集合指向言語SOLのマルチメディアデータ型と手続きに関する拡張とその評価

横尾徳保† 重松保弘†

近年，マルチメディアデータを含んだデータの構造化，およびその表示や検索などをプログラムとして実現するマルチメディアプログラミングの要求が高まっている．しかし，Visual C++やDelphiのような高度なプログラム開発環境においてもAPIに関する専門知識が必要であり，そのプログラミングは容易でない．そこで，マルチメディアプログラミングの簡便なツールを目指して，集合指向言語SOLのデータ型と手続きに関する拡張を試みた．SOLはアルゴリズムの自然なプログラム化を目的として，筆者らが設計および開発を行ってきたプログラミング言語である．このSOLにおいて，静止画像型，動画像型および音声型を基本型として導入し，その集合や写像/対応，代入や関係演算，集合演算を規定することで，マルチメディアデータの構造化が容易に行えるようになった．また，マルチメディアデータに対しても既存の基本型データと同じ形式で不定型変数や標準出力を利用できるため，マルチメディアデータの表示や検索なども容易に記述できる．本論文では，拡張したSOLの言語仕様ならびに処理系の実装について述べる．また，スキーマフリーの簡易データ管理アプリケーションを例として，SOLとDelphiのプログラム記述を比較し，その記述量からSOLの効率性を評価する．さらに，同じ例題に対してJavaを用いたプログラム記述例を示し，手続き型言語の枠にとらわれない比較評価を行う．

A Multimedia Data Type and Procedure Extension of the Set Oriented Language SOL and Its Evaluation

Noriyasu Yokoo† and Yasuhiro Shigematsu†

A desire of multimedia-programming, that treats multimedia-data as simple objects in programming, is increasing in recent years. However multimedia-programming is not easy, because knowledge of technical details on Application Program Interfaces (API) is required. This requirement prevents users to make the programs even if any Integrated Development Environments (IDE), such as Visual C++ and Delphi and so on, are provided. Thus we made a multimedia data type and procedure extension of the set oriented language SOL, which we have designed and developed. Concretely, new data types which were *image type*, *movie type* and *audio type* were introduced as basic types. Several operations concerned with these types, such as set, mapping/correspondence, assignment and so on, were also introduced and implemented, and their usages are similar to other basic types. This paper presents the specification of the extended SOL and its implementation. We evaluate the efficiency of the extended SOL from the number of lines of the program, comparing a SOL program of a data management example which includes a schema-free data structure with a Delphi one. Futhermore we also compare with Java over the framework of the procedural language.

1. はじめに

近年，パーソナルコンピュータ上でマルチメディアデータを容易に扱える環境が整ってきており，マルチメディアデータを含んだデータの構造化およびその表示や検索などをプログラムとして実現するマルチメディアプログラミングの要求が高まっている．たとえば，マルチメディアデータを含む多様なデータからHTML形式のファイルを自動生成するためのデータの構造化[1)]があげられる．しかし，マルチメディアプログラミングにおいては，API（Application Program Interface）に関する専門的な知識が必要であり，またプログラム記述においても初期化や再描画，マルチスレッド化など，プログラム作成者が直接意図しない処理の記述が必要になる．このため，プログラミングを専門としないプログラム作成者にとって，マルチメディアプログラミングは容易でない．仮にVisualC ++[2)]

† 九州工業大学大学院工学研究科
Graduate School of Engineering, Kyushu Institute of Technology

や Delphi[3] のような高度なプログラム統合開発環境が利用できたとしても，マルチメディアプログラミングにおけるこのような問題はほとんど解決されない．

そこで，マルチメディアプログラミングの簡便なツールを目指して，集合指向言語 SOL（Set Oriented Language）のデータ型と手続きの拡張を試みる．SOL はアルゴリズムの自然なプログラム化を目的として設計されたプログラミング言語である[4)~7)]．SOL と同じ集合指向の言語としては SETL[8] や ISETL[9] が知られている．SOL は構文規則に集合，写像/対応，述語論理の概念を取り入れたプログラミング言語であり，プログラム実行中に写像/対応を動的に生成することができるなどの特徴を持つ．

