平成15年度学士論文

大容量XML文書のデータ更新が可能なXML編集ライブラリ

東京工業大学 理学部 情報科学科
学籍番号 00-0151-9
石川　零

指導教官
千葉 滋 助教授

平成16年2月5日

概要

大容量の XML (Extensible Markup Language) 文書を効率よく操作するライブラリの必要性が高まっている。現在、XML は特定のプラットフォームやプログラミング言語に依存しないという特徴から、データ記述言語として適用分野や用途を問わず広く利用されている。その特徴から、XML は複数の企業、団体間でやり取りされるデータの共通のフォーマットとして、あるいはデータベース管理されるような大規模なデータの保管形式として利用され始めている。

大容量の XML 文書の部分的な操作には、データを一旦すべてメモリに読み込む DOM (Document Object Model) 方式に比べてイベント駆動型の SAX (Simple API for XML) 方式の方が効率的である。SAX は XML 文書を先頭から順に読み込み、要素が現れるたびに対応するイベントハンドラを呼び出すという方式を取っている。SAX はイベントの通知を終えた要素のデータを保持しないため、不必要なオブジェクトを生成せず少ないメモリ使用量で XML 文書を処理できる。

SAX はデータの効率的な探索を目指した簡潔なモデルと言えるが、仕組が単純で XML 文書を操作する基本的な機能しか提供されていないため、プログラムが直感的でなく煩雑になりがちである。多くの場合、ユーザはイベントハンドラに、他のイベントの結果を利用するような処理を記述しなければならない。例えば、ある要素の親子関係や兄弟関係をたどってデータを探索したり変換する場合、プログラマは探索条件に一致する要素を、その要素の存在する階層や発見順、名前などの情報を元に選別するためのプログラムを書かなければならない。

本研究では、欲しいデータを XML 文書のルートからの経路 (path) により指定し、かつ path 上にある要素や複数の要素からなる部分構造を効率的に取得、編集できるライブラリとして Xeal を開発した。Xeal は特定の部分構造を指定・抽出するための、XPath をベースとした言語を path の記述に用いる。XPath は XML 文書中の特定の要素を指定する記述方法を定めた規格で、記述の仕方がファイルシステムの指定に似てシンプルで分かりやすいという利点を持つ。しかしながら、XPath は単なる位置指定を目的とした記述方式なので、単体では path 上の要素や部分構造を抽出することができない。また、要素指定に XPath を利用した XSLT

(XML Stylesheet Language Transformations) という言語は、XML 文書の別フォーマットへの変換処理に特化しており、本研究の目指す大容量 XML 文書からの効率的なデータの探索・抽出には向いていない。

Xeal は、XML 文書の部分的なデータを指定する path 記述だけでなく、用途に応じた構造、型でデータを取得するためのアブストラクションも提供する。それにより、ユーザは探索して取り出されるデータの格納先クラスやそれがマッピングされるデータ構造を、path 指定と共に記述することができる。Xeal は、ユーザが記述した path を元に必要なクラスファイルを生成するクラス生成器を提供する。探索されたデータは、生成されたクラスへ指定通りにマッピングされユーザに渡される。

実験では、Xeal と要素の探索に使われる従来の手法とで探索時間の比較を行い、これらの手法に対して Xeal が高速に探索を行えることを確認した。比較した従来の手法として、XSLT を用いてデータを抽出した後にそのデータを DOM で操作するという手法と、DOM で XML 文書をメモリ上に読み込んだ後に必要なデータを XPath で指定して取り出すという手法を用いた。

謝辞

本研究を進めるにあたり、研究の方向付けや進め方について多くの有用な助言を頂き、指導してくださった千葉滋先生に大変感謝しております。

佐藤芳樹氏には、プログラムの細かい仕様から参考になる研究、論文の書き方について、数々の助言を頂きました。西澤無我氏には、実装上の問題や論文の書き方など、私の相談を親身になって聞いて頂きました。また研究の進み具合を気にしてくださった柳澤佳里氏、松沼正浩氏、そして論文のスタイルファイルを作り残して頂いた光来健一先生に感謝いたします。

そして、共に研究活動をおこなった研究室の皆さんに、心から感謝いたします。

目 次

第1章 はじめに **7**

第2章 既存の XML 文書編集ライブラリとその問題点 **10**

2.1 XML 10
2.2 XML 文書編集ライブラリ 12
2.2.1 DOM 12
2.2.2 XPath Engine 13
2.2.3 XML-Java Data Binding ツール 16
2.3 大容量の XML 文書を編集するライブラリへの要求 19
2.3.1 使用メモリ量の節約 19
2.3.2 文書中の要素の参照・更新 19
2.4 大容量の XML 文書の編集が可能な既存技術とその問題点 . 20
2.4.1 SAX 20
2.4.2 XSLT 21
2.4.3 問題点のまとめ 23

第3章 Xeal: データの更新が可能な XML 編集ライブラリ **26**

3.1 特徴 26
3.2 仕様 27
3.2.1 利用例 27
3.2.2 生成されるクラスの仕様 28
3.2.3 拡張した XPath の文法 30

第4章 実装 **33**

4.1 Class Generator 33
4.1.1 Path Parser 33
4.1.2 Class Generator 33
4.2 Element Searcher 33
4.2.1 Forwarding Transrator 34
4.2.2 Path の変換アルゴリズム 36
4.2.3 Element Searcher 38
4.2.4 XML Parser 39

第5章 実験 **40**
5.1 他の API との実行時間の比較 40
5.1.1 実験環境 40
5.1.2 実験結果 41
5.1.3 考察 41

第6章 まとめ **44**
6.1 今後の課題 44

付 録A Xeal の自動生成したクラスの例 **48**

付 録B 拡張した XPath の文法規則 **50**

図目次

2.1 XML 文書の例 11
2.2 図 2.1 の XML 文書を表す DOM の構造モデル 13
2.3 図 2.1 の XML 文書の一部を取り出す XPath 記述 14
2.4 図 2.3 の XPath 記述に一致した要素 15
2.5 XML-Java Data Binding における XML 文書とスキーマ、生成されたクラスと与えられるオブジェクトの関係 16
2.6 図 2.1 の XML 文書の構造を定める XML Schema 17
2.7 図 2.1 を HTML 文書へ変換する XSLT スタイルシートの例 24
2.8 図 2.7 のスタイルシートによる変換結果 25

3.1 Xeal におけるデータバインディングの仕組み 27

4.1 Class Generator 内で行われる処理の流れ 34
4.2 Element Searcher 内で行われる処理の流れ 35
4.3 探索に XML 文書の後戻りが必要な Path による解析の順序 36
4.4 Xpath 式の X-Tree 表現とそれを変換して得られた X-Dag 構造 38
4.5 Element Searcher 内で利用される得られたデータの中間構造 39

5.1 各ライブラリで探索に必要な時間を比較したグラフ 41
5.2 図 5.1 の原点付近を拡大したグラフ 42
5.3 Xeal と SAX における、XML 文書の解析に要する時間の比較 43

第1章　はじめに

現在、データの記述言語として XML が注目されている。XML はタグを使って記述される拡張可能なマークアップ言語である。XML の仕様は標準化団体である W3C によって規定されており、特定のプラットフォームやプログラムに依存した規格ではない。そのため XML は、様々な分野でソフトウェア間、企業・団体間における標準のデータ記述形式として利用されている。

XML が標準のデータ記述形式として利用されるに伴い、データベースで管理されるような大きなデータを XML 文書で取り扱う機会も増えてきた。例えば複数の企業間で、製品のニーズや要望に関するアンケートを集計したデータを交換したり、複数の医療機関で患者の電子カルテの受け渡しをしたりという利用法が挙げられる。

このような大容量の XML 文書に対して必要な操作としては、XML 文書から別のファイル形式へのフォーマットの変更や、必要なデータの抽出、データの追加などといった操作が考えられる。その中で本研究は、XML 文書の一部を抽出し、データの変更を行いそれを元の XML 文書へ反映させるという操作に注目した。このような操作が必要になる例として、医療機関で利用される電子カルテの中から投薬に関するデータを抽出し、そのデータに基づいて薬の費用をカルテへ記入するという操作や、企業の部署名の変更に伴い、社員名簿を表す XML 文書の中から特定の部署名を抽出して変更するといった操作が挙げられる。

大容量の XML 文書の操作に利用される既存の技術としては SAX や XSLT が存在する。SAX は XML 文書を先頭から読み込み、「要素の開始」や「要素の終了」、「文字列の発見」といったイベントを順に発生させてそれをハンドラに通知する。プログラマはあらかじめイベントを受け取った時の処理をハンドラに定義しておく事で、発生したイベントに対応する処理をハンドラに次々と実行させる事ができる。

しかし SAX のハンドラは、ある要素から親子関係や兄弟関係にあたる要素を参照して取り出すといったプログラムを記述するのが大変であるという欠点がある。また SAX のハンドラは、XML 文書の後戻りをすることはできない。そのためデータの更新作業をする場合、データを参照するためのハンドラとは別に、データの変更を元の XML 文書へ反映させるた

めのハンドラをプログラマが記述しなければならない。そのため XML 文書から複雑な構造をしたデータを抽出して更新する場合、記述が大変であるハンドラを 2 つ用意しなければならない。

XSLT は XML 文書の変換規則を記述するための言語である。XSLT では XML 文書を木構造のデータとして考え、XML 文書に含まれるそれぞれの要素を木構造におけるノードと見立てている。それぞれのノードに対してどのような変換を行うかを XSLT の文法に従い記述する事で、文書全体の変換方法を定めることができる。この変換方法が記されたスタイルシートを元に、XML 文書から必要なデータが抽出された XML 文書を作成する事ができる。この抽出された XML 文書を DOM などの技術で操作することにより、大容量の XML 文書を少ないメモリで操作する事が実現できる。

しかし XSLT を用いて XML 文書中のデータを更新するには、SAX の場合と同様、データを抽出するためのスタイルシートの他に、抽出された XML 文書に加えられた変更を元の XML 文書へ反映させるスタイルシートを記述しなければならない。

我々は大容量の XML 文書に対してデータの参照と更新を効率よく行うことのできるライブラリとして、Xeal を提案する。Xeal は、XML 文書中の要素を指定するための言語である XPath を拡張する事で、XML 文書の一部分を取り出して参照する事や、得られたデータを更新することが可能である。また XML 文書に対して、その XML 文書を操作するための API を生成する XML-Java Data Binding を利用する事で、得られた要素を利用者の利用しやすいデータ構造で渡す事ができる。

