関係データベースを利用した XML 版管理手法

横山 昌平† 太田 学† 片山 薫† 石川 博†

†東京都立大学大学院工学研究科 〒192-0397 東京都八王子市南大沢１－１

E-mail: †{shohei,ohta,katayama,ishikawa}@hikendbs.eei.metro-u.ac.jp

あらまし 近年, XML はデータ交換の共通フォーマットのみならず, データベースとしての利用機会が増えている．更新のあるデータを管理する際，重要となるのがデータの版管理である．XML の版管理には RCS の様な行指向の版管理ツールも利用できるが，XML の文法を理解しない為 XML のタグ構造に沿った差分データを得ることが出来ず，効率が悪い．また，XML を関係データベースに格納する研究は盛んであるが，関係データベースに格納した XML の版管理については殆ど議論されていない．本稿では XML の構造に基づく差分データを用いた XML 版管理手法について述べる．提案方式の特徴は一つのバージョンから異なる複数のバージョンが作れ，また任意のバージョンをほぼ一定のコストで検索可能な事である．

キーワード XML，関係データベース，版管理，バージョン，ブランチ，SAX パーサ

Versioning XML Data using Relational Database

Shohei YOKOYAMA† Manabu OHTA† Kaoru KATAYAMA† and Hiroshi ISHIKAWA†

†Graduate School of Eng., Tokyo Metropolitan University, 1-1 Minami-Osawa, Hachioji-shi Tokyo, 192-0397 Japan

E-mail: †{shohei,ohta,katayama,ishikawa}@hikendbs.eei.metro-u.ac.jp

Abstract Recently, XML documents are not only used for data-exchange, but also used as formats for databases. When XML data with update is considered, management of the versions is very important. In this case, however, traditional line-based document versioning systems, e.g. RCS, are inefficient, because they don't preserve the logical structure of the XML data. Furthermore, much research has been carried out on storage using relational databases; nevertheless version management in such an environment hasn’t been researched extensively. In this paper, we propose a new method of versioning for XML data stored using relational database. The main features are that plural versions are created from the previous version of the XML data, and the cost for retrieval of any version is almost equal.

Keyword XML, Relational Database, Versioning, Versions, Branch, SAX Parser

1. はじめに

XML 文書はとりわけ企業間のデータ交換に用いられていたが，最近はデータベースと連携して利用される機会が増えてきている．それは XML で扱うデータ量が増大すると共に，データを効率よく管理することが急務となっている為である. XML データの管理をする為に関係データベースを用いるシステムが複数提案されているが, 我々は XML データの格納だけでなく，更新情報を管理することが重要と考えた.

版管理の一番原始的な方法は，あるリソースに関する全てのバージョンを完全に保存する事である．この方法の問題点は，ストレージコストが版の数に比例して増加することである．たとえば，あるバージョンとその次バージョンの差異がリソース全体の 1%であったとしても, この二つのバージョンを格納する為には，そのリソースの 2 倍の容量が必要となる．そこで，RCS[1]や SCCS[2]のような，バージョン間の差分データのみを格納して，版管理を行う方法が提案された．この二つのツールは主にテキストドキュメントやプログラムソースコードの版管理に利用されている.

XML 文書もテキストファイルであるので, これら二つのツールを版管理に利用することが可能である．しかし, RCS, SCCS 共に行を単位とした差分を基に版管理を行う為，得られる差分データは XML のタグ構造が考慮されていない. XML をデータベースとして利用する場合, XML の構造は検索にとって非常に重要である．提案システムでは XML の構造に即した差分データを利用して版管理を行っている.

また分散環境でデータの共有がされているインターネット・イントラネット環境での版管理はブランチ機能が重要であると考える．ブランチ機能とはある一つのバージョンから複数の異なるバージョンを作り出す事で，複数の開発者が平行して作業を行う際に便利である．提案方式はこのブランチ機能を有する版管理

手法である．

提案方式は本研究室で開発をしている XML データベース Saxophone[3]と連携して動作する．Saxophone は XML のスキーマに依存しない XML 文書の関係データベース格納手法であり，SAX を用いてアクセスすることが可能である．

本稿の構成は以下の通りである．続く 2 節では提案方式と関連研究について述べる．3 節では Saxophone の説明を行う．4 節ではブランチ機能を持つ版識別子について説明し，5 節では版識別子を利用して版管理を行う手法を提案する．最後に 6 節で本稿をまとめる．

2. 提案方式とその関連研究

2.1. 提案方式の概要

提案方式は関係データベースを用いた XML の版管理手法である．特徴はブランチを持てることである．ブランチのあるバージョン系列は木とみなす事が出来る．本研究では木の接点すなわち各バージョンに効率的な識別子“版識別子”を定義している．版識別子はバージョン番号とバージョン識別子で構成されている．バージョン番号は木の深さを示し，バージョン識別子はブランチの分岐点を一意に表す事が出来る

