PostgreSQLを用いた多機能なXMLデータベース環境の構築

油井 誠[†,☆] 森嶋 厚行[††]

我々はオープンソースのRDBMSであるPostgreSQLとそのXML拡張であるXMLPGSQLを利用し，多機能なXMLデータベース環境を構築した．本環境ではXMLをPostgreSQLに格納し，XMLとしてアクセス可能なインタフェース群を提供する．本環境が多機能であるとは，次の機能をすべて持つことを指している．(a) DOM準拠関数によるアクセス機能，(b) XPathによるアクセス機能，(c) 格納されたXMLの更新機能．機能 (a)，(c) はすでにXMLPGSQLが提供しているので，構築にあたっては特に機能 (b) の実装を行った．また，この機能の追加にともない，機能 (c) の変更も行った．本論文ではこれらについて説明する．本論文の貢献は，研究コミュニティが開発した成果などを，オープンソースRDBMSに適用した事例を示すことである．本環境はさらに次の特徴を持つ．(1) 現在まで別々に行われていた研究や開発の成果などを組み合わせ，多くの機能をそれ1つで提供するオープンソースの環境を提供する．(2) 研究プロジェクトではあまり重要視されてこなかった詳細機能の実装も行う．(3) PostgreSQL専用とし，構築にあたってはユーザ定義関数などの，必ずしもすべてのRDBMSがサポートしていない機能も利用する．

Development of a Multifunctional XML Database Environment Based on PostgreSQL

Makoto Yui[†,☆] and Atsuyuki Morishima[††]

We developed a multifunctional XML database environment using PostgreSQL, an open-source RDBMS and XMLPGSQL, an XML extension. The environment decomposes XML documents into fragments and uses PostgreSQL to store them in a set of relations. Users do not have to know that they are stored as relations. It provides a variety of means to access XML documents; (a) DOM functions to build and traverse XML documents, (b) XPath engine to extract information from documents, and (c) update functions to modify documents. Since XMLPGSQL provides functions (a) and (c), our focus was on development of function (b) and modifications of funcion (c). Our main contribution is to show a case where we applied the fruits of research and development activities to an opensource RDBMS. The features of the projects are as follows: (1) we combine separately-developped technologies to construct one integrated environment providing multi-functionality, (2) we implement details that tend to be ignored by research prototype systems, and (3) we do not hesitate to use PostgreSQL's features (such as user-defined functions) to construct the environment.

1. はじめに

XMLは計算機環境におけるデータおよび文書の表現形式の一標準として，先進ユーザや研究者による興味の対象から，より幅広い利用者による現実的な技術となりつつある．大量のXML文書を永続的に格納し，検索，更新などの操作を可能とするためのデータベースシステムは，研究用，商用ともにさかんに開発が行われている[24]．また，オープンソースコミュニティでもいくつかの開発が行われている[2),16)]．これらはデータベースシステムを中心にミドルウェアなどを組み合わせたアーキテクチャで実現されることも多いため，以下ではXMLデータベース環境と総称する．

我々のプロジェクトの目的は，現在まで行われた各種研究開発の成果の一部を組み合わせ，標準に可能な限り準拠した，多機能なオープンソースのXMLデータベース環境を構築することである．特に関係データベースPostgreSQL[12)]にXMLを格納・検索するための多機能な環境を構築することを目的としている．

近年のソフトウェア開発において，オープンソースソフトウェアが果たしてきた役割は大きい．データベース分野ではPostgreSQL，MySQL[11)]などが代表

† 芝浦工業大学工学部工業経営学科
Department of Industrial and Management, Shibaura Institute of Technology
†† 筑波大学知的コミュニティ基盤研究センター
RCKC, University of Tsukuba
☆ 現在，株式会社NEC情報システムズ
Presently with NEC Informatec Systems, Ltd.

的なものである．また，Free Software Foundation [6] や Apache Software Foundation [1] などが提供してきた各種ツールは特に多くの利用者に利用されている．これらのソフトウェアの目的は，多くの人に，便利なツールとして利用されることであり，提案手法の有効性を示すための研究プロトタイプシステムとは目的が異なる．我々の XML データベース環境は前者に分類される．本環境開発の目的は，コードを公開し，多機能なオープンソース XML データベースとして今後の各種研究開発に役立つことである．このプロジェクトは，IPA による未踏ソフトウェア創造事業「未踏ユース」のテーマとして採択されたが，そこにおける管理組織の代表は PostgreSQL の XML 拡張である XMLPGSQL の開発者である [16]．成果は PostgreSQL コミュニティに対しても公開される☆．現在まで，RDBMS の機能を積極的に利用して実装された多機能なオープンソース XML データベース環境は存在しなかった．

一方，研究コミュニティでは RDBMS を利用した XML データベース環境の研究は数多く行われてきたが，大部分のプロジェクトでは XML データベース環境の一部の機能に焦点を当てた開発が行われており，RDBMS を利用した多機能な XML データベース環境を目的としたものは存在しなかった．また研究プロジェクトで作成されたプロトタイプの多くでは，本格的に利用するためには必要であっても各提案手法とは直接関係のない問題は省略するか，もしくは標準とは異なる実装をしていることもあった．

ここで本 XML データベース環境が多機能であるとは，次の機能をすべて持つことを指している．(a) RDBMS に格納された XML に対する DOM 準拠関数によるアクセス機能，(b) XPath 式☆☆によるアクセス機能，(c) 格納された XML の更新機能．詳細は後述するが，我々は機能 (a)，(c) の実現に PostgreSQL の XML 拡張である XMLPGSQL を利用することとし，機能 (b) の実装を行うことによって，本環境を実現した．また，機能 (b) の実装にともない，XMLPGSQL が提供する機能 (c) の変更を行った．本論文ではこれらについて説明する．

本論文の貢献は，研究コミュニティが開発した成果を，RDBMS を用いた多機能なオープンソース XML データベース環境の構築のために適用した事例を示すことである．本環境は次の特徴を持つ．(1) 現在まで別々の問題に焦点を当てて行われていた研究や開発の成果などを組み合わせ，多くの機能をそれ 1 つで提供するオープンソースの環境を提供する．どの手法を組み合わせるかという問題は，多機能なデータベースを構築するという観点から検討が行われている．特に機能 (b) の実装および機能 (c) の変更にあたっては，研究コミュニティで得られた成果を本プロジェクトの主旨にてらし，適切と考えられる組合せを選択した．(2) 研究プロジェクトではあまり重要視されてこなかった詳細機能の実装も行う．(3) PostgreSQL 専用とし，構築にあたってはユーザ定義関数などの，必ずしもすべての RDBMS がサポートしていない機能も利用する．

