Subversion の管理構造に基づいた地図更新管理

広部 一弥† 渡邉 豊英‡

†‡名古屋大学 情報科学研究科 社会システム情報学専攻 〒464-8603 名古屋市千種区不老町

E-mail: †hirobe@watanabe.ss.is.nagoya-u.ac.jp, ‡watanabe@is.nagoya-u.ac.jp

あらまし 時間管理を行う地図情報システムでは，蓄積する情報が持続的に増加し続けるため，過去のデータ構造に過度に依存せずに時間単位でデータを独立に処理できること，情報を蓄積する期間によらずに簡潔に情報を取得できることなどが必要となる．本稿では，ファイルシステム・ベースのバージョン管理システム Subversion のデータ管理構造に着目し，過去のデータ構造に依存せずに地図の部分画像を貼り付ける形で更新する差分情報記述方法と，それを用いて HR-木構造のデータベースで地図の変更情報を管理するデータ構造を提案する．本稿で扱う地図が対象とする情報は大学構内の地理情報であり，地図内の情報はラスター画像矩形を貼り付けていく形で管理される．

キーワード 時空間管理, GIS, バージョン管理

Change Management for Geographic Information Systems Based on Structure of Subversion

Kazuya HIROBE† Toyohide WATANABE‡

†‡Department of Systems and Social Informatics Graduate School of Information Science Nagoya University

Furo-cho, Chikusa-ku, Nagoya, 464-8603, Japan

E-mail: †hirobe@watanabe.ss.is.nagoya-u.ac.jp, ‡watanabe@is.nagoya-u.ac.jp

Abstract In a GIS which manages time, information keeps increasing. Therefore, the GIS should process a transaction without depending on a data structure, moreover it should extract information easily without depending on an amount of transactions. In this paper, we focus on the data management structure of Subversion which is a version management system for file systems. We propose a method to descript differences of changing geographic information by putting images on a map and a data structure for managing geographic information based on Historical R-trees.

Keyword Temporal Spatial Management, GIS, Version Management

1. はじめに

電子地図はその地域の建築物や道路の位置を直感的に表現できるため，近年様々な情報システムで利用されている．時間管理を行う地図情報システムでは，蓄積する情報が持続的に増加し続けるため，過去のデータ構造に過度に依存せずに時間単位でデータを独立に処理できること[1],情報を蓄積する期間によらずに簡潔に情報を取得できることなどが必要となる．

本稿では，過去のデータ構造に依存せずに地図の部分画像を貼り付ける形で更新する差分情報記述方法と，それを用いて HR-木構造[4]のデータベースで地図の変更情報を管理するデータ構造を提案する．データ管理構造に依存しない差分記述を行うことで，地図情報システムから特定期間における差分情報を抽出することが可能となり，分散化された地図情報システム間での情報の交換が容易となる．本稿で扱う地図が対象とする情報は大学構内の地理情報であり，地図内の情報は建物やフロアのような領域を持つオブジェクトにより構成される．領域を持つオブジェクトは，矩形のラスター画像で表現され，常に最新の日時で貼り付けられて更新される．道路のような線情報については陽には扱わない．

２章で，本稿のアプローチを紹介する．３章で，このようなデータ構造を提案するために，バージョン管理システムシステム Subversion の管理構造と差分記述方式を紹介する[3]．４章で，本稿で扱うデータ構造に関してシステム内部の具体的な差分記法，利用者が指定する抽象的な差分記法を提案し，抽象的な差分記

法から具体的な差分記法へ変換する手法を述べる．５章でまとめと今後の展開について説明する．なお，本稿ではリビジョン内の地図データを R-木で管理する．R-木において葉以外のノードは利用者に意識させるべきではないが，Subversion に沿った差分記述では，ディレクトリ・パスに相当するノード構成を記述する必要がある．これによる差分情報のデータ構造への依存を避けるため，さらに抽象的な外部デルタ記述を導入する．

