階層構造グラフによるデータモデルの適用例：木簡データベース

杉山武司 (姫路獨協大学情報科学センター),
森下淳也，大月一弘 (神戸大学国際文化学部),
上島紳一 (関西大学総合情報学部)

グラフ化されたオブジェクトの複合状態である階層構造グラフを用いて未整理の半構造化データを段階的に構造化していくためのプロトタイプシステムを作成した。実装にはオブジェクトデータベースを用いた。

On a Prototype system for wooden slips researches – an application of Hierarchical Structure Graph.

T.Sugiyama(Information Science Center, Himeji Dokkyo University),
J.Morishita, K.Ohtsuki(Faculty of Cross-Cultural Studies, Kobe University),
S. Ueshima(Faculty of Informatics, Kansai University)

We developed the prototype system implementing Hierarchical Structure Graph for wooden slips researches. Databases such as wooden slips have loosely constraint structure, therefore, they construct as so called semistructured data. In our system, users can manipulate their data in a transparent manner by using hierarchical structure graph.

1　はじめに

近年，文献情報データや環境情報データなどの科学技術データベースが注目されてきている [1, 2]．これらはデータが収集された時点で，データに対して大雑把な構造しか与えられていない場合が多い．言わば半構造化の状態のまま，構築されているという特徴がある．また，確定したスキーマを持つ場合でも，そのデータを利用する段階で，視点の違いや仮説などを盛り込んだ思考実験のような自由度を与えたいと考えられる．

木簡研究支援システムは，中国敦煌遺跡から出土した木簡一千本，中国居延遺跡の木簡一万本を対象にしたシステムである [3]．これらの木簡 (図 1) は，発掘された時点で基本的なデータを収集し，その後，木簡の釋読が進むにつれて文字データが収集される．さらに，釋読文から個々の意味が汲み取られるという過程を経て，データベースに格納される．このようなデータは，

我々はこのような大雑把なスキーマしか持たない半構造化状態のデータに対して，元のデータベースのデータを保持しつつ，利用者の視点に応じて，属性付けを行うことで様々な構造化を同時に表現するデータモデル，階層構造グラフを考案した [6]．

本稿では，複雑な構造を持つデータや異種構造を持つデータ郡をオブジェクトとして収容でき、階層構造グラフを用いて構造化できるようなデータモデルの 1 つの実現であるプロトタイプシステムの紹介を行う。

モデルの重要な要素である**スコープ**とオブジェクトの仮想化についても述べる。

階層構造上の部分グラフを**スコープ**と呼ぶ．利用者はスコープに基づいて仮想オブジェクトを操作することで，他の利用者の操作結果の参照や，複数の利用者間の操作を，独立性を保ちながら段階的構造化を行うことができる．

このデータモデルに基づいて作成した、プロトタイプシステムの特徴について述べる。

2 節では，階層構造グラフを説明し，スコープを 3 節で，実装についてを 4 節で述べる．

図 1. 木簡

2　階層構造グラフ

階層構造グラフは，ノードと枝からなるサイクルのない有向グラフである[4]. 各ノードと枝には属性集合を持つオブジェクトへのリンクが張られている．リンクされるオブジェクトはオブジェクト・アイデンティティと属性集合からなる．次のような形式のものである．

$$O \quad = \quad < Oid, \{a_1 : v_1, \cdots, a_n : v_n\} >$$

ここで，Oid はオブジェクト・アイデンティティであり，a_i は属性，v_i は値である．この属性集合の a_i と v_i はそれぞれ次の様に $< attribute >$ と $< values >$ で定義されるデータである．

$$\begin{aligned} < attribute > \quad &::= \quad symbol, \\ < values > \quad &::= \quad < value > | < value >, < values >, \\ < value > \quad &::= \quad int|string|\cdots|Oid. \end{aligned}$$

