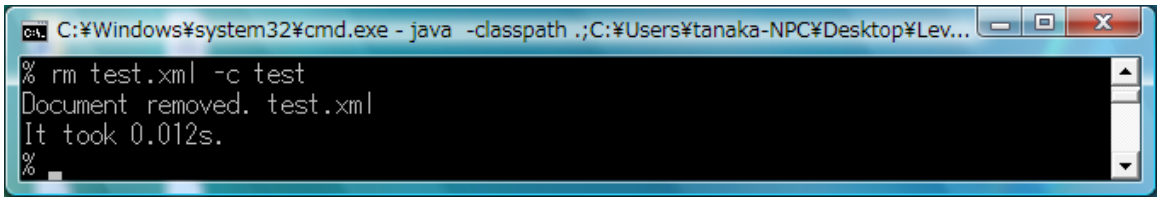

図 3.14: rm コマンド実行結果

find コマンド

find コマンドは、コレクションに格納されている XML 文書に対して、XPath 式による問合せを行い、その結果を出力するコマンドである。本システムでは、コレクション単位で問合せを行うため、図 3.15 に示すように、find の後に collection(”コレクション名”)XPath 式という形で問合せを記述する。

名前空間接頭辞がついている要素または属性に対して、XPath 式で問合せを行う際、図 3.16 に示すように、declare を用いて名前空間を宣言し、同じ名前空間に属していると示す必要がある。なお、シェルにおけるコマンドや引数の識別は、空白区切りで行われる。そのような実装の都合上、XPath 式内に空白が現れる場合、XPath 式をダブルクォーテーションで囲い、XPath 式内のダブルクォーテーションは \ をつけてエスケープしなければならない。

また、find コマンドは、問合せ結果を画面にするだけでなく、ファイルに出力することも可能である。図 3.17 に示すように、find コマンドに-o オプションをつけ、ファイル名を指定した場合、問合せ結果は画面に出力されず、指定したファイルに出力される。