2. アプローチ

Subversion はファイルシステム・ベースのバージョン管理システムである．サーバ内のデータ管理においては overlapping B-木[2]の構造でソース・ツリー構造のバージョンを管理することで，簡潔なノード探索を実現している[3]．また，それぞれのリビジョンにおける差分情報をノードの ID ではなく，リビジョン番号とディレクトリ・パス表記の文字列を用いて表現することで，個々の差分情報をサーバのデータ管理構造に依存せずに扱うことができる．

本稿では, overlapping B-木の考え方を R-木に適用した HR-木でサーバ内データを管理し，データの差分記述方法として Subversion のディレクトリ構造差分表記法であるツリー・デルタを用いる．Subversion のツリー・デルタはノード指定のためにディレクトリ・パス文字列を用いるが，本稿では代替としてノードが保持する MBR(Minimum Bounding Rectangle)の領域をノード指定のために用いる．これを，内部デルタと定義する．ただし，内部デルタは HR-木の木構造変更のようにサーバ内部データ管理構造の変更を具体的に記述しているので，目的としているデータ管理構造に依存しない差分記述とは異なる．そこで，地図に対する抽象的な差分記述を行うための記法として外部デルタを定義する．

図 1 に管理体系を示す．外部デルタでは，対象とする領域とそこに貼り付けるべきラスター画像のみを記述する．外部デルタは，データベース内のデータ構造に依存しない差分記述である．この外部デルタと，データ管理構造内のノード構造を基に，内部デルタを生成する．内部デルタは，データベース内のデータ構造の操作を含む具体的な差分記述である．内部デルタにより時空間データベースを更新する．データベースは HR-木構造によりデータの時間的蓄積量によらず簡潔にデータを参照できる．これにより，データ管理構造に依存しない外部デルタの記述で，高速に参照できる時空間情報を管理する．

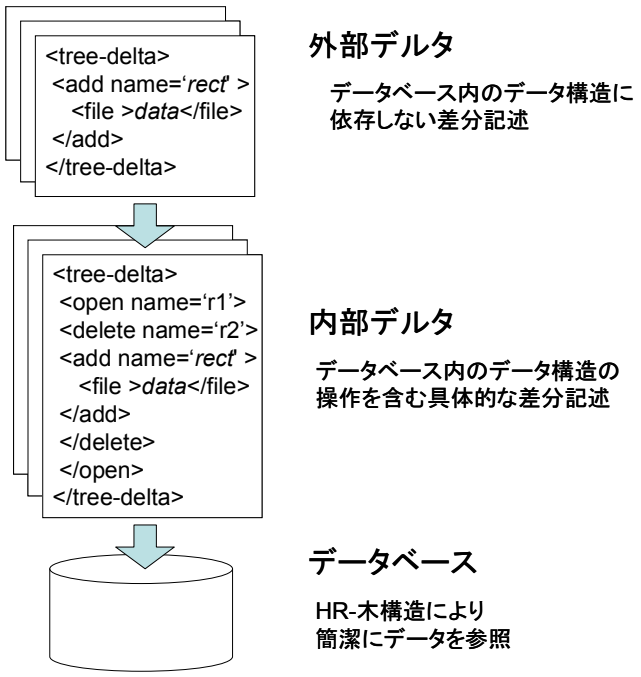

図 1：管理体系

3. Subversion の管理構造

Subversion は，３つの差分表現を持つ．

1. ツリー・デルタ：ディレクトリ構造の変更を表現
2. ファイル・デルタ：ファイル内容の変更を表現
3. プロパティ・デルタ：ファイル作成日のようなファイルに付随する情報を表現

本研究では，1 のツリー・デルタに着目する．

3.1. サーバ内のデータ管理構造

Subversion はサーバ内において，overlapping B-木の管理構造を持つ．これにより，指定したリビジョンのファイル・ツリーは，指定したリビジョンの根ノードから下位にノードを探索していくことで簡単に取得できる．