オブジェクトは3種類あり，それぞれ基本オブジェクト，カテゴリ，関係オブジェクトと呼ばれる．基本オブジェクトは元々の半構造化状態のデータを格納するもので，グラフの最下位のリーフに置かれる．カテゴリはノードに置かれ，視点を表す．関係オブジェクトは枝に置かれ，視点とデータの間に固有の情報が書き込まれる．図2に階層構造グラフを示す。ノードと枝は，リンクされたオブジェクトの属性集合 $\alpha_i, \beta_j, \rho_{kl}$ を各々持つ（四角で表される）。

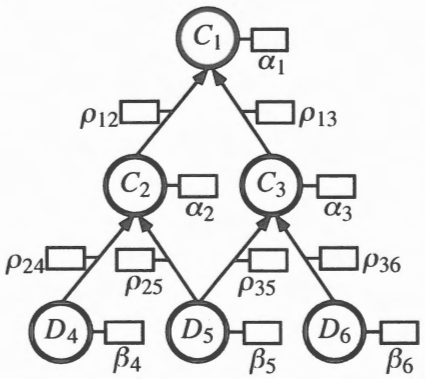

図2. 階層構造グラフ

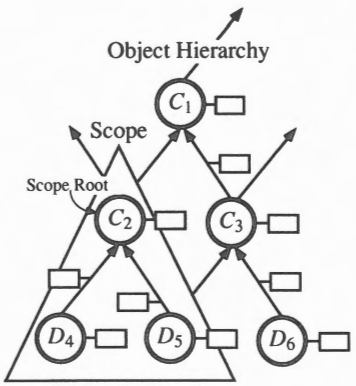

図3. スコープ

仮想オブジェクト

オブジェクトの仮想化は属性集合の演算であるが，利用者はこの仮想化されたものを自分のオブジェクトとして扱いたい．そこで，分類項目のノードから下位に分類されているノードまでの部分有向グラフを**仮想オブジェクト**と定義する．図2のグラフにおいて，C_2 から見た D_4 の仮想オブジェクトを $\tilde{D}_4[C_2]$ と表記する．これに対してノード自身のオブジェクトを実オブジェクトと呼んで区別する．

仮想オブジェクトの属性は，選んだ実オブジェクトの属性に，分類項目のノードからそのオブジェクトに至る総てのパスからの属性を加えて，膨らんだものである．仮想オブジェクトをみることで利用者のオブジェクトに付与した属性があたかも実オブジェクトに付与されたかのように見える．

例えば，図2での C_1 から見た D_5 の仮想オブジェクト，$\tilde{D}_5[C_1]$ ではノード，C_2 と C_3 が間にあるため，その属性は $\alpha_2 + \rho_{25} + \alpha_3 + \rho_{35} + \beta_5$ となる．

3　スコープ

オブジェクト階層上で一つのカテゴリから下位方向へデータオブジェクトまで至るパスの存在するオブジェクト階層の部分階層を**スコープ**と定義する．スコープを決める最上位にあるカテゴリを**スコープ・ルート**と呼ぶ．スコープを指定するためにスコープ・ルートでそれを明記する事もできる．スコープ・ルート C_2 のスコープをスコープ C_2 と呼ぶ．

スコープではスコープ・ルートとはつながらない経路は隠蔽され，スコープ自身が閉じたオブジェクト階層とみなされる．オブジェクト階層，図2とスコープの関係を図3に示す．三角で囲まれたものがスコープである．階層の有向枝である矢印がとぎれているのはスコープ内ではそれが見えないことを表している．

スコープは利用者の視点を反映するものであり，利用者は原則としてこのスコープを決めて，オブジェクト階層の操作を行うものとする．利用者が現在指定しているカテゴリを**カレント・ルート**，対象となる作業範囲を**カレント・スコープ**，カレント・スコープに含まれる全オブジェクトの集合を**対象オブジェクト**という．