自動生成されたクラスには、オブジェクトに加えられた変更を XML 文書へ反映させるためのメソッドが用意されており、Xeal の利用者はこのメソッドを呼び出すだけでデータの更新を行う事が可能である。この処理では、探索時に取り出したデータのファイル内における位置を記憶しておく事で、変更を反映する必要のあるデータの位置を XML 文書の構造を辿らずに見つけることを実現している。

XML-Java Data Binding とは、特定の XML 文書に対してその文書のデータを保持し操作するための Java のクラスを自動生成し、利用者に提供するという技術である。利用者はそのクラスを利用して、XML 文書内の要素の参照や変更を行う事ができる。通常の XML-Java Data Binding では、スキーマと呼ばれる XML 文書の構造を規定する文書をスキーマコンパイラに与える事で、クラスが自動生成されるが、Xeal では、探索に用いられる Path を Class Generator に与えることでクラスを生成するという仕組みを用いている。

また XML 文書から探索したい部分構造を変更する場合も、Xeal では

変更した Path を元にクラスを生成させるだけでよい。SAX や XSLT のように、ハンドラやスタイルシートを変更する必要はない。

そして Xeal は、本来 XML 文書を先頭から読み込むだけでは探索できないような XPath の記述にも対応している。これには Xaos [1] アルゴリズムを参考にして作成した、与えられた Path を先頭から読み込んで探索できる形へ変換するアルゴリズムを利用している。これにより複雑な構造を指定する Path に対しても一致する部分構造を探索する事が可能である。

以下、2 章では既存の XML 編集ライブラリとその問題点について述べる。3 章では Xeal の仕様、利用例について述べる。4 章では Xeal を構成するモジュールと実装のポイントについて述べる。5 章では Xeal と他の編集ライブラリとの、探索に要する時間を比較した実験について述べる。最後に、6 章で本論文をまとめる。

第2章　既存のXML文書編集ライブラリとその問題点

2.1　XML

XML (Extensible Markup Language) [11] とは 1998 年 10 月に W3C (World Wide Web Consortium) より勧告された、拡張可能なマークアップ言語である。

XML の仕様は ISO (International Organization for Standardization) によって制定された、文書の意味構造を記述するための言語である SGML (Standard Generalized Markup Language) のサブセットである。タグの利用や文書の構造化といった特徴を持つ SGML の長所を生かし、よりシンプルで Web 環境に対応した仕様として XML は定められた。

XML の特徴の 1 つとして、マークアップ (Markup) 言語であることが挙げられる。マークアップとは、目的に応じてデータに意味を持たせることを指す。例えば「`50`」というデータがあった場合、これが何を表すのかは不明瞭である。しかし「`<age>50</age>`」と書くことで、この数字が年齢を表すものだと利用者は理解する事ができる。XML の場合、`<`と`>`を使ったタグでデータを括ることでマークアップを行っている。

XML は拡張可能な (Extensible) 言語であるが、ここで拡張可能とは独自のタグを利用した記述が可能なことを指している。例えば XML と同様のマークアップ言語として HTML (Hyper Text Markup Language) がある。しかし HTML では決められた名前のタグしか利用する事ができない。一方、XML では利用者が自由にタグを使って記述することができる。

また XML はメタ言語である。メタ言語は、その言語を元にして別な言語を作ることができる。XML 文書の変換を行うルールを記述する言語である XSLT [15]、数式を記述するための言語である Math ML [12] は XML を元にして作成された言語である。

XML は現在、文書の記述のほかデータを記述する言語として利用されている。データの記述に XML を利用することで得られる利点を挙げる。

- プラットフォームに依存しない形式

  XML 文書は特定の OS やプログラムに依存した形式ではない。これにより、どんな OS やプログラムからも操作する事が可能である。

```
<?xml version="1.0" encoding="SHIFT_JIS"?>
<bibliography>
   <book isbn="4-04-100112-9">
      <title>こゝろ</title>
      <author>夏目漱石</author>
      <published-year>1951</published-year>
      <price>300</price>
   </book>
   <book isbn="4-04-110003-8">
      <title>堕落論</title>
      <author>坂口安吾</author>
      <published-year>1969</published-year>
      <price>420</price>
   </book>
   <book isbn="4-10-100605-9">
      <title>人間失格</title>
      <author>太宰治</author>
      <published-year>1985</published-year>
      <price>286</price>
   </book>
</bibliography>
```

図 2.1: XML 文書の例

- 標準化された言語

  XML は標準化団体である W3C によって制定された言語で、特定の企業によって開発された言語ではない。そのため異なる企業・団体間で利用されるデータの記述形式として XML は広く利用されている。

- 人間にとって理解しやすい形式

  XML 文書はテキスト形式であるため、通常のエディタで編集を行うことができる。またタグに意味のある名前をつける事が可能なので、人間にとってそのデータが何を表しているのか理解しやすい言語であると言える。

- データの内容と表現の分離

  XML 文書はデータを記すもので、それを表現する方法は規定されていない。これにより 1 つの XML 文書から、それを変換して Web

ページとして表示させたり、それをプログラムから読み取ってデータとして利用したりする事が可能になる。

XML の欠点の1つは、データが冗長になってしまうという事である。データがバイナリ形式でなくテキスト形式である事や、マークアップ記述の量が多いことが原因である。

2.2　XML 文書編集ライブラリ

XML で記述されたデータをプログラム上から扱う場合、XML 文書の作成や要素の参照・変更を行うためのプログラムを書かなければならない。それらの作業は、XML 文書の編集ライブラリを利用する事で簡単に行うことができる。

2.2.1　DOM

DOM(Document Object Model) [10] は、XML 文書の内容や構造をプログラムが動的に参照・更新する事を可能にするために開発された API である。W3C により標準化作業が行われ、1998 年 10 月に DOM Level1 が策定された。現在は DOM Level2 が公開されている。

DOM を利用することで、プログラマは XML 文書の構築や、その構造の走査、XML 文書への要素と内容の追加、修正、削除を行うことができる。またそれらの処理を元の文書へ反映させる事ができる。DOM は XML 文書の表す構造と似た構造のオブジェクトを用いて XML 文書を操作する。その構造は極めて木構造に近い形をしており、「構造モデル」と DOM の勧告の中で命名されている。図 2.2 は図 2.1 の XML 文書を DOM の構造モデルで表したものである。

DOM の仕様では XML 文書を表現するための標準的なオブジェクトや、それらの組み合わせ方のモデル、それらを操作するためのインターフェースが定められている。また DOM は言語やプラットフォームに非依存の API である。そのため勧告ではその仕様のみが定められており、実装は複数の団体・企業により、Java や C++などの様々な言語で与えられている。

Java では、DOM での提供するインターフェースは JAXP (Java APIs for XML Processing) の一部として `org.w3c.dom` パッケージに含まれており、Java のインターフェースとして用意されている。これにより利用者は、API の実装に依存しない形でプログラムを記述する事ができ、後にその実装を変更することになってもプログラムを変更せずに済むという利点を得られる。JAXP の提供する DOM のインターフェースは、Crimson [6] や Xerces2 Java [8] といった XML ライブラリによって実装されている。

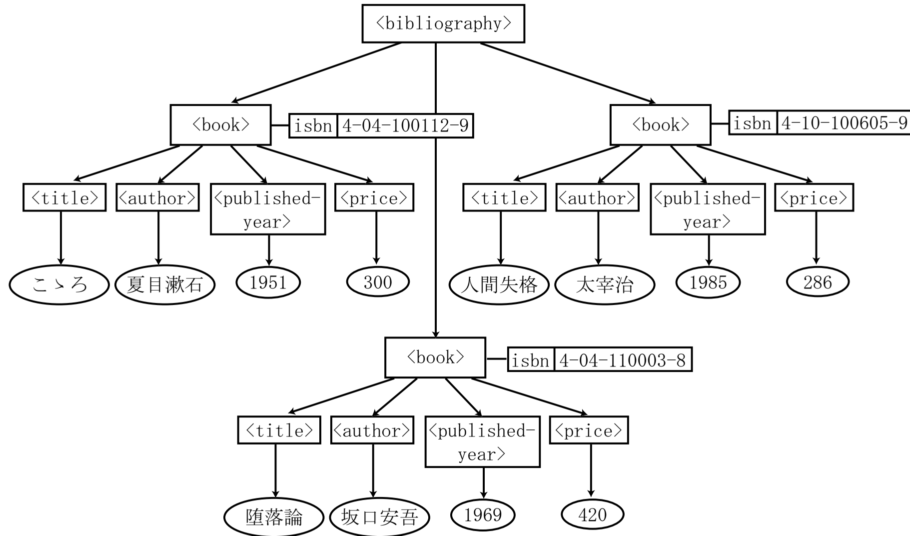

図 2.2: 図 2.1 の XML 文書を表す DOM の構造モデル

2.2.2　XPath Engine

XPath Engine とは、XML 文書の一部を参照するための言語である XPath [13] によって書かれた記述を元に、XML 文書内から指定された要素を取り出すためのプロセッサを指す。代表的な XPath Engine として、Codehaus が開発している jaxen [9] や、Apache Project が開発している Xalan [7] が挙げられる。Xalan はその機能の一部として XPath Engine を備えている。

XPath とは、W3C で勧告された XML 文書の一部を参照するためのアドレッシング言語である。現在は XPath 1.0 が勧告として公開されている。当初は XSLT [15] と XPointer という言語内で XML 文書中の要素を指定するために利用されることを目的としていた。XML の文法を用いずに定義されたコンパクトな文法で書かれている。

XPath では、XML 文書を幾つかの種類のノードから構成される木と考える。XML 文書中に含まれる、あるノードを指し示すためには、起点となるノードからその目的のノードに至る経路を指定すればよい。XPath ではこの経路という概念を用いて要素を指定している。簡単な XPath の例を挙げる。

- `/bibliography/book`

  `/`という文字は XPath の一番左にある場合ルートノードを意味する。これは XML 文書の最上位にある要素 (図 2.1 の XML 文書では `bibliography` 要素) の、さらに親にあたるノードで、探索の起点と

なるノードである。この XPath 式は「ルートノードの子にあたる bibliography 要素の、さらに子である book 要素」を指している。

- /bibliography/book/title/text()

  text() は、探索の対象にテキストノードを指定する記述である。テキストノードとは、XML 文書中に含まれる文字列の事を言う。この XPath 式は「ルートノードの子にあたる bibliography 要素の、さらに子である book 要素の子の title 要素の、子である文字列」を指している。

- /bibliography/book@isbn

  @記号はアトリビュートノード、つまり要素の属性を探索の対象に指定する記述である。この XPath 式は「ルートノードの子にあたる bibliography 要素の、さらに子である book 要素の isbn という属性値」を指している。