なお，Subversion 独自の構造として，ノードのファイル名を親ノードが保持するという特徴を持つ．これはファイル名の変更やコピー操作といったファイルの内容を変更しない操作に対しては，親ノードのみに変更が発生することを意味する．

図 2 に，新しいリビジョンにおいてファイル D.txt の内容を変更した場合のノードを示す．まず，D.txt の葉ノードを生成する．次に，その親ノードを根ノードに到達するまで直前のリビジョンのコピーとして生成する（フォルダ A,C）．その際に，更新のないノードについては直前のリビジョンのノードへリンクを生成する．最後に，根ノードを最新のリビジョン番号(リビジョン 2)にリンクする．

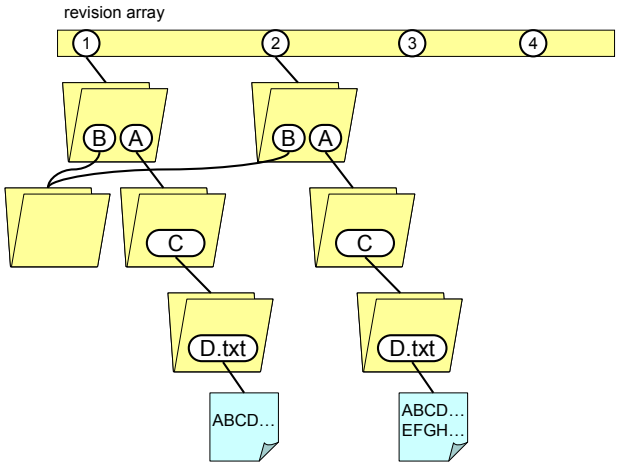

図 2：ファイル D.txt の内容を変更

3.2. Subversion におけるツリー・デルタ表記

Subversion におけるツリー・デルタの記法を説明する．Subversion におけるツリー・デルタの記法は，階層的な表現によって逐次に解釈されるべき変更関数の列挙である．

ツリー・デルタの構成要素を表 1 に示す．ツリー・デルタの根要素は tree-delta である．tree-delta は複数の change をもつ．change は，open，add，delete のいずれかの変更操作である．open はその content に対して変更を加えることを示し，add は content を生成することを示す．delete は content が新しいリビジョンには存在しないことを示す．それぞれの変更操作において name は新しいリビジョンにおける対象ファイル名またはディレクトリ名である．content は変更内容を示し，file または directory のいずれかで表される．file はファイルを表し，ancestor，prop-delta，text-delta を引数として持つ．ancestor は，前のリビジョンにおける content のパスを示す．prop-delta，text-delta はそれぞれ変更後のプロパティ・デルタ，変更後のファイル・デルタを表す．directory はディレクトリを表し，ancestor，prop-delta，tree-delta を引数として持つ．

Subversion において，ツリー・デルタを XML で記述した例を図 3 に示す．この例では，まず，ディレクトリ/dir1 に含まれる file1 の内容 text-delta を変更している．text-delta はファイルの変更内容を表し，テキスト・デルタの差分記述形式で記述される．次に，ディレクトリ/dir1 に含まれるディレクトリ dir2 のディレクトリ名を dir3 に変更している．

表 1：ツリー・デルタの構成要素

tree-delta($change_1$,$change_2$, …$change_n$)	ツリー・デルタの根要素
change:	変更操作
open([name],[content])	name という名称のノードを content に変更
add([name],[content])	name という名称の content の追加
delete([name])	name の削除
name:	変更後ファイル名
content:	変更後の内容
file([ancestor],prop-delta, text-delta)	ファイル
file(ancestor)	
directory([ancestor], prop-delta,tree-delta)	ディレクトリ
directory(ancestor)	
prop-delta:	プロパティ・デルタ
text-delta:	テキスト・デルタ
ancestor:	前リビジョンでのパス
(path, new)	path：パス文字列 new：新しいファイルか否かの真偽値

```
<tree-delta>
  <open name='dir1'>
    <directory>
      <tree-delta>
        <open name='file1.txt'>
          <file>text-delta</file>
        </open>
        <delete name='dir2' />
        <add name='dir3' >
          <directory ancestor='/dir1/dir2' />
        </add>
      </directory>
  </open>
</tree-delta>
```