本論文の構成は次のとおりである．まず，関連研究・ソフトウェアについて説明する．次に，本環境の構築にあたって採用した技術とその理由について説明する．これは本論文の重要な貢献の 1 つである．また，それに従って決定された本環境のアーキテクチャについて説明する．続いて，我々が実装した XPath 処理系と更新処理について説明する．最後に，本環境の簡単な評価を行う．

2. 関連研究・ソフトウェア

ここでは関連研究・ソフトウェアを，本環境の特徴の観点から分類する．

(a) RDBMS を用いた XML データベース環境に関する研究　RDBMS を利用した XML データベース環境の構築に関する研究はデータベースコミュニティでの近年における最もホットなテーマの 1 つである．RDBMS を利用する理由は主に次のようなものである．(1) RDBMS の持つ機能（インデックス機能，SQL によるアクセスなど）を積極的に利用したい．(2) 通常の RDB データとの統合利用が容易である．このアプローチに基づくシステムの例としては，XRel [25]，Edge [7]，XParent [9]，Monet [17]，Saxophone [23] などがある．このアプローチに関する研究のうち本プロジェクトに強く関連するものは，XML の各要素のラベル付けに関するものと XPath 処理に関するものである．まず，論文 19) では XML を RDB に格納するときの，各 XML 要素に対するラベル付けの問題について議論している．特に，XPath 処理と更新時のコストの観点から複数のラベル付け手法を比較している．論文 3) では，論文 19) とは異なるラベル付けの方法（範囲（Range））を用いた番号付けを説明している．論文 8) では，XPath 式を効率良く処理するために R-Tree を

☆ 現在，PostgreSQL 関連のオープンソースソフトウェアポータル GBorg で公開中である（http://gborg.postgresql.org/project/xpsql/）．

☆☆ 詳細は 8.1 節で説明するが，本環境でサポートする XPath 式とはロケーションパス [20] のことである．

用いる手法を提案している．論文 15) は格納された XML の更新を行う際に，レンジによるラベルの更新コストを減らすための番号付けの方法を提案している．一般に，これらの研究プロジェクトにおけるプロトタイプシステムでは提案手法の有効性を示すために必要な機能に絞った実装が行われている．たとえば，XRel や XParent のプロトタイプの現在の実装では XPath における child 軸と descendant 軸以外の処理，名前空間，PI（Processing Instruction）の処理などが省略されている．また，XRel プロトタイプの実装では，XPath 評価における文字列の扱いが本来あるべきものと一部異なる．これらの理由は，各研究における提案手法の本質とは直接関係しない部分だからである．本環境が処理可能な XPath 式は，以上の点を含むより広範囲な表現力をサポートする（表現力の詳細は 8.1 節で説明する）．

(b) オープンソースコミュニティで開発されているソフトウェア オープンソースの XML データベースで有名なプロジェクトとしては Xindice[2] がある．これは XML のネイティブデータベースであり，XPath による検索と XUpdate による更新をサポートする．XQEngine[22] は java で記述された XQuery 処理系である．eXist[5] は XPath と DOM をサポートするネイティブデータベースである．我々のプロジェクトは関連研究 (a) で説明したような，RDBMS の機能を積極的に用いた XML データベース環境の構築というアプローチであり，その点において以上のシステムとは異なる．eXist は RDBMS を XML 格納のためのストレージとしても利用可能であるが，あくまでネイティブデータベースとしての性能を重視しており，RDBMS との組合せを意図しては開発されていない[5),☆]．XMLPGSQL[16] は PostgreSQL のための XML 拡張であり，格納された XML 文書に対して DOM 準拠関数による操作が可能である．後述するように，我々の構築する環境は，PostgreSQL+XMLPGSQL という組合せをさらに拡張し XPath 処理系を追加したものである．データベースに XML を格納し DOM 操作を行うためのソフトウェアとしては他に DBDOM[4] がある．これは，RDBMS の種類を限定しない汎用のミドルウェアを目標に開発が進められている．

(c) 多機能な XML データベース環境 我々の知る限り，現在，DOM と XPath をともにサポートする多機能な XML データベース環境は Tamino[14]，eXist[5] などのネイティブデータベース環境のみである．RDBMS の機能を積極的に用いて XML 管理を行う，多機能 XML データベース環境は商用を含めて存在しない．

☆ ただし，PostgreSQL と組み合わせて利用することが可能であるため，8 章では本環境との XPath の表現力の違いについて評価している．

3. PostgreSQL と XMLPGSQL

PostgreSQL はカリフォルニア大学バークレー校の postgres プロジェクトから派生したオープンソースのオブジェクト関係データベースである．SQL92 エントリレベルの大部分および SQL99 の一部の機能をサポートしている．現在も有志によってサポート・開発が進められており，研究やビジネス用途に広く利用されている．PostgreSQL の特徴の 1 つに，SQL 問合せ中で利用可能なユーザ定義関数による機能拡張がある．

XMLPGSQL[16] は，ユーザ定義関数を用いた PostgreSQL の XML 拡張である．XMLPGSQL を利用すれば，PostgreSQL に XML 文書を格納し，その文書に対して DOM Level 2 Core のサブセットに準拠した関数を用いた操作ができる．

3.1 XMLPGSQL における XML 文書の管理

XMLPGSQL では XML 文書格納用の固定したスキーマを用意する．XML 文書は分解され，そのスキーマに格納される．**図 1** の XML 文書を例に説明する．**図 2** はこの XML 文書を木構造で表したものである．図中の Nx は各ノードに割り振った ID である．XMLPGSQL では，この文書を**図 3** の関係表群として格納する．関係表 xml_node の各タプルは各ノードの情報を格納している（pid と label 属性は XMLPGSQL では用いない．これらについては後述する）．kind 属性はノードの種類を表している．これは

```
<paper>
  <title>XML and Databases</title>
  <authors>
    <author aff="SIT">Makoto Yui</author>
    <author aff="SIT">
     Atsuyuki Morishima
    </author>
    <author>Taro Shibaura</author>
  </authors>
  <abstract/>
  <section>
    start section
    <subsection/>
    end section
  </section>
</paper>
```

図 1 XML 文書インスタンスの一例
Fig. 1 An XML document instance.