出力されたファイルを図 3.18 に示す。

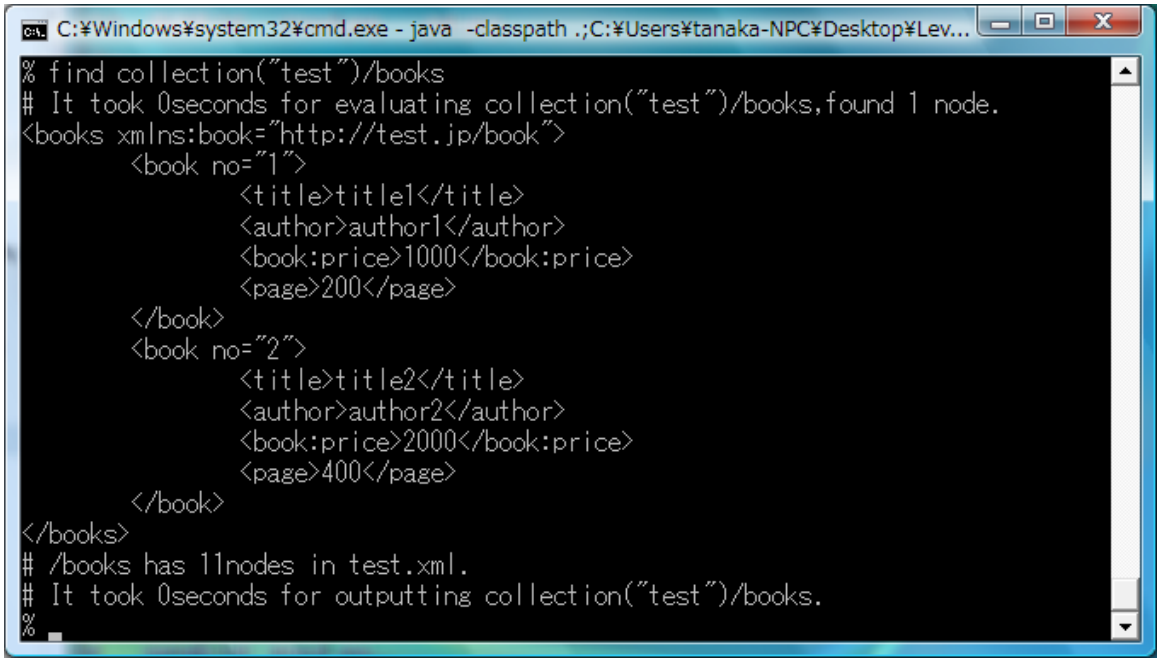

図 3.15: find コマンド実行結果

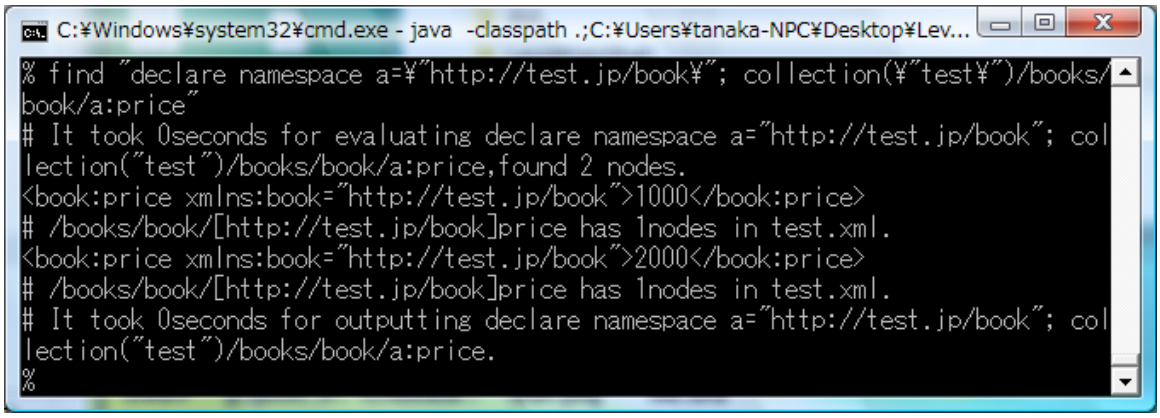

図 3.16: 名前空間宣言時の find コマンド実行結果

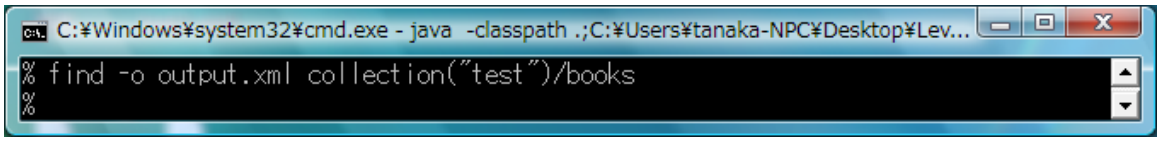

図 3.17: find コマンド実行結果 (ファイル出力時)

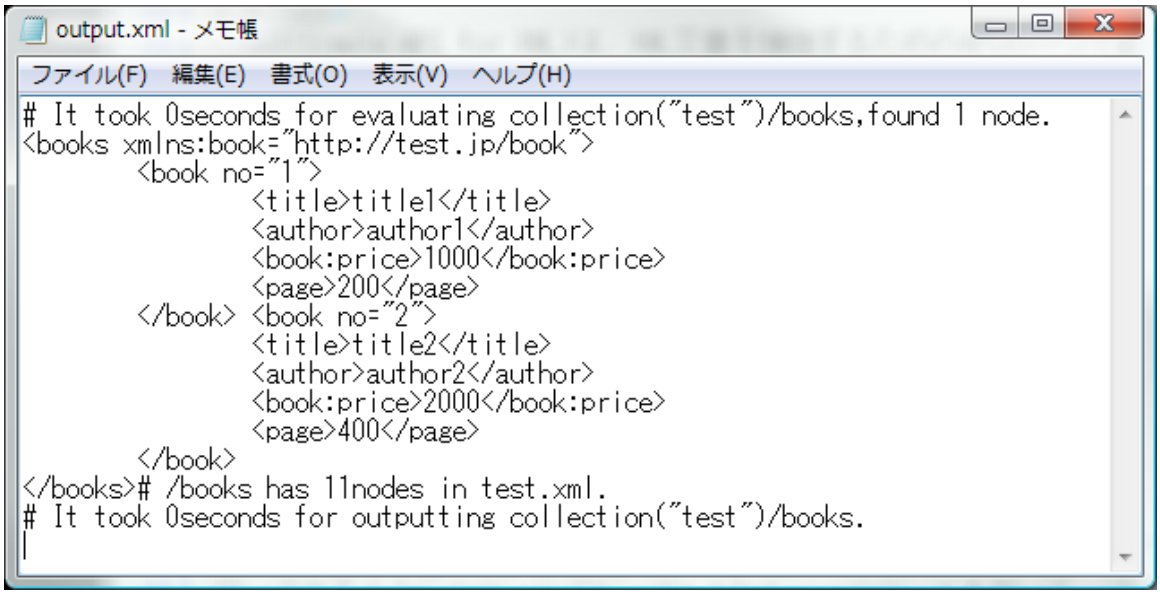

図 3.18: 出力されたファイル output.xml

3.4.3 シェル関連コマンド

シェル関連コマンドとは、シェルのヘルプを表示したり、シェルを終了するためのコマンドで、表 3.4 に示すコマンドがある。

表 3.4: データ定義コマンド

コマンド名	説明
help	コマンド一覧を表示する
quit	シェルを終了する

help コマンド

help コマンドは、コマンド一覧を表示するコマンドである。help コマンドを実行すると、図 3.19 に示すようにコマンド一覧が表示される。これによりユーザは、シェルで使用可能なコマンドとそのコマンドの機能を知ることができる。