スコープはオブジェクトの集合を扱う一つの外延としても機能する．例えば，スコープ W の中から属性，$a_1 = v_1$ であるような基本オブジェクトを検索する場合，

$$\textbf{select * from } W \textbf{ where } a_1 = v_1,$$

という問い合わせをすることで基本オブジェクトの集合が得られるが，この時の W は集合の外延を意味する．

スコープによる利用環境

図4の階層構造グラフにおいて，任意の基本オブジェクトに対して複数個の仮想オブジェクトが定義されるが，これらはグラフ上には，陽には表れない．また，同グラフにおいて，どこまでが分類構造を表し、どこまでが仮想化を表すという区別はなく，利用者の意図や見方によって，分類構造にも仮想化ともみなすことができる．このため，利用者が階層構造グラフを直接参照しながら作業を行うことは困難であると考えられる．そこで，スコープの概念を用いて利用者の利用環境を規定し，操作の簡便性を与える．

仮想オブジェクトを通してみた分類階層は実オブジェクトの分類階層と同一である．スコープの中で仮想オブジェクトを通して新しいカテゴリに分類する場合でも，カテゴリの下位に位置づけられるものは実オブジェクトである．実オブジェクトが分類され，属性継承の働きによって，それがまた，仮想オブジェクトとなる．仮想オブジェクト自身が分類されているのではないことに注意が必要である．

利用者には，各自の欲する仮想オブジェクトと，オブジェクトの分類構造を分離して見せることにする．利用者には，まず，図4に示すような分類階層グラフが与えられる．これは，実オブジェクト間の分類関係を表すもの，即ち，階層構造グラフから属性を取り除いたものである．この分類階層グラフを用いてカレント・スコープを指定すると，そのスコープの対象オブジェクトがカレント・ルートの仮想オブジェクトとして表示されるものとする（図5）．つまり，カレント・スコープを指定すれば，そのスコープ内に分類されている実オブジェクトの数，ならびに，仮想オブジェクトの種類と数がすべて一意に決定する．

図5において，カテゴリも基本オブジェクトと同様に仮想化されており，区別なく表示される．また，実オブジェクトがカレント・カテゴリに対して直接的/間接的に分類されているかどうかは陽には表れないが，間接的に分類されているものに関しては，上位分類に値する仮想オブジェクトの属性はすべて継承される．

カレント・スコープが変更されれば，同一の実オブジェクトに対しても，その仮想オブジェクトは変更される．図6に，カレント・スコープを C_2 に移動させた場合の仮想オブジェクトを示す．

スコープの概念を用いることにより，利用者は，仮想オブジェクトを強く意識しなくてもよいことになる．つまり，分類階層と実オブジェクトのみを意識しておけば，自動的に必要な仮想オブジェクトを利用することができる．

4　実装

グラフ化されたオブジェクトの複合状態である階層構造グラフを用いて未整理の半構造化データを段階的に構造化していくためのプロトタイプシステムを作成した。我々が作成したプロトタイプシステムは、つぎのような機能をもつ。

- 階層構造グラフを構成、編集できる。
- 編集した階層構造グラフを保存できる。
- 階層構造グラフ上を自在にトラバースできる。
- スコープを指定すれば、それらを集約した仮想的なオブジェクトを表示できる。
- 階層構造グラフから、ナビゲーションによって検索ができ、検索結果の部分グラフが得られる。

基本データオブジェクト

階層構造グラフにおけるノードに位置するオブジェクトは、階層構造グラフを構成し、データベースの実質的なデータを保持するもっとも重要なオブジェクトである。これは、つぎの4つのメンバを持つ。