XQuery 1.0 and XPath data model [21]（以下 XPath データモデル）で規定されている 7 種類に対応している（**図 4**）．parent，child，next 属性は関係する各ノード間のリンク構造を表す（**図 5**）．関係表 xml_tag と xml_attribute は，タグおよび XML 要素の属性名の情報を格納する．本論文では，説明に関係のない関係表と属性を一部省略している．実際に利用されるすべての関係表の一覧を**図 6** に示す．

XMLPGSQL の格納形式の特徴は，次のとおりである．(1) XPath データモデルに準拠している．たとえば，XML の処理命令（processing instruction）ノー

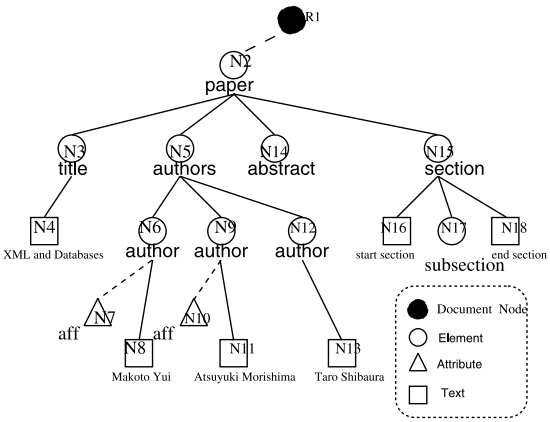


図 2 図 1 の XML 文書を表す木構造
Fig. 2 Tree representation of the XML document in Fig. 1.

種類	kind 属性での値
document	0
element	1
text	2
comment	3
attribute	4
namespace	5
processing instruction	6

図 4 ノードの種類一覧
Fig. 4 Kinds of nodes.

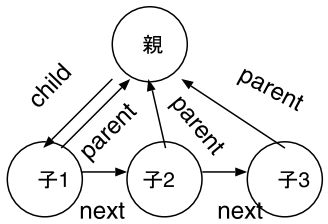


図 5 XMLPGSQL で管理されるリンク情報
Fig. 5 Link structure managed by XMLPGSQL.

関係表名	説明
xml_node	ノードを管理
xml_attribute	属性名を管理
xml_document	文書を管理
xml_namespace	名前空間を管理
xml_pi	処理命令を管理
xml_tag	タグ名を管理

図 6 XMLPGSQL で利用する関係表一覧
Fig. 6 Relations managed by XMLPGSQL.

関係表 xml_node

id	did	kind	tag	parent	child	next	value	(pid)	(label)
R1	D1	0			N2				1
N2	D1	1	E1	N1	N3			P1	1.1
N3	D1	1	E2	N2	N4	N5		P2	1.1.1
N4	D1	2		N3			XML and Databases	P3	1.1.1.1
N5	D1	1	E3	N2	N6	N14		P4	1.1.2
N6	D1	1	E4	N5	N8	N9		P5	1.1.2.1
N7	D1	4	A1	N6			SIT	P6	1.1.2.1.0
N8	D1	2		N6			Makoto Yui	P7	1.1.2.1.1
N9	D1	1	E4	N5	N11	N12		P5	1.1.2.2
N10	D1	4	A1	N9			SIT	P6	1.1.2.2.0
N11	D1	2		N9			Atsuyuki Morishima	P7	1.1.2.2.1
N12	D1	1	E4	N5	N13			P5	1.1.2.3
N13	D1	2		N12			Taro Shibaura	P7	1.1.2.3.1
N14	D1	1	E5	N2		N15		P8	1.1.3
N15	D1	1	E6	N2	N16			P9	1.1.4
N16	D1	2		N15		N17	start section	P10	1.1.4.1
N17	D1	1	E7	N15		N18		P11	1.1.4.2
N18	D1	2		N15			end section	P10	1.1.4.3

関係表 xml_tag

id	name
E1	paper
E2	title
E3	authors
E4	author
E5	abstract
E6	section
E7	subsection

関係表 xml_attribute

id	name
A1	aff

図 3 図 1 の XML 文書の関係表への格納
Fig. 3 Relations storing the XML document in Fig. 1.

ドに対するアクセスなどにも対応できる．これは通常の研究プロトタイプシステムなどでは省略されているところである．(2) DOM 操作を意識した構造になっている．たとえば兄弟間のリンクなどを明示的に保存している．

4. 本環境で採用したアプローチ

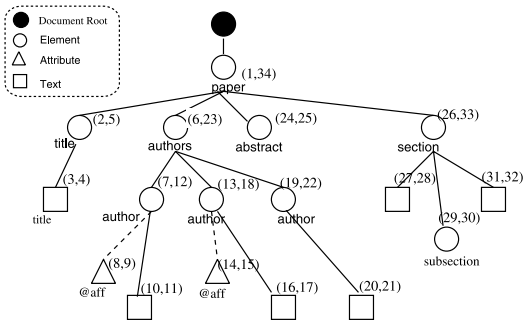
PostgreSQL を用いて多機能な XML データベース環境を実現するにあたり，どのようなアプローチを採用するかは重要な問題である．一般に各アプローチは異なる長短所を持っており，多くの機能を 1 つの環境で提供するときには，様々な点を考慮した選択を行う必要があるからである．本章では我々が採用したアプローチとその理由について述べる．

関係データベースを用いた XML データベース 本環境開発のそもそもの動機は，オープンソースの RDBMS である PostgreSQL に本格的な XML 格納・検索機能を追加したいということである．特に日本においては PostgreSQL は最もメジャーなオープンソース RDBMS であり，実務に利用されている例も多数ある[13]．データを PostgreSQL で管理している利用者が，同じシステムで XML 処理も行いたいという要求を持つことは自然であると考えられる．その最も大きな理由として考えられるものは，他のデータとの自然な統合利用である．たとえば SQL を用いて XML 文書と関係データの統合利用を行うことができれば，現在の RDBMS 利用の自然な拡張として XML 文書を扱える．