- /descendant::book/title

  descendant::book はその直前の要素に対して子孫関係にある book という要素を指している。子孫関係とは子要素の子要素など、そのノードよりも深い位置に存在する要素との関係の事を指す。この XPath 式は「ルートノードの子孫である book 要素の、子である title 要素」を指している。

  <axisname>:: という記述は、XPath の文法の中で axis (基準点) と呼ばれる。axis の種類は XPath 1.0 の勧告内で定められており、他にも親関係を指定する parent 、先祖関係を指定する ancestor 、同じ階層でその要素より後ろにある要素との関係を指定する preceding-sibling などの axis が定められている。

その他に XPath では、文字列や数値、ブール値を操作する基本的な関数も提供している。図 2.3 は図 2.1 の XML 文書の一部を参照する XPath の記述である。

```
/bibliography/book[number(published-year/text())
                    >1960 and number(price/text())<400]
```

図 2.3: 図 2.1 の XML 文書の一部を取り出す XPath 記述

この path は、bibliography 要素の子要素である book 要素のうち、published-year 要素の内容を数字に変換したものが 1960 より大きく、かつ price 要素の内容を数字に変換したものが 400 より小さいという条

件を満たすものを指定している。図 2.4 は、この path に一致した要素を表している。

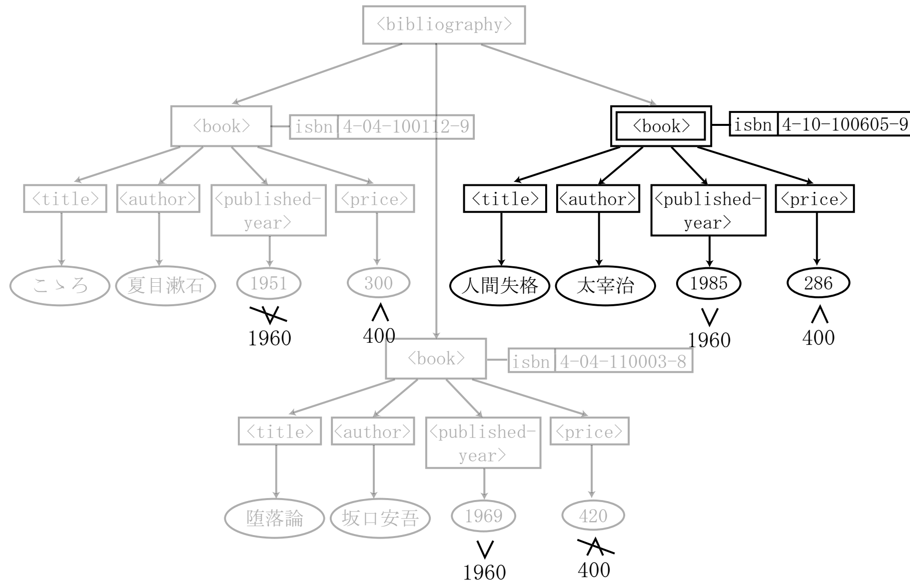

図 2.4: 図 2.3 の XPath 記述に一致した要素

Xalan Java, jaxen といった XPath Engine はそれ単独で XML 文書から要素を取り出すことはできず、別の XML 文書編集ライブラリと共に用いられる。Xalan Java の場合、DOM の実装を与える Xerces2 Java などと連携することにより、XML 文書を表す DOM のモデル構造から、与えられた XPath 記述に一致する要素を取り出してプログラムに渡す事ができる。図 2.4 の場合、Xalan Java による探索の結果渡されるのは二重の四角に囲まれた `book` 要素であり、プログラムはその `book` 要素を介して子要素である `title` 要素や `author` 要素の内容を参照することができる。

XPath Engine を用いて DOM の文書モデルから要素を取り出すという方法により、利用者はプログラミングの負担を軽減する事ができる。なぜなら DOM の文書モデルを辿って特定の要素を取り出すプログラムを書くには、ループ表現や条件式をプログラム内に沢山記述することになり、プログラマにとって負担になるからである。XPath Engine を用いることで、このような探索のためのプログラムを記述する負担を解消する事ができる。

2.2.3　XML-Java Data Binding ツール

XML 文書の構造が記されたスキーマと呼ばれる文書を元に、その XML 文書を操作するための Java クラスやインターフェースを自動生成するプログラムが存在する。これらのプログラムを、XML-Java Data Binding ツールと呼ぶ。DOM を用いると XML 文書を木構造のデータとして扱うことができるが、木構造のデータを直接扱うプログラムは、要素を取り出すためにループや条件分岐を多用しなければならない。そのようなプログラムの記述はプログラマにとって負担となる。XML-Java Data Binding ツールを用いると、プログラマは自動生成された Java クラスを利用して、XML 文書を操作するプログラムを書くことができる。実行時には XML 文書から自動生成されたクラスへデータがマッピングされ、プログラマはそのクラスのオブジェクトを介して文書内のデータを参照・修正する事が可能になる。またオブジェクトへ加えた変更を、元の文書へ反映させる事も可能である。図 2.5 は、XML 文書と XML 文書のスキーマ、生成されたクラスと実行時に与えられるそのクラスのオブジェクトの関係を示したものである。

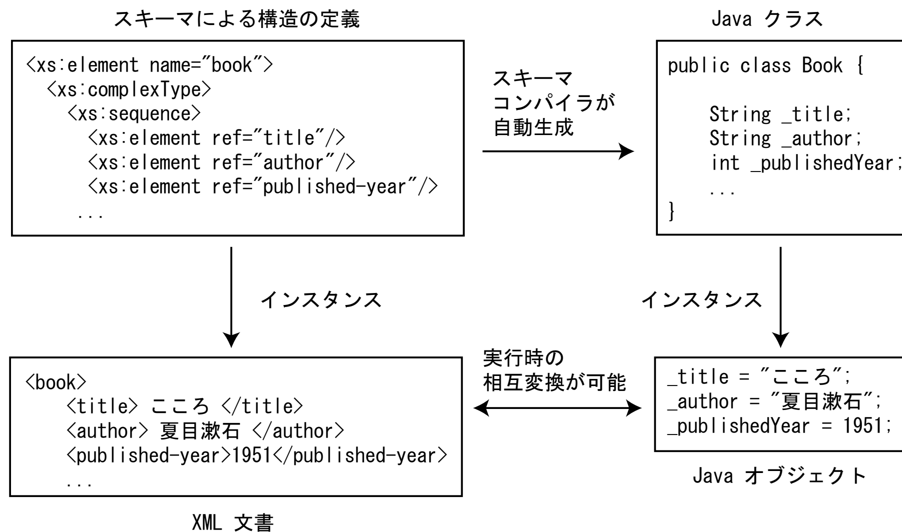

図 2.5: XML-Java Data Binding における XML 文書とスキーマ、生成されたクラスと与えられるオブジェクトの関係

スキーマを記述するための言語として代表的な、XML Schema [14] と呼ばれる言語によって書かれた、図 2.1 の XML 文書のスキーマを図 2.6 に示す。

