筑波大学大学院博士課程

システム情報工学研究科修士論文

関係データベースに基づく XPath2.0 処理器の開発 ―問合せ結果再構築について―

佐用　健

(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，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実施している「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」における，「研究開発プロジェクト」の成果をまとめたものである．

プロジェクトでは，システム情報工学研究科の天笠俊之准教授から委託を受け，4 人のチームで「XPath2.0 処理器の開発」を行い，その成果として XPPP (PathParserProcessingPlatform)を作成した．XPPP は， XML (Extensible Markup Language)データを関係データベースに格納し， XPath(XML Path Language)式を SQL 問合せ言語に変換し，XPath 式にて指定されたデータを XML データに再構築し出力する関係データベースに基づいた XML データベースである．

天笠准教授の研究では，XML データを関係データベースに格納するアルゴリズムや，XPath 式を SQL 問合せに変換するアルゴリズムの研究などを行っている．その際 XML データベース(XML データベース)を書き換えることが必要となってくるが，オープンソースの XML データベースは規模が大きすぎるため，書き換えが容易でない．また，研究室の先輩が開発した XML データベースを書き換えることも考えられるが，そのシステムは特定の研究分野に特化しており，書き換えが難しい．そこで各自が XML データベースを開発していた．そこで書き換えの容易な XML データベースの開発が求められた．

報告者の担当部分である「問合せ結果再構築」の機能は，受け取った問合せ結果である要素から，その子要素以下の要素を取得し，その関係表を XML データに再構築する機能である．関係表から XML データに再構築するには，一般的に用いられる方法として DOM(Document Object Model)が挙げられる．しかし DOM は XML ツリーをメモリに展開してから出力する方式なので，メモリが枯渇してしまうことから大容量のデータには対応できない．この問題を解決するため SAX(The Simple API for XML)フィルターである XMLWriter クラスを用いた．このクラスは XML データをストリーミング処理するので，大容量データを出力する際に，メモリ容量を必要とせず処理できるという特徴がある．

本報告書では，チームとして共通部分である，XPPP 開発の計画，およびその概要と拡張性を意識した設計について述べた後，報告者の担当部分である「問合せ結果再構築」の実現方法の詳細について報告する．

目次

図目次

表目次

第1章　はじめに

XML は当初，構造化文書を記述するためのメタ言語として誕生したが，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，文書のみならずデータの表現形式としても急速に普及した[1]．よって，今後 XML データが増大すると予想される．

そこで，そのデータを管理するためのデータベースが重要となってくる．それらの膨大なデータを長期間維持・管理・活用していくためには，より使いやすく，かつ柔軟なデータ管理方式が必要になる．現在はデータの管理方法として関係データベースが一般的に用いられているが，関係データベース はシンプルで使いやすい反面，文書データなどの非構造データの扱いや，利用するアプリケーションやデータ構造の変化に対し，柔軟に対応できないといった欠点も挙げられる．これらの欠点を補う新しいデータベース管理方式として，XML データベースが期待されている．

XML データベースは，厳密なスキーマ定義が必須ではないため，関係データベース に比べてデータ構造の拡張性が高く，データベース設計の途中の変更にも柔軟に対応できる．また，アプリケーションデータの XML データ化などの影響から，XML データが増加しているため，XML データベースの普及・研究開発はますます加速するものと思われる．

研究開発プロジェクトの教員提案テーマの一つとして，天笠俊之准教授からソースコードの書き換えが容易で，信頼性のある XPath2.0 処理器を開発してほしいとの委託があった．天笠准教授は XML データを関係データベースに格納するアルゴリズムや, XPath 式を SQL 問合せに変換するアルゴリズムなどの研究を行っている．研究を行う際は XML データベースに変更を加えることとなる．既存の XML データベースを書き換えるには以下のような問題点があった．すなわち，オープンソースの XML データベースは規模が大きすぎ，システムの全体を把握することが難しく, 変更を加えにくい. また, 研究室で過去に開発された XML データベースを書き換える方法もあるが，このシステムは特定の研究分野に特化しており変更を加えにくく，また信頼性の保証もできない．現状では各自が XML データベースを開発している，しかしそのシステム開発に時間がかかりすぎること，信頼性が保証できないことが問題となっている．

そこで報告者を含む 4 人のチームで，関係データベースを用いた XPath2.0 処理器，XPathProcessingPlatform (XPPP)の開発を行った. XPPP を用いて研究者は XML データを関係データベースに格納したり，関係データベースに格納されている XML データを XPath 式で問合せ，その問合せ結果を XML データとして受け取ることができる．画面は CUI(コマンドラインインタフェース)を提供している. コマンドは既存の XML データベース(ネイティブ XML データベース)と同様にしている．

システムは，以下の四つの機能から構成されている．XML データを関係データベースに格納できるよう変換し，関係データベースに格納する機能(XML データ格納)，XPath 式を構文解析し，中間表現にする機能(XPath2.0 パーサ)，中間表現を SQL 問合せに変換し問合せを行い，XML ノードを問合せ結果として取得する機能(SQL トランスレータ)，問合せ結果である XML ノードからそのノードよりも階層が下であるノード(子孫ノード)を取得し, その問合せ結果を XML データに再構築し出力する機能(問合せ結果再構築), これらの機能構成で

ある．

その機能の実装の容易性を考慮した結果，本システムでは XPath2.0 フルセットではなく，独自に定義した XPath2.0 サブセットを実現することとした．サブセットには，XPath2.0 の中核部分であるパス式の解釈と述語が解釈できるようになっている．これらは実装の容易性だけでなく，実用性も兼ねたサブセットを実現している．

XPPP を開発するに際して，ウォーターフォール開発モデルを採択した．信頼性のあるシステムを開発するためであり，また「PBL 型システム開発」の授業の復習も兼ねている．要件定義工程から外部設計工程まではシステム全体を全員で進めたが，内部設計工程より四つの機能をチームメンバー人がそれぞれ責任を負って開発することとした．

本報告書ではまず XML や XPath2.0 など本システムに関わる技術要素を説明する．次いでチームプロジェクトの計画，進め方などを説明する．次いで本システムの概要を説明した後担当部分の実現方法を説明し，プロジェクトを振り返り，全体のまとめを記載する．

第2章　XML について

本章では，本報告書で対象とする XML とその問合せ言語である XPath，XML データを扱うデータベースシステムである XML データベースなどの技術要素について説明する．

2.1　XML と XML データベース

XML [2]は，1998 年に W3C(World Wide Web Consortium)によって勧告された，データの記述方法である．マークアップ言語を記述するメタ言語であるため，文書内に付加情報を与える目印（タグ）を付けることができる．

マークアップ言語である HyperText Markup Language(HTML) では，決められたタグしか用いることができないが，XML では，ユーザが自由にタグを作ることができるため，データに様々な意味を持たせることができる．更に，タグ内には属性を記述することができ，データに付加的な情報を加えることができる．また，データをテキスト形式で表現しているため，マルチプラットフォーム環境でのデータ交換・処理に適している．

図 2-1 に XML データの構造を説明するための図を示す．

```
<catalog>
    <book category = ‘recommendation’>
        <title>XML</title>
        <author>W3C</author>
    </book>
    <book>
        <title>XPath</title>
    </book>
</catalog>
```

図 2-1　サンプル XML データ

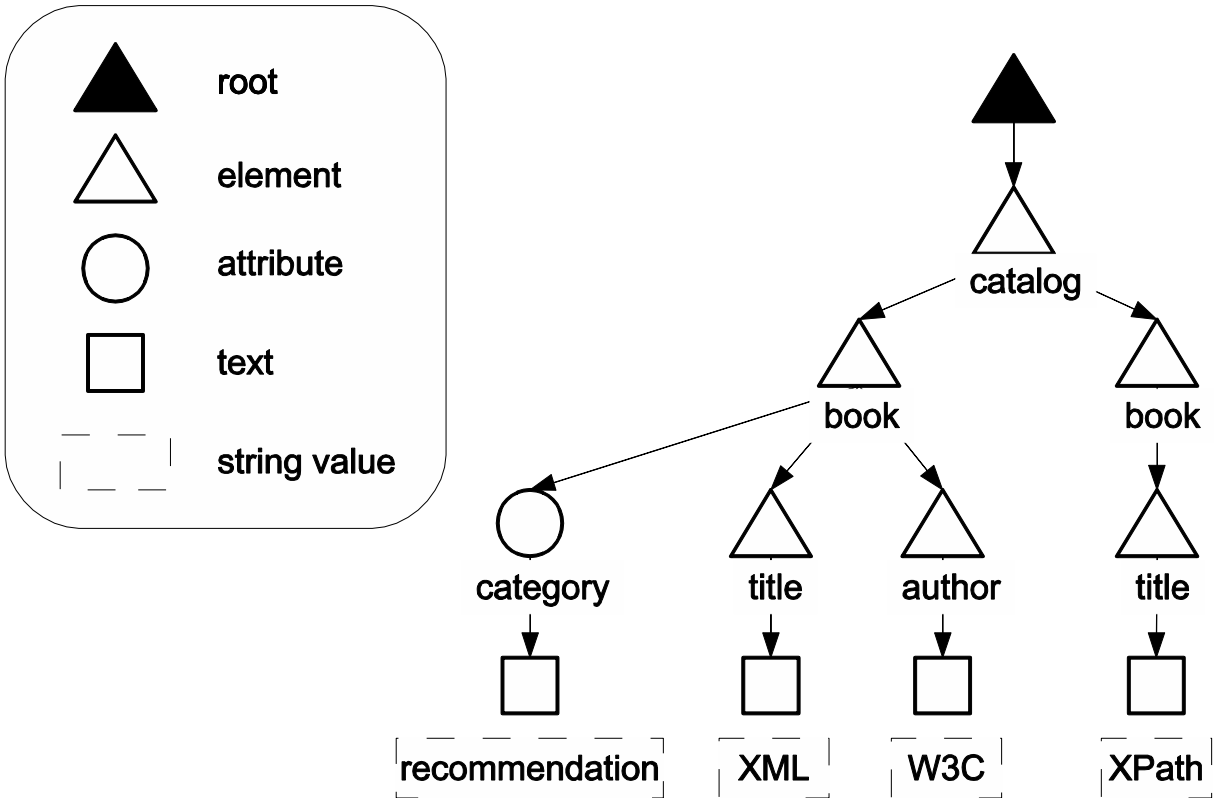

図 2-2　XML データの木構造

XML データの構造を説明する．まず，XML データは開始タグと終了タグがネスト構造をとっており，つまり木構造となっている．図 2-1 の XML データを木構造に展開した図が図 2-2 である．

XML データにはルートノードと呼ばれる最上位のノードが必ず存在する．図 2-1 の要素である<catalog>がルートノードとなる．ルートノードの開始タグは，いかなる他のタグよりも早く現れ，ルートノードの終了タグは，いかなる他のタグよりも後に現れる．XML データは，最低でも 1 つの要素を持たねばならないが，それがルートノードである．以下に XML データを構成する要素，属性，名前空間，テキストについて説明する．

2.1.1　要素

XML データにおいて，もっとも基本となる情報単位が要素である．

要素は開始タグと終了タグでテキスト(内容)を囲うことで表現する．開始タグは<要素名>，終了タグは</要素名>で記述する．要素にはテキストを持つ通常の要素と，テキストを持たない空要素がある．空要素は<要素名></要素名>もしくは<要素名/>で表す．

また，要素は内部に子要素を含むことができ，そうすることで XML データは木構造を取っている．

2.1.2　属性

属性は，XML データの中で，要素に対して付加的な情報を付け加えるために使用される．付加される情報は名前と値のペアで表現され，開始タグの中に記述される．<book category=’recommendation’>このように，属性の値はシングルクォートかダブルクォートで括らなくてはいけない．

なお，要素と属性には，属性が要素に従属している，という関係がある．

一つの要素内に複数個属性を挿入することができるが,そうする上で注意すべき点がある．それは同じ名前の属性を一つの要素に挿入できないことである．例えば，<book category = ‘recommendation’ genre = ‘English’>は正しい使い方であるが，<book category = ‘recommendation’ category = ‘English’>は誤った使い方なので注意が必要である．

2.1.3 XML 名前空間

XML 名前空間(Namespaces in XML[3])は，1999 年に W3C によって勧告された，XML データの中で使われる要素や属性の名前を，URI 参照により特定される名前空間に結びつけることにより修飾するための単純な方法を提供するものである．

名前空間 URI で要素を修飾することにより，複数の語彙を混在させることが可能となる．例として title という要素の名前を取り挙げる．この title はある XML データの中ではタイトルという意味を持たせるよう使われている．しかし，ある XML データの中では肩書きという意味で用いられていることがある．該当するタグにはどのような意味があるかを指し示すもの，それが XML 名前空間である．

```
<body xmlns:book=‘http://Ex.com/ns/book/’
      xmlns:prof=‘http://Ex.com/ns/profile/’>
 <book:title>XPath2.0</book:title>
 <prof:title>CEO</prof:title>
<body>
```

図 2-3 名前空間を持つ XML データ

なお，要素と XML 名前空間には，XML 名前空間が要素に従属している，という関係がある．

XML 名前空間を使用する際にはまず XML 名前空間を宣言する必要がある．図 2-3 を用いて説明する．名前空間は属性と同様，要素に挿入する形で宣言する．’xmlns’を前に修飾し，その後に接頭辞である’book’の名前を挿入し，その値として名前空間 URI を挿入する．

次いで，XML 名前空間と関連付けたい要素，あるいは属性の名前に対して，関連付けたい URI と結びついた接頭辞を修飾する．こうすることで XML 名前空間が結びつくこととなる．

接頭辞を用いて，名前空間を宣言したが，接頭辞なしでも名前空間を宣言することが可能である．この名前空間をデフォルト名前空間という．名前空間が有効な範囲では，接頭辞がない要素に関しては，常にデフォルト名前空間が結びついていることとなる．

次いで，XML 名前空間の有効範囲について説明する．接頭辞付き名前空間は要素，ならびに属性に対して有効であり，その有効範囲は宣言された要素以下の子要素内で用いることができる．デフォルト名前空間も宣言された要素以下の子要素内で用いることができるが要素にのみ有効であり，属性には影響を及ぼさない．

XML 名前空間を使用する際は一つ注意点がある．同一要素内における複数個の名前空間宣言である．同じ接頭辞を二回以上宣言してはならない，デフォルト名前空間を同じ要素内

で二回以上宣言してはならない，同じ名前空間 URI を別の接頭辞を用いて，同じ要素内に宣言してはならないといった規則である．例えば図 2-3 のルートノード内の宣言は許されるが，<xmlns:book='http://Ex.com/ns/book/' xmlns:book='http://Ex.com/ns/book2/'> は許されない宣言である．

2.2 XPath2.0

2.2.1 XPath2.0 概要

XPath2.0[4]とは，2007 年に W3C によって勧告された，XML 文章の中の要素や属性の位置を指定するための言語である．

XPath は，XML データを木構造としてモデル化し，要素や属性への位置を指定できるようする．また，指定したノードに対して，条件判定を行ったり，文字列計算をしたり，計算結果を使用したりすることも可能である．