quit コマンド

quit コマンドはシェルを終了するコマンドである。quit コマンドを実行すると、図 3.20 に示すようにシェルが終了し、コマンドプロンプトへ戻る。

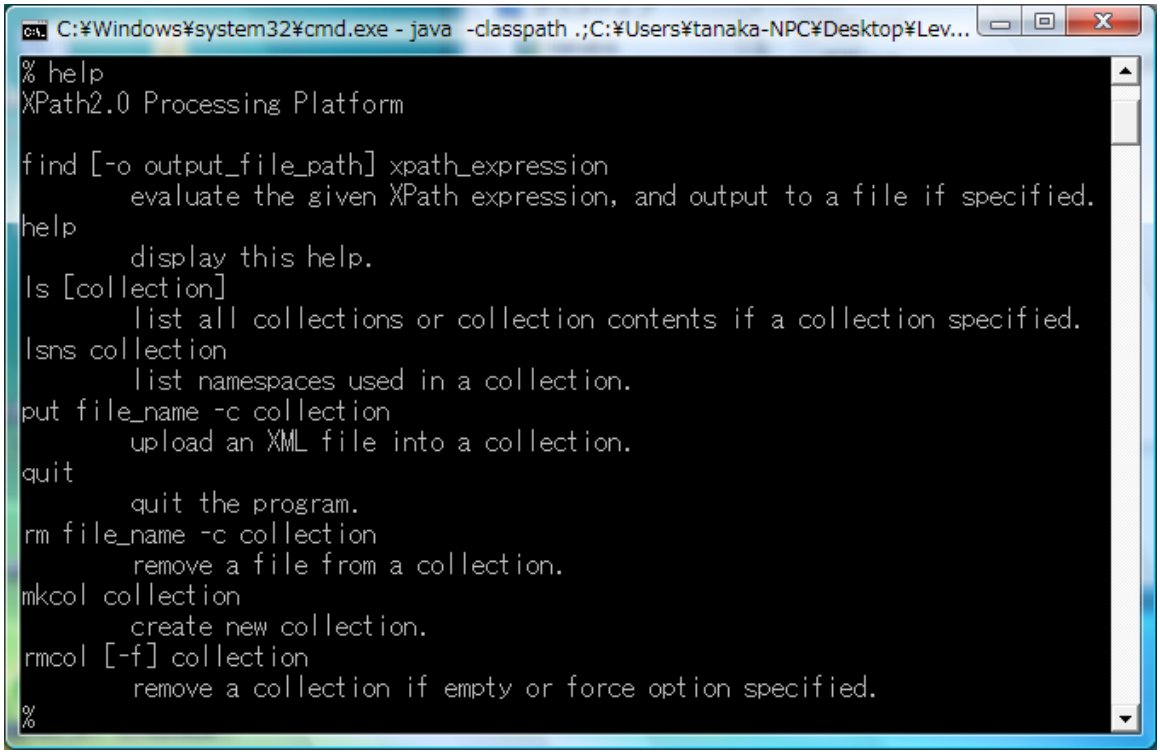

図 3.19: help コマンド実行結果

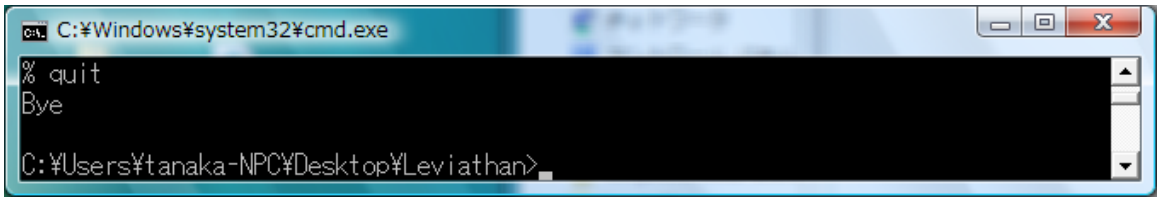

図 3.20: quit コマンド実行結果

第4章　XPath式の構文解析および中間表現への変換

4.1　中間表現 LeviathanGraph

LeviathanGraph は、XPath 式から SQL 問合せへの変換を行う際の中間表現として表される有向グラフである。LeviathanGraph は四角形のノードと矢印のエッジから構成され、以下の XPath 式が入力された場合、図 4.1 のような構文木を生成する。

/ catalog / book[compare(title, ”XML”) = 0] / author

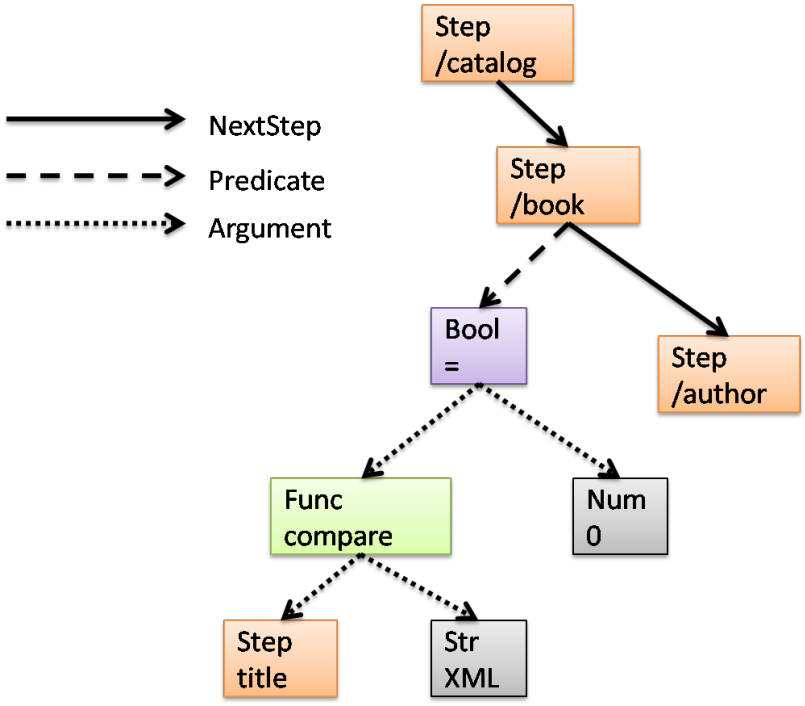

図 4.1: LeviathanGraph の構文木

ノードは、構文解析で導出された記号に応じて、表 4.1 に示すような種類のノードが作成され、各ノードには、各ノードの種類に応じた値が設定される。また、ノード間には、表 4.2 に示すような現在のノードが矢印で示す先のノードとどのような関係となっているか表現す