図 2.6 のスキーマを、Castor の用意する Source Generator に与える事

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">

  <xs:element name="bibliography">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="book"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="author"/>
        <xs:element ref="published-year"/>
        <xs:element ref="price"/>
        <xs:element ref="publishing-company"/>
      </xs:sequence>
      <xs:attribute name="isbn" use="required"
                                      type="xs:string"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="xs:string"/>
  <xs:element name="published-year" type="xs:string"/>
  <xs:element name="price" type="xs:string"/>
  <xs:element name="publishing-company" type="xs:string"/>
</xs:schema>
```

図 2.6: 図 2.1 の XML 文書の構造を定める XML Schema

で、このスキーマに添って書かれた XML 文書を操作するための class のソースコードが出力される。これらのクラスを用いる事で、XML 文書の編集が簡単になる。

XML-Java Data Binding の利点

XML-Java Data Binding ツールの利点として、XML 文書である事を意識せずに XML 文書の操作が可能であるということが挙げられる。DOM で XML 文書の構造モデルを操作する場合、XML 文書またはそのスキーマを見て XML 文書がどのような構造をしているか分からない限り、XML 文書を操作する事ができない。一方 XML Data Binding を利用して得られたクラスはその XML 文書の構造を表しているため、そのクラスの構造を知る事で得られたデータの操作を行う事が可能で、元の XML 文書やそのスキーマを見る必要はない。

また DOM の構造モデルは自由に要素の追加が行えるため、スキーマの定義する構造に当てはまらない XML 文書が作成されてしまう恐れがある。一方 Data Binding ツールによって自動生成されたクラスは、そもそもスキーマの定義に当てはまらない要素の追加が不可能であるような構造をしており、そのためスキーマ定義を無視した XML 文書の作成を防ぐことができる。

現在 Java-XML Data Binding Framework としては JAXB、Castor、Relaxer というツールが代表的である。

Castor

Castor [2] は ExoLab Group によって開発されているオープンソースのマッピングツールである。XML Schema というスキーマ言語に対応している。Castor 独自の機能として、XML 文書を既存のクラスへマッピングさせる事が可能である。これはクラスと別に、マッピングの方法を定義する定義ファイルを記述する事で実現している。またリレーショナルデータベースから SQL を利用して得られたデータから Java のオブジェクトへのマッピングなど、XML-Java Data Binding 以外のマッピング機能も備えたツールである。

JAXB

JAXB (Java Architecture for XML Binding) [5] は Sun Microsystems によって開発されている Data Binding ツールである。DTD と XML Schema というスキーマ言語に対応している。現在は Java Web Services

Developer Pack の一部として利用する事が可能である。

Relaxer

Relaxer [17] は浅海智晴氏によって開発されたマッピングツールである。RELAX NG と DTD というスキーマに対応している。生成するクラスに Composite パターンや Visitor Pattern を利用するためのメソッドを加える機能を備えている。これにより、アプリケーション開発の手間をさらに削減することができる。

2.3　大容量の XML 文書を編集するライブラリへの要求

データの記述に XML を利用する場合、データの増大に伴い文書の容量が非常に大きくなる場合が考えられる。ここではそのような大容量の XML 文書を編集するためのライブラリに必要な性能と機能について述べる。

2.3.1　使用メモリ量の節約

XML 文書全体をメモリ上に展開して操作する DOM や XML Data Binding Framework を用いて行う大容量の XML 文書の操作では、実行時に大量のメモリが必要になる。そのため、文書の一部または 1 つの要素をメモリ上に乗せるなど、別の手法で XML 文書を操作する必要がある。

2.3.2　文書中の要素の参照・更新

XML 文書中の要素の参照という操作は XML 文書からデータを得る上で不可欠だが、文書内の要素の更新が必要な場合も少なくない。これは大容量の XML 文書でも同様で、参照と更新、両方の機能を利用者に提供する必要がある。

要素の参照と更新が必要な例

現在、医療分野では電子カルテの導入が進んでいる。この電子カルテを例にとると、例えば、電子カルテを表す大容量の XML 文書の中から医薬品の情報のみを取り出し、医療費を計算してその結果を元の電子カルテへ反映させるといった状況が挙げられる。

他にも省庁間で電子文書の交換をする場合や電子申請など、大容量の XML 文書を取り扱う機会は多いと考えられる。

2.4　大容量の XML 文書の編集が可能な既存技術とその問題点

前節で、大容量の XML 文書の編集ライブラリに必要な性能と機能を述べた。この節では SAX と XSLT という少ないメモリ消費量で動作が可能な XML 文書編集ライブラリについて述べ、それぞれにおける問題点を指摘する。

2.4.1　SAX

SAX(Simple API for XML) [4] は XML-DEV というメーリングリスト内で行われた議論の中から開発された、XML 文書処理のためのイベント駆動型 API である。1998 年 5 月に SAX 1.0 が開発され、現在は SAX 2.0 が公開されている。SAX は XML 文書中の要素を認識するたびにイベントを生成させ、それをアプリケーションのハンドラに通知する。利用者はハンドラを実装したクラスを記述し、それを用意された XML 文書の Parser に与えることで、Parser により発生したイベントに応じてハンドラは処理を行う。

利点

- 少ないメモリの消費量

  SAX は先頭から順に XML 文書を読み込んでいき、イベントを生成し終わった要素のデータはメモリ上から解放する。そのため文書全体をメモリ上に展開する DOM や XML Data Binding Frameworks と違い、メモリの消費量が少ない。

欠点

- 対話的なデータの更新が困難

  SAX では `XMLFilter` クラスを用いる事で、先頭から読み込んで得られた XML 文書中の要素に対し、順にデータの変更処理を行うことはできるが、指定されたデータを利用者に渡し、変更されたオブジェクトを元の XML 文書へ反映させるための処理を行うための機

能はついていない。

SAX のようなイベント駆動型のライブラリで、探索したデータを一度利用者に渡し、その変更を元の文書へ反映させるといった対話的な変換を行うには、XML 文書を一度走査してデータを探索し、その変更をもう一度走査して反映させる、といった 2 度の走査が必要である。そのため SAX では、処理を行う速度を重視して、データを反映させる機能を付け加えなかったのだと考えられる。

XML 文書の変換・更新処理を行うことが可能であるが、SAX 自体がイベント処理のため、利用者に参照されたデータを渡し、その変更を XML 文書へ反映させる、といった処理には使用できない。なぜなら SAX は 1 つのハンドラでデータの参照と更新を同時に行うことはできず、そのため得られたデータを変更して元の XML 文書へ反映させたい場合は、もう 1 つデータを反映させるためのハンドラを記述しなければならない。これはプログラマにとって負担となる。

- ハンドラの記述にかかる負担

  プログラマは SAX を利用するためにハンドラを記述する必要がある。SAX のハンドラでは、DOM や XML Data Binding Frameworks のように XML 文書内の要素の親子関係を利用してデータを取り出す記述を書くのが難しい。また取り出したいデータの構造が変わった場合、その変更にかかる作業も大きな負担になる。

2.4.2　XSLT

XSLT (Extensible Stylesheet Language Transformations) [15] は XML 文書の変換規則を記述するための言語である。XSLT は、XSL と呼ばれる XML の表示や印刷のためのスタイルを指定する言語の一部であった。しかし XSLT の機能は表示や印刷だけにとどまらず、XML 文書から XML 文書・または別なフォーマットへの文書への変換を行う事が可能である。これは XML を利用する様々な分野で有効であり、そのため XSL から切り離されて独立した仕様となった。

XSLT では XML 文書を木構造に見立て、そのノード 1 つ 1 つに対してどのような変換を行うかを指定する事で、文書全体の変換を規定する。XSLT によって変換規則が記されたファイルをスタイルシートと呼ぶ。図 2.7 は図 2.1 の本のデータを表す XML 文書を Web での閲覧が可能な HTML 文書へ変換するためのスタイルシートである。変換された HTML 文書が図 2.8 である。

XSLT を用いて書かれたスタイルシートを用いて XML 文書の変換を

行うには、XSLT Processor を利用する必要がある。XSLT Processor は XSLT スタイルシートと XML 文書を入力として受け取り、変換後の XML 文書を出力して、ブラウザやプログラムへ出力を行う。XSLT スタイルシートと XSLT Processor を用いる事で、大容量の XML 文書から必要なデータのみを取り出した小さな XML 文書の生成を行う事が出来る。この変換後の XML 文書に対し DOM や SAX など XML 編集ライブラリを共に用いることで、大容量の XML 文書からデータを参照する事が可能になる。

XSLT の有効な利用例として、企業が製品を紹介する Web サイトを挙げる。全製品のデータが入った 1 つの XML 文書から、全ての製品の名前と金額のみを記した一覧ページや、1 つの商品の詳細を記したページ、また特定のジャンルの商品のみの情報を記したページなど異なった HTML 文書を、それぞれ XSLT スタイルシートを用いて出力することができる。各ページに同一のデータを記述するという手間も省ける上、商品のデータを 1 つの XML 文書で管理するため、データの修正・変更にも強い。

Apache Project の Xalan-Java という XSLT Processor は、入力となる XML 文書を DOM の構造モデルの形や `java.io.InputStream` 型で受け取り、与えられたスタイルシートに従って変換した上で、 DOM の構造モデルの形や `java.io.OutputStream` 型で返すことができる。

利点

- 少ないメモリの消費量

  XSLT では文書中の要素を先頭から読み込み、発見した要素に対してスタイルシートで定められた変換を行う。変換を終えた要素のデータはメモリ上から解放されるので、SAX と同様にメモリの消費量が少ない。

- 要素の指定に XPath の利用が可能

  テンプレートの内部で XPath 式を用いる事が可能である。XPath 式は、データを参照したり、次のパターンマッチングの対象を決めるのに利用される。SAX と違い、要素の親子関係を用いたデータの変換が容易になる。

欠点

- データの対話的な更新が困難

  XSLT スタイルシートを用いて XML 文書の変換を行う場合、XML

文書を先頭から読み込んで得られた XML 文書中の要素に対して順に変換処理を行うことはできるが、SAX の場合と同様、探索して得られたデータを一度利用者に渡し、その変更を元の文書へ反映させるといった対話的な変換を行うには、2 度 XML 文書を走査する必要がある。そもそも XSLT スタイルシートはパターンマッチングを用いて変更を行うための言語で、スタイルシートを用いて抽出された XML 文書へ加えられた変更を、元の XML 文書へ反映させるような機能をサポートしていない。

- 複雑なデータの処理

  XSLT ではデータの更新を行う際、複雑な関数の記述を行う事ができない。SAX は Java のプログラムで記述できるため、外部の関数やライブラリを利用し、非常に複雑な処理を行うことも可能だが、XSLT は独自に決められた関数を用いなければならないからである。

2.4.3　問題点のまとめ

XSLT も SAX も大容量の XML 文書を少ないメモリで操作する事が可能な編集ライブラリだが、両方とも XML 文書の更新という点では問題点がある。なぜなら、SAX でも XSLT でもデータ更新という機能はついておらず、もし SAX で文書の更新を行いたい場合は、データの取り出すための `ContentHandler` とは別に、文書の更新をするための `XMLFilter` を記述しなければならない。また XSLT では、取り出したデータに加えた変更を元の XML 文書へ更新するという機能は、XML 文書の変換規則を記すための言語としては範囲外であると言える。また探索したい要素の構造を変更する場合、SAX ではハンドラの記述を、XSLT ではスタイルシートをそれぞれ変更しなければならず、これは保守性に欠けていると考えられる。

```
<?xml version="1.0" encoding="shift_jis"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:bib="http://www.example.com/bibliography">

  <xsl:output method="html" encoding="shift_jis"/>

  <xsl:template match="/">
    <html><body>
      <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="bib:bibliography">
    <table border="2"><tr>
        <td>id</td>
        <td>「題名」著者</td>
        <td>出版年</td>
      </tr>
      <xsl:apply-templates/>
    </table>
  </xsl:template>

  <xsl:template match="bib:book">
    <tr>
      <td><xsl:value-of select="@id"/></td>
      <td>
         「<xsl:value-of select="bib:title/text()"/>」
         <xsl:value-of select="bib:author/text()"/>
      </td>
      <td>
        <xsl:value-of
             select="bib:published-year/text()"/>
      </td>
     </tr>
  </xsl:template>
</xsl:stylesheet>
```

図 2.7: 図 2.1 を HTML 文書へ変換する XSLT スタイルシートの例