図 3：ツリー・デルタの XML 表記

4. 地図の管理構造

4.1. サーバ内のデータ管理構造

サーバ内のデータ管理構造は HR-木の形を持つ．ただし，Subversion で親ノードが子ノードの名称を保持することに従って，親ノードが子ノードの MBR を保持する．

4.2. 内部デルタ

内部デルタの記法を定義する．記法は Subversion のツリー・デルタ記法に準拠する．ただし，ファイル・

パスおよび，ファイル名，ディレクトリ名に相当する部分は変更するノードの MBR となる．また，ファイル・ノードは HR-木において葉ノードを指し，ディレクトリ・ノードは葉でないノードを指す．

ツリー・デルタは，階層的な表現によって逐次に解釈されるべき変更関数の列挙である．

以下の３つのパラメタを導入する．ツリー・デルタの階層を逐次に探索してゆく際には，この３つのパラメタを保持して解釈することを前提とする．

・revision：特定のリビジョン
・source-dir：ソース・ツリーの中で，これから変更しようとするノードの MBR
・target-dir：構築中のディレクトリ

ツリー・デルタの根においては，source-dir, target-dir は総て同一に根ノードの MBR を保持する．ツリー・デルタを探索するにつれて，target-dir は生成するノードの MBR を保持し，source-dir はオリジナルのソース・ツリーにおいて対応するノードの MBR を保持する．

ツリー・デルタは，以下のように表現される．

$$tree\text{-}delta([change_1,change_2,..change_n]) \quad (1)$$

change が，どのようにして *source-dir* のコンテンツを *target-dir* へと編集するかを示す．３種類の *change* が存在する．

・open
・delete
・add

open は，以下のように表現される．

$$open([mbr],[content]) \quad (2)$$

open は，*source-dir* と *target-dir* の両方を *mbr* に持つノードに変更する．*content* は変更対象となる *file* または directory を示す．

delete は，以下のように表現される．

$$delete([mbr]) \quad (3)$$

source-dir に含まれる *mbr* という MBR を持つのノードは，*target-dir* には存在しないことを表す．

add は，以下のように表現される．

$$add([mbr], [content]) \quad (4)$$

source-dir には現在はまだ存在しない *mbr* という MBR を持つノードは，*target-dir* に存在することを表す．*content* は，新しいディレクトリ・エントリが示すことになるファイルまたはディレクトリである．

なお，*source-dir* に存在するが，*change* にて指定されないノードは，*target-dir* において変更が発生しないことを示す．

content は，ファイルまたはディレクトリの変更内容である，次の形式のうちの 1 つである．

・file
・directory

file は，以下のように表現される．

$$file([ancestor], text\text{-}delta)$$

または，

$$file(ancestor) \quad (5)$$

file は現在のノード *target-dir* が，葉ノード *f* についての変更であることを示す．*ancestor* は，*f* が以前のリビジョンにおいて R-木のどの葉ノードであるかを示す MBR である．*text-delta* は，*f* が *ancestor* を元に変更されたラスター画像矩形である．*text-delta* が省略された場合は，ラスター画像矩形の内容に変更がないことを示す．これは，MBR の変更や HR-木の分割のようなツリー形状のみの変更を表す．

directory は，以下のように表現される．

$$directory([ancestor], tree\text{-}delta)$$

または

$$directory(ancestor) \quad (6)$$

directory は現在の *target-dir* ノードが，葉でないノード *s* についての変更であることを示す．*ancestor* は，*s* が以前のリビジョンにおいて R-木のどのノードであるかを示す MBR である．*tree-delta* は，*s* が *ancestor* からどのように構成されたかを示すツリー・デルタである．省略された場合は，*s* の下位葉ノードにあるラスター画像矩形の内容に変更がないことを示す．これは，MBR の変更や R-木の分割のようなツリー形状のみの変更を表す．