るエッジが結ばれる。

表 4.1: ノードの種類と値

種類	値
Step	/と述語を除いたステップ (軸，ノードテスト) から成り， 省略構文で記述されたステップはそのままノードの値となる
Bool	比較演算子 (<, <=, =, !=, =>, >)、論理演算子 (and, or)
Func	関数名
Num	数字
Str	文字列

表 4.2: エッジの種類と意味

種類	意味
NextStep	現在のノードから次に進むべきノード
Predicate	ステップ中に表れた述語
Argument	関数，比較・論理演算子における引数

4.2 XPath 式の構文解析および中間表現への変換

XPath2.0 サブセットの文法を定義するにあたり、委託元から次の要件が挙げられた。

- 以下に示す軸を解釈できること
  - child, descendant, attribute
- 述語で論理演算子 (and, or) を利用できること
- 述語で比較演算子 (<, <=, =, !=, =>, >) を利用できること
- 以下に示す関数を利用できること
  - compare, contains, not
- パス式を中心とすること

この要件に沿って、XPath2.0 の文法から本システムで不要だと思われる文法を排除した。パス式を中心とするシンプルな問合せを心がけ、繰り返し機能の For 式や条件を判定する条件式などのプログラミング的な式をサブセットの定義から外し、それに伴い、変数参照、算術演算、集合演算といった要素も定義から外した。これとは逆に、XQuery の文法で定義され

ている declare による名前空間の宣言や collection への問合せといった XPath2.0 のフルセットには存在しない文法の追加も行った。以上のことを考慮し、複数の XML 文書に対して名前空間を用いて問合せが可能な XPath2.0 サブセットの文法を定義した。

XPath2.0 パーサの実装は、JavaCC を用いて行った。JavaCC は、入力文字列を左から右へ構文解析をしていき、左端導出を行う LL 法となっている。XPath2.0 パーサに入力された XPath 式は、左端から XPath2.0 サブセットの文法に照らし合わせていく。非終端記号が出てくるたび、生成規則を適用していき、これを繰り返し、終端記号に到達すると返り値を返し始める。解析途中でロジックに従い、ノードを生成し、ノード間のエッジを結び、XPath 式を LeviathanGraph へ変換していく。XPath 式が右端へ到達し、返り値を返し終えると構文解析が完了する。

4.3 具体例による中間表現への変換手順

以下の XPath 式が入力された場合、XPath 式から中間表現への変換がどのように行われるのかを示す。

```
collection(”sample”)/ catalog / book[compare(title, ”XML”) = 0] / author
```

入力された XPath 式を左端から解析すると、まず、collection(”sample”) 部分が解析され、問合せを行うコレクション名が sample と認識される。次に、/catalog 部分が解析され、図 4.2 に示すように Step ノードが作成され、値として/catalog が設定される。

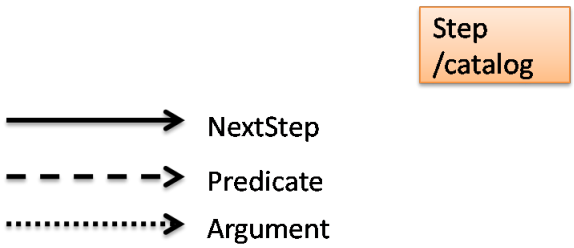

図 4.2: 中間表現への変換 1

catalog までの解析を終えると、/book 部分が解析され、先ほどと同様に図 4.3 に示すような Step ノードが作成される。

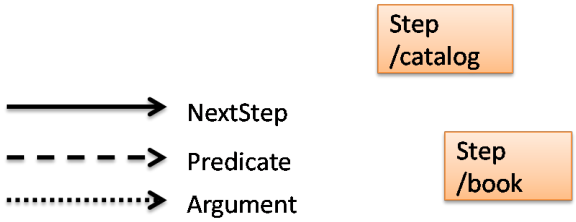

図 4.3: 中間表現への変換 2

次に、パーサで”[”が導出されると述語部分の解析を開始する。解析が進むと図 4.4 に示すように Func ノードが作成され、値として compare が設定される。

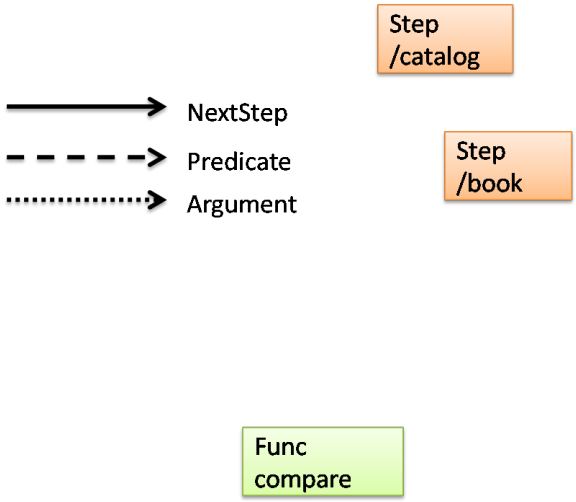

図 4.4: 中間表現への変換 3

Func ノードが作成されると、関数の引数部分の解析を開始する。title 部分が解析され、図 4.5 に示すように title という値を持つ Step ノードが作成され、Func ノードの引数として、Argument エッジが結ばれる。

次に、”,”が導出されると、二つ目の引数の解析が行われる。”XML”部分が解析され、図 4.6 に示すように XML という値を持つ Str ノードが作成され、Func ノードの引数として、Argument エッジが結ばれる。