この版識別子は，関係データベースの問い合わせ機能を用いて任意のバージョンの検索，あるバージョンの祖先及び子孫の検索を行う事が出来る．

また，提案方式はバージョン間の差分データのみを格納する方式を取っている．この手法では任意のバージョンを取り出す際，それまでの差分データを集約する必要がある．提案研究はこのバージョン検索にかかるコストにも留意している．詳しくは後述するが，バージョン検索にかかるコストは，取り出すバージョンに関わらずほぼ一定である．

つまり本研究の特徴は以下の 3 点に集約できる．

- 関係データベース上にブランチ機能付き XML 版管理システムを構築する．
- 関係データベースの問い合わせ機能で木構造の接点を容易に検索できるバージョン識別子を定義する．
- 検索にかかるコストが常にほぼ一定である．

次節では，本研究が持つこれらの特徴について，関連研究との比較を行う．

2.2. 関連研究

XML をスキーマに依存せず関係データベースに格納する手法として xRel[4] [5]がある．これは XML 文書の持つ木構造をノードで分割し、ノードのタイプに従い関係データベースに格納する手法である．格納された各要素は木構造の経路をＩＤに持ち、経路表記を基に問い合わせを行うことが出来る．問い合わせを重視した xRel に対し Saxophone は SAX API を用いて XML 文書にアクセスすることを目標としている．

ブランチを有するバージョンの系列は木構造となる．提案方式はこの系列から特定のバージョンを取り出す為，各ノードに識別子を付けている．木のノードに効率的な識別子を付ける(Labeling Scheme)研究は多数されている[6] [7] [8] [9]．Labeling Scheme の代表的なアプローチの一つに半構造データを *k*-ary tree とみなし，接点に識別子を付ける手法[7]がある．ある接点の子要素が *k* 未満の時は，仮想的にノードが有ると仮定する．この手法では，識別子から木の位置が特定できるが，*k*-ary tree と大きく構造が異なるデータに対しては，仮想ノードの数も非常に多くなる等，効率の面で問題がある．この問題に対し，佐藤らはレベル毎に *k* の値を変化させる手法[9]を提案している．しかしながら，番号の枯渇を防ぐ為に木の高さを規定する必要があり，バージョンの更新が木の高さ方向に進む版管理システムには向かない．

Labeling Scheme の別のアプローチとしてプレフィクスにより木の親子関係を規定する Prefix Labeling 手法[8]がある．この手法は，ある識別子のプレフィックスとなる識別子を持つノードは祖先である，という特徴を持っている．プレフィクスの検索は関係データベースの機能を用いて容易に行うことができるため，提案手法で利用するのに適していると考えられる．一方で欠点は，skew のある木の末端では識別子が非常に長くなってしまうことである．提案方式ではこの Prefix Labeling のアプローチを基本とした識別子をバージョン検索に利用しているが，木の高さを表す値と組み合わせることによって，また任意の n 進数で識別子を構成することによって，それぞれ，木の高さ方向及び幅方向で識別子長の増加を防いでいる．

RCS で利用する差分情報は挿入情報と削除情報のどちらかで構成される．初期状態は NULL であり，バージョン１のデータはその NULL に対して挿入情報からなる．そして任意のバージョン v のデータはバージョン v-1 のデータとの差分から成る．差分データのみを保存することによって，データの冗長性を省きストレージコストを低減させている．しかしながら，この方法では基準となるバージョン１から検索したいバージョンまで全てのデータを走査しなければバージョンを復元することが出来ない．つまり隣接する二つのバージョン間の差分情報で版管理を行う場合，バージョン検索の計算コストがバージョンの数に比例して線形増加する事が問題視されている．

たとえばバージョン *n* から *n+1* への変換操作を $f_1(n)$ とすると，任意のバージョン *s* を検索する操作 *R* は以下のようになる．

$$R_1 = \sum_{k=1}^{s} f_1(k)$$

この R を一定にする方法として，バージョンの系列上でそのデータが有効である期間をデータに付加する方式の版管理手法[10] [11]*が考えられる．例えば，データの有効期間 $b \rightarrow d$ の要素 E について考える．この要素は $b \le s \le d$ を満たす任意のバージョン s で存在する．XML を構成する全ての要素・属性の存在を検証する操作を $f_2(x)$とすると，任意のバージョン s を検索する操作 R は

$$R_2 = f_2(s)$$

となり，バージョン検索にかかるコストは取り出すバージョンに依存しない．これは前述した隣接する二つのバージョンの差分情報を用いる手法に比べて効率がよい．また，ストレージのコストも差分情報しか格納しない為, 前者と同等である. さらに天笠らの手法[11]は前述した xRel との連携も可能[12]である．しかしながら，ブランチの有る系列では，バージョン番号が同じであってもブランチ毎に異なるデータが存在するため, バージョンを一意に識別することが不可能である．