上記の file エレメントと directory エレメントのなかで，*ancestor* は以下の２項で表される．

$$ancestor = ([mbr],[new]) \quad (7)$$

mbr は，リビジョンにおける新しいまたは変更されたノードの，前のリビジョンにおける MBR は *mbr* であることを示す．これが *file* エレメントの属性として現れたとき，エレメントのテキスト・デルタは *mbr* に適用される．これが *directory* エレメントの属性として現れたとき，*mbr* は，エレメントのツリー・デルタを解釈する際の新しい *source-dir* として扱われる．*new* は，ノードが R-木の中で *ancestor* を持たないことを示す真偽値である．*new* が真かつテキスト・デルタが付随する場合，デルタは空のノードに新しいノードとして適用される．*new* が真かつツリー・デルタが付随する場合，デルタは *source-dir* が空のノードであるかのように評価される．*mbr* と *new* の両方が省略された場合，これは同じリビジョン番号を持つ *ancestor=('mbr',)* と同じである．

4.3. 外部デルタ

内部的なデルタ記法では，R-木の構造の変更方法についても記載するが,地図データの更新を考えた場合,内部のツリー構造を参照できない利用者や外部システムが指定することは困難である．利用者や外部システムからの更新のために，外部的なデルタ記法を定義する．

外部的なデルタ記法においては，ノードの操作を記述しない．また，MBR が一致する葉ノードが HR-木内に存在する必要はない．

内部デルタと外部デルタの記述の比較を表 2 に示す．外部デルタで使われるのは tree-delta,add,file の 3 つのみである．従って，tree-delta は常に以下の形式で構成される：

$$tree\text{-}delta(change_1,change_2,..change_n) \quad (8)$$
$$change_n = add(mbr,file([ancestor],\ text\text{-}delta\ ))$$

表 2：内部デルタと外部デルタの比較

	内部デルタ	外部デルタ
tree-delta	○	○
open	○	×
delete	○	×
add	○	○
directory	○	×
file	○	○

5. 外部デルタから内部デルタへの変換

外部デルタから内部デルタへの変換は，以下の手順で行う．

1. 領域指定の変更
2. R-木構造変更記述の追加

領域指定の変更では，外部デルタで ancestor に指定された領域と，直前リビジョンの R-木内の葉ノードを比較し，R-木内の葉ノード領域に合わせて，関数と ancestor を変更する．

R-木構造変更記述の追加では，R-木の領域分割に伴うツリー構造の変更記述を追加する．

5.1. 領域指定の変更

外部的なデルタ記法においては，R-木内の葉ノードが持つ領域と異なる領域の指定を許すため，データベース内のノードを参照し，データベース内のオブジェクトの持つ座標へと変換する．

この際，以下の点の考慮が必要となる．

- 指定された領域と完全に一致するオブジェクトの上書き
- 完全に上書きされる旧オブジェクトの削除

外部デルタが以下の形式で表されるとき，

add(rect,file(ancestor=rect0,data0))

以下の手順で置き換える：

1. rect0 と完全に領域 rect n が一致する葉ノードが存在する場合，以下の操作に置き換える．
   open(rect0,file(data0))
2. rect0 と完全に領域 rect n が一致するノードが存在しない場合，R-木の挿入アルゴリズムに従い，rect0 を挿入する親ノード p を探索し，3～5 の処理を行う．
3. p の領域 rect p に対し，以下の操作に置き換える．
   add(rect p,file(data0))
   p の下位に存在する葉ノード c を総て取得する．
4. c のうち，その領域 rect c が rect0 に完全に含まれる葉ノード x が存在する場合，すべて x の領域 x に対して以下の操作を追加する．
   delete(rect x)