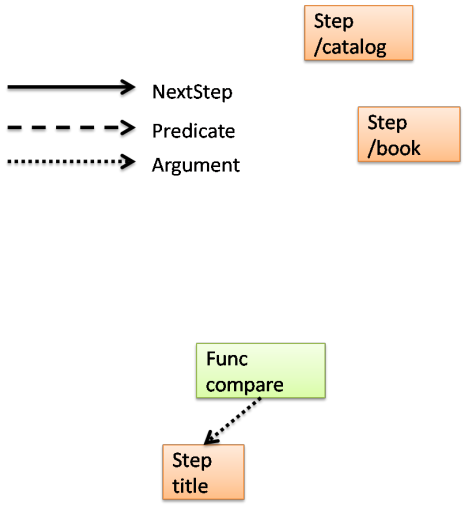

図 4.5: 中間表現への変換 4

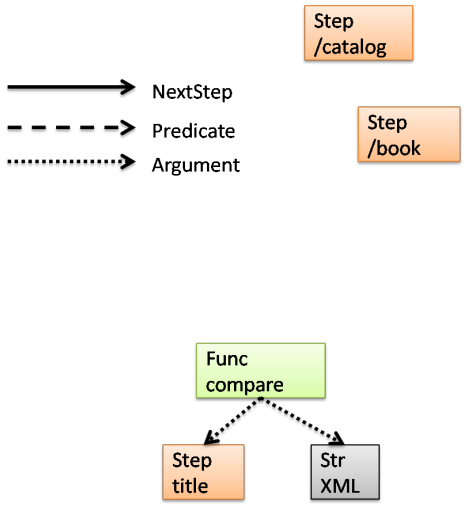

図 4.6: 中間表現への変換 5

) までの解析を終えると、関数の引数部分の解析を終了する。次に、=が解析され、図 4.7 に示すように=という値を持つ Bool ノードが作成される。

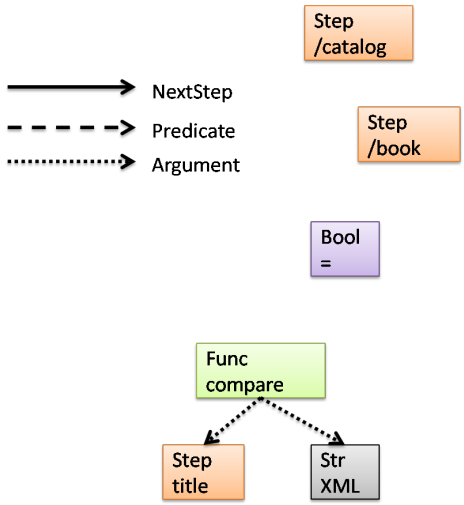

図 4.7: 中間表現への変換 6

さらに、0 が解析され、図 4.8 に示すように 0 という値を持つ Num ノードが作成される。

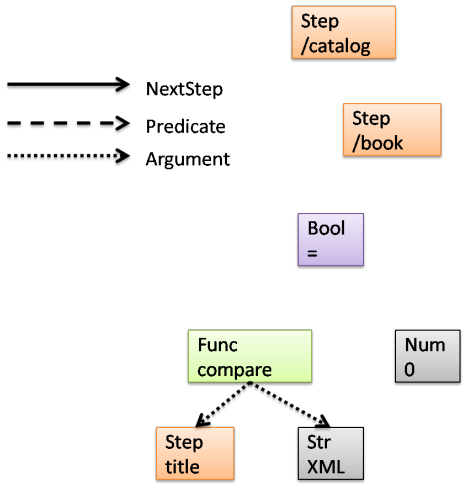

図 4.8: 中間表現への変換 7

0 までの解析を終えると、図 4.9 に示すように Bool ノードの引数として、Func ノードと Num ノードにそれぞれ Argument エッジが結ばれる。

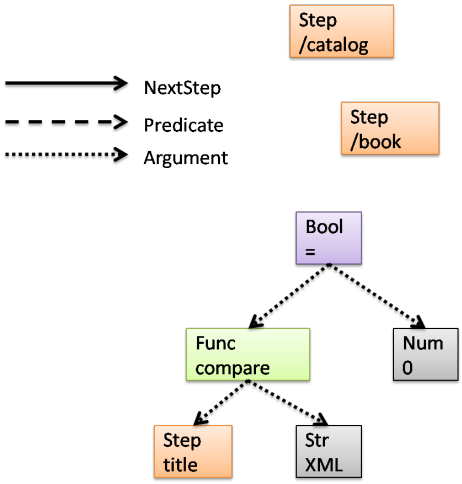

図 4.9: 中間表現への変換 8

次に、] まで解析を終えると、述語部分の解析が終了し、図 4.10 に示すように/book を値に持つ Step ノードの述語として、Bool ノードに Predicate エッジが結ばれる。