提案方式での版は木構造の経路式表現を用いた識別子を用いてブランチのあるバージョンを一意に判別する．この識別子を利用してノードの有効期間を記述することにより，ブランチのある系列で効率良くバージョン検索が可能である．

3. Saxophone

Saxophone[3]はスキーマに依存せず複数の XML 文書を格納できる XML データベースであり，関係データベース上に構築される．特定の OS や RDBMS に依存せず多くの環境を利用することが出来る．例えば我々は PostgreSQL7.2 + Red Hat Linux 及び Microsoft SQL Server 2000 + Microsoft Windows 2000 Advanced Server 上で実装及び実験を行っている．

3.1. 格納手法

3.1.1. データ構造変換

XML 文書を関係データベースに格納する際問題になるのはデータ構造の違いである. XML は木構造を持つのに対して，関係データベースはテーブル構造である．Saxophone では XML を SAX イベントに分解することによって，木構造を関係データベースに格納可能な形式に変換する．SAX (Simple API for XML)は XML 処理を行う最も低レベルの API で，高速処理・低メモリ使用量を特徴としている．SAX パーサは XML 文書

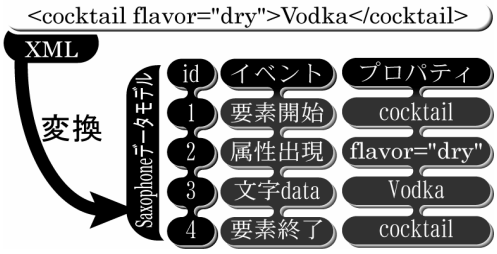


図 1 : Saxophone データモデルの例

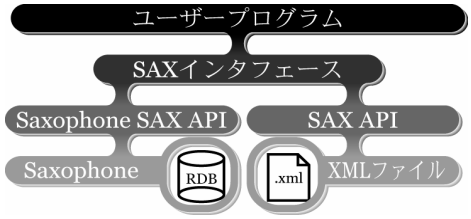


図 2: 提案システム Saxophone の透過性

を先頭から走査し，開始タグ等発見した XML の構造に応じてイベントを発生させる．このイベントをイベントの種類とそのプロパティと言う二つの属性を持つ列と見ることによって, XML をテーブル構造に変換することが出来る(図１)．このデータ構造を Saxophone データモデルと呼ぶ.

3.1.2. 関係データベース格納

前節の手順で構成した Saxophone データモデルを関係データベースに格納する．関係データベースのテーブル構成は Saxophone データモデルのスキーマには依存しない．例えば XML 文書によく見られる要素の繰り返しを排除し，ストレージコストを低減したスキーマ[3]等が考えられる.

3.2. 呼び出し

関係データベースに格納した XML 文書は，アプリケーションから既存の SAX パーサと同等のインターフェースを用いて呼び出すことが出来る.

つまり，プログラマは関係データベースの存在を気にする必要は無く，また SQL，XML Query 等の問い合わせ言語を新たに習得する必要も無い. XML ファイルに対する SAX パーサの呼び出し手順をもって，Saxophone にアクセスすることが出来る(図２).

4. 版識別子

この節では，ブランチのあるバージョン系列で任意のブランチに属する任意のバージョンを一意に指定することが出来る識別子について述べる．版識別子はバージョン番号とブランチ識別子から構成される.

4.1. ブランチ機能

単純なバージョン系列とブランチのあるバージョン系列の例を図３に示す．この例ではバージョン３に属する文書の版が４つあることを示している．このようにブランチのあるバージョン系列では，従来の単純

* この研究は単純な版管理システムではなく，「XML のある部分が最後に変更されたのは何時か」のように時制に留意した問い合わせを可能とする事も目的としている.

なバージョン番号での管理は困難である．そこで，バージョン系列を木とみなし，接点に識別子を付け，それを用いて版管理を行う．

4.2. ブランチ識別子の構成法

まず用語を定義する．あるバージョンの直系の子バージョンを幹と呼び，それ以外をブランチと呼ぶ．例えば図３のバージョン１にはブランチが二つあると言うことが出来る．ブランチ識別子は n 進数(n=3,4,5･･･)で構成され，根となるバージョン１は 1 となる．ブランチの分岐点以外の場所では，一つ前のバージョンと同じブランチ識別子を持つ．またブランチの分岐点の場合は以下のようになる．ブランチ識別子 A の分岐点が幹 B_0 および k 個のブランチ B_1,･･･,B_k を持つとき，任意の幹・ブランチのブランチ識別子 B_x は，以下のようになる．

$$B_x = nA + x$$

つまり十進数のブランチ識別子で A=11 の時，そこから派生する3番目のブランチ B_3 は以下の通りである．

$$B_3 = 10 \times 11 + 3 = 113$$

図３のバージョン系列はブランチ識別子を用いて表現すると図４のようになる．

詳しくは後述するが，このブランチ識別子から，木の祖先を示すことが出来る．B_x(0≦x＜n)の時，あるブランチ識別子 A の祖先の識別子は必ず A のプレフィクスとなる．この性質を B_x(n≦x)の時にも拡張するため，オーバフローフラグの概念を取り入れた．

n-1 のシンボル(十進数の時の 9)をオーバフローフラグとして利用することにより，枝の数の制限をなくすことが出来る．オーバフローも考慮し一般化したブランチ識別子の構成法は以下の通りである．

$$B_x = n^{\left\lfloor \frac{x+n-1}{n-1} \right\rfloor}(A+1) - n + \{x \bmod (n-1)\}$$