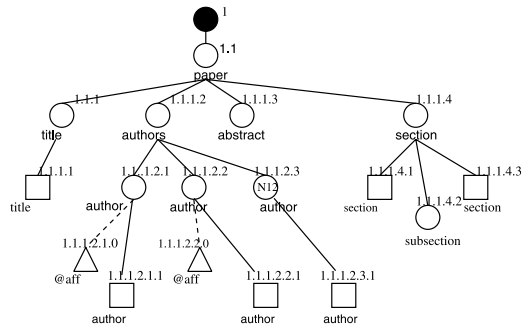
モデル写像アプローチ XML を関係データベースに格納する主要な方法としては，(a) CLOB（Character Large OBjects）などを利用して XML 文書を丸ごとアトミックな値として格納する方法と，(b) XML 文書を分解しデータベーススキーマに写像する方法がある．本環境では後者の分解する方法を採用した．理由は，(1) DOM 操作や XPath 式評価の際，SQL 処理系やインデックスの仕組みを有効利用したいこと，および，(2) XML 文書の部分更新を行いたいこと，である．方法 (b) はさらに，DTD をスキーマ構造に反映させる**構造写像アプローチ**と，固定したデータベーススキーマを用いる**モデル写像アプローチ**に大きく分けることができる[24]．本環境ではモデル写像アプローチを採用している．理由は，DTD が不明であるような整形式 XML 文書や，DTD が変更される頻度の高いような XML 文書の格納も行うためである．また，モデル写像アプローチでは DTD ごとにスキーマを作成する必要がないため，XML 文書を扱うためのハードルが低くなるという利点も考慮した．

エッジ指向とパス指向 関係データベースを用いたモデル写像アプローチとして現在までに提案された主要な方法に，エッジ指向とパス指向のものがある．Edge[7] に代表されるエッジ指向の手法では，XML インスタンスを木構造で表した後，各辺の情報に分解して関係表に保持する．各辺の情報は基本的に親子ノードの組 (n_1, n_2) で表される．したがって，XPath 式を評価するためには，一般に多くの結合演算が必要となる．図 3 で示すように XMLPGSQL はエッジ指向の手法の変種である．一方，XRel[25] に代表されるパス指向の手法では，XML インスタンスを木構造で表した後，各ノードに対するルートノードからのパス（たとえば図 2 のノード N6 に対しては `/paper/authors/author`）をそれぞれ文字列として保存する．XPath の評価は，この文字列に対するパターンマッチを利用して行われる．したがって，パスの各ステップごとに結合演算を行う必要がある純粋なエッジ指向に比べて結合演算数の削減が期待できる．これらのエッジ指向とパス指向の手法はそれぞれ相反するものではなく，これらを組み合わせた手法も存在する[9]．後述するように，我々の環境もこれらを組み合わせる．理由は，(1) エッジ指向の手法が，リンクをたどる DOM 操作との相性が良いこと，および (2) 特定の形式をした XPath 式では，結合演算を必要としないパス指向の手法が優れている[9]からである．この選択は，特に DOM 操作と XPath 演算の機能をともに実現するという目的から行われた．

Dewey 順序によるラベル付け 一般に XML の要素列では順序が重要（たとえば論文の著者の列）であるため，XML 文書を表す木構造は順序木となる．この順序木の各ノードに対して上手くラベル付けをすることにより，XPath 式の処理などを効率良くする手法が研究されている．このラベル付けには，範囲（Range）を用いるもの[8] や Dewey 順序を用いるもの[19] などがある．典型的には，範囲によるラベル付け（**図 7** (a)）では，XML 順序木における各ノードの前置順と後置順の順序の対をラベルとする．Dewey 順序によるラベル付け（図 7 (b)）では，XML 順序木の各ノードのラベルは，親のラベルの後ろに兄弟間の順序を連結したものである．Dewey 順序を用いるものは XPath 検索の性能とデータ更新の性能のバランスが良いことが知られている[19]．本環境では DOM 操作によるデータの更新をサポートする必要があるため，Dewey 順序によるラベル付けを採用した．範囲を用いたラベルに関しても，更新に強いラベル付け手法の研究が行われている[3),10),15)] が，これらでは一般にノード間の先



(a) 範囲（前置順，後置順）によるラベル付け



(b) Dewey 順序によるラベル付け

図 7 ラベル付け手法
Fig. 7 Labeling methods.

祖–子孫関係の計算のみが考慮されている．

以上の結果として，PostgreSQL を用いて多機能な XML データベースを構築するために，次章で説明するアーキテクチャを採用することに決定した．

5. アーキテクチャと利用法

5.1 アーキテクチャ

まず，本環境のアーキテクチャのポイントをまとめると次のようになる．

(1) PostgreSQL+XMLPGSQL という枠組みをベースに開発する．理由は，XMLPGSQL が (1) モデル写像アプローチ，エッジ指向のスキーマを採用しており，我々の目的に合致しているうえ，すでに DOM 準拠関数を実装していること，(2) XPath データモデルに従った実装が行われており，処理命令（Processing Instruction）やコメントなど，実際のシステムに必要な細かな情報を扱うための枠組みも提供していること，の 2 点である．本環境は，この枠組みに XPath 処理系などを追加することによって実現する．

(2) XML 文書格納のためのスキーマは，次の情報

pid	path
P1	#/paper
P2	#/paper#/title
P3	#/paper#/title#/
P4	#/paper#/authors
P5	#/paper#/authors#/author
P6	#/paper#/authors#/author#/@aff
P7	#/paper#/authors#/author#/
P8	#/paper#/abstract
P9	#/paper#/section
P10	#/paper#/section#/
P11	#/paper#/section#/subsection

図 8 追加する関係表 xml_path
Fig. 8 Additional relation xml_path.