```
<html>
<body>
  <table border="2">
    <tr>
       <td>id</td>
       <td>「題名」著者</td>
       <td>出版年</td>
    </tr>
    <tr>
       <td>4-04-100112-9</td>
       <td>「こころ」夏目漱石</td>
       <td>1951</td>
    </tr>
    <tr>
       <td>4-04-110003-8</td>
       <td>「堕落論」坂口安吾</td>
       <td>1969</td>
    </tr>
    <tr>
       <td>4-10-100605-9</td>
       <td>「人間失格」太宰治</td>
       <td>1985</td>
    </tr>
  </table>
</body>
</html>
```

図 2.8: 図 2.7 のスタイルシートによる変換結果

第3章 Xeal: データの更新が可能なXML編集ライブラリ

3.1 特徴

Xealは、XPathを拡張した文法によって記述されたpathに一致する部分構造をXML文書から取り出し、それをJava-XML Data Binding技術を利用し、利用者にとって扱いやすいオブジェクトの形として渡すことができるライブラリである。

通常、Java-XML Data Bindingを行うツールでは、スキーマを元にXML文書を操作するAPIを自動生成するが、Xealでは、探索に用いるpathを元にそれらのクラスを自動生成する。利用者は、スキーマではなくpathをXealのクラスファイル生成器に与える事で、XML文書の一部を操作するクラスを得る事ができる。そのクラスを用いる事で、XML文書から探索されたデータを扱いやすい形で操作できる。図3.1は、Xealにおけるpathと Javaクラス、そしてXML文書の関係を示している。以下で、Xealの特徴的な機能や仕様について説明する。

XML文書の更新

利用者はXML文書の更新を簡単に行うことができる。自動生成されたクラスのうち、XML文書から得られたデータを保持するためのクラスには`marshal`メソッドが用意されている。利用者は生成されたクラスのインスタンスとしてXML文書から探索して抽出されたデータを受け取るので、それに変更を加えた後にこのメソッドを呼ぶと、そのオブジェクトに加えられた変更を、元のXML文書へ反映させることができる。

Java-XML Data Binding の利用

XealはJava-XML Data Bindingを利用し、探索して得られたデータを扱うためのクラスファイルを自動生成する。この自動生成されたクラスを利用する事で、利用者はXSLTやSAXのスタイルシート・ハンドラの記述という手間を省く事ができる。

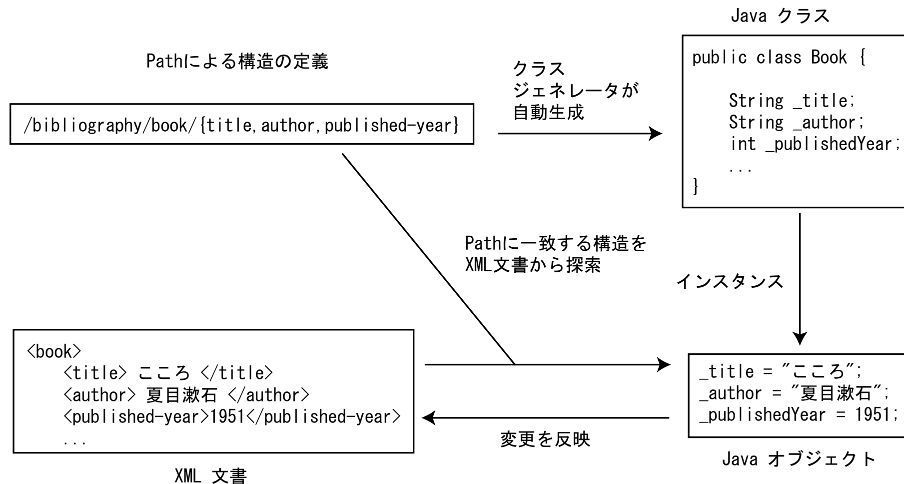

図 3.1: Xeal におけるデータバインディングの仕組み

拡張した XPath の利用

Xeal では XPath を、要素を指定するものではなく、XML 文書中の部分構造を指定するための言語として利用している。そのために、XPath に対して拡張を行った言語を path の記述言語に定めている。

また、XPath 中の axis の記述に制限がある XSLT とは違い、`ancestor` や `following-sibling` といった全ての axis を使って XML 文書中の要素を指定する事ができる。

3.2　仕様

Xeal は大きく分けて 2 つのモジュールから成り立つ。1 つは、path を元に XML 文書から得られたデータを扱うためのクラスファイルを自動生成する Class Generator、もう 1 つは、生成されたクラスファイルが実行時に要素を探索するために利用する Runtime Library である。

3.2.1　利用例

Xeal を利用する例を示す。探索の対象となる XML 文書として、図 2.1 の XML 文書を例に取る。この XML 文書から次のような path を用いてデータの探索を行う。

```
/bibliography/(this[])book/{title/text(),
                author/text(), (int)price/text()}
```

this[] キーワードは、Book クラスを生成して、かつそれを Bibilography クラスで配列として持たせるという意味を持つ。この path を、次のようなコマンドを利用する事で Xeal の Class Generator へ与える。

```
/> java xeal.generator.ClassGenerator /bibliography/
(this[])book/{title/text(), author/text(), (int)price/
text()}
```

この命令によって生成されたクラスのフィールドとメソッドは付録に示す。これら生成されたクラスのインスタンスとして、利用者は XML 文書から探索して得られたデータを受け取る事ができる。例えば得られたデータから本のデータを出力するプログラムは、次のように書く事ができる。

```
FileInputStream in =
       new FileInputStream("bib.xml");

Bibliography bib =
                Bibliography.bind(in);

Book[] books = bib.getBook();
for(int i=0;i<books.length;i++) {
    System.out.println("book number:"+i);
    System.out.println(books[i].getTitle());
    System.out.println(books[i].getAuthor());
    System.out.println(books[i].getPrice());
}
```

3.2.2　生成されるクラスの仕様

上の例を用いて、生成されるクラスの仕様について説明する。
生成されたクラスには、次のメソッドとフィールドが付加される。

- path フィールド
- <fieldname>_position_in_file フィールド
- <fieldname>_size フィールド
- フィールドの Setter/Getter メソッド

- bind メソッド
- reflect メソッド

それぞれのフィールド・メソッドの機能と用途を説明する。

path フィールド

path フィールドは String 型の static フィールドである。利用者により与えられた path が保存される。利用者より与えられた path を代入する命令が、コンストラクタに加えられる。

<fieldname>_position_in_file フィールド

このフィールドは long 型の値、または long 型の配列である。XML 文書から探索して得られたデータが<fieldname>フィールドへ代入される時に、そのデータがある XML 文書内の位置が代入される。このフィールドに保存された値は、reflect メソッド内で利用される。そのためアクセス修飾子は private である。

<fieldname>_size フィールド

このフィールドは、<fieldname>フィールドが配列や List クラスなど、複数のデータを格納する場合、クラスに付加される。XML 文書から探索して得られたデータが代入される時に、該当したデータの数を保存する。

marshal メソッド内でこの値とフィールドのサイズが比べられ、一致しないときは SizeUnmatchedException 例外が発生する。

Setter / Getter

これらは、そのクラスのフィールドへ参照・代入を行うためのメソッドである。

bind メソッド

bind メソッドは static メソッドで、XML 文書へのパス名を引数に取る。返り値は、path に一致する要素を引数の XML 文書から探索し、その型のオブジェクトに変換したものである。

reflect メソッド

reflect メソッドは、元の XML 文書へ変更を反映させるためのメソッドである。

3.2.3　拡張した XPath の文法

XML 文書中の部分構造を指定できるよう拡張した XPath 1.0 の文法について説明する。

XPath は本来、XML 文書中の要素を指定する経路の記述を目的とする言語で、XML 文書中の部分構造を指定する事はできない。また XML 文書から得られたデータをどのような Java の型・クラスで得たいかを指定する記述も XPath に盛り込む必要があった。

拡張した文法は 4 つあり、以下でそれを説明する。

探索要素を複数指定するための記述

XML 文書中の部分構造を指定させるために、XPath に複数の要素を探索するための記号として波カッコ記号 {} を導入した。

```
/bibliography/book/{title/text(), author/text()}
```

このような記述により、book 要素の子要素として title 要素と author 要素を指定する事ができる。生成される Book クラスには title と author、2 つの要素を格納するためのフィールドが生成される。

Java のデータ型を指定するための記述

XML 文書内の要素をどのような Java のデータ型で受け取りたいかを指定させるために、キャスト記述 (<typename>) を導入した。path を元に生成されるクラスのフィールドは、指定がない限り String 型となるが、XML 文書中のデータを int 型や double 型といった数値を表現する型で受け取りたい場合も考えられる。また path に一致する要素が複数ある場合、それらを配列で得たいか、java.util.Vector クラスのオブジェクトとして得たいか、java.util.LinkedList クラスのオブジェクトとして得たいかは、利用者の状況によって異なる。それらを指定するために導入された文法である。下の例を用いて、キャスト記述の利用法について説明する。

```
1. /bibliography/book/{title/text(), (int)price/text()}
```

1. の path では、price という名前を持つ要素に対して int 型のキャストを行っている。この記述により price に対応して生成されるフィールドの型は int 型になる。XML 文書の内容を表す String 型からこれら premitive 型へのキャストは、それぞれのラッパークラスのメソッドである perseInt、parseDouble といったメソッドを利用している。

```
2-1. /bibliography/book/
                {title/text(), (String[])author/text()}
2-2. /bibliography/book/{title/text(),
             java.util.LinkedList<String>)author/text()}
```

2-1. の path では、author という名前を持つ要素に対して String[] 型のキャストを行っている。この記述により author に対応して生成されるフィールドの型は String[] 型になる。例えばある XML 文書内の book 要素の子要素に title 要素が複数存在すると仮定する。path のキャスト記述を省略すると、その中で先頭にある title 要素の内容のみがオブジェクトへマップされる。一方、(String[]) というキャスト記述を行うと、そこにある全ての title 要素の内容が配列の形でオブジェクトへマップされる。2-2. の path は、配列ではなく java.util.LinkedList クラスをデータ型に指定した例である。これら java.util.List 型のインターフェースを実装するクラスを用いる時は、<>記号で要素に対応する型も同時にしていしなければならない。

```
3-1. /bibliography/(java.util.Vector<this>)book/
                       {title/text(), author/text()}
3-2. /bibliography/(this[])book/
                       {title/text(), author/text()}
```

3-1. の path は、book という名前の要素に対応するクラスを生成し、探索して発見された要素を java.util.Vector クラスを用いて複数得るための記述である。Bibliography クラスの book 要素に対応するフィールドは、java.util.Vector 型となる。型に this を指定すると、その要素に対応するクラスを生成しつつ、複数の要素を探索するという意味になる。3-2. の path は、java.util.Vector クラスでなく、配列を用いて複数の要素を得たい時の記述である。

クラスの生成を抑制する記述

path 上の要素に対して、対応する Java クラスの生成を抑制するための記述を丸カッコ記号 () として用意した。探索時に用いる path 上の要素に

対応するクラスを全て生成すると、利用者にとって無駄のあるデータ構造となる可能性がある。例を挙げて説明する。

```
(a) /bibliography/book[@isbn="xxx"]/
                   {title/text(), author/text()}
(b) /(bibliography)/book[@isbn="xxx"]/
                   {title/text(), author/text()}
```

a は特定の isbn 属性を持つ book 要素を探索したい時の path である、この path では bibliography と book の両方に対応するクラスが出力されるが、Bibliography クラスは子要素に Book クラスのオブジェクトを持つだけのクラスであるため、冗長なデータ構造となっている。

一方、丸カッコ記号 () を用いて書かれた b の path では、bibliography 要素に対応するクラスは生成されず、利用者は無駄のない構造でデータを得ることができる。

新たにクラスを生成する記述

path 上の要素に対し、要素に対応しないクラスを新たに生成するために、+記号を導入した。探索に用いた path によって生成されるデータ構造が利用者にとって不便な場合が考えられる。この記述を利用すれば利用者は得られるデータ構造を変更する事ができる。