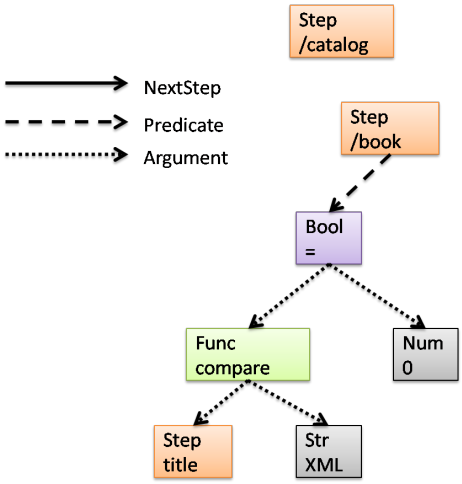

図 4.10: 中間表現への変換 9

さらに、/catalog を値に持つ Step ノードの次のステップとして、/book を値に持つ Step ノードに NextStep エッジが結ばれる。

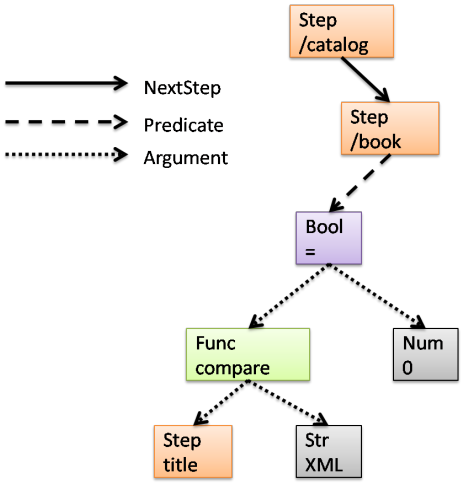

図 4.11: 中間表現への変換 10

最後に、/author 部分の解析を終えると、/author を値に持つ Step ノードが作成され、/book を値に持つ Step ノードと NextStep エッジが結ばれる。

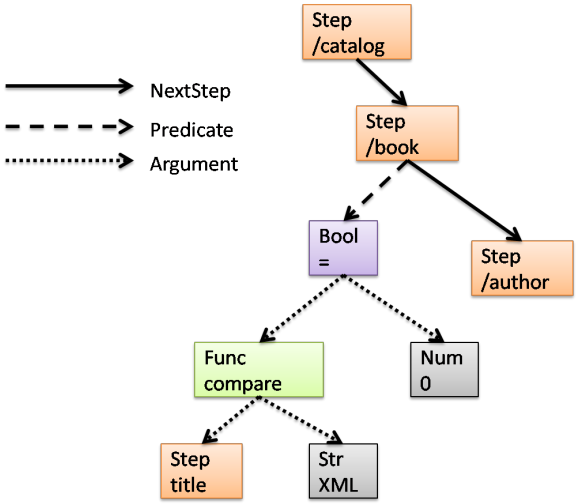

図 4.12: 中間表現への変換 11

以上のような流れで、XPath 式から中間表現への変換が行われる。このように作成された中間表現をもとに SQL が生成される。

4.4 問題点と解決策

開発を担当した XPath2.0 パーサにおいて発生した問題点として、XPath 式で、ある文字を使用すると構文解析エラーが起きてしまうため、その文字が使用できないという点が挙げられる。使用できない文字は、XPath 式による問合せを行う際、コレクション名を指定する collection や名前空間を宣言する declare などの文字である。したがって、これは XML 文書中に collection などの要素が存在した場合、その要素を指し示すことができないという意味であり、以下に示すような XPath 式による問合せは行えない。

collection(”sample”)　/ collection

このような問題が発生した原因として、トークン定義の規則の順序が挙げられる。トークンとは、プログラミング言語のソースコードを構成する単語や記号であり、ここでは XPath 式を構成する単語や記号となる。XPath 式が入力されると、スキャナは JJ ファイルに記述されたトークン定義の規則に従い、字句解析を行う。XPath 式から、適切な文字列を値とするトークンが生成されるとともにトークンに種類が与えられ、パーサはそのトークンをもとに構文解析を行う。トークンの種類とは、トークンの文法上の分類である。例えば、上記の XPath 式の一つ目の collection は、コレクション名を指定するためのトークンとして分類される。

JJ ファイルに記述されたトークン定義の規則には、優先順位が存在し、複数の規則に当てはまる文字列は先に JJ ファイルに記述してある規則を優先する。つまり、collection という文字列は、ノードテストを意味するトークン定義の規則とコレクション名を指定するためのトークン定義の規則に当てはまるが、ノードテストを意味するトークン定義の規則より先にコレクション名を指定するためのトークン定義の規則が記述してあったため、二つ目の collection は、コレクション名を指定するためのトークンとして分類されてしまった。よって、二つ目の collection は、本来パーサによってノードテストを意味するトークンとして解析されるはずだが、一つ目の collection と同様にコレクション名を指定するためのトークンして解析されたため、構文解析エラーが発生した。

この問題を解決するために、トークン定義に状態を持たせる方法をとった。JavaCC では、スキャナに状態があり、スキャナがある状態に遷移すると、その状態専用のトークン定義の規則しか働かなくなる。そこで、コレクション名を指定するためのトークン定義の規則や名前空間を宣言するためのトークン定義の規則に状態 (仮に、A とする) を持たせる。解析を行う際、スキャナの状態を最初は A にしておき、XPath 式のコレクション名を指定する部分、名前空間を宣言する部分の解析を終えたら、スキャナの状態を DEFAULT の状態にする。このようにスキャナの状態を遷移させることで、上記の XPath 式の一つ目の collection はコレクション名を指定する種類、二つ目の collection はノードテストを意味する種類のトークンとなり、構文解析エラーの発生を防ぐことができた。これにより、XPath 式で使用できない文字があるという問題は解決した。

第5章　開発プロジェクト

5.1　開発体制

本プロジェクトは、開発プロセスとしてウォーターフォール型のプロセスを採用した。このプロセスを採用した理由として、科目「PBL 型システム開発」においてメンバー全員がウォーターフォール型のプロセスによってシステム開発を経験したことと、実社会においてウォーターフォール型のプロセスが多く採用されていることが挙げられる。ウォーターフォール型のプロセスは要件定義工程、外部設計工程、内部設計工程、実装工程、単体テスト工程、結合テスト工程、総合テスト工程の七つの工程から構成される。