を XMLPGSQL のスキーマに追加したものを採用する．

- Dewey 順序を用いたラベル情報
- パスに関する情報

具体的には次の 2 つの変更を行う．(a) 新たな関係表 xml_path（**図 8**）を追加する．これは XML 文書のパス情報を格納する．各パスは ID を持つ．この情報は，次で説明する拡張された関係表 xml_node から参照される．(b) 3 章で説明した関係表 xm_node（図 3）に，pid と label 属性を追加する．pid は，ルートノードからそのノードへのパスを表すパスの ID（xml_path への外部キー）であり，label は，各ノードに Dewey 順序を用いたラベルを割り振ったものである．たとえば，図 7 (b) のノード N12 へのルートからのパスは/paper/authors/author であり，Dewey 順序を用いたラベルは 1.1.1.2.3 である．

現在，Dewey 順序情報を格納している label 属性の型は，PostgreSQL でサポートされているユーザ定義型を用いて実装している．現実装では INTEGER（4 バイト）の配列として実現しているが，Dewey 順序情報の効率の良い実装は自明な問題ではなく他の方法も考えられる[19]．それらの手法の比較検討および適切な手法の採用は次期版で行う予定である．

(3) PostgreSQL のユーザ定義関数として，XPath 式を評価するための関数を実装する．XPath 式の評価は，(2) で追加したラベル情報とパスに関する情報を用いて行う．

(4) DOM 操作に関しては，基本的に XMLPGSQL の機能をそのまま利用する．ただし，更新操作については，本環境で新たに追加した情報も更新されるように仕組みを拡張する．

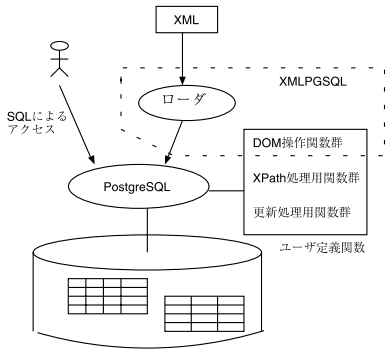


図 9 本環境のアーキテクチャ
Fig. 9 Architecture.

図 9 にアーキテクチャを示す．XML 文書はローダによって分解され，上記 (2) で説明した関係表群に格納される．XML に関する各処理機能は，PostgreSQL のユーザ定義関数を用いて実装されている．ユーザは SQL を通じてそれらの関数を利用することにより，XML の各種操作を行う．用意されているユーザ定義関数は次の 3 つに分類される．(1) DOM 操作関数群，(2) XPath 処理用関数群，(3) 更新処理用補助関数群．これらのうち，(1) は XMLPGSQL で用意されている．詳しくは後述するが，更新処理には (1) と (3) の関数が利用される．我々は (2)，(3) を実装した．次の節では (2) のグループの関数およびその利用法について説明する．

5.2 XPath 処理関数とその利用法

PostgreSQL のユーザ定義関数機能を用いて，SQL 問合せから呼び出し可能な次の 3 つの関数を実装した．
(1) xpath_eval（VARCHAR e）．文字列 e で与えられた XPath 問合せを評価し，ドキュメント ID とノード ID を持つ 2 項関係表を返す．付録 A.1 に本関数で評価可能な XPath 式を示す．基本的には，XPath 式は**ロケーションステップ**（以下 Step）を / で区切って並べたものである．本関数の特徴は，XPath [20] のすべての軸（axis）（**図 10**）を利用可能であるなど，表現力が大きいことである．**図 11** は格納されている関係表が図 3 と図 8（すなわち図 2 の XML）であるときの実行例である．
(2) xpath2sql（VARCHAR e）．文字列 e で与えられた XPath 問合せに対応する SQL 問合せを文字列として返す．たとえば，次の SQL 文を発行すると，`/paper//author[3]` を評価するための SQL 問合せを含む単項関係表を返す．

```
SELECT xpath2sql('/paper//author[3]')
```

実は，先の xpath_eval 関数が返すものは，この SQL 問合せを実行した結果である．実際にどのような SQL 問合せが作成されるかは次章で説明する．
(3) toString（INTEGER id）．ノード ID id に対応する XML の構成要素を文字列化する（すなわち，XML 要素を文字列で表現したものを返す）．これは，xpath_eval と組み合わせて利用する．

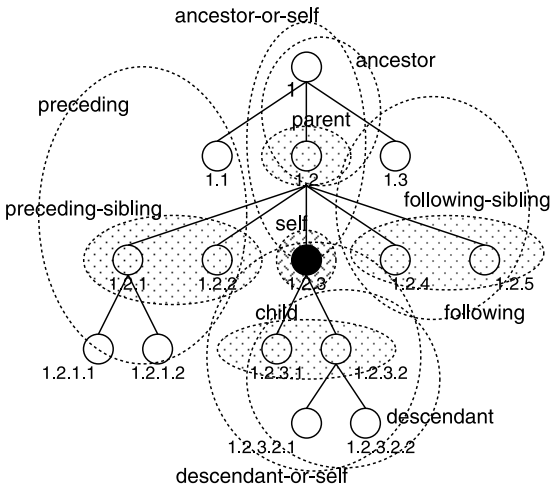


図 10 Dewey 順序ラベルを用いた軸の評価
Fig. 10 Axis evaluation using Dewey order labels.

```
SELECT * FROM xpath_eval('/paper//author[3]')
```

did	id
D1	N12

図 11 xpath_eval 関数の利用例とその結果
Fig. 11 Example for xpath_eval function and its results.

6. XPath 式から SQL 問合せへの変換

図 3 のリレーション群を利用した最もナイーブな XPath 式処理の方法は，リレーション xml_node の child 属性や next 属性をたどりながら木の探索を行い，与えられた XPath 式を満たすノードを見つけることである．しかしその処理を行うための SQL 問合せは xml_node の結合演算を多数含むものとなり，現実的ではない．そこで，我々は 4 章で述べたように，Dewey 順序によるラベルを用いた XPath 処理手法[19] をベースとした実装を行った．また，特定の形式を持つ XPath（部分）式の処理に関しては，パス情報を用いた手法[25] が少ない結合演算で実現可能であることが知られているため，この手法も一部応用している．以下では本実装による XPath 式から SQL 問合せへの変換について説明する．

6.1 Dewey 順序を用いた XPath 処理

Dewey 順序ラベルを用いた XPath 処理の基本的なアイデアは次のとおりである．すなわち，起点ノー

ド n と，ある軸（descendent など）が与えられたとき，n のラベルと他のノードのラベルを比較すれば，その軸に対応するノードの集合を求めることができる（図 10）．たとえば，n のラベルを l とした場合，n の descendent-or-self 軸の要素のラベルはその prefix が l であるものである．また，n の following-sibling 軸の要素のラベルは，末端の値が l のものより大きいことを除き，l と同じである．

このような条件を用いると，各ロケーションステップごとに xml_node の自己結合を順に行うことにより，XPath 式の実行を行うことができる．XML インスタンスの各辺ごとに結合演算を行う必要はない．