```
/bibliography/(this[])book/{title/text(),
      +publish/{@isbn,
       publishing-company/text(),published-year/text()}
```

この例では、publishing-company 要素の内容と published-year 要素の内容と isbn 属性の値を 1 つにまとめて扱うための、Publish クラスの生成を行う。利用者がこれらの情報を 1 つにまとめて別なモジュールへ渡したりという場合に有効である。

第4章　実装

Xeal は大きく分けて二つの部分から成り立つ。1 つは XML Data Binding を行う際、必要なクラスを生成する Class Generator、もう 1 つは自動生成されたクラスが実行時に利用する、指定された path に一致する要素を XML 文書から探し出す Element Searcher である。

4.1　Class Generator

Class Generator が行う処理の流れを図 4.1 に示す。以降で Class Generator を構成するモジュールの、それぞれの役割と機能について説明する。

4.1.1　Path Parser

Path Parser は、XPath を拡張した文法に沿って書かれた path の記述を `String` 型で受け取り、それを Class Generator や Element Searcher が扱いやすいオブジェクトの形に変換するモジュールである。XPath の文法は W3C の定めた勧告 [13] に記述されており、拡張した文法については節 3.2.3 で述べている。

4.1.2　Class Generator

Class Generator は、Path Parser から受け取った path のオブジェクト表現を元に、Java のクラスファイルを生成する。クラスファイルの生成ではソースコード生成、バイトコード生成を選択する事が可能である。バイトコード生成には、Javassist [16] を利用している。

4.2　Element Searcher

Element Searcher が行う処理の流れを図 4.2 に示す。以降でそれぞれの役割と機能について説明する。

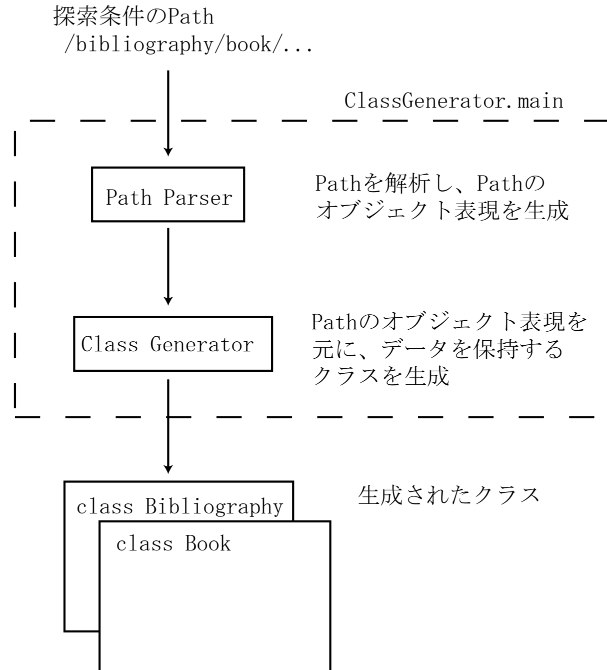

図 4.1: Class Generator 内で行われる処理の流れ

4.2.1　Forwarding Transrator

XML 文書を先頭から順に一度だけ読み込むだけで path に一致する要素を探索可能な状態にするため、これは Path Parser より得られた path を表すオブジェクトを、それが可能な形に変換するモジュールである。

XPath 記述の中には、XML 文書を後戻りして解析しなければいけないものが存在する。例を用いてその違いを説明する。

1. `/bibliography/book[author/text()="坂口安吾"]/following-sibling::book`
2. `/bibliography/book[author/text()="坂口安吾"]/preciding-sibling::book`

例えば 1 の path は、`author` 子要素の内容が`"坂口安吾"`であるような `book` 要素より、同じ階層にあってそれより後に出現する `book` 要素を指定している。一方、2 の path は、`author` 子要素の内容が`"坂口安吾"`であるような `book` 要素より、同じ階層にあってその前に出現する `book` 要素を指定している。1 の path に一致する要素の探索は、XML 文書を前向きに解析する事で行うことができるが、2 の path に一致する要素の探索には、解析中に文書を後戻りする必要が生じる。これを説明したのが図 4.3

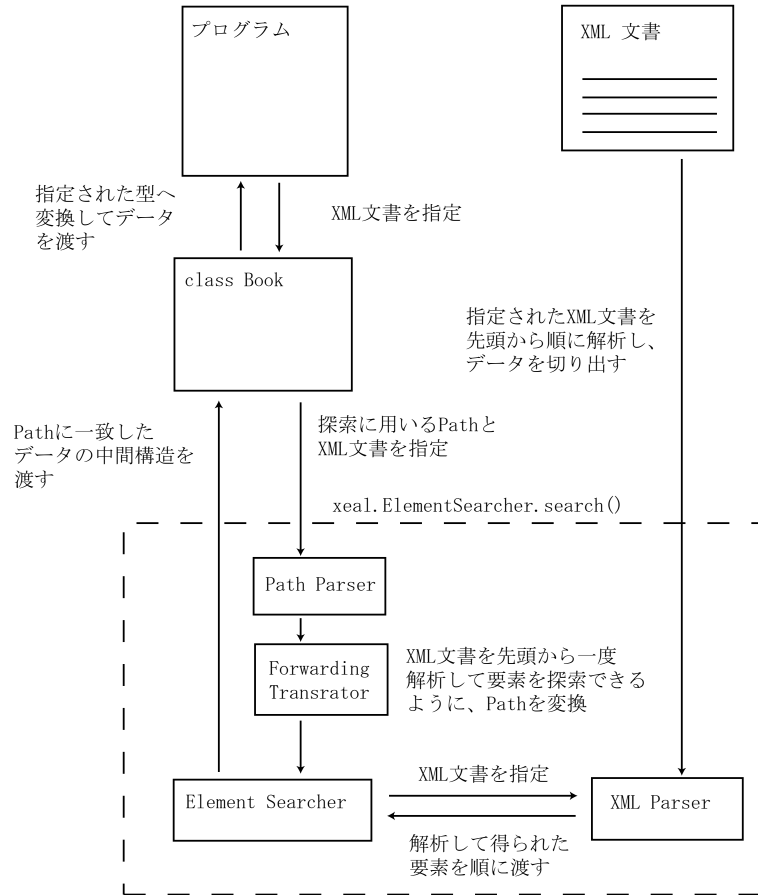

図 4.2: Element Searcher 内で行われる処理の流れ

である。

XPath の文法で後戻りの必要な axis は reverse axis と呼ばれる。

`preciding-sibling` や `ancestor`、`parent`、`preciding` が reverse axis にあたる。この axis を含んだ path は探索中に XML 文書を後戻りする必要が生じる。

Xeal では大容量の XML 文書の操作を可能にするため、XML 文書を一度だけ先頭から順に読み込んで、Path に一致する要素を探索する。そのため reverse axis を含む解析に後戻りが必要な Path を元に探索を行う事が出来ない。reverse axis を含む Path に一致する要素の探索を可能にするために、Xeal では reverse axis を含んだ Path の変換を行っており、その変換を行うのがこの Forwarding Transrator モジュールである。

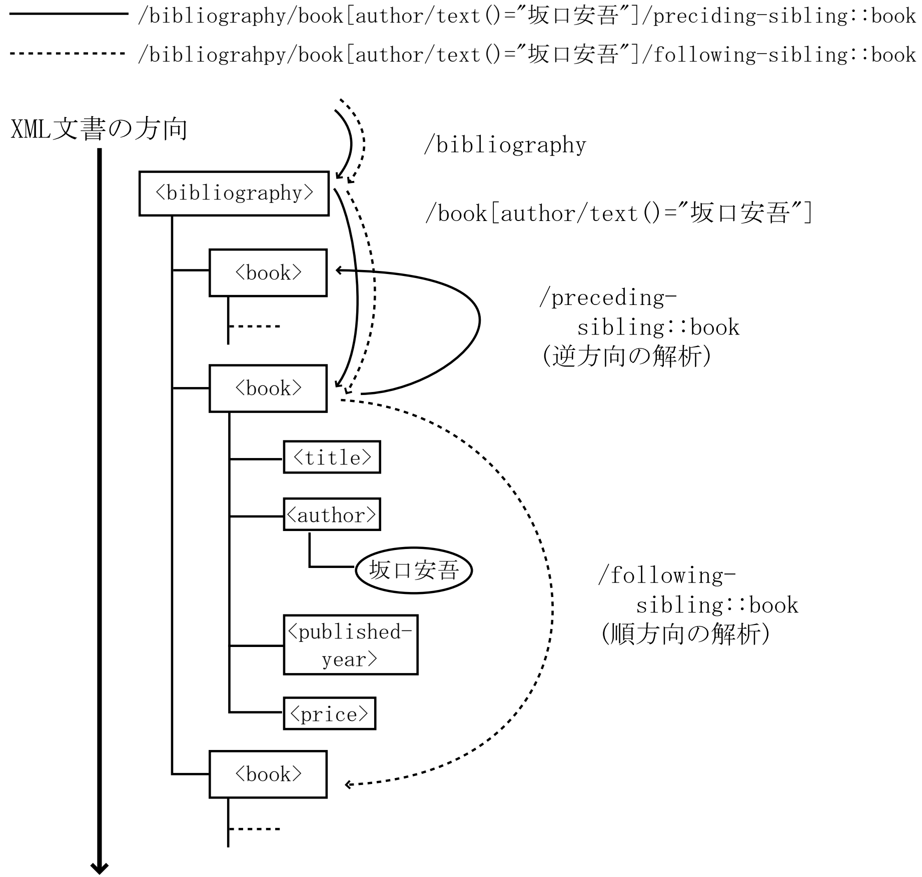

図 4.3: 探索に XML 文書の後戻りが必要な Path による解析の順序

4.2.2　Path の変換アルゴリズム

XPath で書かれた任意の path を、reverse axis を持たない等価な path に変換するアルゴリズムとして、Dan Olteanu らによって研究されている rare アルゴリズムがある [3]。path を rare アルゴリズムを用いて変換した例を挙げる。