例えば十進数のブランチ識別子で A=11 のバージョンから発生する9番目及び19番目のブランチ識別子は

$$B_9 = (11+1)100 - 10 + 0 = 1190$$

$$B_{19} = (11+1)1000 - 10 + 1 = 11991$$

となる．オーバフローを用いれば，ブランチの数にかかわらず，プレフィクスの関係を用いて，祖先を示すことが出来る(図 5).

4.3. エポックポイント

分岐点以外のバージョン間ではブランチ識別子は変化しない(図 4). ブランチ識別子は可変長文字列型として関係データベースに格納する事を目的としている．すなわち桁が大きくなるとそれだけ多くの格納コスト

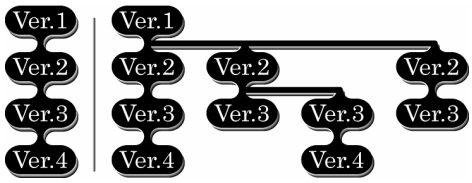


図 3: 単純な系列(左)とブランチのある系列(右)

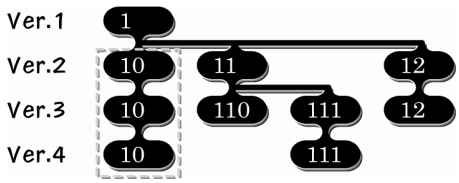


図４: ブランチ識別子構成例

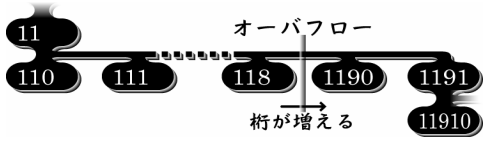


図５: オーバフローの例

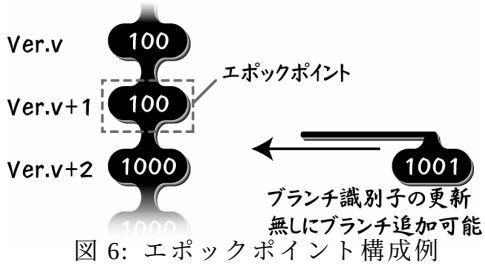


図 6: エポックポイント構成例

が必要となり，またすぐに枯渇してしまう．そこでバージョン番号とブランチ識別子の組でバージョンを特定する事によって消費するリソースを低減させている．例えば図 4 でバージョン４･ブランチ識別子 10 であれば，一番左の系列のバージョン４と特定できる．

この手法で問題となるのが更新に伴いブランチ識別子の付け替えが発生する事である．提案方式では，予め後々ブランチが発生すると予想されるバージョンを指定する事で頻繁な更新を回避する事が出来る．このバージョンをエポックポイントと読んでいる．エポックポイントのバージョン A とその次のバージョン B の関係は以下のようになる．

$$B = nA$$

ブランチ識別子 100 のバージョン v+1 をエポックポイントに指定した際の前後のブランチ識別子は図 6 の様になる．

4.4. バージョン系列指定

ブランチ識別子を用いれば、任意のバージョンからそのバージョンの祖先・子孫を特定する事が可能である．例えばブランチ識別子 111 の親となるブランチ識別子は 11 であり、そのまた親となるブランチ識別子は 1 である．このように，あるブランチ識別子 A のプレフィクスとなるブランチ識別子は全て A の祖先のバー

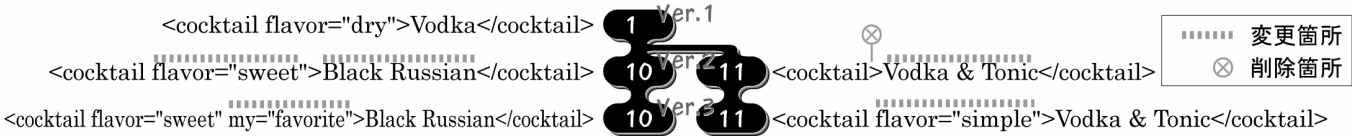


図 7: 複数のバージョンを持つ XML 文書の例

ジョンである．また，Aがプレフィクスとなるブランチ識別子は全てAの子孫である．

前述したようにブランチ識別子は可変長文字列型として関係データベースに格納して利用する．SQL を用いれば，ある文字列が別の文字列のプレフィクスになるかを問い合わせることが出来る．LIKE 述語と%記号，そして各データベースの文字列関数を用いることにより可能である．