- オブジェクトID
- 上位オブジェクトのリスト
- 下位オブジェクトのリスト

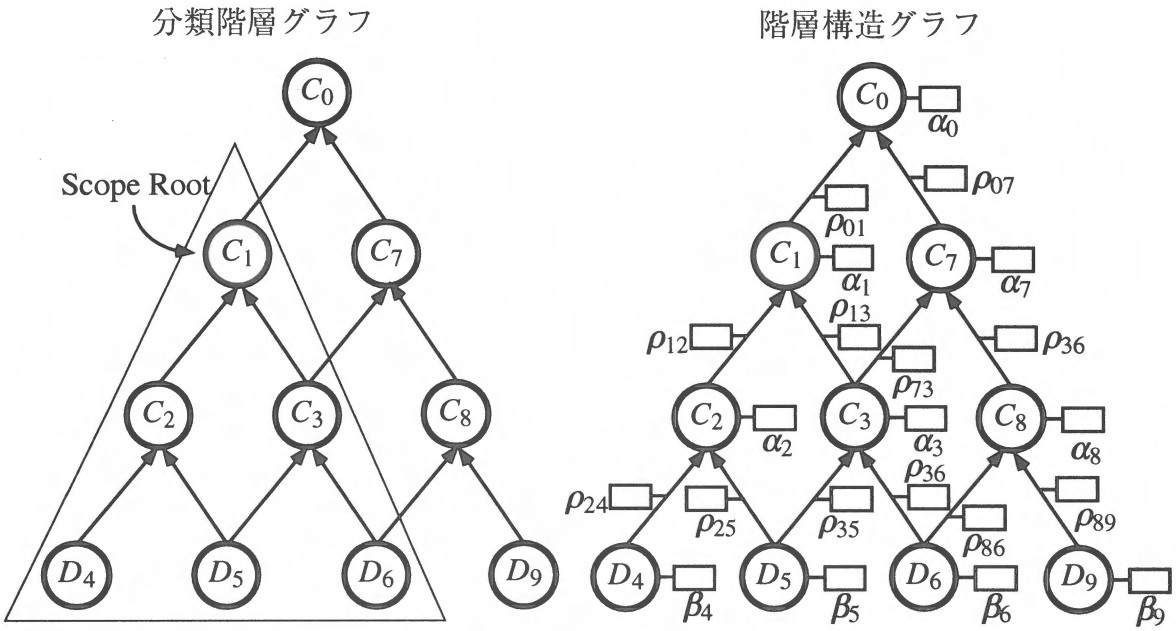

図 4. 分類階層グラフ

- 属性、属性値のハッシュテーブル

オブジェクト ID は、オブジェクトを識別するものでシステムの中でユニークである。上位、下位オブジェクトのリストは階層構造グラフのエッジを構成するためのもので、セッションのなかで、リストの要素が動的に変化したりリストが伸縮したりする。このことによって、階層構造グラフの形状が変化する。属性、属性値のハッシュテーブルは、このオブジェクトの内容を決定するもので、検索の対象であり、セッション中に動的に変化する。

永続化オブジェクト

データベースエンジン部の記述には JAVA 言語を採用した。データベースには ObjectDesign 社の ObjectStore PSE Pro を用いた。これは JAVA 言語から透過的にアクセスできる ODBMS である。ODBMS を使う理由は、トランザクション処理の機能が備わっていることとオブジェクトの永続化ができる点にある。

我々のシステムでは、セッション中に、あらゆるオブジェクトが絶えず生成あるいは消滅し、セッションが終わると 1 次記憶の中からは消える。つぎのセッションで以前のセッションの最終状態を 1 次記憶の中に復

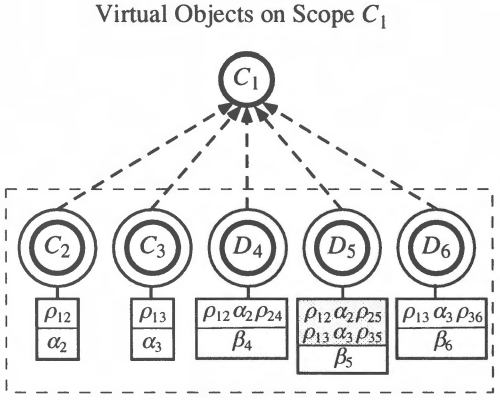

図 5. C_1 による仮想オブジェクト

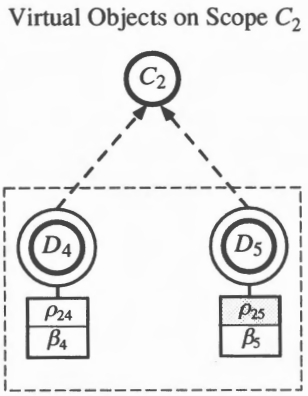

図 6. C_2 による仮想オブジェクト

元できるようにしなければならないが、これを保証するのが永続化オブジェクトである。たとえば、オブジェクト anObject をデータベースに登録 (永続化) するにはつぎのようにする。

```
Database db = Database.create(dbName,
    ObjectStore.ALL_READ |
        ObjectStore.ALL_WRITE);
// Start a transaction
Transaction tr =
   Transaction.begin(ObjectStore.UPDATE);
db.createRoot("theRoot", anObject);
tr.commit();
```