例 1：

```
/descendant-or-self::subsection/following::*
```

この式は問合せ SQL1 に変換される☆．

SQL1 例 1 の XPath 式から変換された SQL 問合せ

```
SELECT xn2.id
FROM xml_node xn1, xml_tag xt1, xml_node xn2
WHERE descendent-or-self(root, xn1.dewey)(a)
AND xn1.tag = xt1.id(b)
AND xt1.name = 'subsection'(b)
AND following(xn2.dewey, xn1.dewey)(c)
AND xn2.kind = 1(d)
```

SQL1 の where 節において，(a) は `/descendant-or-self`，(b) は `subsection`，(c) は `following`，(d) は `*` を評価するための条件である．まず，`/descendant-or-self` はルート要素自身もしくはその子孫の要素（図 10）であるので，(a) の `descendant-or-self` 関数が Dewey 順序ラベルを用いてその関係にあるものを選択する．次に，そのノードの中で，タグが `subsection` であるものだけが (b) で選択される．さらに，それらのノードに対して `following` の関係にあるものが，(c) で選択される．`following` 関数は Dewey 順序ラベルを用いてその関係にあるものを選択する．最後に，この XPath 式において `*` は要素である（すなわち名前空間や属性でない）ことという条件を表すので，(d) でノードの種類を要素に限定している．この手法の特徴は，すべての軸を処理できることである☆☆．

6.2 パス情報の利用

実際には，本関数は与えられた XPath 式が prefix として単純絶対正規表現（simple absolute regular expressions）[25] を持つとき，パス情報を利用してその部分の XPath 式を評価する SQL 問合せを作成する．単純絶対正規表現とは，XPath 式を省略記法で書いたとき[20] ☆☆☆，`/` で始まり，`/` もしくは `//` で区切られたものである．次に示す例 2 の XPath 式は，prefix として単純絶対正規表現 `/paper/authors/author` を持つ．

例 2：

```
/paper/authors/author/following::node()
```

この式は SQL2 に変換される．

SQL2 例 2 の XPath 式から変換された SQL 問合せ

```
SELECT xn2.id
FROM xml_path xp1, xml_node xn1, xml_node xn2
WHERE xp1.path = '#/paper#/authors#/author'(a)
AND xn1.pid = xp1.pid(a)
AND following(xn2.dewey, xn1.dewey)(b)
```

SQL2 の where 節において，(a) は `/paper/authors/author`，(b) は `following` を評価するための条件である（`node()` は「任意の種類のノード」の意であるので特別な条件を必要としない）．(a) によって，`/paper/authors/author` のパスで到達可能なノードが選択される．それらのノードに対して `following` の関係にあるものが，(b) で選択される．この例から分かるように，単純絶対正規表現である prefix の処理部分に関しては，単純にラベルを利用した場合に比べて，パス情報を利用したほうが結合演算の回数が少なくなる可能性がある．本関数は，単純絶対正規表現である prefix の処理部分は，以上のようなパス情報を利用した結合演算を行う．このような，パス情報を用いた処理を行う SQL の生成は XRel[25] で提案された手法である．ただし，任意の軸（following-siblings など）を実行する部分は，ラベルを用いた処理が行われる．

7. XML 文書の更新処理

本環境では XMLPGSQL の DOM 準拠関数（appendChild や insertBefore）を用いて，XML 文書の更新を行う．本環境では XPath 処理のために追加した情報があり，XMLPGSQL で実装されている更新操作だけでは不十分なので，更新処理の追加を行った．ここでは格納された XML 文書に新たなノード

☆ 簡単化のため，ドキュメント ID は省略する．

☆☆ 高速化のため，実際には parent，child 軸に関しては `parent` 属性を用いた等結合が用いられる．

☆☆☆ たとえば，'//' は'`/descendant-or-self::node()/`' の略である．

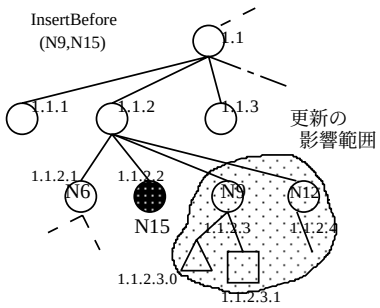


図 12 ラベルの更新処理
Fig. 12 Label updates.

が追加されたときの処理について説明する．例として**図 12** の XML 文書の N6 と N9 の間に N15 が追加されたとする．このとき，新たなノードに対応するタプルを関係表 xml_node に追加する以外に，次の 3 つの処理を行う必要がある．

リンク情報の更新 関係表 xml_node の各リンク属性（child，parent，next）の更新を行い，適切に張り替える必要がある．

ラベルの張替え xml_node の label 属性を変更する必要がある．Dewey 順序によるラベル付けでは，更新の影響範囲は限定されている[19]（図 12）．

パス情報の更新 新たなパスが現れた場合には，関係表 xml_path を更新する必要がある．ただし，「新しい種類のパスかどうか」を調べるためには xml_path を検索しなくてはならずコストがかかるため，次のような工夫をしている．すなわち，更新時にはそのチェックは行わず，追加されたノードのパス情報は無条件に xml_path に追加する．重複しているパス情報は，定期的に動作するガベージコレクタによって除去される．

以上のうち，リンク情報の更新処理は，XMLPGSQL の DOM 準拠関数で直接行われる．ラベルの張替えとパス情報の更新は，トリガを利用して xml_node の変更時に更新処理関数群（図 9）を呼び出すことによって行う．

8. 評　　価

ここでは，本環境でサポートする XPath 式の表現力，XPath 処理の性能，XML 部分更新処理の性能について評価を行う．我々は XRel[25] を，XPath 式の表現力および XPath 処理の性能に関する比較対象として選んだ．その理由は，XRel が RDBMS の機能を積極的に用いた XML データベース環境であり，かつ PostgreSQL と組み合わせて利用可能だからである．

	可	不可	可能な割合
本環境	27	4	87%
XRel	13	18	42%
eXist	17	14	55%

図 13 表現力の評価
Fig. 13 Expressive power.