```
変換前: /descendant::book/
            preceding::title[ancestor::bibliography]
変換後: /descendant-or-self::bibliography/
                   desendant::title[following::book]
```

変換前の path には、`preceding`、`ancestor` の 2 つの reverse axis が含まれている。変換後の path に一致する要素の集合は変換前の path に一致する要素の集合と等しいが、変換前の path に含まれていた 2 つの reverse

axis はそれぞれ除去されている。

しかし、変換前の path と rare アルゴリズムで変換された path では、探索に利用する経路が異なる。この例では、変換前の path は book 要素を辿って title 要素を探索するのに対し、変換後の path では bibliography 要素を辿って title 要素を探索している。Xeal では、この経路の途中にある要素に対応するクラスの生成を行うため、rare アルゴリズムを用いて変換した path では異なるクラスファイルを生成してしまう事になり、同一の探索を行う事が出来ない。

そこで Xeal の変換アルゴリズムは、Charles Barton らの提案する Xaos アルゴリズム [1] を参考にした。Xaos アルゴリズムは、reverse axis を含む XPath 記述に一致する要素を、XML 文書を後戻りせずに一度だけ読み込む事で探索するためのアルゴリズムである。

Xaos アルゴリズムではまず与えられた XPath 記述を X-Tree という木構造で表し、それを X-dag と呼ばれる構造に変換している。その X-dag 構造を元に、XML 文書を先頭から読み込み一致する要素を探索するという手法を取っている。X-Tree 構造は 1 つの step を木構造の node に見立て、step と step を結ぶ axis を木構造の Edge に見立てている。X-dag 構造とは、Xpath 記述を木構造で表した X-Tree 構造に対し、次のような変換を与える事で行うことができる。

- step から伸びている枝の axis が forward axis である場合は何も行わない
- step から伸びている枝の axis が reverse axis である場合、axis を対応する reverse axis に変更し、枝の親子関係を逆にする
- 変換の結果 step に入ってくる枝が存在しない場合は、Root からその step の方向を向いた枝を追加する。`ancestor-or-self` をその枝の axis とする。

このような変換により得られた X-dag には reverse axis が含まれていない。

また変換された X-dag 構造に一致する XML 文書の部分構造を探索した後で、得られた部分を元の path に一致する形へ変換する事も難しくない。

Xeal はこの Xaos アルゴリズムを利用して path 記述を変換し、得られた構造に一致する要素を XML 文書から探索している。ただし Xaos で変換が可能な path は、`child`、 `parent`、 `descendant`、 `ancestor` のうちいずれかの axis のみを持ち、述語を持たないものに限定している。Xeal では `following-sibling`、 `following` といった reverse axis に関しても上と同様の変換を行うことで、その他の axis にも対応している。図 4.4 は

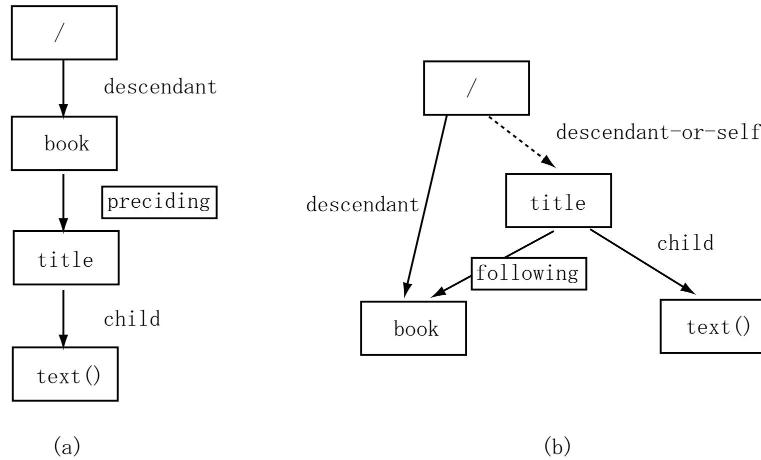

図 4.4: Xpath 式の X-Tree 表現とそれを変換して得られた X-Dag 構造

以下の Xpath 式の X-Tree 表現と、それを変換して得られた X-Dag 構造である。

このアルゴリズムによって変換された path の構造は dag (非循環有向グラフ) になっているため、path に一致する要素を XML 文書内から探索する Element Searcher では、path の dag 表現に一致する要素の探索が可能であるような実装を行っている。

4.2.3 Element Searcher

Element Searcher は、Forwarding Transrator により XML 文書の前向きの検索が可能な状態に変換された、path を表すオブジェクトを元に、XML Parser から受け取った XML 文書中の要素が path に一致するかどうかを確かめる。XML Parser からの要素の受け取りが全て完了した後、path に一致した要素はデータの中間構造として、生成されたデータを保持するためのクラスへ渡される。この中間構造では、DOM のように全ての要素が同じクラスのオブジェクトで表されている。この中間構造を受け取ったデータを保持するクラスは、渡すべきオブジェクトの構造へ変換して元のプログラムに返す。図 4.5 は、データの中間構造の様子を表している。

我々はこの Element Searcher を、SAX のパッケージに含まれている `org.xml.sax.ContentHandler` インターフェースを実装したクラスとして作成した。そのため Xeal の XML Parser を SAX のパッケージに含まれている `org.xml.sax.XMLReader` クラスと入れ替える事も可能だが、`XMLReader` クラスを使った場合は、データの変更を XML 文書へ反映させ

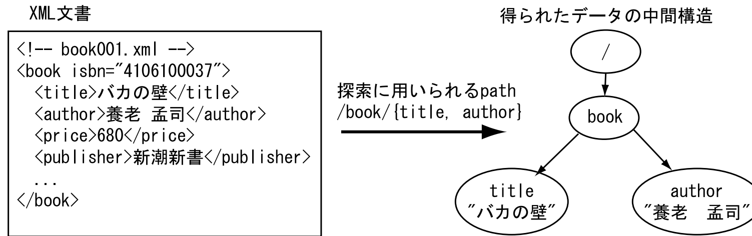

図 4.5: Element Searcher 内で利用される得られたデータの中間構造

る事ができないという不都合が生じる。なぜなら XML 文書への変更の反映のために、得られた要素の先頭文字の位置という情報を入手する必要があり、`XMLReader` からはその情報を利用することができないからである。そのため我々は XML Parser モジュールを独自に用意した。

ただし Element Searcher にこの `ContentHandler` インターフェースを実装させる事により、我々の XML Reader と SAX の `XMLReader` とで XML 文書を解析しイベントを全て起こし終わるまでの時間を比較する事ができる。また SAX の `XMLReader` の実装が変更され、得られた要素の先頭位置が分かるようになった場合は、我々のライブラリ内で SAX を実装に利用できるのではと考えている。

4.2.4　XML Parser

XML Parser は XML 文書を先頭から順に解析して、切り出された要素を元にイベントを発生し、Element Searcher の適切なメソッドを実行している。解析できない XML 文書の場合、`org.xml.sax.SAXException` 例外が返される。現在は XML 文書が整形式 (well-formed、XML の文法 [11] に沿っている XML 文書) であるかを判別することはできない。解析が不可能だった場合は `xeal.xml.XMLSyntaxException` が返される。

第5章　実験

5.1　他の API との実行時間の比較

XML 文書の一部を取り出し、それをプログラムへオブジェクトとして渡すことのできる API を用意し、要素を取り出してオブジェクトを作成するまでに必要な時間の比較を行った。用意した API は以下の 4 つである。() 内は実装に用いたライブラリの名前である。

- Xeal(本ライブラリ)
- SAX
  得られた要素を格納するためのクラスと SAX のハンドラを記述し、XML 文書から必要な要素を絞り込んでそのクラスのオブジェクトへ格納するという方法
- XSLT(Xalan) + DOM(crimson)
  XSLT を用いて XML 文書から必要な要素を絞りこみ小さな XML 文書を作成、その XML 文書から DOM ツリーを作るという手法
- DOM(Crimson) + XPath Engine(Xalan)
  DOM を用いて DOM ツリーを作り、そこから Xalan の XPath エンジンを用いて XPath 記述に一致する要素を絞りこむという手法

XSLT と SAX では、実験に使用した XPath と同じデータを絞り込むスタイルシート、ContentsHandler をそれぞれ用意し、それを用いて実験を行った。

用意した XPath 記述によって絞り込まれるデータの量は、元の文書の 200 分の 1 程度の大きさである。つまりこの path は文書中のごく一部のデータを得るような記述である。

5.1.1　実験環境

- マシン (Sun Blade 1000, UltraSPARC ⊠ 750MHx × 2, 1024MB, Solaris 8)
- Java version 1.4.0_01

- VM の最大使用可能メモリ 64MB
- 検索対象の XML 文書 XMark により作成、117KB から 118MB のものまで 11 個を用意
- 検索に用いた XPath 記述 /site//closed_auction/price/text()

5.1.2　実験結果

図1が実験の結果を表したグラフである。図2は図1の原点付近を拡大したものである。

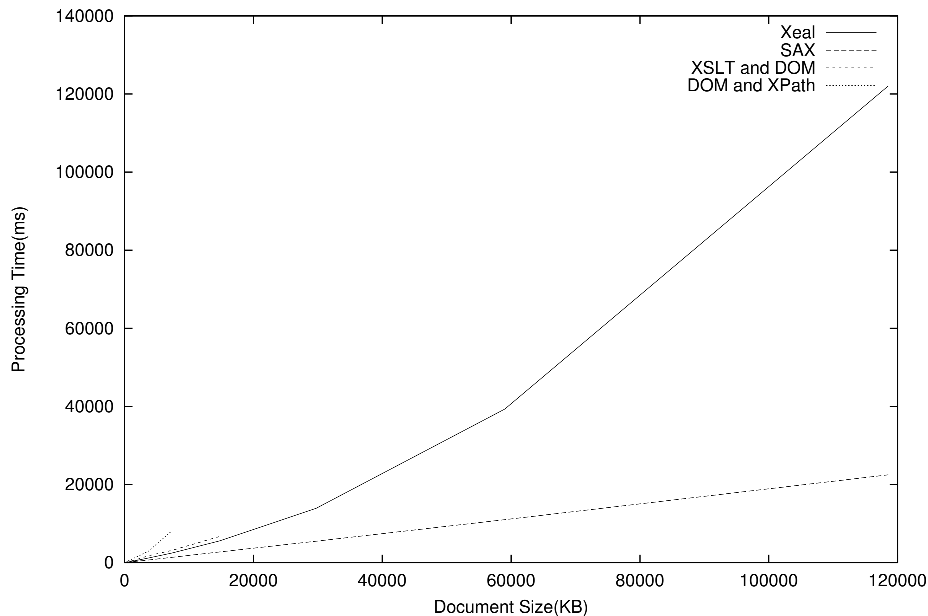

図 5.1: 各ライブラリで探索に必要な時間を比較したグラフ

5.1.3　考察

Xeal は XSLT と DOM の組み合わせ、DOM と XPath Engine の組み合わせより高速で、かつ大容量の XML 文書を操作の対象とする事が可能である事が分かった。

Xeal に比べ SAX の実行時間が非常に良い理由は 2 つ考えられる。1 つは、今回用意した SAX の `ContentsHandler` が、この実験に使った特定の path の探索のみを目的として書いたものだからである。Xeal でも、与

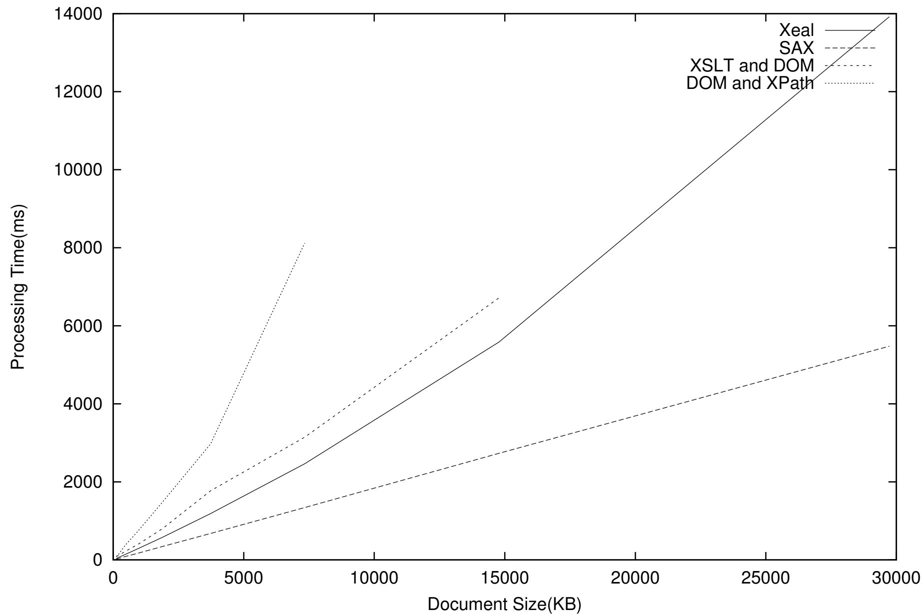

図 5.2: 図 5.1 の原点付近を拡大したグラフ

えられた Path に基づいてその Path に一致する要素を探索するハンドラを生成し、それを用いて検索を行えば、その部分の差は多少埋まると考えている。もう 1 つは、XML 文書を解析する XML Parser の性能の違いである。

図 5.3 は、ローカルマシンで Xeal の XML Parser と SAX の XML Parser の時間を比較したものである。両方とも、XML 文書を走査し、ある要素の名前を見つけたら変数をインクリメントさせるという簡単な命令を行っている。この結果より Xeal と SAX では XML Parser の性能に開きがあり、それは実行速度の差に影響を与えている要因の 1 つであると言える。

特定の条件に一致する要素のみを探索する `ContentsHandler` を記述して用いれば Xeal に比べ高速に探索を行う事が可能だが、SAX のハンドラ記述は XML 文書の構造を用いたデータの探索を行いにくく、探索の対象の構造が複雑になるほど記述が難しくなる。また探索対象となる XML 文書中の構造や、探索対象の XML 文書の構造が変わった場合、Xeal では path 記述を元にクラスファイルを生成する事で変更に対応できるが、SAX を用いた場合、データを格納するためのクラスとハンドラの変更をプログラマが記述しなければならない。これは保守性に欠けると言えるだろう。

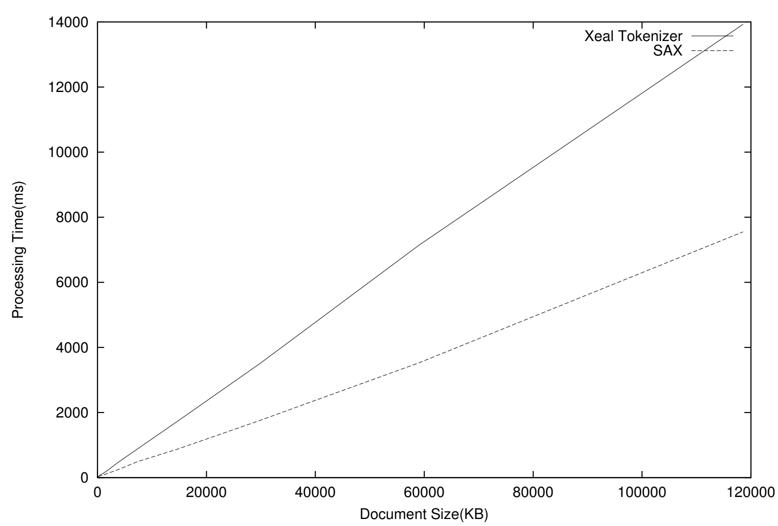

図 5.3: Xeal と SAX における、XML 文書の解析に要する時間の比較

第6章　まとめ

本研究では、XML 文書からのデータの参照と更新に長けた XML 文書編集ライブラリとして Xeal を提案した。Xeal は XML 文書内から要素を探索するための path を元に、探索して得られたデータを操作するためのクラスを生成する。その生成されたクラスを利用する事で、プログラマは扱いやすいオブジェクトとして抽出されたデータの参照や変更を行うことができる。

Xeal では path を記述する文法として XPath を利用した。また探索したデータを操作するクラスを自動生成する技術として XML-Java Data Binding を参考にした。実験の結果、大容量の XML 文書からデータを探索する技術の 1 つである XSLT より高速にデータを探索することができたが、SAX よりは低速である事が明らかになった。

6.1　今後の課題

- 探索時間の高速化
  Xeal の XML Parser は自ら実装したものを利用しており、実験の結果 SAX に比べて大幅に探索時間がかかることが確かめられた。XML 文書を先頭から順に読み込んで解析することのできるライブラリは SAX を初め多数存在するので、その中から適したものを Xeal の実装に利用する事により探索にかかる時間が向上すると考えられる。他に XPath の記述を、XML 文書を後戻りせずに一度解析するだけで要素の探索が可能な形へ変換するアルゴリズムに関しても改善の余地がある。より探索を行いやすい形へ変更する事で探索時間の高速化を行えると考えられる。

- 探索の条件を動的に与えることの実現
  探索して得られたデータを保持する Java のデータ構造は変えずに、探索の条件だけを変更したい場合が考えられる。探索の条件をプログラムの中から動的に与える事ができれば、そのつどクラスファイルを生成する必要はなく、利用者にとってより便利ではないかと考えられる。

- Java のデータ構造を指定する記述の柔軟性
  探索して得られたデータを保持する Java のデータ構造を、利用者がより自由に決められる事ができれば便利なのではないかと考えられる。現在では path の記述に依存した形になるので、より記述を柔軟に行えるような方法を探し出したい。

参考文献

[1] Barton, C., Charles, P., Goyal, D., Raghavachari, M., Funtoura, M. and Josifovski, V.: An algorithm for streaming xpath processing with forward and backward axes, *ICDE* (2003).

[2] Exolab Group: Castor, http://www.castor.org/.

[3] Olteanu, D., Meuss, H., Furche, T. and Bry, F.: XPath: Looking Forward, *Proc. of the EDBT Workshop on XML Data Management (XMLDM)*, LNCS, Vol. 2490, Springer, pp. 109–127 (2002).

[4] SAX Project: SAX (Simple API for XML), http://www.saxproject.org/.

[5] Sun Microsystems: Java Architecture for XML Binding (JAXB), http://java.sun.com/xml/jaxb/.

[6] The Apache XML Project: Crimson, http://xml.apache.org/crimson/index.html.

[7] The Apache XML Project: Xalan-Java, http://xml.apache.org/xalan-j/index.html.

[8] The Apache XML Project: Xerces2 Java Parser, http://xml.apache.org/xerces2-j/index.html.

[9] The Codehaus: jaxen, http://jaxen.org/.

[10] World Wide Web Consortium: Document Object Model (DOM), http://www.w3.org/DOM/.

[11] World Wide Web Consortium: Extensible Markup Language (XML), http://www.w3.org/XML/.

[12] World Wide Web Consortium: Mathematical Markup Language (MathML), http://www.w3.org/Math/.

[13] World Wide Web Consortium: XML Path Language (XPath), `http://www.w3.org/TR/xpath/`.

[14] World Wide Web Consortium: XML Schema, `http://www.w3.org/XML/XMLSchema`.

[15] World Wide Web Consortium: XSL Transformations (XSLT) Version 1.0, `http://www.w3.org/TR/xslt/`.

[16] 千葉滋: Javassist Home Page, `http://www.csg.is.titech.ac.jp/~chiba/javassist/index.html`.

[17] 浅海智晴: Relaxer, `http://www.asahi-net.or.jp/~dp8t-asm/java/tools/Relaxer/index_ja.html`.

付 録 A　Xealの自動生成したクラスの例

3.2.1 の利用例において、自動生成されたクラスのフィールドとメソッドを記す。