登録されたオブジェクトを復元するには

```
// Start a read-only transaction
Transaction tr =
 Transaction.begin(ObjectStore.READONLY);
Object b = (Object) db.getRoot("theRoot");
b.print();
tr.commit();
```

などとする。復元されたオブジェクトが他のオブジェクトをメンバとして包含している場合、そのメンバオブジェクトも同時に復元されるので、ナビゲーションによってアクセスするようなオブジェクトについては、まったく入出力について考慮する必要がないと言える。あるオブジェクトが他のオブジェクトを内包する形で、いもづる式に繋がっている場合、根元のオブジェクトを 1 次記憶に復元したからといって、それに繋がるすべてのオブジェクトが 1 次記憶にロードされるわけではなく、アクセス可能な状態 (hollow な状態) に置かれるだけである。実際にそのオブジェクトにアクセス (touch) したときに、1 次記憶にロードされ、活性化 (active な状態) される。また、どんなオブジェクトでも永続化できるわけではなく、Vector や Hashtable など、あらかじめ大きさが不定の collection 型のオブジェクトは永続化できない。その場合は、Vector の代わりに OSVector など ODBMS で提供されている collection を使う必要がある。われわれが階層構造グラフのノードを実装するのに使ったオブジェクトのクラス定義はつぎのようなものである。

```
public class BasicObject {
   // Constants
   public static
     final String SET = "BasicObjectSet";
   public static
     final String MAP = "BasicObjectMap";
   // Private Members
   private  int        id_;
   private  OSVectorList    supers_;
   private  OSVectorList    subs_;
   private  OSHashMap       attrs_;
   .......
 }
```

メンバ id_は、ノードになる BasicObject を識別するため、また、グラフ全体を管理するときに使うための識別子である。メンバ supers_、メンバ subs_はそれぞれ上位ノード、下位ノードとなるオブジェクトを保持するためのリストである。メンバ attrs_は属性、属性値を保持するための連想リストである。

検索機構

このシステムの検索機構には 2 つある。1 つはナビゲーションを用いるものと、もう 1 つは ODBMS に備わっているトラバースを用いる方法である。ここではナビゲーション検索のみを述べる。もっとも単純な検

索ルーチンはつぎのようなものである。これは、データオブジェクト内の連想リストを探索して、その有無を返す。

```
public boolean
simplefind(String k, String v) {
   Enumeration enum = getAttrs().keys();
   while (enum.hasMoreElements()) {
     String key = (String)enum.nextElement();
     String val = (String)getAttrs().get(key);
     if (k.equals(key) && v.equals(val))
          return true;
   }
   return false;
}
```

この検索ルーチンはオブジェクト内のトラバースによる検索で、平均の検索速度は連想リストの要素数によって変わるが、連想リストの要素数はこのシステムの場合高々数10から数100のオーダーであるのでほとんど問題にならない。

つぎの検索ルーチン findSubs はデータオブジェクトの下位につながるすべてのサブオブジェクトをナビゲーションによって走査して、その属性、属性値の有無を調べ、それを持つオブジェクトのリストを返すものである。これは自分自身を呼び出す再帰関数になっているが、われわれの想定では、上位、下位の段数が著しく増えることはオブジェクト同士の意味の関連の深さから考えられない。

```
public FList findSubs(String k, String v) {
    FList result = new FList();
    OSVectorList l = getSubs();
    for (int i = 0; i < l.size(); i++) {
       BasicObject x = (BasicObject)l.get(i);
       if (x.simplefind(k, v)) {
          result.addElement(x);
       }
       result.append(x.findSubs(k, v));
    }
   return result;
}
```