XRel はパス指向の XPath 処理系研究プロトタイプシステムとして実装されており DOM 操作や XML の部分更新などは考慮されていない．本環境は RDBMS を用いた XPath 処理の実装に関して XRel の手法を参考にしている．したがって，本環境の性能は XRel と類似した傾向を示すと考えられる．また，ネイティブデータベースとして開発されている eXist☆についても PostgreSQL との組合せで利用可能であることから表現力に関して本環境との比較を行った．

8.1 表　現　力

本環境において実行可能な XPath 式は付録 A.1 に説明したとおりであるが，この表現力を説明するために簡単な調査を行った．**図 13** は，XPath の仕様書[20] の 2 章「Location Paths」に現れる 31 の XPath 式の例のうち，本環境，XRel，eXist において実行可能な XPath 式の個数を数えたものである．本環境で不可能な 4 つのうち，2 つは `last()` 関数を利用したものである．残りの 2 つは，Predicate の中で論理和演算を行うものである．本環境で可能な数が XRel と eXist に比べ多い理由は，本環境では XPath のすべての軸に対応しているからである．また，NodeTest に関しては本環境ではほぼすべての種類，eXist ではいくつかの種類に対応している．XRel の現在の実装では文字列の解釈が一部 XPath データモデルと一致しない点があるが，XRel に関しては「原理的にはできる」はずの点についてはできるものとして評価している．ただし，本プロジェクトの観点からいえば，「実装していないが原理的にはできる」ことではなく「実際にできる」ことが重要である．

8.2 XPath 処理の性能

この実験の目的は本環境の XPath 処理系が合理的な性能を持つことを示すことである．ここでいう合理的な性能とは，本環境の開発にあたって参考にした手法のシステムと大幅に異ならず，かつ性能の差が説明できるという意味である．これは，本環境の実装が，参考にしたシステムと比較して特に非効率でないことを主張する根拠として示すものである．本実験は，RDBMS を用いた XML データベース環境実現のため

☆ 現バージョンは 0.9 である．

表 1 性能評価のための問合せ
Table 1 Queries for performance evaluation.

	XPath 式	特徴
Q1	/site/open_auctions/open_auction	単純な経路式
Q2	/site/open_auctions/open_auction/bidder[1]/increase	position() 関数
Q3	/site/open_auctions/open_auction/[bidder/personref/@person=”person32”]/reserve	文字列の比較
Q4	/site/closed_auctions/closed_auction[price/text()>=40]/price	数値の比較
Q5	/site/regions//item	// が 1 つ
Q6	//open_auctions//description	// が複数
Q7	//closed_auction/following-sibling::*	/ と // 以外の軸
Q8	//from/parent::node()	ノードテスト

表 2 XPath 処理の実験結果 (ms)
Table 2 Experimental results of XPath processing (ms).

	0.01		0.03	0.05
	本環境	XRel		
Q1	18	65	29	33
Q2	164	368	314	564
Q3	43	145	55	95
Q4	28	—	987	2276
Q5	33	66	55	63
Q6	38	73	54	64
Q7	298	—	2665	5554
Q8	65	—	85	94

の各手法の優劣性の議論に焦点を当てたものではない．これに関する議論は本論文のスコープから外れるため，各研究論文[9),19),25)]をご参照いただきたい．本実験では XMark [18)] データベースを作成し，様々な特徴を持つ XPath 式（**表 1**）の処理時間を計測した．Q1〜Q6 は XMark ベンチマークで利用される XQuery 問合せから各特徴を持つ XPath 式を抜き出したものである．Q7，Q8 は適切なものが XMark に存在しなかったため我々が作成した．これらの問合せのうち本環境および XRel 双方が処理可能な問合せに関しては，本環境は XRel に類似の SQL 問合せを生成する．

実験結果を**表 2** に示す．実験環境は CPU が Mobile Pentium3 866 MHz，メモリが 384 MB，OS は RedHat Linux（カーネル 2.4.20）である．まずデータベースのスケールファクタを 0.01 とし，本環境および XRel について計測を行った．各値は 3 回実行した平均である．

次のようなことが分かる．まず，本環境と XRel でともに実行可能な XPath 式に関しては，どちらのシステムも似た傾向の結果を示している．すなわち，異なる XPath 式間の相対的な実行時間の大小関係がほぼ同様である．これは，6 章で示したように，本環境はパス情報を利用可能な部分では，XRel と同様の SQL 問合せを作成するからである．ただし，XML を格納する関係表の構造がまったく同じではないため，それ

表 3 部分更新処理実験結果 (ms)
Table 3 Experimental results of partial updates (ms).

影響範囲のノード数	計測時間 (s)	1 ノード 1 回あたり (ms)
0	7.5	250
15	9.4	310
45	9.5	316
150	10.8	360

が結果に影響している．この実験では全体的に本環境のほうがやや速い傾向があるが，その一因は次の理由による．すなわち，本環境では，親子関係を調べる必要があるときには parent 属性を用いた等結合を利用しているが，XRel ではつねに範囲ラベルを用いた θ 結合を利用し，やや遅くなる場合があるからである．本環境に関しては，スケールファクタ 0.03，0.05 の場合についての計測結果についても示す．Q4，Q7 は 0.01 の場合にのみ極端に速度が速いが，これは実験環境のメモリサイズとの関連が推測される（実行プランは 0.03 および 0.05 の場合と同じであった）．その点以外においては，Q4，Q7 を含め特筆すべき性能劣化は認められない．

8.3 XML 部分更新処理の性能

本環境における XML 部分更新処理の性能は更新の影響範囲のサイズに影響を受ける（図 12）．本実験では更新の影響範囲のノード数を変えて処理時間の計測を行った．データベースのスケールファクタは 0.01 である．結果を**表 3** に示す．表中の計測時間は，単一ノードの挿入時間ではなく，3 ノードから構成されるサブツリーの挿入を 10 回連続して行った場合の合計処理時間である．本実験では影響範囲のノード数増加にともなう極端な性能劣化は認められなかった．

9. おわりに

本論文では，PostgreSQL を用いた多機能な XML データベース環境の構築について説明した．本環境は XML 文書を分解して関係表に格納し，DOM 操作，XPath 式評価，文書の部分更新をすべてサポートする