基本的な記述方法は，軸方法::ノードテスト[述語]/～～となる．これをロケーションパスといい，'/'で区切られた一つの記述をロケーションステップと呼ぶ．

ロケーションパスは'/'ではじまるものを絶対パス，始まらないものを相対パスとして展開する．絶対パスではルートノード(後述する)がコンテキストノードとなり，相対パスでは基準ノードがコンテキストノードになる

なお，コンテキストノードとは XPath で指定されたノードを意味する．ノードについては後述する．

2.2.2 ノード

XPath では，XML データをノードという単位に分解し，木構造に展開する．ノードには以下のような種類のノードが存在する．

- ルートノード：全てのノードを包括するノード．通常，ルートノードは 1 個の要素ノードを子に持つ
- 要素ノード：マーク付け文書の要素に当たるノード．属性ノードとテキストノードは，必ず要素ノードを親に持つ
- 属性ノード：要素ノードが示す要素の内容のうち，属性に当たるノード
- テキストノード：要素ノードが示す要素の内容のうち，他のノードに属さない平文に当たるノード

2.2.3 軸方向(Axis)

軸方向とはコンテキストノードから対象となるノードへの樹関係を示すものである．頻繁に用いられる軸方向には略記法が定義されている以下のような種類の軸が存在する．

- ancestor-or-self 軸：コンテキストノード自身とその先祖．'ancestor-or-self::'と表記する
- ancestor 軸：コンテキストノードの先祖．'ancestor::'と表記する
- parent 軸：コンテキストノードの親．'parent::'と表記する．'parent::node()'は'..'と省略も出来る
- preceding 軸：コンテキストノードより先に記述されたノード（兄や伯父を含む）但し先祖，属性ノードは除く．'preceding::'と表記する
- preceding-sibling：コンテキストノードの兄（自身より先にある兄弟ノード）但しコンテキストノードが属性の場合は空．'preceding-sibling::'と表記する
- self 軸：コンテキストノード自身．'self::'と表記する．'self::node()'は'.'と省略出来る

- following-sibling 軸：コンテキストノードの弟（自身より後にある兄弟ノード）但しコンテキストノードが属性又は名前空間ノードの場合は空．’ preceding-sibling::’と表記する
- following 軸：コンテキストノードより後に記述されたノード（弟や叔父を含む）但し子孫，属性ノードは除く．’following::’と表記する
- child 軸：コンテキストノードの子供．’ child::’と表記する．ロケーションステップでは軸方向を省略すると"child"が自動的に補完される．
- descendant 軸：コンテキストノードの子孫．” descendant::”と表記する
- descendant-or-self 軸：コンテキストノード自身とその子孫．’descendant-or-self::’と表記する．"/descendant-or-self::node()/"は"//"と省略出来る．
- attribute 軸：コンテキストノードの属性ノード．コンテキストノードが要素ノードでなければ空．"attribute::"は"@"と省略出来る．

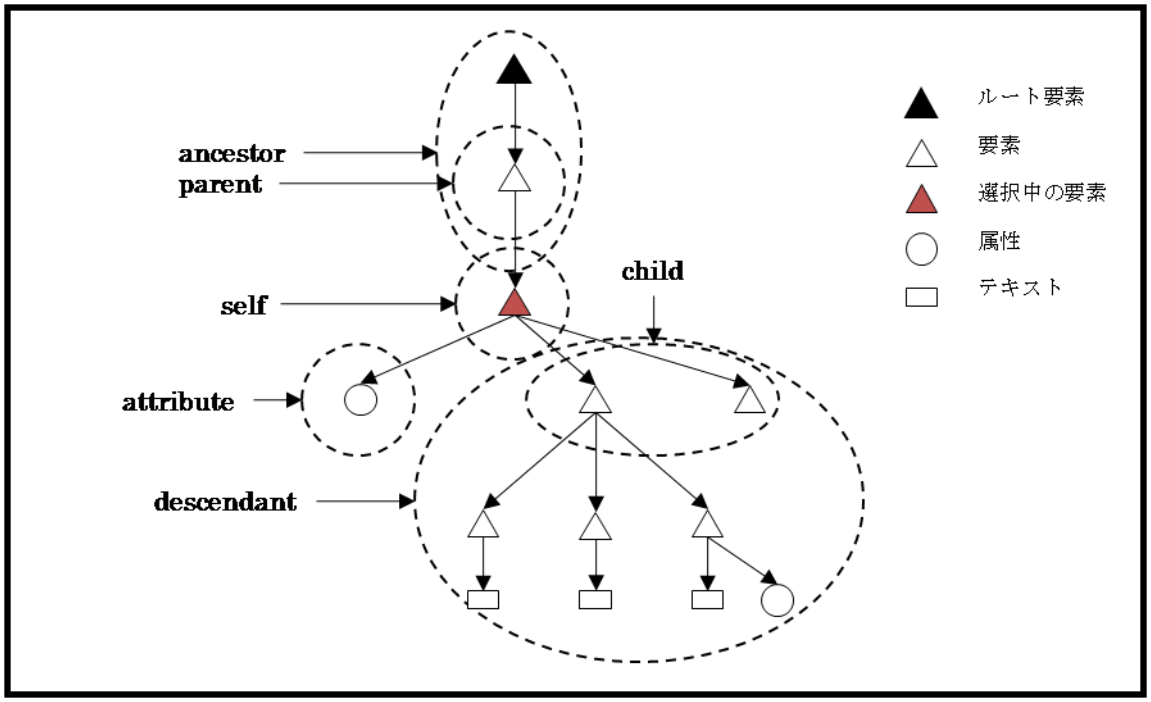

図 2-4　DOM ツリーを用いた軸解説図

2.2.4　　ノードテスト

ノードテストとは軸方向から該当するノードを指定するものである．以下の記法が存在する．

- node()：全てのノード
- *：全ての主ノード．’attribute::’であれば属性，それ以外であれば要素ノード
- text()：テキストノード

2.2.5　　述語

述語とは，ノードテストで選出されたノード集合を，さらに振り分けする条件を記述する部分である．評価には式を用い，式の値が真のノードのみが選出される．

述語には式と関数が記述される．以下のような式がある．

・ ロケーションステップ：ロケーションステップは式としても使える
・ ブール値：論理演算子(and，or)，算術演算子(<，<=，=，!=，=>，>)
・ 数値演算子：算術演算子(+,-,*,/,mod)
関数にはノード集合関数，ブール値関数，数値関数といったものがある．

以上の XPath に関連する知識を踏まえたうえで，具体例を用いて XML データに対して XPath の問合せを行う過程を説明する．

例えば図 2-5 のような XML データで，XPath 式

/drink/coffee[@id="1"]

で問い合わせると，まずロケーションステップの/drink/coffee は/child::drink/child::coffee であるので，要素 drink の子供である，coffee が選択される．次に述語の部分において，属性名 id の値が 1 である要素が選択されることとなる．よって属性の名前 id の値が 1 である要素 coffee が選択されることとなる．

```
<drink>
  <coffee id="1">
    <price>150</price>
  </coffee>
  <coffee id="2">
    <price>130</price>
  </coffee>
</drink>
```

図 2-5　XPath 式によって選択される XML データ

2.3　XML データベース

XML データベースとは，XML データを扱うための機能を持つデータベースである．

格納の対象となるのはXML データそのものである．格納された XML データに対して，XPath や XQuery[5]などの問合せ言語で問合せを行う．

XML データベースは, XML の特長を生かすため, 関係データベース では困難とされた，データ構造の変更を可能としている．これにより，データベースの詳細な論理設計が不要であり，設計を途中で変更できるため，非常に柔軟性・拡張性に富んだデータベースだと言える．さらに，昨今のシステム開発の現場では，アジャイル開発手法を採用する場合も増えてきており，XML データベースの柔軟性や拡張性はこのような反復的なシステム開発に適している. 以上の事から, 今後 XML データベースの開発と利用が広がっていくと予想される．

2.3.1　ネイティブ XML データベース

ネイティブ XML データベースとは，XML データをその構造のまま格納・操作を行うことができる XML データベースである．それにより XML が本来持つ木構造，メタ情報管理と

いう優位性を最大限活用することができる．

ネイティブ XML データベースの構造について説明する．関係データベースやオブジェクトデータベースなどの既存のシステムは基本的に構造化されたデータを念頭に設計されている．これに対して XML データは半構造性を持っており，ゆるやかな構造は持つものの，それは可変的である．このため，既存の技術をそのまま応用するだけでは十分とは言えない．XML データの性質はアリケーションによって決まるので，アプリケーションの要求に応じて，適切な処理手法を選択することが重要である

2.3.2 関係 XML データベース

関係 XML データベースとは既存の関係データベースを用いて XML データを格納・操作するデータベースである．

データを扱うに当たって関係データベースを用いるメリットについて説明する．まず，技術の蓄積が豊富でありシステムの信頼性が高いことが挙げられる．関係データベースは開発が始められてから 30 年以上に渡っている．次いで，そのシステムの数と種類が多いことである．世界中に普及しており，エンタープライズ系の製品からオープンソースのものまで開発が行われている．次いで，既に関係データベースが普及していることから，他の関係データベースと連携が取りやすいと言ったことが挙げられる．

そのように既に確立されたシステムであるが，XML データを扱う際には考慮すべき点がある．というのも，XML データは木構造であるが，関係データベースに格納されるデータは関係表という違いがあるからである．よって木構造から関係表へのマッピングと，関係表に置いても要素の親子関係を維持する必要がある．

2.3.3 木構造データから関係表へのマッピング

XML データを関係表に格納する最も単純な方法は，関係データベースが提供する可変長文字列の機能を用いて，XML データを長大な文字列として単一のカラムに格納する方法である．しかしこうした場合，データの出し入れの単位は XML データそのものとなってしまう．したがって，XPath や XQuery を用いて，文書全体よりも粒度の細かい要素や属性の抽出を行う検索を処理することができない．それらを扱うには関係データベース自身の機能を用いることができず，専用の索引機構が必要になってしまう．

XML データをそのまま文字列として扱うのではなく，関係表とのマッピングを考える手法は，大きく構造写像アプローチとモデル写像アプローチの二つに分類することができる．構造写像アプローチは，個別のＸML データの構造（スキーマ）に基づいて関係スキーマを設計する方法である．一方，モデル写像アプローチは，XML データモデルに基づいて関係スキーマを設計する手法である[6].

2.3.4 ノードラベリング

ノードラベリングとは XML データを関係表に格納する際, XML ノードにラベルを付与することである．このラベルを用いて XML ノード間の親子・先祖・子孫関係を関係表の中でも保持している．

ラベルの付与の際，属性と XML 名前空間に対してどのようなラベルが振られるかを説明する．2.1.2，2.1.3 節で述べたように，同じ要素内に置いて存在する属性，XML 名前空間は要素に従属しているとみなされるため，要素と同じラベルが付与される．

ノードラベリングの重要性を説明する．XPath や Xquery を用いて関係データベースに格納されるがその際に，どのようなノードラベル手法が用いられるかによって，検索の速度が

変わってくる．また，ノードラベリングは XML データの更新にうまく対応できていない．というのは，データの追加，削除，更新などの操作が行われると，大々的なラベルの付け替えを行わなければならない. ノードラベリングの必要性の大きさと, 課題点があることから，ノードラベリング手法の研究が活発に行われている．

ノードラベリングには代表的な手法として二つの手法がある．PrePostOrder と DeweyOrder である．PrePostOrder は一番単純な範囲ラベルであり，木構造を深さ優先探索し，行きがけと帰りがけに PreOrder と PostOrder の対をラベルとして付与するものである．PreOrder の値は，XML データを格納する際，ノードの開始タグが読み込まれた時点で付与される．PostOrder の値は，ノードの終了タグが読み込まれた時点で付与される．

図 2-6 に示すサンプルデータに対して, どのようなラベルを付与するかを, 図 2-7 に示す.

```
<catalog>
    <book>
        <autohr>W3C</author>
        <title>XML</title>
    </book>
    <book>
        <title>XPath</title>
    </book>
</catalog>
```

図 2-6　ノードラベリング説明用 XML データ

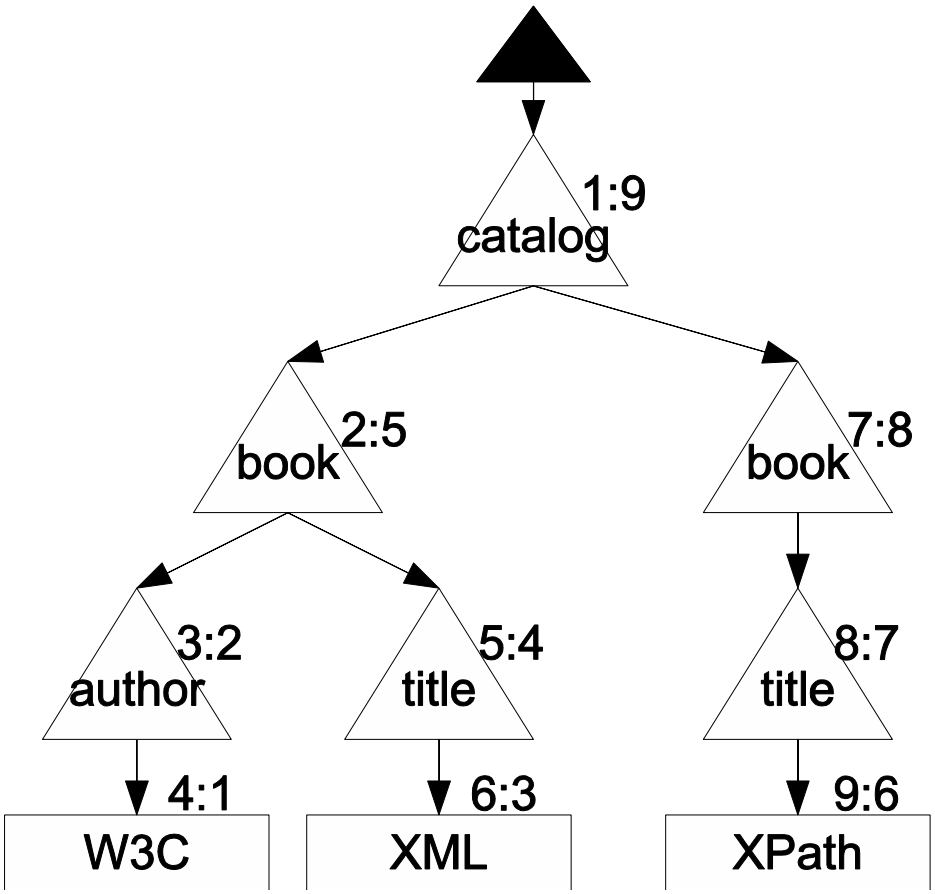

図 2-7　PrePostOrder のラベル

DeweyOrder は一番単純な接頭辞ラベルであり， 本の章立てのように同じ親を持つ XML ノードに順番を付け，親のラベルと結合するというものである．

DeweyOrder では，ノードの先祖子孫および親子関係はラベルが部分一致するかどうかで判定できる．また，デューイオーダの辞書式順序によって，文書順と兄弟判定も可能になる．