例えばブランチ識別子 110 自身及びその子孫のバージョンを取り出すには以下の構文を用いる．

SELECT * FROM *database*
WHERE *branch_id* LIKE ’110%’;

この問い合わせによって，110 をプレフィクスに持つブランチ識別子が選択される．

一方あるバージョンの祖先を調べる場合，指定した文字列のプレフィクスになる文字列をデータベース中から検索する必要があり LIKE 述語を用いる事は出来ない．そこで文字列関数を用いる．

postgreSQL では position(*substring* in *string*)関数を用いて *string* 内の *substring* の位置を知ることが出来る．つまり position(*branch_id* in ‘110’)が 1 を返す時，*branch_id* は自分自身又は祖先であると言える．

Microsoft SQL Server 2000 でも同等の役割を持つ CHARINDEX 関数がある，また他の多くの関係データベースで同じ機能を持つ関数を利用することが出来る．

本稿では説明の為に十進数のブランチ識別子を用いているが，ブランチの数が多くオーバフローが頻繁に発生すると予想されるならば，任意の数を用いることができる．また，途中から変更することも可能である．例えば一つのバージョンから最大で 998 のブランチが発生する場合，十進数を用いるよりも千進数を用た方がストレージコストを低減できる．なお，検索は文字列として行われる為，各数のシンボルには各ロケールの文字セットや UNICODE 文字を含む任意の文字を割り振ることが出来る．

5. 版管理手法

Saxophone データモデルを拡張し，各 SAX イベントに版識別子を付加する事により，各バージョンの差分データのみを格納した版管理を行うことが出来る．図 7 のバージョン系列を持つ XML 文書を例に説明する．

バージョン１の Saxophone データモデルは図 1 で示

図 8: Saxophone データモデルの拡張

したが，版識別子で拡張したものを図 8 に示す．このように，各 SAX イベントを表すタプルにそのイベントが発生したブランチ識別子とバージョン番号が付加されている．1.1 節で述べた差分管理の二つの手法，即ち差分を挿入削除情報で表す手法及びデータのバージョン系列内での有効期間を付加することにより管理する手法に分類すると，この提案方式は後者にあたる．ただし，各イベントに付加されている版識別子は，そのイベントの有効期間の開始時期を表している．提案方式では，終了時期を明記しない．終了時期は，そのバージョン以降で同じイベントＩＤを持つイベントが発生した時点となる．

5.1. バージョン間差分データ

提案方式ではストレージコストの低減のため，バージョン間の差分のみをデータベースに格納する．XML 文書間の差分を取得する手法は本研究室で開発した xPatch/xoDiff[3]で行うことが出来る．このツールは patch/diff と類似した手続きを用いて XML の構造を考慮した差分を取得するツールである．差分情報は diff 出力と同じフォーマットで挿入・削除箇所を示す．

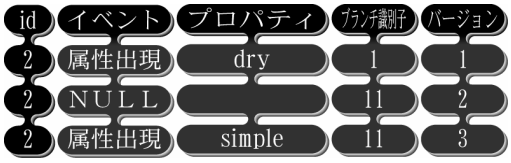
まず XML 文書の文字列(イベント id３)に注目する．Vodka という文字列が，それぞれ，Black Russian 及び Vodka & Tonic に変更されていることがわかる．すなわち，バージョン２のブランチ識別子{10,11}で文字列イベント{Black Russian , Vodka & Tonic}が有効になったと言うことが出来る．またこのイベントに関して，バージョン２と３の間には差異が無いためバージョン３として格納されるデータは無い．このバージョン系列におけるイベント id３のデータは図 9 の通りである．

5.2. NULL イベント

Saxophone データモデルに付加されるのは，イベントの有効期間の開始時期であるため，イベントが削除されたことを示すことが出来ない．そこで SAX イベントを拡張し NULL イベントというのを定義した．NULL

id	イベント	プロパティ	ブランチ識別子	バージョン
3	文字data	Vodka	1	1
3	文字data	Black Russian	10	2
3	文字data	Vodka & Tonic	11	2

図 9: バージョン情報(id=3)



id	イベント	プロパティ	ブランチ識別子	バージョン
2	属性出現	dry	1	1
2	NULL		11	2
2	属性出現	simple	11	3

図 10: バージョン情報(id=2)

イベントは何も無い事を示すイベントである．イベント id=2 のイベントはブランチ識別子 11 のバージョン２で削除されているが，これは NULL イベントの開始として図 10 の様に表される．

ブランチ識別子 11 のバージョン２を検索すると，この NULL イベントが結果となる．

5.3. 新規イベント挿入

ブランチ識別子 10 のバージョン３では属性出現イベント my=”favorite”が発生している．SAX イベントの順序では３番目であるが，イベント id の付け替えを避けるため，前後のイベント id の間の数で構成する．図の例の場合，前後直近のイベント id は 2 と 3 であるため，この新しく発生したイベントの id は 2.1 となる．