要件定義工程では、委託元教員にヒアリングを行い、メンバー全員でシステム化する範囲やシステムが満たすべき要件をまとめ、各々が担当する部分を決定した。本システムの開発に携わるチームメンバと各々が担当した部分を表 5.1 に示す。

表 5.1: チームメンバと担当部分

メンバ名	担当
上田 保祐	XML 文書格納
佐用 健	問合せ結果再構築
田中 勇也	XPath2.0 パーサ
羽鳥 貴之	SQL トランスレータ

外部設計工程では、XPath サブセット定義書といった特定のメンバーに依存する作業以外は、システムの全体像を理解するといった意味あいもあり基本的に担当者を定めずに作業を行った。また、要件定義・外部設計工程を学習期間と位置づけ、XML,XPath、JavaCC、SAX、JDBC などの本システムの核となる技術について、学習を行った。

これ以降の工程は、各々が担当する部分に関する作業をメインとして、設計・実装・テストを進めた。

チームミーティングは週に二回程度、タスクの進捗状況、システムに関する議論、議論に基づく意志決定、技術調査内容の報告などを行った。また、ミーティングがこれだけでは不十分だと感じた場合、朝ミーティングを実施するなど臨機応変な対応を取る体制をとった。委託元教員とは、二週間に一度程度ミーティングを実施する体制をとり、プロジェクトの進捗状況の報告や開発を進める上で生じた疑問点の相談、成果物のレビューなどを行った。

5.2 開発環境

本プロジェクトにおける開発環境を表 5.3 に示す。委託元からプラットフォームに依存しないシステムが要求されたため、開発言語は Java を採用した。チームで開発を行う際、メンバ間でのソースコードの連携やバグの混入をなるべく防ぐため、統合開発環境が重要となる。本プロジェクトでは、Java によるソフトウェア開発のデファクトスタンダードとなっている Eclipse を使用した。これにより、Subversion によるソースコードの管理、JUniit や djUnit といったテストツールを活用することができたため、効率的な開発が行えた。また、TortoiseSVN を導入することで、各種成果物や技術調査報告、ミーティング議事録などの情報共有を容易に行えるようにした。

表 5.2: 開発環境

開発言語	Java 1.5 JavaCC 4.2
統合開発環境	Eclipse 3.5 Galileo
テストツール	JUnit4 djUnit 0.8.4
外部 JAR	Connector-J 5.1.10 Xerces 2.9.1 XMLWriter 0.2
データベース	MySQL 5.0
バージョン管理システム	Subversion(サーバ) TortoiseSVN(クライアント)
OS	Windows Vista

5.3 開発計画と実績

本プロジェクトにおける開発計画の予定と実績を表 5.3 と図 5.1 に示す。当初プロジェクトは、2009 年 7 月 2 日の要件定義を皮切りに 2009 年 12 月 15 日に総合テストを終了する予定だったが、単体・結合テストでの遅れが総合テストに影響したため、7 日遅れの 2009 年 12 月 24 日に委託元へシステムを納入する結果となった。このような遅延が起きた原因は、筆者の担当部分である XPath2.0 パーサにおいて、4.4 節で示した問題が発生したためである。問題を解決するために、内部設計の見直しまで作業が後戻りしてしまい、デバッグ作業が遅々として進まず、進捗の遅れを取り戻すことができなかった。

また、要件定義工程でも 5 日の遅れが生じた。こちらの遅延の原因として、本システムに関連する XML、XPath などの知識が不足していたため、委託元からの要求を理解し、要件定義書にまとめる作業に時間がかかったことが挙げられる。しかし、こちらの進捗の遅れは、外

部設計を短縮することで取り戻すことができた。要件定義・外部設計工程を学習期間と定めていたが、要件定義書をまとめる上で先行して学習していた部分があり、その学習コストを削減できたためである。

表 5.3: 予定と実績

工程	予定期間	実績期間
要件定義	2009 年 7 月 2 日～2009 年 7 月 31 日	2009 年 7 月 2 日～2009 年 8 月 7 日
外部設計	2009 年 8 月 3 日～2009 年 9 月 11 日	2009 年 8 月 10 日～2009 年 9 月 11 日
内部設計	2009 年 9 月 14 日～2009 年 10 月 9 日	2009 年 9 月 14 日～2009 年 10 月 9 日
実装	2009 年 10 月 13 日～2009 年 11 月 2 日	2009 年 10 月 13 日～2009 年 11 月 2 日
単体結合テスト	2009 年 11 月 4 日～2009 年 12 月 2 日	2009 年 11 月 4 日～2009 年 12 月 9 日
総合テスト	2009 年 12 月 3 日～2009 年 12 月 15 日	2009 年 12 月 10 日～2009 年 12 月 24 日

図 5.1: 予定と実績

5.4　プロジェクトの成果物

本プロジェクトは、ウォータフォール型の開発プロセスを採用し、各工程ごとに成果物を作成した。各工程における成果物を以下に示す。

要件定義工程

委託元からのヒアリングをもとに、現状の問題点や委託元が要求するシステム像を把握し、ユースケース図、アクティビティ図を作成した。ユースケース図は、ユーザから見たシステムの機能を表す図であり、本システムが提供する機能を記述した。アクティビティ図は、フローチャートのような動作の流れを記述するための図で、本システム導入前の研究プロセスと導入後に予想されるプロセスを記述した。これらの図を要件定義書としてまとめ上げ、要件定義工程を終了した。