図 2-6 に示すサンプルデータに対してどのようなラベルを付与するかを図 2-8 に示す．

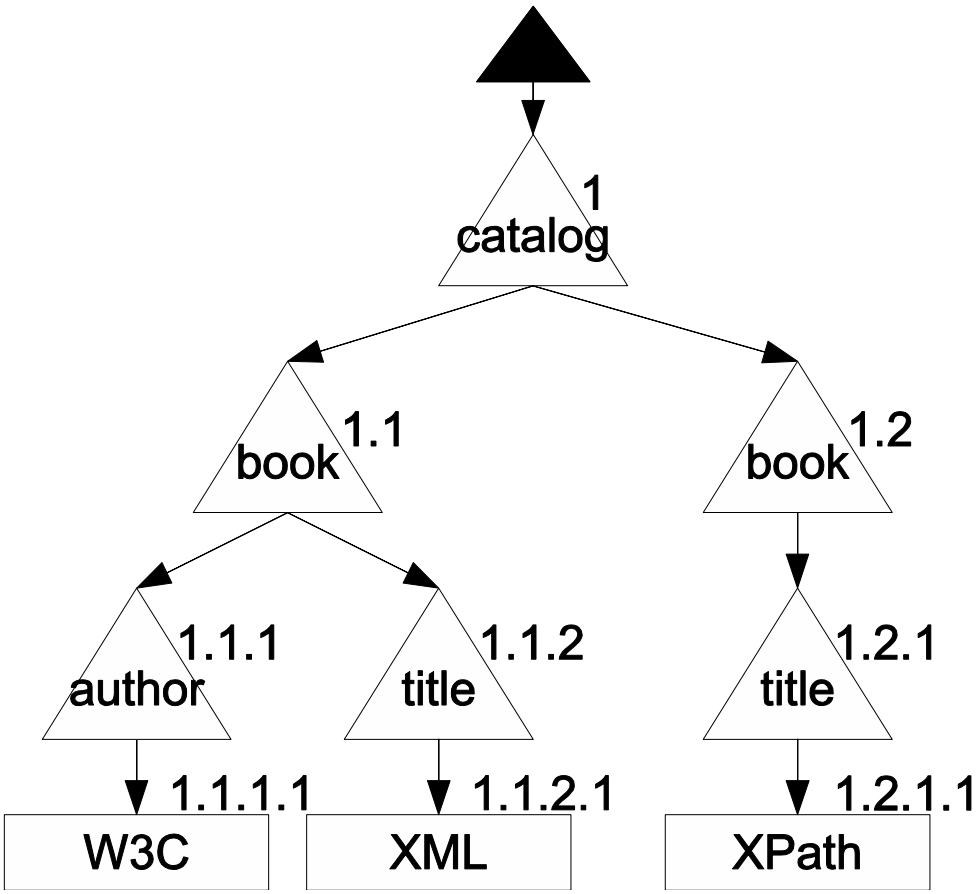

図 2-8　DeweyOrder のラベル

2.4　DOM

DOM(Document Object Model)は 1998 年に W3C により勧告された XML パーサにアクセスするための API の一つである. W3C が公式に公開した唯一の API である. DOM には, Level 1，2，3 などいくつかのレベルがある．レベルが大きい方が，より新しく高機能である.

DOM は，XML データを DOM ツリーと呼ばれるツリー構造として扱う．そのため，XML パーサが XML データ全体を読み込んだ後でなければ, アクセスすることができない. また, DOM ツリーは通常，メモリ上に展開されるため，大規模な XML データを処理する場合には，その分メモリ容量も要求される．その代わり，DOM を利用すれば，XML データの順番に関係なくアクセスできる.

XML パーサとは，XML データを，アプリケーションソフトが利用しやすい形に変換するソフトウェアである．変換時に，XML データが文法に照らして正確に記述されているかどうかを同時に検証する.

2.5　SAX

Simple API for XML (SAX)とは，アプリケーションソフトが，XML データの解釈・検証を行なう「XML パーサ」の標準の一つである．DOM と並んで最も広く利用されている API の一つである.

XML データを一つの木構造に変換する DOM と違って，XML データを先頭から一行ずつ順に読み込んで，要素が現れる度に対応する処理手順を呼び出すという方式を用いているため，巨大な XML データを扱ってもメモリ容量を必要とせず高速に処理できるという特徴がある．

第3章　開発計画

本章では，チームプロジェクトについて説明する．具体的にはプロジェクトにおける開発体制，初期の開発計画や成果物の位置づけ，システム分担について説明する

3.1　開発体制

プロジェクトでは，表 3.1 に示すメンバで構成されるチームによって開発を行った．

委託元である天笠准教授とは，具体的な要件確定と進捗状況の確認，および技術的なアドバイスを頂くため，定期的なミーティングを行い，共同で開発を進めた．

それぞれのメンバのチーム内の役割分担も決めた．役割としてはプロジェクトマネージャー(PM)，渉外，議事録，ミーティングの議事進行，の 4 つに分け，4 人で工程毎にローテーションしていくこととした．これは，システム，ないしチームの全体像を把握することの大切さ，技術を，全員が PM を体験，取得できるようにするためである．

3.2　初期開発日程

プロジェクトの初期スケジュールを，図 3-1 に示す．天笠准教授が「PBL 型システム開発」の授業で学んだことを生かしてほしいとの要求からウォーターフォールモデルで開発を行った．

スケジュールにおいて，外部設計工程とテスト工程の期間を他の工程よりも長期間行うスケジュールを立てた．外部設計工程の期間を長く設定した理由として，XML や XPath などに関する知識が乏しかったため，技術調査に時間を割くことが必要と考えたためである．テスト工程の期間を長く設定した理由として，信頼性の高いシステムを開発してほしいとの天笠准教授からの要求があったためである．

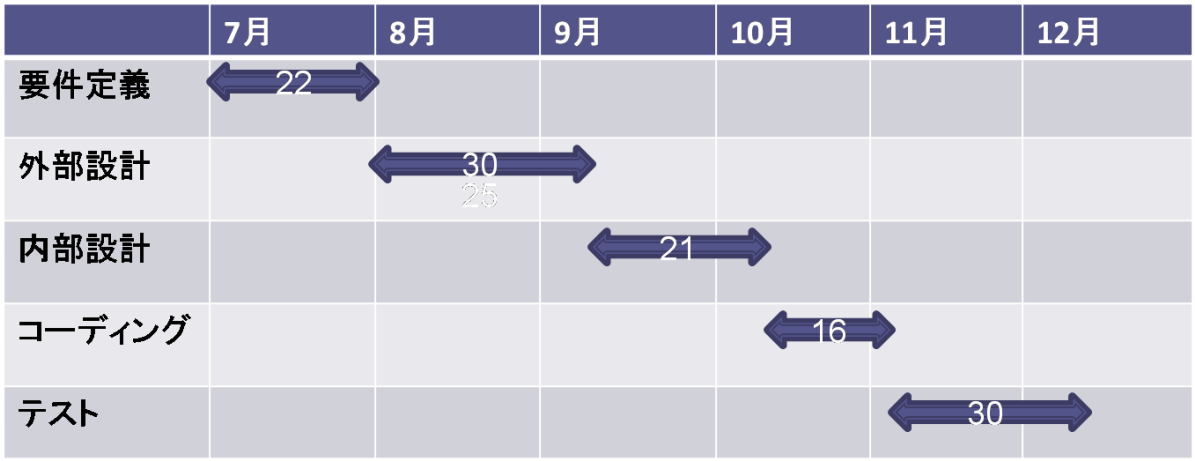

図 3-1　初期開発スケジュール

3.3　成果物の位置づけ

本システムを作成するにあたって，作成される成果物の位置づけについて説明する．

本来ウォーターフォールモデルによりシステム開発を行う場合，委託元との契約を明確にするため，要件定義行程では要件定義書，外部設計工程では外部設計書，内部設計工程では内部設計書などである．

本システムの開発において,委託元である天笠准教授とのミーティングを通して成果物は,本文書作成と開発システムを仕上げることが必須項目となり，工程ごとの締めは，我々自身で決定することとした．しかし，要件漏れを防ぐために要件定義書は作成し，天笠准教授とレビュを行い，同意を得た．外部設計書，内部設計書は作成していない．

本文書の巻末に，要件定義書，追加要件定義書，クラス図などを添付した．

3.4　システムの分担について

プロジェクトメンバのそれぞれの分担を表したものを表 3-1 に示す．

表 3-1　システム分担

担当者	担当	説明
上田	XML データ格納	XML データを関係データベースに格納する部分
佐用	問合せ結果データ再構築	SQL 問合せ結果を XPath 式の結果に変換する部分
田中	XPath2.0 パーサ	XPath 式構文解析する部分
羽鳥	SQL トランスレータ	XPath 式の構文解析結果を SQL 問合せに変換する部分

第4章　　システムの使い方

本章では開発システムを利用面から説明する．具体的にはシステムの概要からムの概要，要件，ソフトウェア構成，システムの使い方について説明する．

4.1　　システムの概要

我々が開発したXPath2.0処理器であるXPathProcceingPaltformの概要を図4-1に示す．

利用者は，CUI(コマンドラインインタフェース)を用いて，登録したい XML データをシステムに格納したり，XPath2.0による問合せを行うことができる．

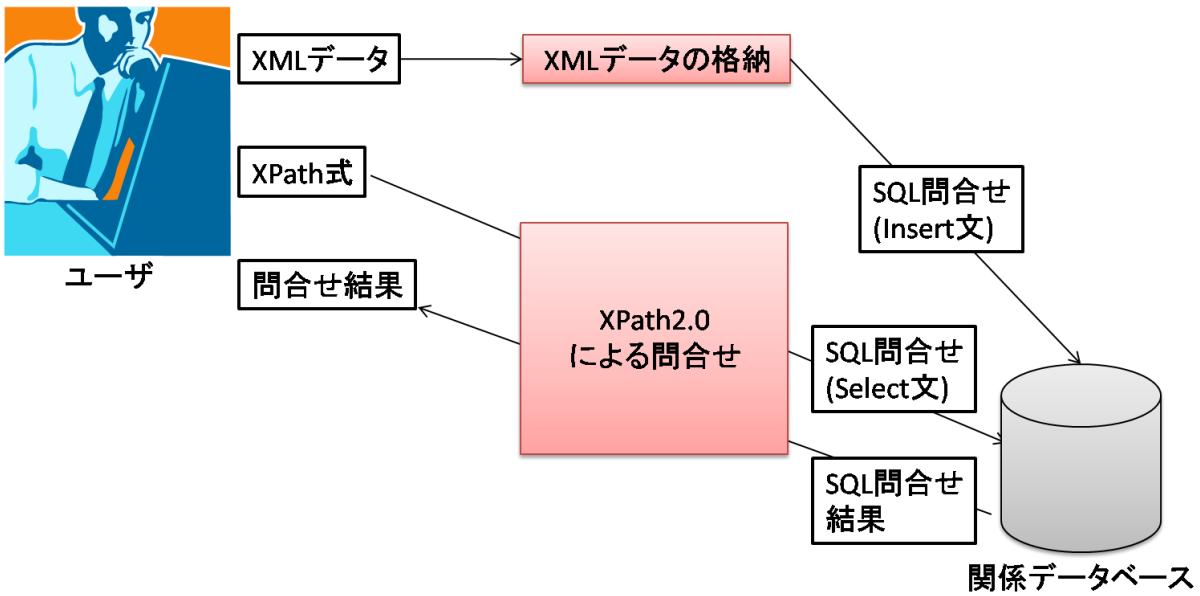

図 4-1　システム概要図

4.2　　システムの利用手順

システムの利用手順について説明する．

① XML データベースに関する新たなアルゴリズムを考案する
② 考案したアルゴリズムに基づき本システムに変更を加える
③ 設定ファイルを作成し，システム起動時に読み込ませる
④ XML データを本システムを用いて関係データベースに格納する
⑤ XPath 式で格納されたデータに対して問合せを行う

4.2.1　　設定ファイル

システムを起動する際設定ファイルを読み込ませる必要がある．この設定ファイルの中身について説明した表を表に示す．

表 4-1　設定ファイル

変数名	説明
db_url	JDBC ドライバの URI を代入する．値が間違っていたり，代入されなかった場合はエラーとなる
db_user	データベースの ID を記入する．値が間違っていたり，代入されなかった場合はエラーとなる
db_password	データベースのパスワードを記入する．値が間違っていた場合は
algorithm_id	ノードラベリング方法を記入する．データベースがシステムによって自動生成される場合は，代入された値によってテーブルの構造をノードラベリングに対応させた形となる．代入されなかった場合はエラーとなる
prompt	プロンプトの表示を代入する．値が与えられなかった場合はエラーとなる

4.2.2　コマンド一覧

本システムはユーザに CUI を提供する．CUI コマンドの一覧を表 4.4.1.1 に示す.

表 4-2　コマンド一覧

コマンド名	コマンドの説明
find	XPath 式を評価し，問合せ結果を表示する
help	システムで使用できるコマンドの一覧とその説明を表示する
ls	引数としてコレクション名が与えられればそのコレクションに格納された XML ファイルのファイル名の一覧を表示する．引数がなければ，データベースに存在するコレクション名の一覧を表示する
lsns	引数として与えられたコレクション名の中に存在する XML 名前空間の情報を一覧表示する
put	XML ファイルをデータベースに格納する．ファイルのパスは相対・絶対パスともに使用できる．格納するコレクションも指定する
mkcol	引数の名前のコレクションを作成する
quit	システムを終了する
rm	指定された XML ファイルを削除する
rmcol	コレクションを削除する．コレクションの中にデータがある場合は，削除できない．オプションを用いると削除できる

4.2.3　コマンドの使い方と実行結果

find コマンド

find コマンドは，CUI に入力された XPath 式に対する問合せ結果を表示するコマンドである.

XPath 式で使用することのできる軸は child, descendant, descendant-or-self, namespace, attribute, self, parent, ancestor, ancestor-or-self である．述語において使用できるのは論理演算子(and，or)，算術演算子(＜，＜＝，＝，！＝，＝＞，＞)，関数(fn:compare ，fn:contains, , fn:not)である.

なお，XPath 式において名前空間と関連づいたパス式を用いる場合には名前空間宣言を行う必要がある.

・ 名前空間と関連のないパス式を用いる場合

コマンドの記法	find collection("*コレクション名*")*XPath 式*
コマンド例	find collection("SAMPLECOLLECTION")/catalog1
コマンド例の実行結果	# It took 0seconds for evaluating collection("SAMPLECOLLECTION")/catalog1,found 2 nodes. <catalog1 xmlns:NS1="http://www.NS/"> <book category1="recommendation"> <NS1:title>XML</NS1:title> <author>W3C</author> </book> <book> <title>XPath</title> </book> </catalog1> # /catalog1 has 6nodes in sample1.xml. <catalog1 xmlns:NS2="http://www.NS2/"> <book category2="recommendation"> <NS2:title>XML</NS2:title> <author>W3C</author> </book> <book> <title>XPath</title> </book> </catalog1> # /catalog1 has 6nodes in sample2.xml. # It took 0seconds for outputting collection("SAMPLECOLLECTION")/catalog1.