この方法を用いればイベント id の付け替えを行うこと無く新しいイベントを挿入することが出来る．この小数を用いたイベント id の構成には，構成法及び最適化等については議論があるが，本稿では詳述しない．

5.4. バージョン検索

本節では，これまでに述べた方法で Saxophone に格納したバージョン系列から特定のバージョンを取り出す手法について説明する．Saxophone が特定の RDBMS やそのスキーマ構成に限定されないことは前述したが，ここでは現在我々が Microsoft SQL Server 2000 上に構築しているシステムを例に説明する．なお実験システムの概要を以下に記す．

ハードウエア：DELL PawerEdge 2650
CPU：Xeon 2GHz ×2
メモリ：512MB
OS：Microsoft Windows 2000 Advanced Server
RDBMS：Microsoft SQL Server 2000 Enterprise Edition
使用ライブラリ：ADO.NET

またデータベースは Saxophone データモデルをそのまま格納するスキーマで構成されている．

5.4.1. テーブル構成

Saxophone はユーザ認証などに利用するテーブルを除くと，二つのテーブルから構成される．XML 文書のメタデータを格納する resources テーブルと Saxophone データモデルを格納する saxophone テーブルである．各々のスキーマを図 11 に示す．

resources

主キー	Field名	型名	説明
⚿	rid	int	ブランチ系列固有id
	f_path	varchar	hostname上でのファイルパス
	f_url	varchar	web公開アドレス
	owner	int	リソースの所有者id
	subscriber	varchar	リソースの閲覧者グループid
	lastmodify	datetime	最終更新日時
	hostname	varchar	リソースの登録元hostname
	head bid	varchar	上位ブランチとの分岐点ブランチ識別子
	rg	int	バージョン系列固有id
	explanation	text	このリソースの説明

saxophone

主キー	Field名	型名	説明
⚿	rid	int	ブランチ系列固有id
	eid	int	イベントid
⚿	bid	varchar	ブランチ識別子
⚿	birth	int	バージョン番号
	type	char	イベントの種類
	property	text	イベントのプロパティ
	rg	int	バージョン系列固有id

図 11: 関係データベーススキーマ

エンドユーザが取り出したいバージョンの版識別子を指定することは困難である．そこで，ブランチと関連付けてファイルパスおよび URL を格納することにより，ユーザはファイルパスとバージョン番号をキーにバージョン検索することが出来る．

5.4.2. 検索アルゴリズム

取り出したいバージョンの版識別子が判っている場合，以下の条件を満たしているタプルがそのバージョンで有効な SAX イベントである．

1. ブランチ識別子(bid フィールド)の値が検索対象の版識別子のプレフィクスを持つもの
2. 同じイベント ID(eid フィールド)を持つタプルが複数存在する場合は birth が最大値のもの

この二つの条件に合致するタプルを eid の昇順で取り出すことによって，任意の版における SAX イベントのプロパティと順序を保護している．

Saxophone には一つのデータベース中に複数の異なるリソース及びバージョン系列を格納することができる．つまり実際には，上記の操作に加えてデータベース中から取り出したいバージョン系列を特定する操作が必要となる．また，ユーザが検索する際，ブランチ識別子で指定させるのは難しい，よって提案方式ではそれぞれのブランチに異なる URL を付け，ユーザはファイルパスとバージョン番号を用いて検索することが出来るような仕組みになっている．

提案システムの検索機能はストアドプロシージャ内に隠蔽されている．ストアドプロシージャとはサーバサイドで実行される一連の処理であり，Microsoft SQL Server 2000 の場合，SQL の独自の拡張である Transact-SQL 構文を用いて記述できる．

ファイルパスとバージョン番号を引数に持ち，その

```
CREATE PROCEDURE sax_user.getEvents(@ckeyword varchar(40),@F_PATH varchar(1024), @VERSION INT) AS
DECLARE @RC int
  --①ログイン処理                                                                                          ①
EXEC @RC = auth @ckeyword
if(0 <> @RC )
BEGIN
  --②ファイルパスで表されるリソースの親との分岐点のバージョンが持つブランチ識別子を得る                      ②
  DECLARE @HBID VARCHAR(1024);
  SET @HBID =(select head_bid from resources where f_path =  @F_PATH);
  --③ファイルパスで表されるリソースのRIDを得る                                                              ③
  DECLARE @RID INT;
  SET @RID = (select rid from resources where f_path =  @F_PATH);
  --④ファイルパスで表されるリソースのRGを得る                                                               ④
  DECLARE @RG INT;
  SET @RG = (select rg from resources where f_path =  @F_PATH);
  --⑤ridに属するイベントのうち分岐点以降(すなわち@HBIDがbidフィールド値のプレフィクスとなるレコード)
  --    および以前(bidフィールドが@HBIDのプレフィクス)のレコードで得たいバージョンのイベントからbidを得る      ⑤
  DECLARE @BID VARCHAR(1024);
  SET @BID = (select top 1 bid from saxophone where rg = @RG and ((1 = CHARINDEX(bid,@HBID))
             or (1 = CHARINDEX(@HBID,bid)  and rid = @RID)) and birth = @VERSION);
  --⑥自分とその祖先について(すなわちbidが@BIDのプレフィクス)、eidごとにbirthの最大値を求める＝>birthの最大値は「最近に生まれたイベント」を意味する =>一時テーブル  ⑥
  select eid,MAX(birth) as birth into #valid_event  from saxophone
   where  rg = @RG and birth <= @VERSION and 1 = CHARINDEX(bid,@BID)  group by eid;
  --⑦自分とその祖先について、一時テーブルのハッシュを元に、全イベントをとりだす。イベント順でソートすることで、SAXイベント順を保証できる  ⑦
  select saxophone.rid,saxophone.eid,event_type.event,saxophone.property
   from saxophone,#valid_event,event_type
   where saxophone.rg = @RG and event_type.type = saxophone.type
    and #valid_event.eid = saxophone.eid and saxophone.birth = #valid_event.birth
    and  1 = CHARINDEX(saxophone.bid,@BID) and saxophone.type != 0 order by saxophone.eid;
 return 1;
END
ELSE
BEGIN
  --ログイン失敗
  return 0;
END
```