5.2. R-木構造変更記述の追加

外部的なデルタ記法においては，R-木内の中間ノードを指定しないため，データベース内のノードを参照し，R-木構造に対する変更を追加する．

前のリビジョンの R-木構造と，それに対して変更を加えた結果，生成される R-木構造との差分が内部デルタとなる．図 4 に，オブジェクトを追加した際の内部デルタの生成例を示す．この例では，オブジェクト J の追加によるノード C の領域分割が行われ，ノード H,I が生成がされている．この場合，内部デルタにはノード A'の選択と，その子ノードであった C の削除，A'の子ノード H,I の生成，H,I への E,F,G,J の追加が記述される．

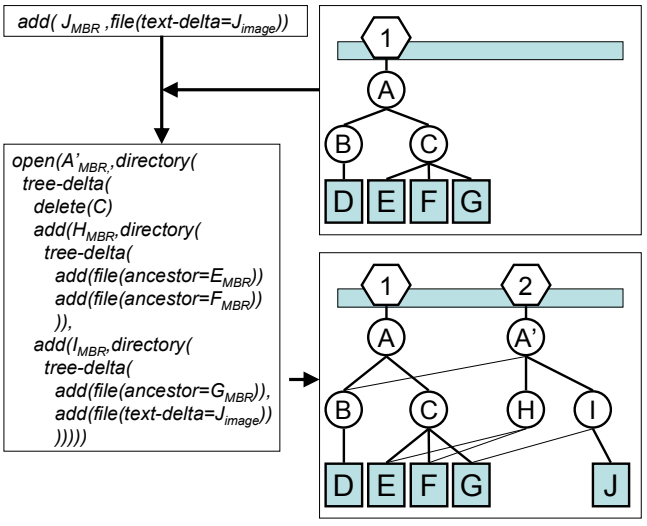

図 4：オブジェクトの追加による内部デルタ

6. おわりに

本稿では，過去のデータ構造に依存せずに地図の部分画像を貼り付ける形で更新するための外部デルタの記述方法と，それを用いて overlapping B-木構造のデータベースで地図の変更情報を管理するためのデータ構造を提案した．

外部デルタにおいてデータ構造と独立した差分記述を行うことで，特定の期間における地図差分を抽出可能であり，分散化された地図情報システムにおける更新情報のやりとりを容易に行うことが可能である．

現段階では，ラスター矩形を貼り付けて地図を更新するため，外部デルタに記載されるのは画像の追加のみであるが，地図情報システムとして使う場合にはこのほかに，建物の名称のような属性情報の添付，建物内部のような異なるレベルの地図への遷移，道路や地下配管図のようにレイヤーの異なる地図への重ね合わせなどを考慮する必要がある．

今後，属性情報，レベル，レイヤーの扱いを検討し，本稿の方式を拡張する．

参　考　文　献

[1] 根岸幸生，青木秀晃，笠原直，郭薇，川崎洋，大沢裕，“時空間管理のための地理情報システム STIMS,” DB/DE 研　夏のワークショップ 2003,pp.195-202 (2003).

[2] F.W. Burton, J.G. Kollias, D.G. Matsakis and V. G. Kollias, “Implementation of Overlapping B-Trees for Time and Space Efficient Representation of Collections of Similar Files,” The Computer Journal, Vol.33,No.3,pp.279–280 (1990).

[3] Collab.Net “Subversion Design, ” http://subversion.tigris.org/design.html .

[4] M.A. Nascimento and J.R.O. Silva, “Towards historical R-trees.” Proceedings of the 1998 ACM symposium on Applied Computing, pp. 235–240 (1998).