・ 接頭辞つき名前空間と関連づいたパス式を用いる場合

コマンドの記法	Find “declare namespace 接頭辞名 = ¥”名前空間URI¥”;collection (¥"*コレクション名*¥")*XPath 式*”
コマンド例	find "declare namespace NS1 = ¥"http://www.NS/¥”;collection(¥"SAMPLECOLLECTION¥")/catalog1/book/NS1:title
コマンド例の実行結果	# It took 0seconds for evaluating declare element namespace NS1 = "http://www.NS/";collection("SAMPLECOLLECTION")/catalog1/book/NS1:title,found 1 node. <NS1:title>XML</NS1:title> # /[http://www.default/]catalog1 has 1node in sample1.xml. # It took 0seconds for outputting declare element namespace NS1 = "http://www.NS/";collection("SAMPLECOLLECTION")/catalog1/book/NS1:title.

・デフォルト名前空間と関連づいたパス式を用いる場合

コマンドの記法	Find “declare default element namespace 接頭辞名 = ¥”名前空間 URI¥”;collection (¥"*コレクション名*¥")*XPath 式*”
コマンド例	find "declare default element namespace ¥"http://www.default/¥";collection(¥"SAMPLECOLLECTION¥")/catalog1"
コマンド例の実行結果	# It took 0seconds for evaluating declare default element namespace "http://www.default/";collection("SAMPLECOLLECTION")/catalog1,found 1 node. <catalog1 xmlns="http://www.default/" xmlns:NS4="http://www.NS4/"> <book category3="recommendation"> <NS4:title>XML</NS4:title> <author>W3C</author> </book> <book> <title>XPath</title> </book> </catalog1> # /[http://www.default/]catalog1 has 6nodes in sample3.xml. # It took 0seconds for outputting declare default element namespace "http://www.default/";collection("SAMPLECOLLECTION")/catalog1.

名前空間が複数個関連づいたパス式を用いる場合は，その個数分だけ名前空間宣言が必要となる．デフォルト名前空間一つと接頭辞つき名前空間一つが関連づいたパス式を用いて問合せを行う場合の XPath 式のコマンドの記法を記載する

find “declare default element namespace 接頭辞名 = ¥”名前空間 URI①¥”; declare namespace 接頭辞名 = ¥”名前空間 URI②¥”;collection (¥"*コレクション名*¥")*XPath 式*

なお，実行結果には XPath 式の問合せ結果だけでなく，ノード数などシステムが付加したデータも表示している．区別がつくよう，付加したデータには先頭に’＃’を付けた．

help コマンド

システムに用いることのできるコマンド名とその使い方，説明を一覧として画面に表示するコマンドである．

コマンドの記法	help
コマンド例	help
コマンド例の実行結果	XPath2.0 Processing Platform find [-o output_file_path] xpath_expression evaluate the given XPath expression, and output to a file if specified.

	help display this help. ls [collection] list all collections or collection contents if a collection specified. lsns collection list namespaces used in a collection. put file_name -c collection upload an XML file into a collection. quit quit the program. rm file_name -c collection remove a file from a collection. mkcol collection create new collection. rmcol [-f] collection remove a collection if empty or force option specified.

ls コマンド

格納されている XML ファイルの情報，もしくはデータベースに存在するコレクション名を一覧表示するコマンドである．

引数としてコレクション名が与えられればそのコレクションに格納された XML ファイルのファイルの情報を一覧表示する．引数がなければ，データベースに存在するコレクション名を一覧表示する．

・ コレクション名が引数として与えられた場合

コマンドの記法	ls *コレクション名*
コマンド例	ls SAMPLECOLLECTION
コマンド例の実行結果	sample1.xml 5 SAMPLECOLLECTION Fri Jan 15 17:05:33 JST 2010 sample2.xml 6 SAMPLECOLLECTION Fri Jan 15 17:06:38 JST 2010

・ 引数与えられなかった場合

コマンドの記法	ls
コマンド例	ls
コマンド例の実行結果	SAMPLECOLLECTION SAMPLECOLLECTION2 SAMPLECOLLECTION3

lsns コマンド

引数として与えられたコレクション名の中に存在する XML 名前空間の情報を一覧表示するコマンドである

コマンドの記法	lsns *コレクション名*

コマンド例	lsns SAMPLECOLLECTION
コマンド例の実行結果	1　http://www.w3.org/2000/xmlns/ 2　http://www.w3.org/XML/1998/namespace 5　http://www.NS/ 6　http://www.NS2/

put コマンド

XML ファイルをデータベースに格納するコマンドである．ファイルのパスは相対・絶対パスともに使用できる．

コマンドの記法	put *格納ファイルのパス* –c *コレクション名*
コマンド例	put src/leviathan/sample1.xml -c SAMPLECOLLECTION
コマンド例の実行結果	sample1.xml is stored in SAMPLECOLLECTION. Element:6, Attribute:1, Text:11 It took 0.031s.

格納ファイルに存在した要素の数や格納に要した時間も表示する．

mkcol コマンド

引数として入力された名前のコレクションを作成するコマンドである．

コマンドの記法	mkcol *コレクション名*
コマンド例	mkcol SAMPLECOLLECTION
コマンド例の実行結果	created SAMPLECOLLECTION

quit コマンド

システムを終了するコマンドである．

コマンドの記法	quit
コマンド例	quit
コマンド例の実行結果	Bye

rm コマンド

指定された XML ファイルを削除するコマンドである．

コマンドの記法	rm *XML ファイル名* –c *コレクション名*
コマンド例	rm sample3.xml -c SAMPLECOLLECTION2
コマンド例の実行結果	Document removed. sample3.xml It took 0.016s.

rmcol コマンド

指定されたコレクションを削除するコマンドである．オプションでコレクション内に XML ファイルが格納されている場合でもコレクションを削除できる，その場合は当該コレクション内の XML ファイルも削除される．

コマンドの記法	rmcol *コレクション名*
コマンド例	rmcol SAMPLECOLLECTION3
コマンド例の実行結果	delete SAMPLECOLLECTION3

4.2.4 コレクション構造

システムにはコレクションという概念を取り入れている．これはファイルシステム上におけるフォルダと概念的に同じである．

一つのデータベースに対し，複数個のコレクションを取ることがでる．また，そのコレクション内に複数の XML ファイルを格納することができる．しかし，コレクションの中にコレクションを作成するといった，コレクションの階層構造をとることはできない．

4.3 システムの動作環境

システムのソフトウェア動作環境を表 4-3 に示す．

表 4-3 ソフトウェアの動作環境

ソフトウェアの名称	バージョン
JRE	1.5
JavaCC	4.2
JDBC ドライバ	5.0.8
MySQL	5.0

第5章　　システムの内部構成

本章ではシステムの内部構成について説明する．具体的にシステムの機能説明，XPath2.0 のサブセット，システムの要件，システムの設計を説明する．

5.1　　システムの概要

我々が開発した XPath2.0 処理器である XPathProcceingPaltform の概要を図 5-1 に示す．

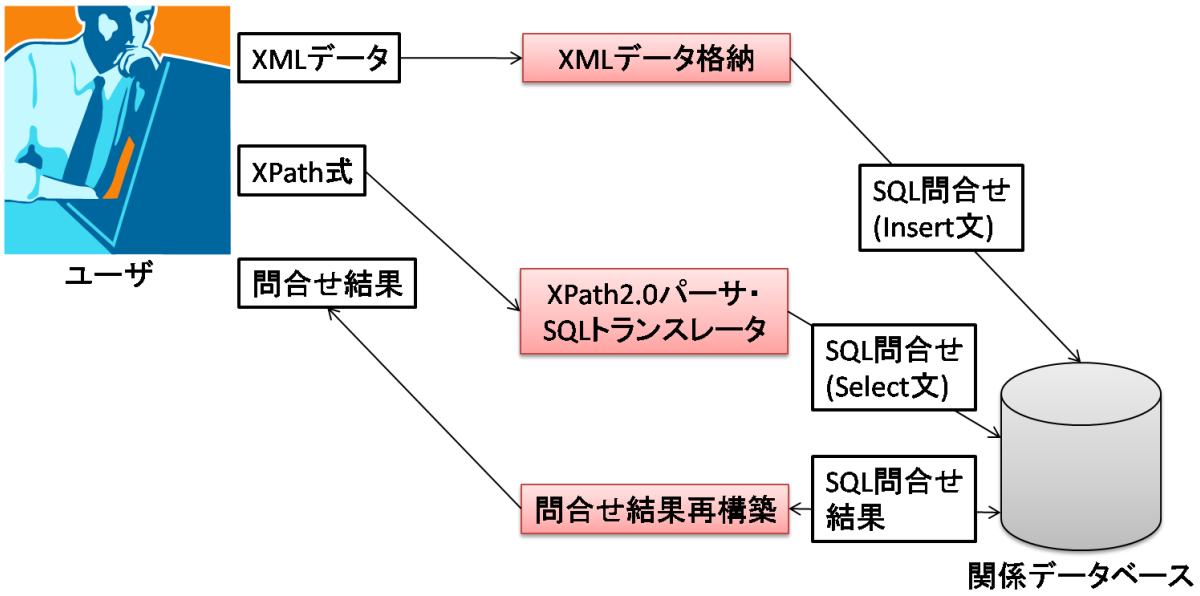

図 5-1　システム概要図

XPPP は関係データベースに基づいた XPath2.0 処理器である．以下の機能を実装している．

・ 利用者から与えられた XML データを変換し，関係データベースに格納する機能
・ 入力された XPath 式を対応する SQL 問合せに変換し，データベースに問合せを行う機能
・ SQL 問合せの結果を XML データに再構築して利用者に返却する機能

5.2　　XML データを関係データベースに格納する機能

XML データを関係データベースに格納する機能について説明する．この機能は木構造である XML データ関係データベースに関係表として格納する機能である．

5.2.1　　関係スキーマ

XML データを関係データベースに格納する方法を関係スキーマとしてモデル写像アプロ

ーチを取り入れている．本システムの ER 図(Entitiy-RelationalDaigram)を示す．

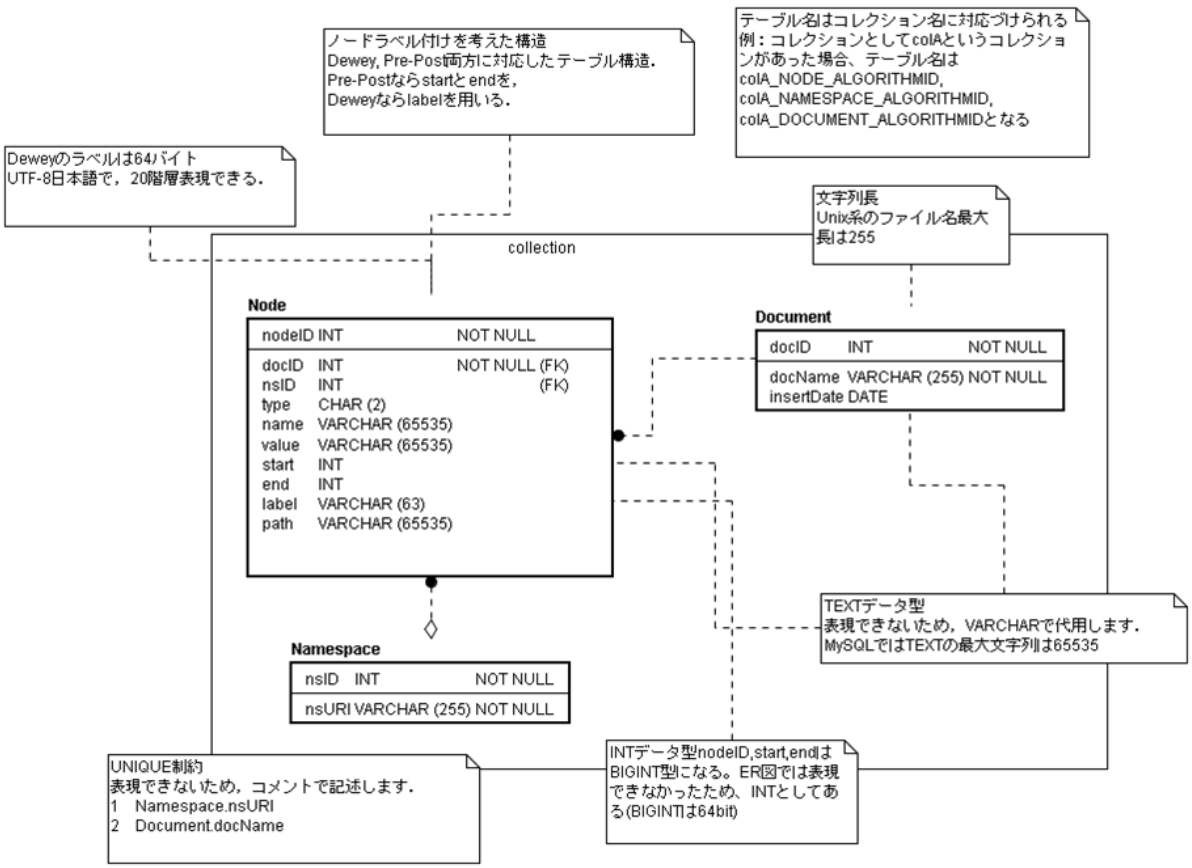

図 5-2 Entitiy-RelationalDaigram

Node テーブルはノードを格納するテーブルである．Namespace テーブルは XML 名前空間を格納するテーブルである．Document テーブルは XML ファイルの情報を格納するテーブルである．Node テーブルのカラムに関する説明を表 5-1 に，Namespace テーブルのカラムに関する説明を表 5-2 に，Document テーブルに関する説明を表 5-3 に示す．

表 5-1 Node テーブルのカラム一覧

カラム名	カラムの説明
nodeID	オートインクリメントされる ID である
docID	格納された XML ファイルを識別する ID である
nsId	関連づいた XML 名前空間を識別する ID である
type	どの種類のノードかを識別する ID である．要素であれば'1e'，名前空間であれば'2n'，属性であれば'3a'，テキストであれば'4t'という値が格納される
name	ノードの名前である
value	ノードの値である
start	PreOrder のラベルの値である．このカラムは PrePostOrder 用のカラムであり，DeweyOrder がノードラベリング手法として選ばれた場合は，カラムは存在しない
end	PostOrder のラベルの値である．このカラムは PrePostOrder 用のカラムであり，DeweyOrder がノードラベリング手法として選ばれた場合は，カラム

	は存在しない
lacel	DeweyOrder のラベルの値である．このカラムは DeweyOrder 用のカラムであり，PrePostOrder がノードラベリング手法として選ばれた場合は，カラムは存在しない
path	ノードのパスである

表 5-2　Namespace テーブルのカラム一覧

カラム名	カラムの説明
nsId	オートインクリメントされる ID である
nsURI	名前空間 URI の値である

表 5-3　Document テーブルのカラム一覧

カラム名	カラムの説明
docId	オートインクリメントされる ID である
docName	XML ファイルの名前である
insertDate	XML ファイルが格納された日付である