図 12: バージョン問い合わせのストアドプロシージャ

バージョンの XML 文書を取り出すストアドプロシージャを図 12 に示す

Saxophone は XML データベースとして独自にユーザ管理機能を持っている為，まずユーザ認証を行う(①). 次にファイルパスからそのブランチの分岐点が持つ(②)ブランチ識別子(head_bid)，(③)リソース id(rid)及び(④)ブランチ系列(rg)を検索する．そして head_bid を元に，自分の検索しようとしているバージョンが実際に持つブランチ識別子を取り出している(⑤).

CHARINDEX(*substring*,*string*)は文字列の開始位置を返す関数で，この関数が 1 を返すとき，*substring* は *string* のプレフィクスであると言うことが出来る．ここでは前述したブランチ系列指定手法を用いて，検索キーとして与えたファイルパスのブランチ自身そして祖先及び子孫のバージョンからブランチ識別子を検索している．このことにより，例えば図 7 のブランチ識別子 11 に属するリソースのバージョン 1 が検索された時，その祖先であるブランチ識別子 1 に属するバージョン 1 を返すことが出来る．また，最新バージョンよりも新しいバージョンが指定されたとき，最新のバージョンを返すことが出来る為, int 型の最大値を与えれば最新バージョンが得られる．

次に同じイベント id を持つタプルの内, 取り出したいバージョンに最も近いバージョンに属するデータのみを取り出す(⑥)．ここではイベント id とそのバージョン番号の最大値を一時テーブルに格納している．

次に⑥で作った一時テーブルのイベント id とバージョン番号を元に saxophone テーブルからイベントを呼び出す(⑦)．この際イベント id で昇順にソートしてやることで, SAX イベント順は保障される. また NULL イベントは無視する．

この様にある要素が有効である期間を付加する手法は，diff を用いた RCS 等の手法に比べ検索効率が良い．diff を用いる場合，呼び出したい版の深さに比例した回数のバージョン復元操作が必要であるが，この手法では文字列のプレフィクス検索とバージョン番号の比較操作のみで検索を行うことができる．

前述した天笠らの手法[11]もこの方式をとっている. 提案手法との違いは，提案手法がデータの出現時を表す識別子のみを付加するのに対し，データの消滅時期も付加するところである．この手法を利用すれば，図 12 の⑦に相当する操作が必要なくなり, バージョン検索にかかるコストがより一定になると予想できる．しかしながら，ブランチのあるバージョン系列では，あるブランチでは消滅しているが，別のブランチではまだ有効である等，消滅時期を一意に表すことが不可能であるため, 提案システムに適用することは出来ない. しかしながら我々は関係データベースの機能である B Tree インデクスによって，手続き⑥のコストをほぼ一定に保てるのではないかと考えた．この点に関して実験を行い検証した．

5.5. バージョン検索のコスト

次の XML 文書を用いて，バージョン検索にかかるコストを比較する．

バージョン 4n+1 (n=0,1,2,3,…24)：
<VERSION even=”no”>A</VERSION>

バージョン 4n+2 (n=0,1,2,3,…,24)：
<VERSION even=”yes”>A</VERSION>

バージョン 4n+3 (n=0,1,2,3,…,24)：
<VERSION even=”no”>B</VERSION>

バージョン 4n+4 (n=0,1,2,3,…,24)：
<VERSION even=”yes”>B</VERSION>

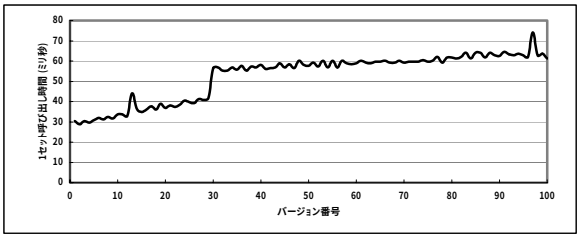


図 13: バージョン検索コスト

なお，ブランチが複数ある場合をシミュレートするため，すべてのバージョンをエポックポイントに指定している．すなわちバージョン *n* はバージョン *n-1* を分岐点とするブランチとみなすことができる．これによって 99 階層の擬似的なブランチを構築している．