外部設計工程

要件定義工程の成果物をもとに、クラス、属性などのクラス構造を表す分析クラス図、オブジェクト間のメッセージを表す分析シーケンス図を作成した。本システムは、データベースへのアクセスを伴うシステムであるためテーブル間の関係を記述した ER 図が必要となるが、外部設計段階ではきちんと定めることが難しいため、概念レベルの ER 図として作成した。XPath サブセット定義書は、XPath2.0 パーサ、SQL トランスレータの担当者が中心となり、XPath2.0 のフルセットから文法を定義した。また、外部設計工程での話し合いにより、シェル上でコマンドを入力し、データベースを操作するシステムとすることが決定した。オープンソースの XML データベースを参考にシェルで使用可能なコマンドを定義し、シェルコマンド定義書としてまとめた。

内部設計工程

物理 ER 図は概念 ER 図、設計クラス図は分析クラス図、設計シーケンス図は分析シーケンス図をもとにして、それぞれ作成した。また、設計クラス図、設計シーケンス図もとに、クラス定義書を作成した。クラス定義書は、クラスの概要、クラス内のフィールド・メソッド、メソッドの概要・引数・戻り値・処理の流れなどを詳細に記述した資料である。物理 ER 図からデータベースの関係スキーマが明らかになったので、コマンドを実行した際、SQL 問合せを SQL 定義書にまとめた。

実装工程

内部設計工程の成果物をもとに、各メンバーは自分が担当となっている部分をメインとして、実装を行った。成果物の一部が不完全な箇所があり、実装を進める上で悩んだ箇所が合ったが、メンバー間で話し合いを行い、実装を完了した。表 5.4 に本システムのソースコードの規模を示す。

表 5.4: ソースコードの規模

全体	約 1.4K Step
有効行	約 9k Step
コメント行	約 3k Step

納入時

以下のファイルを DVD でまとめ、委託元へ納入した。

- JAR ファイル
- サンプル設定ファイル
- README.txt
- ソースコード
- javadoc

JAR ファイルは、本システムを起動するために必要な class ファイルをまとめたファイルであり、サンプル設定ファイルには、起動時に必要なパラメータが記述されている。javadoc は、ライブラリのリファレンスとして HTML 形式で記述されたドキュメントである。

第6章　まとめ

本プロジェクトでは、天笠准教授からの委託により、XPath2.0 処理器「XPathProcceingPlatform」の開発を行った。要件に従い、関係データベースに XML 文書を格納するとともに、XML 問合せ言語である XPath2.0 のサブセットによる問合せを行い、問合せ結果を XML 文書に再構築する機能を実装した。

筆者が開発を担当した XPath2.0 パーサでは、XPath 式による問合せで使用できない文字が表れるという問題が発生した。しかし、スキャナの状態の遷移させることで問題を解決し、要求を満たすシステムを納入することができた。この問題は、筆者の不勉強が原因で起こった問題であり、未知なる技術に対する技術調査の重要性について改めて認識することとなった。また、この問題の影響により当初の開発計画から若干遅れたものの、委託元は納期の遅延についてそれほど気にとめていなかったため、本プロジェクトは成功したプロジェクトといえる。

今後、本システムはオープンソースとして公開することを目指す。研究者が、本システムを再利用することにより、短時間で実験に使用するシステムを開発し、従来より効率的な研究活動が行えるようになることを期待する。

謝辞

本報告書を作成するにあたり、研究開発プロジェクトの委託元教員として数々のご助言とご指導を頂いた天笠俊之准教授に深く感謝いたします。お忙しい中、本プロジェクトにおける個人的な相談にも丁寧に対応して頂き、ありがとうございました。

田中二郎教授には、指導教員としてだけではなく、「高度 IT 人材育成のための実践的ソフトウェア開発専修プログラム」の推進室長として、ご指導頂きました。本当にありがとうございました。

「高度 IT 人材育成のための実践的ソフトウェア開発専修プログラム」の専任教員でいらっしゃる菊池純男教授、駒谷昇一教授には、2 年間大変お世話になりました。講義のみならず、進路などでもご指導頂き、有意義な学生生活を送ることができました。両先生に感謝いたします。

また、本プロジェクトのチームメンバーである上田保祐君、佐用健君、羽鳥貴之君にもお世話になりました。私の至らない点を指摘して頂いたことで、自分自身を振り返ることができました。

最後に私の学生生活を支えてくださった方々、温かく見守っていただいた家族に心から感謝いたします。

参考文献

[1] 清水 敏之, 鬼塚 真, 江田 毅晴, 吉川 正俊, “ XML データの管理とストリーム処理に関する技術 ”, 電子情報通信学会論文誌：Vol.J90-D, No.2, pp. 159-184, 2007 年 2 月

[2] W3C: ”XML Path Language (XPath) 2.0 ”, http://www.w3.org/TR/xpath20/, W3C Recommendation 23 January 2007

[3] W3C: ”XML Path Language (XPath) Version 1.0 ”, http://www.w3.org/TR/xpath/, W3C Recommendation 16 November 1999

[4] W3C: ”XQuery 1.0: An XML Query Language ”, http://www.w3.org/TR/xquery/, W3C Recommendation 23 January 2007

付 録A　要件定義書

付 録B　XPathサブセット定義書

付 録C　シェルコマンド定義書

付 録D　設計シーケンス図

付 録E　クラス定義書