5.2.2　実装ノードラベリング手法

XML データを関係データベースに格納する際のノードラベリング手法として，PrePostOrder と DeweyOrder を実装した．

5.3　XPath 式を SQL 問合せに変換する機能

XPath 式を SQL 問合せに変換する機能について説明する．この機能は，XML データへの問合せ言語である XPath 式を，SQL 文に変換する機能である．

処理の流れとしては，まず XPath 式を構文解析して中間表現を作成する．その後中間表現を SQL 文に変換している．なお，構文解析の部分は JavaCC を用いて実装を行う．

SQL 文は XPath 式によって指定されたノードを選択する select 文である．なお XPath のフルセットを実装することはなく，天笠准教授からの要件であるサブセットのみを実装する．

5.3.1　LeviathanGraph

Leviathan Graph とは，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．/catalog/book[fn:compare(/title, "XML") = 0]/author という XPath 式が入力された場合，図 5-3 の例のように表現される．

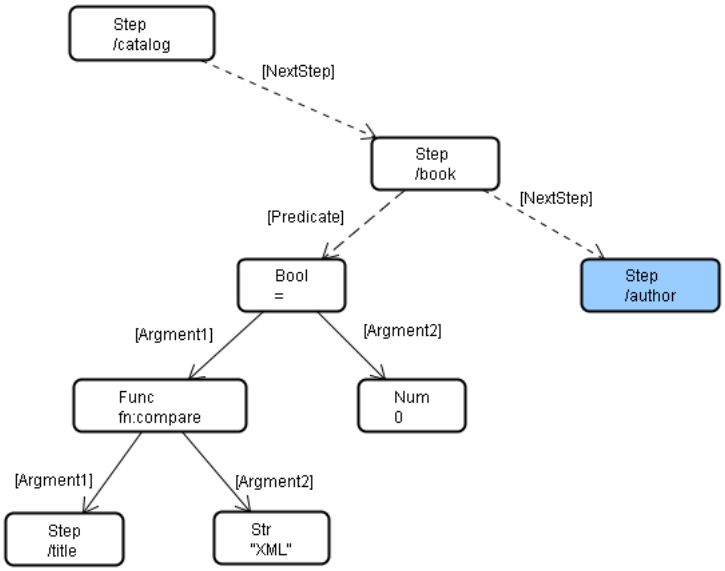

図 5-3　LeviathnGraph の例

Leviathan Graph はノードと有向エッジから構成される．最終的な問合せ結果として求めたい部分を青いノードで表している．

ノードには Step，Bool，Func，Num，Str の 5 種類あり，ノードの種類に応じた値が設定されます．表に Leviathan Graph におけるノードの種類と値の詳細を示す．

表 5-4　ノードの種類と値

種類	値
Step	ステップノード．/とステップ(軸，ノードテスト)から成る．
Bool	真偽値ノード．比較演算子，and，or
Func	関数ノード．関数のサブセットで定義した関数
Num	数値ノード
Str	文字列ノード

エッジは，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を表現している．エッジの種類と意味を示す．

表 5-5　エッジの種類と意味

種類	意味
NextStep	次のステップ
Predicate	ステップの述語
Argument	関数，比較演算子，論理演算子の引数

5.4　問合せ結果リレーションを XML に再構築する機能

この部分は「SQL トランスレータ」部分より SQL 問合せ結果を受け取る．この問合せ結果は特定要素(XPath 式により選択された部分のルート要素に対応するレコード)のみを含む．開発部分は，受け取った問合せ結果に対応する部分 XML データを再構築する機能である．

このため，関係データベースより部分 XML データに含まれるレコードを取得し， XML データに変換し，ユーザに提供するという処理を行う．

5.5 システムの要件

システムの要件を説明する．

5.5.1 機能要件

機能要件について，入出力と XPath の項目に分類した表を表 5-6 に示す．

表 5-6 機能要件一覧

関連項目	要件の内容
格納	ノードラベリング手法はDeweyOrderとPrePostOrderの二つを実現する
	格納するデータは絶対パスで指定できるようにする
出力	データベースを介する操作や他のライブラリを用いて行う処理において発生するエラーは, その際に利用する API で取得したエラーをそのまま出力する形式にする
	XPath 問合せを実行し，ファイルに実行結果を出力するときは，画面には結果を出力しない
	出力結果に名前空間宣言を挿入する
XPath	XPath 式の問合せの際に，軸として，descendant, child, attribute が使用できるようにする
	述語は使用できることとし，<,<=,=,=>,>,not,contains, compare が使用できるようにする
	XML データを格納する際，コレクション指定を可能となるようにする
	XPath 問合せ式の中で, 問合せに必要な名前空間を宣言できるようにする

5.5.2 非機能要件

システムの非機能要件について，ソフトウェアの品質特性の項目により分類した表を表 5-7 に示す．

表 5-7 非機能要件一覧

関連項目	要件の内容
移植性	DB の種類に依存しない形にする
	マルチプラットフォームに対応させる
保守性	完結しているシステムを開発する
	コンパクトなシステムにする
	ノードラベル付けの変更が容易な設計にする
効率性	数 GB レベルのデータを扱えるようにする
信頼性	ウォーターフォールモデルで開発する
	エラー処理をきちんと行う

5.6 XPath2.0 のサブセット

本システムは XPath2.0 のフルセットを実装せず，サブセットを独自に定義している．サブセットについて説明する．

5.6.1 XPath2.0 サブセット定義

サブセットを定義するに当たり，満たすべき要件を述べる．まずは軸として child，desendant，namespace，attribute を解釈できることである．次いで，述語において，論理演算子(and，or)を利用できること，また算術演算子(＜，＜＝，＝，！＝，＝＞，＞)を利用できることである．次いで fn:compare ，fn:contains,，fn:not が利用できることである．

本システムは研究者が研究で用いることを目的としており，研究者が理解し，変更しやすいシステムというものが求められている．そのため XPath2.0 の基礎であるパス式に重点を置いており，算術演算子(+,-,*,/,mod)を含まない，XML データの値には型を設定しない，といった特性を持つサブセットを定義する

5.6.2 XPath2.0 からの変更点

XPath2.0 には様々な概念がある．ここでは，XPath2.0 の概念を説明しながら，本サブセットの範囲を定義した．

基本式　Primary Expressions

基本式とは，XPath2.0 の基本要素となるものである．即値，変数参照，Context Item 式，関数呼び出しで構成される．即値とは，数値や文字列のことである．本サブセットでは，数値は整数のみとする制限を加える．また，変数参照も本サブセットには含まれない．関数については，関数のサブセットで定義されているものが使うことができる．

パス式　Path Expressions

パス式は，そのパスによって選択されているノードの順序付き集合を返す．ノードの順序付き集合をノードシーケンスと呼ぶ．パス式は複数のステップから構成されており，段階的にノードが選択されていく．ステップにはフィルター式と軸ステップがあり，軸ステップは軸とノードテストで構成されている．軸とは，これまでに選択したノードを起点として，どの方向へノードを探索するかを示すものである．ノードテストは，軸で選択された方向に対して，どのようなノードを選択するかという条件である．軸は child，descendant，descendant-or-self, namespace, attribute, self，parent，ancestor，ancestor-or-self が本サブセットに含まれる．

軸ステップでは述語を指定することで，軸とノードテストを用いて選択したノードシーケンスから，条件にあうノードのみを抽出したシーケンスを作成することができる．角括弧で式を囲んだもの述語と呼び，例えば，child::*[2]のように書くことで，「すべての子ノード」に「2 番目」という意味を追加することができ，「すべての子ノードの中で，ドキュメント順に数えて 2 番目のノード」という意味を表すことができる．

シーケンス式　Sequence Expressions

シーケンスとは，複数の数字や文字列やノードを順序付き集合として表したものである．一般的な式をカンマで区切る，または，1 to 10 のように数値の範囲を指定することで，シーケンスを作成することができる．シーケンス式は本サブセットには含まれないため，説明は

省略する．

数式　Arithmetic Expressions

数式とは，算術演算子(+,-,*,div,idiv,mod)を用いた式のことである．数式は本サブセットには含まれないため，説明は省略する．

比較式　Comparison Expressions

比較式とは，ノードや即値，シーケンスなどを比較するための二項演算子を用いた式である．比較対象の型によって異なる関数がマッピングされ，関数が具体的な比較を行う．本サブセットは，数値同士，文字列同士，真偽値同士，ノードと文字列の比較式を含む．なお，ノードと文字列の比較は，属性ノードの場合は属性の値を，要素ノードの場合は対応するテキストノードの値と比較する．

論理式　Logical Expressions

論理式には，二項演算子の and と or がある．引数を 2 つとり，それらを論理演算する．引数にはノードや即値が指定でき，真偽値ではない引数は，Effective Boolean Value(以下，EBV)という真偽値に変換してから演算を行う．EBV は fn:boolean 関数の戻り値と定義されている．fn:boolean 関数の定義については，W3C による XPath2.0 勧告 [1]の 2.4.3 項を参照してください．本サブセットはすべての論理式を含む．

For 式　For Expressions

ループを実現するための，簡易プログラミング構文である．本サブセットは For 式を含まない．

条件式　Conditional Expressions

条件分岐を実現するための，簡易プログラミング構文です．本サブセットは条件式を含まない．

量数式　Quantified Expressions

量数式は，シーケンスを集合ととらえた述語論理である．存在命題と呼ばれる some と，全称命題と呼ばれる every を用いて，シーケンスから真偽値を生成する．本サブセットは量数式を含まない．

型を用いた式　Expressions on Sequence Types

型を用いて，比較や演算を行うものである．本サブセットは型を用いた式は含まない．

5.6.3　文法のサブセット

本サブセットの文法の中心的な部分を以下に定義する．サブセットの全容は，に記載した．

これらの文法は，W3C による XPath2.0 勧告の文法を元に作成しており，文法番号は勧告で用いられている文法番号と対応づけられいる．OptionDeclare，NamespaceURI，Prefix は，XQuery の declare 文を実現するために追加したものである．XQuery の declare 文を，名前空間に対してのみ使えるように修正してある．XPath と XQuery で文法番号がかぶってしまうため，大きな文法番号をあてている．

表 5-8　文法のサブセット

1	XPath	OptionDeclare* PathExpr
3	ExprSingle	OrExpr
8	OrExpr	AndExpr ("or" AndExpr)*
9	AndExpr	ComparisonExpr ("and" ComparisonExpr)*
10	ComparisonExpr	PathExpr (GeneralComp PathExpr)?
22	GeneralComp	"=" | "!=" | "<" | "<=" | ">" | ">="
25	PathExpr	("/" RelativePathExpr?) | ("//" RelativePathExpr) | RelativePathExpr
26	RelativePathExpr	StepExpr (("/" | "//") StepExpr)*
27	StepExpr	FilterExpr | AxisStep
28	AxisStep	(ReverseStep | ForwardStep) Predicate*
29	ForwardStep	(ForwardAxis NodeTest) | AbbrevForwardStep
30	ForwardAxis	("child" "::") | ("descendant" "::") | ("attribute" "::") | ("self" "::") | ("namespace" "::")
31	AbbrevForwardStep	"@"? NodeTest
32	ReverseStep	(ReverseAxis NodeTest) | AbbrevReverseStep
33	ReverseAxis	("parent" "::") | ("ancestor" "::") | ("ancestor-or-self" "::")
34	AbbrevReverseStep	".."
35	NodeTest	NameTest
36	NameTest	QName | Wildcard
37	Wildcard	"*" | (NCName ":" "*") | ("*" ":" NCName)
38	FilterExpr	PrimaryExpr | (ContextItemExpr Predicate*)
40	Predicate	"[" ExprSingle "]"
41	PrimaryExpr	Literal | ParenthesizedExpr | FunctionCall
42	Literal	NumericLiteral | StringLiteral
43	NumericLiteral	IntegerLiteral
46	ParenthesizedExpr	"(" ExprSingle ")"
47	ContextItemExpr	"."
48	FunctionCall	QName "(" (ExprSingle ("," ExprSingle)*)? ")"
71	IntegerLiteral	Digits
74	StringLiteral	("" (EscapeQuot | [^"])* "") | ("'" (EscapeApos | [^'])* "'")
75	EscapeQuot	""""
76	EscapeApos	"'"
81	Digits	[0-9]+
997	OptionDeclare	"declare" "namespace" "element" Prefix "=" NamspaceURI
998	NamespaceURI	"'" ([^<&"])* "" |　"'" ([^<&'])* "'"
999	Prefix	NCName

5.6.4　　関数のサブセット

本システムでは以下の 3 種類の関数をサポートする．fn:compare と fn:contains の関数に引数に制限がある．

① fn:compare (Argument1, Argument2) as xs:integer

Argument1 と Argument2 を文字列として比較する．両方の文字列が同じものである場合は 0 を返し，最初の引数が 2 番目の引数より小さい場合は 1 を返し，それ以外の場合は -1 を返す．

② fn:contains (Argument1, Argument2) as xs:Boolean

Argument1 と Argument2 は，文字列と文字列，もしくは，文字列とノードとなる

Argument1 と Argument2 を文字列として比較します．最初の引数が 2 番目の引数を含む文字列であった場合は true を返し，それ以外の場合は false を返す．

③ fn:not (Argument1) as xs:boolean

Argument1 と Argument2 は，文字列と文字列，もしくは，文字列とノードをとる．

引数の真偽値を反転させます．引数に真偽値，真偽値を返す論理式をとり，真偽値が true の場合は false を返し， false の場合は true を返す．

5.7 システムの設計

図 5-4 にシステムの論理クラス図を示す．

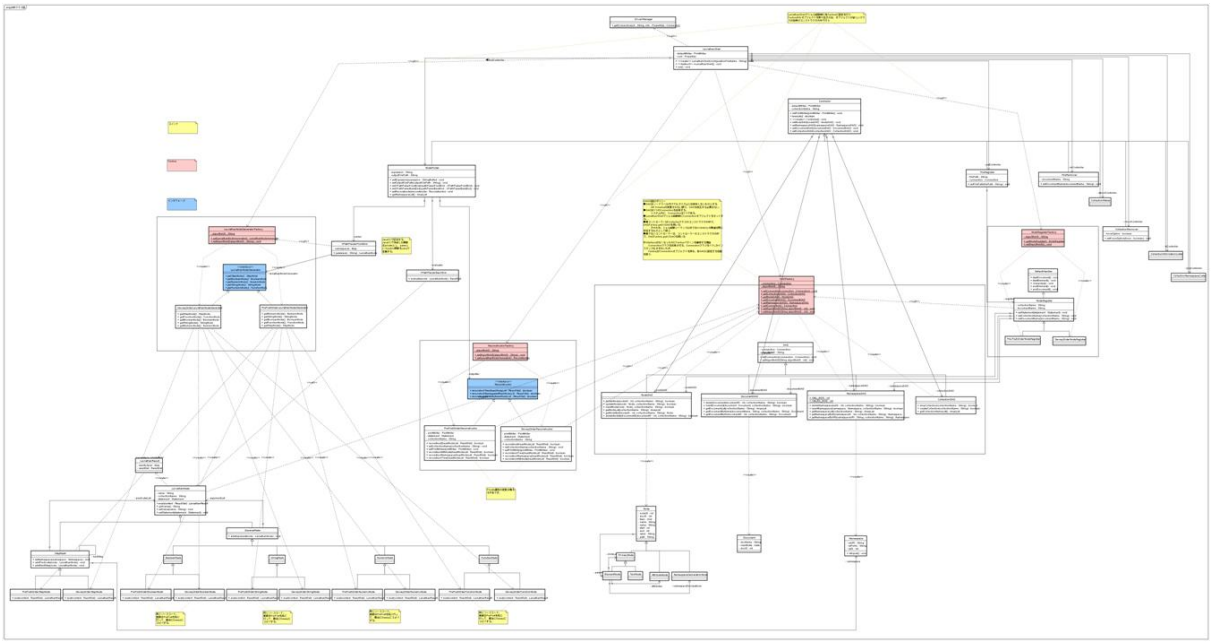

図 5-4 論理クラス図

設計を行ったうえでの方針について説明する．

まず，一つのシェルコマンドに対し，一つ親コントローラを設けた．そして，親同士の Controller の呼び出しはできないこととした．これは，コントローラの責任分担を意識したものである．