この XML 文書を各バージョンにつき 10 回呼び出す事を 1 セットとし，それを 25 回行い，平均をプロットする．結果を図 13 に示す．バージョン３０以降の検索時間でほぼ一定の結果が得られた．それ以前の部分の検索時間は劇的に下がっているが，これは提案方式のアルゴリズムによるものでは無い為，RDBMS 側でなんらかのクエリ最適化がなされたものと考えられる．

この結果より，図 12 の⑥に関するコストは版・ブランチの数に依存するものの，検索時間は，検索したいバージョンがブランチのどこに位置するかに関係なくほぼ一定であると言える．

6. まとめと今後の課題

本稿では，木構造を持つバージョン系列から任意のバージョンを特定する版識別子を提案し，それを利用したブランチ機能つき XML 版管理手法を提案した．

版識別子はバージョン番号とブランチ識別子で構成される．ブランチ識別子は以下の特徴を持つ．

- ブランチ識別子Ａのプレフィクスになるブランチ識別子ＢはＡの祖先である．
- エポックポイントを効果的に用いることにより，バージョンの更新や新規ブランチ作成の際にブランチ識別子の更新が生じない．
- バージョン番号と組み合わせる方式によりバージョン識別子の枯渇が起きにくい．

さらにバージョン検索に関する実験を行い，検索コストが任意のバージョンでほぼ一定である事を示した．

今後は以下の課題に取り組む予定である．

- 現在 SAX を用いて XML 文書全体を取り出す事は可能であるが，これを拡張して XML への問い合わせを行う手法を考えている．
- ブランチのマージ等，バージョンシステムとしての発展を考える．
- 提案方式の応用について考える．例えば格納した XML 文書を URL で指定することが出来る機能を利用して，版を考慮した XHTML のプロクシサーバの実現が考えられる．

謝辞

本研究の一部は，文部科学省科学研究費特定領域研究(2)[情報学:A02](課題番号:14019075)，東京都立大学総長特別研究費(特別重点研究)，日本データベース学会･マイクロソフト株式会社共催 2002 年度データベース研究支援プログラムによる．

文　献

[1] Walter F Tichy, "RCS-A System for Version Control," Software-Practice & Experience, 15, 7, pp.637-654, July 1985.

[2] Marc J. Rochkind, "The Source Code Control System," IEEE Transactions on Software Engineering, SE-1, 4, pp.364-370, Dec. 1975.

[3] 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)論文集，A5-3，http://www.ieice.org/iss/de/DEWS/proc/2002/ ，Mar. 2002.

[4] 吉川正俊，志村壮是，植村俊亮，“オブジェクト関係データベースを用いた XML 文書の格納と検索”，情報処理学会論文誌：データベース第 40 巻，第 SIG6(TOD3)号，pp. 115-131, 1999

[5] M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases," ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 110-141, 2001

[6] Edith Cohen, Haim Kaplan, and Tova Milo, "Labeling Dynamic XML Trees," Proc. PODS 2002, pp. 271-281, 2002.

[7] Quanzhong Li and Bongki Moon, "Indexing and Querying XML Data for Regular Path Expressions, "Proceedings of the 27th VLDB Conference, Roma, 2001.

[8] Serge Abiteboul, Haim Kaplan, and Tova Milo, "Compact labeling schemes for ancestor queries," Proc. SODA 2001, pp. 547-556, 2001.

[9] 佐藤隆士，里本智彦，小畑喜平，潘洪涛，“階層構造を識別可能な木節点の番号付け”，第 13 回データ工学ワークショップ(DEWS2002)論文集，A4-8，http://www.ieice.org/iss/de/DEWS/proc/2002/ ，Mar. 2002.

[10] Chien, S.Y., Tsotras, V. J., Zaniolo, C., and Zhang, D., "Storing and Querying Multiversion XML Documents using Durable Node Numbers," 2nd International Conference on Web Information Systems Engineering(WISE), Kyoto Japan, 2001

[11] T. Amagasa, M. Yoshikawa, and S. Uemura, "A Data Model for Temporal XML Documents," Proc. of the 11th International Conference and Workshop on Database and Expert Systems Applications (DEXA2000), London-Greenwich, U.K., Sept. 2000.

[12] 天笠俊之，吉川正俊，植村俊亮，“XML 文書のためのバイテンポラルデータモデル”，電子情報通信学会技術研究報告〔データ工学〕,DE2001-66, Jul. 2001