多機能な環境である．通常の研究プロトタイプシステムと異なり，処理命令・コメント・テキストノードの扱いなど，細かな点も可能な限り省略することなく標準に従って実装している．本環境の実現にあたっては，現在までの各種研究開発の成果を多機能な環境を実現するという観点から再検討し，必要な技術を選択し，組み合わせた．結果として，現時点では他に存在しない，RDBMS 用いた多機能なオープンソースの XML データベース環境を実現した．また，実装した XPath 処理系がある程度大きな表現力と合理的な性能を持つことが分かった．本環境構築の目的は，コードを公開し，多機能なオープンソース XML データベースとして今後の各種研究開発に役立つことである．今後の課題としては，XPath 式の表現力のさらなる拡大や，インデックスの工夫などを含めたさらなるチューニングなどがあげられる．今回は XMLPGSQL にできるだけ手を入れない方針で環境構築を行ったが，開発の段階で XMLPGSQL にも改良の余地が多くあることを発見した．それも含めた全体的な処理性能のチューニングも今後の課題である．

謝辞 XMLPGSQL に関する疑問について快くお答えいただき，未踏ユースにおける本テーマの管理組織代表としても暖かく見守っていただきました小松誠氏に感謝いたします．また，XRel プロトタイプシステムのコードをご提供いただき疑問にも快くお答えいただきました名古屋大学の吉川正俊教授と奈良先端科学技術大学院大学の天笠俊之助手に感謝いたします．本研究の一部は情報処理振興事業協会（IPA）の平成 14 年度未踏ソフトウェア創造事業「未踏ユース」，文部科学省科学研究費補助金若手研究（B）（課題番号 15700108）の支援による．

参 考 文 献

1) The Apache Software Foundation. http://www.apache.org/
2) Apache Xindice Home Page. http://xml.apache.org/xindice/
3) Cohen, E., Kaplan, H. and Milo, T.: Labeling Dynamic XML Trees, *Proc. PODS 2002*, pp.271–281 (2002).
4) DBDOM Home Page. http://dbdom.sourceforge.net/
5) eXist Home Page. http://exist.sourceforge.net/
6) GNU's Not Unix !—the GNU Project and the Free Software Foundation. http://www.gnu.org/
7) Florescu, D. and Kossmann, D.: A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Technical report RR-3680, INRIA (1999).
8) Grust, T.: Accelerating XPath location steps, *Proc. SIGMOD 2002*, pp.109–120 (2002).
9) Jiang, H., Lu, H., Wang, W. and Yu, J.: Path Materialization Revisited: An Efficient Storage Model for XML Data, *Proc. Australasian Database Conference 2002* (2002).
10) Li, Q. and Moon, B.: Indexing and Querying XML Data for Regular Path Expressions, *Proc. VLDB 2001*, pp.361–370 (2001).
11) MySQL Home Page. http://www.mysql.com/
12) PostgreSQL Home Page. http://www.postgresql.org/
13) 日本 PostgreSQL ユーザ会：PostgreSQL 使用サイトリンク集．http://www.postgresql.jp/projects/links/ippan.html
14) Software AG Home Page. http://www.softwareag.com/tamino/
15) 江田毅晴, 天笠俊之, 吉川正俊, 植村俊亮：XML 木のための更新に強い節点ラベル付け手法，日本データベース学会 Letters，Vol.1, No.1, pp.35–38 (2002).
16) 小松 誠：進化する XMLPGSQL，Software Design 2002 年 4 月号，pp.19–26 (2002).
17) Schmidt, A., Kersten, M., Windhouwer, M. and Waas, F.: Efficient Relational Storage and Retrieval of XML Documents, *Proc. WebDB* (*Informal Proceedings*), pp.47–52 (2000).
18) Schmidt, A., Waas, F., Kersten, M., Florescu, D., Carey, M., Manolescu, I. and Busse, R.: Why And How To Benchmark XML Databases, *SIGMOD Record*, Vol.30, No.3. pp.27–32 (2001)
19) Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E. and Zhang, C.: Storing and Querying Ordered XML using a Relational Database System, *Proc. SIGMOD 2002*, pp.204–215 (2002).
20) W3C: XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath
21) W3C: XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/query-datamodel/
22) XQEngine Home Page. http://www.fatdog.com/
23) 横山昌平, 太田 学, 片山 薫, 石川 博：XML パーサを考慮した応用向き関係データベース構成法，第 13 回データ工学ワークショップ（DEWS2002）(2002).
24) 吉川正俊：データベースの観点から見た XML の研究，2002 年情報学シンポジウム講演論文集，pp.25–31 (2002).

25) Yoshikawa, M., Amagasa, T., Shimura, T. and Uemura, S.: XRel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases, *ACM TOIT*, Vol.1, No.1, pp.110–141 (2001).

付　　録

A.1 本環境でサポートする XPath 式

- 構文：

```
LocationPath ::= '/' RelativeLocationPath?
RelativeLocationPath ::= Step { '/' Step }
Step ::= AxisSpecifier NodeTest Predicate*
AxisSpecifier    ::=    AxisName '::'
AxisName    ::=    'ancestor'
   | 'ancestor-or-self'
   | 'attribute'
   | 'child'
   | 'descendant'
   | 'descendant-or-self'
   | 'following'
   | 'following-sibling'
   | 'namespace'
   | 'parent'
   | 'preceding'
   | 'preceding-sibling'
   | 'self'
```

- 現時点で，NodeTest は，名前空間接頭辞以外がすべて可能である（次の版ではすべて実装予定）．
- Predicate は，”position()=定数” および “相対 XPath 式 [θ 定数]” が可能（[...] は省略可）．
- 各種省略記法（“//” や，position() の代わりに”[定数]” など）が可能．

(平成 15 年 3 月 25 日受付)
(平成 15 年 7 月 2 日採録)

（担当編集委員　原 隆浩）

油井　誠

2003 年芝浦工業大学工学部工業経営学科卒業．現在，（株）NEC 情報システムズ勤務．XML とデータベース等に興味を持つ．

森嶋　厚行（正会員）

1993 年筑波大学第三学群情報学類卒業．1998 年同大学大学院工学研究科修了．博士（工学）．1998 年〜2001 年日本学術振興会特別研究員．1999 年〜2000 年 AT&T Labs-Research 客員研究員．2001 年芝浦工業大学工学部情報工学科講師．現在，筑波大学知的コミュニティ基盤研究センター助教授（図書館情報学系所属）．情報統合，XML/WWW データベース等に興味を持つ．ACM，IEEE-CS，電子情報通信学会，日本データベース学会各会員．