```
public class Bibliography {
   private String path;
   Book[] _book;
   long[] _book_possition_in_file;
   int _book_size1;

   public Book[] getBook() {
      ...
   }
   public void setBook(Book[] x) {
      ...
   }
   public static Bibliograhpy bind(InputStream in) {
      ...
   }
   public void marshal() {
      ...
   }
}

public class Book {

   String _title;
   String _author;
   int _price;
   long _title_possition_in_file;
   long _author_possition_in_file;
   long _price_possition_in_file;
```

```
   public void setTitle(String x) {
      ...
   }
   public void setAuthor(String x) {
      ...
   }
   public void setPrice(int x) {
      ...
   }
   public String getTitle() {
      ...
   }
   public String getAuthor() {
      ...
   }
   public int getPrice() {
      ...
   }
   public static Book bind(InputStream in) {
      ...
   }
   public void marshal() {
      ...
   }
}
```

付 録B 拡張したXPathの文法規則

XPathを新たに拡張したことにより、XPath 1.0の仕様に含まれるEBNF記法を用いて定められた文法規則も拡張される。拡張文法によって変更された文法とその内容をここで記す。なお文法の番号は、XPath 1.0の勧告内の文法の番号に対応している。

```
[4] Step          ::= SingleStep
                    | '{' SingleStep (',' SingleStep)* '}'

 SingleStep       ::= '(' JavaType ')' ContentOfStep
                    | '(' ContentOfStep ')'
                    | '+' ContentOfStep

 ContentOfStep    ::= AxisSpecifier NodeTest Predicate*
                    | AbbreviateStep

 JavaType         ::= JavaTypeName
                    | '[' ']'
                    | JavaTypeName '[' ']'
                    | JavaListName
                    | JavaListName '<' JavaTypeName '>'

 JavaTypeName     ::= NCName
 JavaListName     ::= NCName
```

またキャスト記述を導入するにあたり、XPath 1.0の仕様の"3.7 Lexical Struture"で定められている字句解析のルールの、最初の条件を次のように変更した。

- 変更前のルール

  「前にトークンが存在し、そのトークンが'@'、'::'、'('、'['、Operatorでないならば、*をMultiplyOperatorとして、NCNameをOperatorNameとして認識しなければならない。」

- 変更後のルール

  「前にトークンが存在し、そのトークンが'@'、'::'、'('、'['、Operator でないならば、* を MultiplyOperator として、NCName を OperatorName として認識しなければならない。ただし前に存在するトークン列が、'(' JavaType ')' という構造に一致するならば、NCName は NCName として認識しなければならない。」