次いで，DAO の役割は controller とのオブジェクトの受け渡し，SQL 文の作成，DB との接続・切断とし，データベースとの接続のタイミングは DAO 生成時，切断のタイミングは DAO 破棄とした．

次いで，システムの拡張性を考えて，図 5-5 のように Factory パターンを用いた設計を行っている．

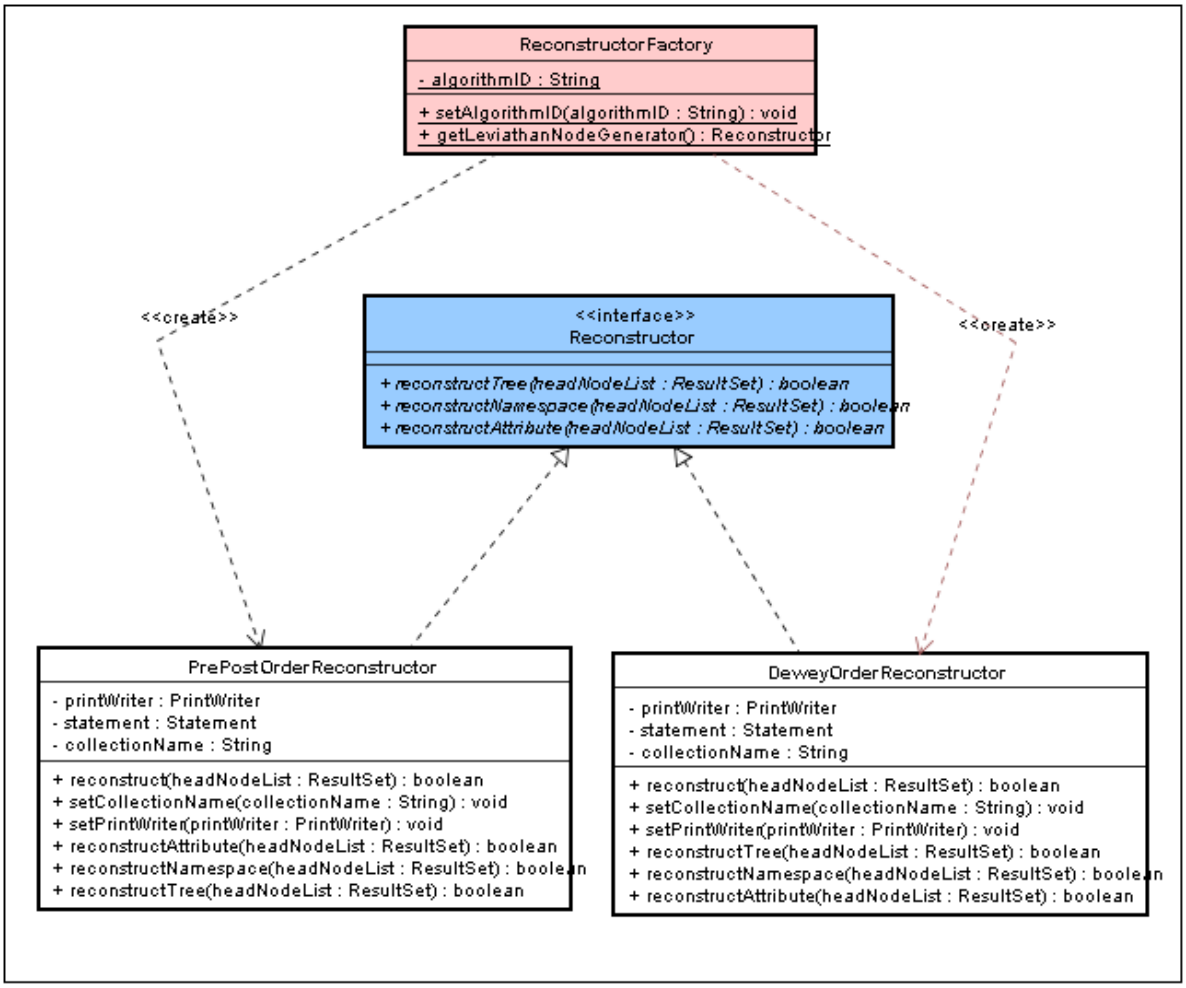

図 5-5　Factory パターンを用いた論理クラス図

図 5-5 は図 5-4 中の問合せ結果再構築の部分を抜粋したものである．

Factory パターンとはデザインパターンの一種であり，オブジェクトを生成する処理と生成したオブジェクトを使用する処理を分離させるパターンである．

オブジェクト生成の専門家である Factory が，オブジェクトの使用者から生成するオブジェクトの種類や生成手順を隠す． オブジェクトの使用者は Factory に生成を依頼するだけで，オブジェクトの生成手順や種類を意識する必要はなく，望むオブジェクトを使用できる状態で手に入れることができる．

もし生成するクラスの種類や作成手順が変更されても，Factory の中を手直しするだけですむ利点がある．

テスト工程においてもメリットがある．使用したいオブジェクトがまだ作成されていないときも，Factory にモックオブジェクトと呼ばれる擬似オブジェクトを生成させることで，オブジェクトの使用者に影響を与えることなく平行開発を行うことができる．テスト用のモックオブジェクト生成すれば，テストの効率化を高めることができる．

第6章　問合せ結果再構築機能の開発

本章では担当部分における技術要素，概要，機能実現にあたっての問題点と解決策ついて説明する．

6.1　開発部分概要

報告者の開発部分である，問い合わせ結果の再構築について概要を述べる．

この部分は「SQL トランスレータ」部分より SQL 問合せ結果を受け取る．受け取った関係表の情報が要素であるか属性であるかにより異なる．

まず要素を受け取った場合を説明する．この問合せ結果は特定要素(XPath 式により選択された部分のルート要素に対応するレコード)のみを含む．開発部分は受け取った問合せ結果に対応する部分 XML データを再構築する機能である．このため，関係データベースより部分 XML データに含まれるレコードを取得し，XML データに変換しユーザに提供する処理を行う．

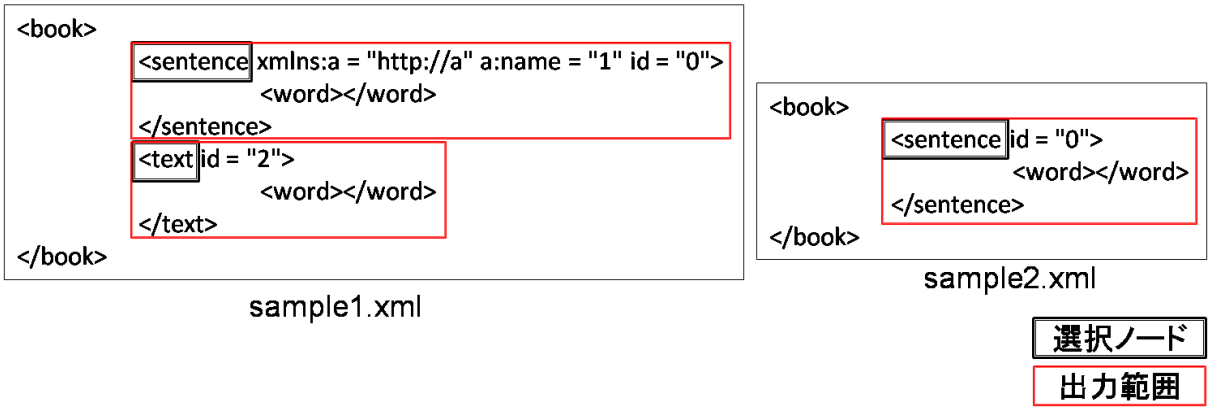

図 6-1　選択されたノード(要素)と出力範囲

```
# It took 1.5 seconds for evaluating
   collection("sampleCollection")/book/child::*,found 3 nodes.
<sentence xmlns:a= "http://a" a:name = "1" id = "0">
<word></word>
</sentence>
# /book/sentence has 2 nodes in sample1.xml.
<text id = "2">
<word></word>
</text>
# /book/text has 2 nodes in sample1.xml.
<sentence id = "0">
<word></word>
</sentence>
# /book/sentence has 2 nodes in sample2.xml.
# It took 2.0 seconds for outputting collection("sampleCollection")/book/child::*.
```

図 6-2　要素が選択された場合の出力形式

格納されていた XML データ以外にも，XPath 式を SQL に変換し，先頭ノードを選択するまでの時間, 先頭ノードの数, 先頭ノードとその階層の下にあるノードの数, を表示する．尚，XML データとシステムが付加させた情報を区別するためにシステムが付加させた情報の先頭には'#'を付加して表示している．

次に，「SQL トランスレータ」から属性の情報を受け取った場合について説明する．属性のレコードを受け取った場合は，そのレコードを XML データに変換して表示する．図 7-1 と同じ状態のコレクションに対して，XPath 式　**//book/@id**　で問合せ際に選択されるノードについて，図 6-3 に記す．選択されたノードが出力範囲となる．

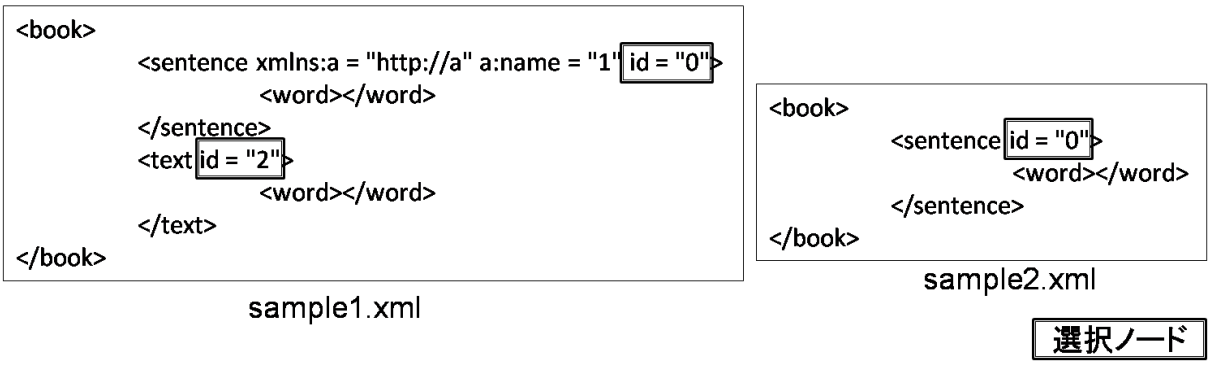

図 6-3　選択されたノード(属性)と出力範囲

```
# It took 1.0 seconds for evaluating collection("sampleCollection")/book//@id, found
   3 attributes.
<LeviathanResultid = “0”/>
# /book/sentence/@id,insample1.xml.
<LeviathanResultid = “2”/>
# /book/text/@id,insample1.xml.
<LeviathanResultid = “0”/>
# /book/sentence/@id,insample2.xml.
# It took 1.1 seconds for outputting collection("sampleCollection")/book//@id.
```

図 6-4　要素が選択された場合の出力形式

要素名，LevaithanResult の空要素タグに出力する属性の名前，値を追加する．

6.2　　機能要件

問合せ結果の再構築の部分における，天笠准教授の要求を以下に示す．

① ノードラベル付けアルゴリズムの変更が容易な設計にすること
② XPath 式の結果を CUI に出力すること
③ ファイルに実行結果を出力するときは，画面には結果を出力しないこと
④ 出力結果に名前空間宣言を挿入すること
⑤ 数 GB レベルのデータを出力できるようにすること

6.3　　特定要素をルート要素とする部分木の取得方法

「SQL トランスレータ」から受け取った問合せ結果が特定要素であった場合は，その特定要素をルート要素とする部分木をデータベースから取得する必要がある．その取得方法について説明する．これはノードラベリング手法により異なる．簡単に説明するため具体例を用いて説明する．

6.3.1　　DeweOrder の場合

ノードラベリング手法が DeweyOrder の場合は特定要素の DeweyOrder(Node テーブルの label カラム)の値を用いて条件問合せを行う．DeweyOrder ではノードの先祖子孫および親子関係は label の値が部分一致するかどうかで判定できる．例えば **/catarog** の XPath 式で問合せが行われた場合，部分木を取得する SQL 問合せは

select * from テーブル名 where label like '1%';

となる．この SQL 問合せを行うことで部分木を取得している．

6.3.2　　PrePostOrder の場合

ノードラベリング手法が PrePostOrder の場合は特定要素の PreOrder(Node テーブルの start カラム)と PostOrder(Node テーブルの end カラム)の値を用いて条件問合せを行う．PrePostOrder の場合，先頭要素の(PreOrder，PostOrder)の値を(PreH，PostH)，その先頭

要素をルート要素とする部分木のあるノードの(PreOrder, PostOrder)の値を(PreD, PostD)とすると必ず以下の式が成り立つ.

PreH＜PreD　かつ　PreH>PreD

また, 部分木でないノードの場合は上記の式は必ず成り立たない. よって **/catarog** の XPath 式で問合せが行われた場合，部分木を取得する SQL 問合せは部分木を取得する SQL 問合せは

select * from テーブル名 where start>1 ,end<9;

となる．この SQL 問合せを行うことで部分木を取得している.