ここに、FList は OSVectorList クラスに append 関数その他を追加するために、派生クラスとしたものである。

```
public class FList extends OSVectorList {
   public FList append(OSVectorList a) {
     Enumeration enum = a.elements();
     while (enum.hasMoreElements()) {
       this.addElement((Object)enum.nextElement());
     }
     return this;
   }
   ..........
}
```

上位につながるオブジェクトの検索にも同様の方法を用いた。これによって、検索結果を表す新しいグラフオブジェクトの作成が可能になった。

仮想オブジェクト

あるオブジェクトの下位につながる1つのオブジェクトへのすべてのパスを集約して仮想的に1つのオブジェクトにみせる機構を、スコープの頂点となるオブジェクトが包含する下位のサブオブジェクトの連想リストを走査して、それらを全て含むようなハッシュを持つあたらしいオブジェクトを一時的に生成することによってに実現した。

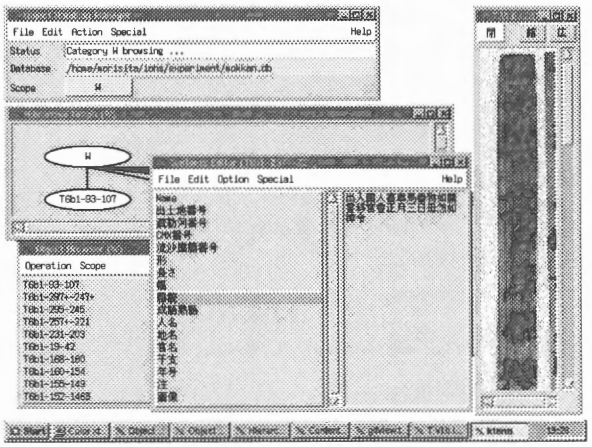

図7. プロトタイプシステム

GUIの実装

すでに稼働しているExpectとTcl/Tkで記述されたフロントエンドをそのまま利用した。

5 終わりに

階層構造グラフモデルにおけるスコープによる視点操作について述べた。階層構造グラフモデルに基づく木簡研究支援のためのプロトタイプシステムを作成した。このシステムはデータベースエンジン部とユーザインターフェイス部から成る。データベースエンジン部にODBMSを採用したため以前のLispによるプロトタイプシステムに比べて大量データの場合の安定性およびパフォーマンスが著しく向上した。

参考文献

[1] IEEE Computer Society, "Special Issue on Scientific Databases," Bulletin of the Technical Committee on Data Engineering, Vol.16, No.1, Mar. 1993.

[2] Zdonik, S., "Incremental Database Systems: Databases from the Ground Up," Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, Washington DC, USA, pp.408–412, May 1993.

[3] Ueshima, S., Ohtsuki, K., Morishita, J., Qian, Q., Oiso, H. and Tanaka, K., "Incremental Data Organization for Ancient Document Databases,", Proc. of the Fourth International Conference on Database Systems for Advanced Applications(DASFAA'95), pp.457–466, Singapore, Apr. 1995.

[4] 森下淳也, 上島紳一, 大月一弘, 杉山武司, "階層構造グラフを用いた半構造化データの段階的構造化手法に関する検討," 情処研報, Vol.97, No.7, DBS96–111,pp.9–16, Jan.1997.

[5] 森下淳也, 上島紳一, 大月一弘, 杉山武司, "階層構造グラフにおける属性の取り扱い方に関する検討," 信学技報, Vol.96, No.469, DE96–79, pp.31–36, Jan.1997.

[6] 上島紳一, 森下淳也, 大月一弘, 杉山武司, "階層構造グラフを用いた半構造化データの構造化手法," 信学技報, Vol.96, No.469, DE96–79, pp.31–36, Jan.1997.

[7] "ObjectStore PSE/PSE Pro for Java API User Guide," ObjectDesign,Inc.,1998.