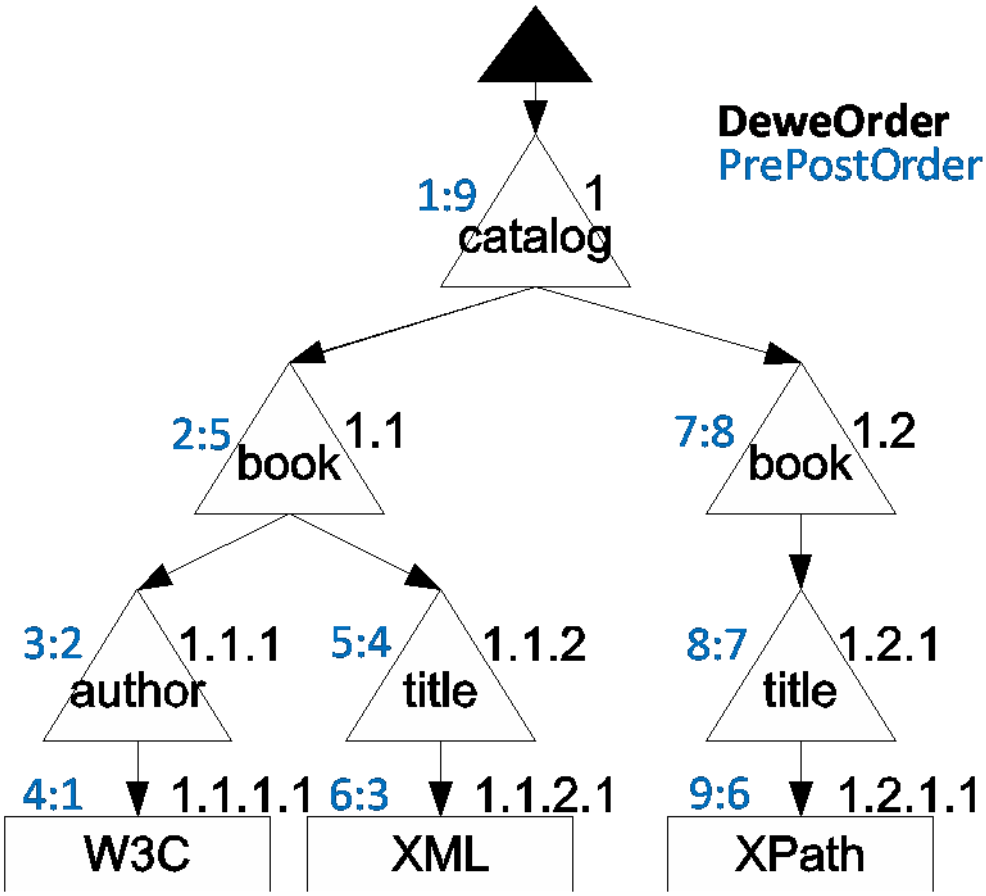

図 6-5　DeweyOrder，PrePostOrder のラベリング

6.4　　関係表から XML データを構築する方法

特定要素をルート要素とする部分木の取得を行って得た関係表を XML データに構築する方法を説明する．XML データを出力するには一般的には DOM が用いられることが多い．DOM は全ての XML データをメモリ上に木構造にするまでは出力処理を行わず，大容量のデータを扱わう場合には用いることができない．天笠准教授の要件「数 GB レベルのデータを出力できるようこと」がる．よって DOM を用いることはできない.

XML データを一つの木構造に変換する DOM と違って，XML データを先頭から一行ずつ順に読み込んで，要素が現れる度に対応する処理手順を呼び出すという方式を用いているため，巨大な XML データを扱ってもメモリ容量を必要とせず高速に処理できるという特徴がある.

報告者の担当部分において, 数GBレベルのXMLデータを取り扱えるようにしてほしい, といった要求がある．このレベルのデータを，DOM を用いて処理しようとすると，メモリにそこで開発システムでは，ストリーム・ベースの XML 出力を処理する特殊な SAX フィルターである，XMLWriter [7]クラスを用いた.

なお，XMLWriter クラスを継承した DataWriter クラスがある．DataWriter クラスを用いることを最初検討していたが開始タグを出力すると自動的に改行が行われてしまった．この原因については解明していないが DataWriter クラスを用いないこととした.

6.4.1 XMLWriter クラスのメソッド

XMLWriter クラスが提供するメソッドの中で今回用いたメソッドについて説明する.

startElement メソッド

開始タグを出力する関数である．引数により名前空間や属性の情報が入った要素の開始タグと，要素のみの開始タグを使い分けることができる.

要素のみの開始タグを出力したい場合は
startElement(java.lang.String localName)
を用いる．名前空間や属性を挿入したい場合は
startElement(java.lang.String uri, java.lang.String localName, java.lang.String qName, org.xml.sax.Attributes atts)
を用いる.

endElement

終了タグを出力する関数である.
endElement(java.lang.String localName)
である.

6.4.2 名前空間・属性の出力方法

名前空間は startElement の'uri'に'qName'値を入れるのではなく, AttributeImpl[11]クラスのオブジェクトに値を代入し，そのオブジェクトを代入することにより，出力している．本来属性を出力する際に用いるのが AttributeImpl クラスであるが startElement の'uri'に'qName'値を入れる方法では一つのタグ内に複数の名前空間を挿入することができないため，名前空間を出力する際でも AttribuImpl クラスを用いる．属性も同じクラスを用いる.

6.4.3 テキストの出力方法

テキストの出力は PrintStream[10]クラスの
print(String s)
メソッドを用いる．XMLWriter クラスにもテキスト出力用であるメソッド
characters(java.lang.String data)
を備えているがエスケープ処理を自動で行ってしまうので日本語を出力できなかったので, PrintStream クラスを使用した.

上述の startElement・endElement・print 関数を用いてデータを出力していく．次いで, 関係表のソート方法について説明する.

DeweyOrder の場合

ノードラベリング手法が DeweyOrder の場合は label の値を用いてソートを行えば出力処理したい順序に並び変わる．よってデータベースより部分木のレコードを取得する SQL 文は以下のようになる

select * from テーブル名 where label like '1%' order by label;

PrePostOrder の場合

ノードラベリング手法が DeweyOrder の場合は start の値を用いてソートを行えば出力処理したい順序に並び変わる．よってデータベースより部分木のレコードを取得する SQL 文は以下のようになる

select * from テーブル名 where start>1 ,end<9 order by start;

次いで，関数をどのようなタイミング・手順で呼び出し，XML データを構築したかについて説明する．簡単に説明するため，要素とテキストのみの関係表をどのように XML データに出力するかを説明する．特定要素とそれ以外の部分木で処理が異なる．まず特定要素の処理手順を記す

① 関係表の先頭行を読み込み，name カラムの値を用いて開始タグ形成し，出力する(startElement メソッドを用いる)
② path カラムより階層を求める
③ name カラムの値を「終了タグとして出力すべき値」として，一時的に保存する
④ 関係表の次の行を読み込む

次いで特定要素以外の部分木の処理手順を記す.

① path カラムより階層を求める．求めた階層と前回読み込んだ階層を比べ，終了タグを出力するかどうか判断する(出力する場合は endElement メソッドを用いる)
② type カラムの値が element であれば開始タグ形成し,出力する，処理③へ．値が text の場合はテキストを出力する(print メソッドを用いる)，処理④へ
③ name カラムの値を「終了タグとして出力すべき値」として，一時的に保存する
④ 関係表の次の行があれば次の行を読み込み処理①へ．次の行がなければ保存している終了タグを全て出力し処理を終える

階層とは Node テーブルの path カラムから算出した値である．path はそのノードのパス式が格納されており，そのパス式の'/'の数より求めている．この階層の値を用いて，終了タグを出力するかどうかを判断する．また前行(一つ前に処理した行行)のノードの種類と今行(現在処理している行)のノードの種類も関係する．場合分けして説明する．

前行が要素であり，今行が要素の場合

今行の階層が，前行の階層より小さい場合は，その差分足す 1 の数だけ終了タグを出力する．

今行の階層が，前行の階層と同じ場合は，その一つだけ数だけ終了タグを出力する．

前行がテキストであり，今行が要素の場合

今行の階層が，前行の階層より小さい場合は，その差分の数だけ終了タグを出力する．

前行が要素であり，今行がテキストの場合

今行の階層が，前行の階層の大きさ足す 1 より小さい場合は，その差分の数だけ終了タグを出力する．

前行がテキストであり，今行がテキストの場合
今行の階層と前行の階層の差分の数だけ終了タグを出力する．

次いで，名前空間と属性が関係表に存在する場合について説明する．開始タグを出力する際，出力対象の要素・名前空間・属性の情報すべてをシステムが持っていなければならない．このことと，終了タグの判定条件に要素・テキストの関係としていることから，要素を読み込んだ処理の内部処理として名前空間・属性を出力の考慮に入れる．つまり，大きくは要素とテキストの関係性で処理を行っていき，要素の内部処理の中において名前空間・属性を考慮し開始タグとして出力する．

しかし，ここで問題になってくるのが，関係表の並び順である．同じ開始タグ内にある要素･名前空間･属性には同じノードが振られている．そのため type カラムでは要素に'1e'，名前空間に'2a'，属性に'3a'，テキストに'4t'という値を保持している．

DeweyOrder の場合

select * from テーブル名 where label like '1%' order by label,type;

PrePostOrder の場合

select * from テーブル名 where start>1 ,end<9 order by start,type;

6.5 大規模データの扱い

6.3 節・6.4 節で述べた処理を行えば小規模のデータ出力であれば問題なく XML データを構築し出力することができる．しかし大規模なデータを出力する場合には考慮しなくてはならない点がある．それは ResultSet[12]クラスの性質である．

ResultSet クラスとは SELECT 実行後にデータを取得するためのインタフェースである．この取得の際にデフォルトとして対象レコードを完全に摘出しメモリに保存するといった性質がある．つまり問合せ結果を逐次処理するのではなく一度全てメモリに保存してから，次の処理に移行するのである．

特定要素から部分木を取得する際，部分木が大規模であればメモリに全て保存するのではなく，問合せ結果を一行取得・逐次処理を行わなければならない．そこでステートメントを発行する際に FetchSize を設定する．FetchSize とはレコードを何行読み込んだらストリーミング処理に移行するか，の数字である．

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
```

この処理を行うとストリーミング処理を行うことができ，メモリに負荷を与えることがなくなる．

しかし，Fetch サイズを変更すると新たな問題点が発生した．出力に時間がかかり過ぎることである．この時間がかかり過ぎる原因として，Fetch を行って次のデータを取得しようとした際, SQL 文に ORDER BY 句が存在した場合に Fetch するたびに並び変える処理が行われてしまう．この問題を解決するため，出力すべきデータをソートした形の View を作成し，そのデータに対し，StreamingResultSet を用いて出力することとした

6.6 名前空間への対応

前述した通り，名前空間はあるノードで宣言すると，要素ではそのノード内とその下の階

層のノードで有効となる．有効となっている名前空間を使用する際，名前空間を宣言しなおすことはしない．以下の図 6-6 の XML データを格納した際の Node テーブル,Namespace テーブルを表に示す．ノードラベリング手法は DeweyOrder とする．

```
<catalog xmlns:NS = 'http://www.NS/'>
    <book category = 'recommendation'>
        <NS:title>
            <NS:author>W3C</NS:author>
        </NS:title>
    </book>
</catalog>
```

図 6-6　格納対象 XML データ

表 6-1　Node テーブルに格納されたデータ

nodeId	nsId	type	name	value	label
1	null	1e	catalog	null	1
2	null	2n	NS	http://www.NS/	1
3	null	1e	book	null	1.1
4	null	3a	caregory	recommendation	1.1
5	3	1e	NS:title	null	1.1.1
6	3	1e	NS:author	null	1.1.1.1
7	null	4t	null	W3C	1.1.1.1.1

表 6-2　Namespace テーブルに格納されたデータ

nsId	nsURI
1	http://www.w3.org/2000/xmlns/
2	http://www.w3.org/XML/1998/namespace
3	http://www.NS/

表 6-1 の Node テーブルからは，便宜上 docId，path カラムを省略した形を取っている．

図 6-5 において，XPath 式　**/category/book**　で問い合わせた際の表示部分は図 6-7 に記載された出力となる．

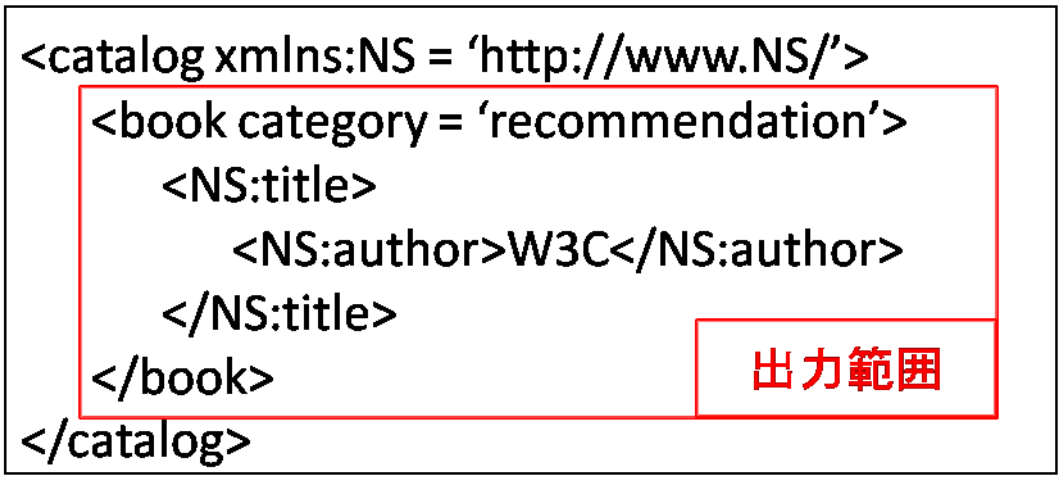

図 6-7　名前空間に対応できていない出力データ

このように XML 名前空間の接頭辞である'NS'がどのように定義されたものかが出力結果からだけでは把握できなくなってしまう．出力の際，接頭辞にどのような XML 名前空間が結びついているかについての情報は，使用している接頭辞については表示してほしいとの天笠准教授の要求があったので，考慮する必要がある．この問題についてどのように解決したかについて説明する．

解決策として，Namespace テーブルに格納された名前空間を，要素を出力する前に全て読み込み，そのデータを HashMap[13]クラスのオブジェクトに格納する．'key'に nsId を，'value'に名前空間 URI を挿入する．デフォルト名前空間の場合は null を key とする．要素・属性ノードの場合名前空間に関連づいている場合は nsId カラムに値が格納されているので，nsId カラムに値がある場合は，その値を HashMap に問い合わせて名前区間 URI を得る．そして出力する．その処置を行って出力したデータを図に示す．

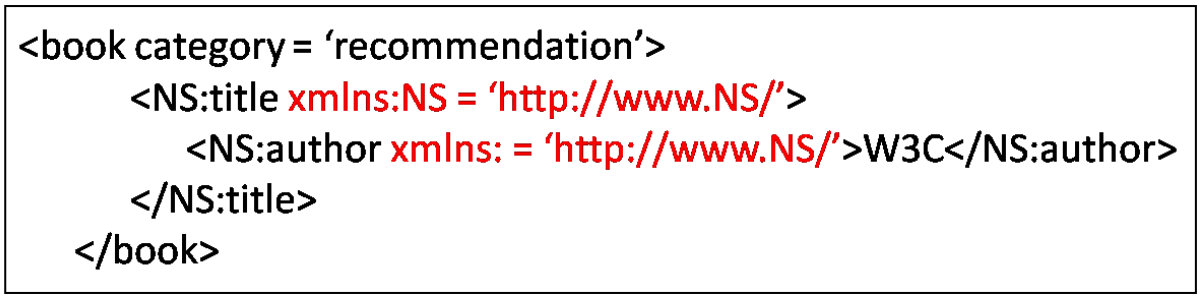

図 6-8　重複する名前空間宣言

接頭辞 NS がどのような URI と関連づいているかは把握できるようになった．しかし同じ NS:author 要素の開始タグにおいて既に有効な名前空間を再度宣言している．これは誤りではないが通常の XML データではこのようなことをしない．

そこで次の方法として HashMap クラスと Vector[14]クラスを用いて名前空間への問題に対応した．まず HashMap クラスについて説明する．HashMap クラスは名前空間の容れ物として用いる．'key'に prefix を，'value'に名前空間 URI を挿入する．

次いで Vectir の使い方を説明する．Vector は HashMap の容れ物として用いる．HashMap オブジェクトを部分木の階層ごとに作成し，Vector オブジェクトに格納する．そのイメージ図を示す．

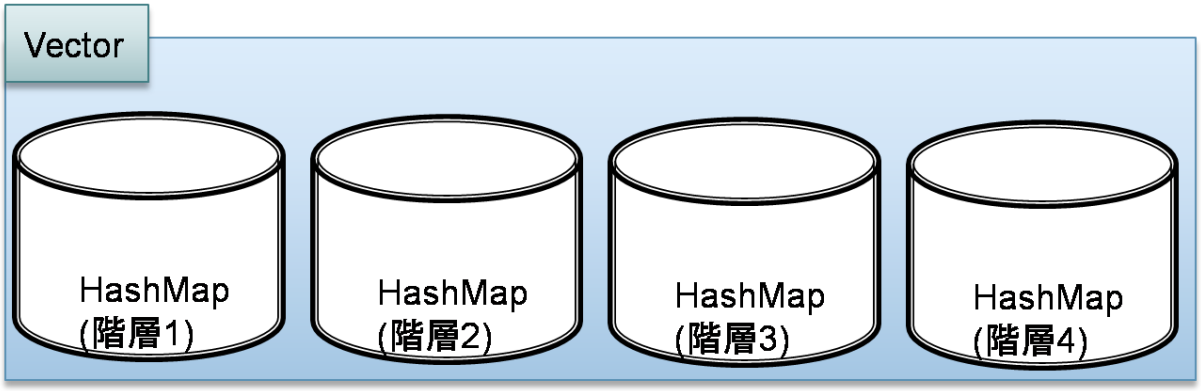

図 6-9　名前空間対応イメージ図

名前空間が関連づいたレコードを読み込むたびにそのレコードの階層に対応する HashMap に読み込んだ名前空間を格納する．図 6-7 の出力範囲を出力する際に，どのように名前空間に対応するかを説明する．

① nodeId='4'の要素に nsId が付与されているので，その名前空間を HashMap(階層 3)に格納する
② 階層 2，階層 1 の順に同じ名前空間が挿入されているかどうか確かめる．挿入さえていないので，当該名前空間を出力する．
③ nodeId='5'の要素に nsID が付与されているのでその名前空間を HashMap(階層 4)に格納する
④ 階層 3，階層 2，階層 1 の順に同じ名前空間が挿入されているかどうか確かめる．階層 3 に挿入されているので当該名前空間は挿入されない
⑤ 階層 4 の終了タグが出力されたので，階層 4 の HashMap を初期状態に戻す
⑥ 階層 3 の終了タグが出力されたので，階層 3 の HashMap を初期状態に戻す

上記の手順を行うことにより図のデータを出力できる．

```
<book category = 'recommendation'>
        <NS:title xmlns:NS = 'http://www.NS/'>
            <NS:author >W3C</NS:author>
        </NS:title>
    </book>
```

図 6-10　名前空間対応出力データ

第7章　問合せ結果再構築の評価

本章では担当部分について評価を述べる.

7.1 担当部分の要求とその評価

担当部分である要求を以下に記す.

① ノードラベル付けアルゴリズムの変更が容易な設計にすること
② XPath 式の結果を CUI に出力すること
③ ファイルに実行結果を出力するときは，画面には結果を出力しないこと
④ 出力結果に名前空間宣言を挿入すること
⑤ 数 GB レベルのデータを出力できるようにすること

これら要求のうち，②，③，④に関してはテスト工程において，要求を満たしていることを確認した.

①の要求に関しては，Factory パターンを用いていること，新しいノードラベリング方法を追加する際に書き換える箇所が SQL 文のみであることから容易であることが推測される.よっておそらく満たしていると考えている．しかし，この項目に関して，天笠准教授からの評価をして頂いてはいないので今後評価していただく必要性がある．また，実際に新たなノードラベリングを導入し，検証を行うなどはしていないのでこちらも検証を行う必要性がある.

次に，⑤の要求に関しては，PrePostOrder，DeweyOrder 両方のノードラベリング方法において，1.11GB の XML データを出力できることを確認した．しかし，それ以上のデータについては検証を行っていない.

7.2 出力に要する時間

出力に要する時間を以下の表に示す．XMark[15]を用いて作成したデータをテストデータとして使用した．ノードラベリング手法は PrePostOrder を使用した.

データの大きさ	出力に要する時間
33KB	約 1 秒
1.12MB	約 1 秒
11.3MB	45 秒
111MB	約 3000 秒
1.11GB	約 46800 秒

次にオープンソースの XML データベースである eXist[16]の出力に要する時間を表に示す.同様に XMark のテストデータを使用した.

データの大きさ	出力に要する時間
33KB	約 0.5 秒

1.12MB	約 1 秒
11.3MB	約 9 秒
111MB	測定不可(Java heap error のため)
1.11GB	測定不可(Java heap error のため)

この違いに関しては eXist の出力の際のアルゴリズムを調べられていないので原因は未明である．今後調査する必要性がある．また，大容量データの出力時間に関してはその規模のデータを出力できる XML データベースと比較し，実使用に耐えうるかを検証する必要がある．

本システムの出力時間に関して，今後改善できる点としては FetchSize の大きさが挙げられる．現在は FetchSize の大きさを 1 レコードにし，ストリーミング処理をおこなっているが FetchSize の大きさを変えることにより，出力時間が変わるかもしれない．変更の可能性があるかどうかをまず確認すべきである．

第8章　プロジェクトの振り返り

本章では，プロジェクトの振り返りとして，システム開発計画とその実績，プロジェクトの反省点について説明する．

8.1　開発計画とその実績

システム開発計画とその実績を図 8-1 に示す．

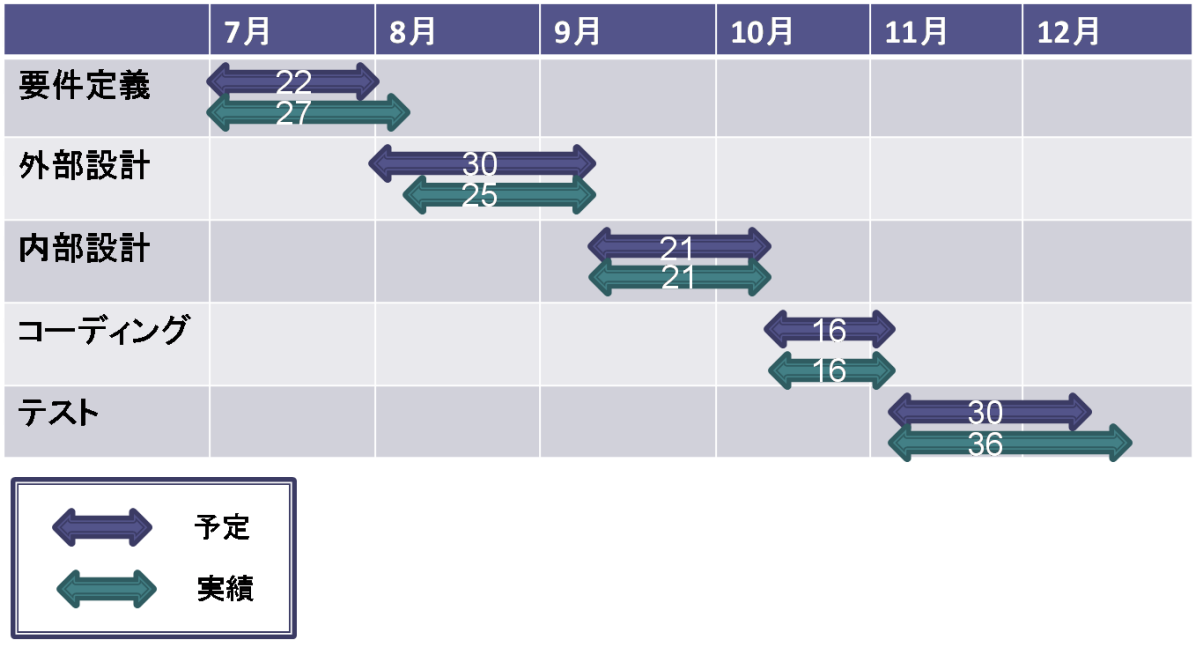

図 8-1　システム開発計画とその実績

要件定義工程が長引いた原因について説明する．これは委託元である天笠准教授と，我々のチームとのシステムに関する認識にずれが生じており，要件の確認，修正を何度も行ったためである．

認識のずれの原因としては，ミーティングの少なさが原因であったと考えられる．要件定義工程において，チーム内ミーティングを週一回しかとらず，チーム内で十分な議論をすることができなかった．また，天笠准教授とのミーティングも二週間に一回のペースと，要件定義工程としては少なく，要件を吸い上げるのに十分なミーティングの時間を取れていなかった．

次に，外部設計工程を予定よりも早く締めることができた理由を説明する．元々外部設計工程においては，開発システムに対する知識の乏しさから，勉強期間も混ぜて，他の工程よりも長い期間を設定していた．しかし，予定よりもシステムに関する知識を，早く習得できたためである．知識習熟の早さの原因として二つ考えられる．

まず，要件定義工程の反省を生かして，週に二回は必ずミーティングを設けたことである．このことにより，メンバ間で議論，技術調査報告などを頻度高く行うことができた．

次に，システムのプロトタイプを開発したことが要因と考えられる．プロトタイプを開発することにより，システムの全体像や必要な知識，リスクなどを把握することができた．

テスト工程が長引いた原因について説明する．この原因としては XPath2.0 パーサにおいて，バグが収束することなく発生したためである．この件に関して，報告者自身の反省点としては，JavaCC の習熟不足が挙げられる．そのため，技術調査など，何かしら手伝える部分があるはずであったが，関与することができなかった．関与することができれば，遅延を少なくできた可能性はある．

8.2 プロジェクトの反省点

プロジェクトの反省点を述べる．

まず内部設計を軽視していた点が挙げられる．その時点で内部設計の重大さに気付いておらず，念入りな設計を行わずに実装工程に入ってしまった．これにより，設計に何度か手戻りが発生してしまった．ただ，実装しなければ中々設計しづらいということも確かに挙げられる．この問題を解決するため，スケールの小さいプロトタイプを作成することが重要であるとの知見を得た．

次に工程全般に言えることだが，重要な部分はどこかを考え，その重要な部分から取りかかるべきであった．例えば，本システムを開発するにあたって，数 GB レベルのデータを扱うことが要求にあったが，その検証をテスト工程の終盤で検証し，大きな問題が発生することとなった．早い段階での技術調査，また委託元の要求優先順位なども考慮したうえでシステム開発を進めるべきである．

次に, 自身の担当部分でない部分に関してあまり把握できていなかったことが挙げられる．これに関しては，当初は関係がないようであっても，後に担当部分に影響が及ぶことがあった．この対策としては，ミーティング時に理解できないことが議題に挙がった場合にはすぐに質問をすることが挙げられる．もちろん，大幅な進行の妨げとならないようどこが理解できていて，どこが理解できていないのかを明確に相手に伝えることが大事である．

第9章　結論

本報告書では，関係データベースを利用した XPath 処理器 XPPP の開発と，プロジェクトにおける報告者の担当である，問合せ結果再構築について述べた．

XPPP の開発では，要件である拡張性を考慮した設計だけでなく，使用性についても XML データベースに似せることで研究者が違和感なく使えるシステムになったと言える．XPPP が普及し，研究が更に活発になることを願う．

報告者の担当部分である問合せ結果再構築についてであるが，拡張性について，変更点が SQL 文のみとすることができた．大容量なデータである 1GB の XML データを出力できることも確認できた．最低限の要件をクリアできたと言える．しかし，出力までの時間を短縮するための検証や，単体テストを行わなかったことによる信頼性への検証などが今後に残された課題である．

謝辞

本報告書を作成するに当たり，日頃よりご指導下さいました筑波大学大学院システム情報工学研究科田中二郎教授に深く感謝致し，御礼申し上げます．

また，プロジェクトの委託元である同大学院システム情報工学研究科の天笠俊之准教授には，プロジェクト開始から終了まで，一貫して適切なご指導を賜りました．また「PBL 型システム開発」に置きましてもシステムについて適切なアドバイスを頂きました事を，深く感謝し，御礼申し上げます．

本報告書の審査を快く引き受けてくださいました，同大学院システム情報工学研究科の菊池純男教授にも深く感謝し，御礼申し上げます．

チームメンバであった上田保佑氏と田中勇也氏，羽鳥貴之氏には，プロジェクト進める上で，何度も助けていただきました事を深く感謝致します．本当にありがとう．

「高度 IT 人材の育成のための実践的ソフトウェア開発専修プログラム」でお世話になりました，同大学院システム情報工学研究科駒谷昇一教授，コースの皆様にも多大な協力とご意見を頂きました事に，深く感謝し，御礼を申し上げます．

最後に，7 年間という長い間，学業を支え続けてくれた家族に感謝致します．

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
[2] Extensible Markup Language:http://www.w3.org/XML/
[3] Namespaces in XML:http://www.w3.org/TR/1999/REC-xml-names-19990114/
[4] XML Path Language (XPath) 2.0:http://www.w3.org/TR/xpath20/
[5] XQuery 1.0: An XML Query Language:http://www.w3.org/TR/xquery/
[6] 天笠俊之，吉川　正俊，"XML データベース技術概説"，オペレーションズ・リサーチ，pp.365-372，2005 年 6 月号
[7]Class XMLWriter : http://fisheye5.cenqua.com/browse/~raw,r=1.1/txw/www/javadoc/com/sun/txw/runtime/XMLWriter.html
[8]Class DataWriter : http://fisheye5.cenqua.com/browse/~raw,r=1.1/txw/www/javadoc/com/sun/txw/runtime/DataWriter.html
[9]Class AttributeImpl : http://java.sun.com/j2se/1.4.2/docs/api/org/xml/sax/helpers/AttributesImpl.html#AttributesImpl(org.xml.sax.Attributes)
[10]Class PrintStream：http://java.sun.com/j2se/1.5.0/docs/api/java/io/PrintStream.html
[11]Class AttributeImpl : http://java.sun.com/j2se/1.4.2/docs/api/org/xml/sax/helpers/AttributesImpl.html#AttributesImpl(org.xml.sax.Attributes)
[12]Interface ResultSet：http://java.sun.com/j2se/1.4.2/docs/api/java/sql/ResultSet.html
[13]Class HashMap　:http://java.sun.com/j2se/1.4.2/docs/api/java/util/HashMap.html
[14]Class Vector：http://java.sun.com/j2se/1.4.2/docs/api/java/util/Vector.html
[15]XMark: http://www.xml-benchmark.org/
[16]eXist:http://exist.sourceforge.net/

付録 A　要件定義書

付録 B　追加要件定義書