このような特徴を持つ SOL において，静止画像型，動画像型および音声型（以降，これらの型をまとめてマルチメディア型と呼ぶ）を基本型として導入し，マルチメディア型データの集合や写像/対応，代入や関係演算，集合演算を規定したことで，マルチメディアデータを含むデータの構造化が容易に実現できるようになった．また，マウスの範囲指定による静止画像データの標準入力，実行時に変数の型が決定される不定型におけるマルチメディアデータの利用を実現し，さらにウインドウ管理に必須であるマルチスレッド化や初期化，再描画の処理を言語処理系で行うようにしたことで，プログラム作成者は API を意識することなくマルチメディアデータの表示や検索などの操作を容易に記述できるようになった．また，容量が大きなマルチメディアデータが利用できるようになることから，写像/対応のファイル入出力形式を改良した．

本論文では，拡張した SOL の言語仕様ならびに言語処理系の実装について述べるとともに，新たに作成した標準関数および標準手続きを紹介する．また，スキーマフリーの簡易データ管理アプリケーションを例として，SOL と同じ Pascal ベースのプログラミング言語である Delphi とのプログラム記述を比較することで，SOL の記述の効率性を評価する．さらに，同じ例題に対して Java を用いたプログラム記述例を示し，手続き型言語の枠にとらわれない比較評価を行う．最後に，まとめと今後の課題について述べる．

2. 言語仕様の拡張

2.1 基本型の追加

マルチメディアデータを扱うデータ型として，静止画像型（型名：image），動画像型（型名：movie）および音声型（型名：audio）を導入した．また，マルチメディア型は基本型であり，集合や組の要素，写像や対応に利用することが可能である．このためマルチメディアデータに対しても，簡便にデータ間に関連付けを行うことができる．変数宣言はそれぞれ次のように行う（以降，静止画像型を中心に説明する）．

（宣言例） **var** *image_data* : image;
（宣言例） **var** *movie_data* : movie;
（宣言例） **var** *audio_data* : audio;

表 1 関係演算とデータ型
Table 1 Relational operators and operand types.

演算子	意味	演算結果の型	データの型の組合せ
＝	等しい	論理型	マルチメディア型とマルチメディア型
≠	等しくない	論理型	マルチメディア型とマルチメディア型
∈	要素である	論理型	マルチメディア型と集合型
⊂	含む	論理型	集合型と集合型

表 2 集合演算とデータ型
Table 2 Set operators and operand types.

演算子	操作	演算結果の型	非演算部の型	演算部の型
∩	共通集合	集合型	集合型	集合型
∪	和集合	集合型	集合型	集合型
−	差集合	集合型	集合型	集合型

2.2 入 出 力

マルチメディア型データの入出力はファイル入出力および標準出力を規定しており，他の基本型データと同様に標準手続き read()，write() を用いる．マルチメディア型データの標準出力ではデータの種類に対応したウインドウが生成され，そのウインドウ上にデータが表示（再生）される．静止画像型データに関しては，オプション：[*width*,*height*] によるサイズ指定表示を可能にした．構文規則[6] の〈文〉の定義に

[〈手続き名〉[(〈式〉[：〈組型〉]*)]]

を追加しており，次のように記述する．

（記述例） write(*image_data*);
（記述例） write(*image_data*：[100,200]);

静止画像型データに関しては，マウスの範囲指定による標準入力を新たに規定した．また，マルチメディア型データを含む集合，写像/対応も標準手続き read()，write() を用いて簡単に入出力を行うことができる．

2.3 演 算

マルチメディア型データに対して規定した関係演算および集合演算をそれぞれ**表 1**，**表 2** に示す．これらの基本的な演算を実装することで，マルチメディアデータをキーとしたプログラミングを容易に行える．

表 3 標準関数
Table 3 Standard functions.

関数名	戻り値の型	引数の型	説明
getwinnum	整数型 (*bool*)	–	ウインドウ番号の取得
readclick_image	論理型 (*bool*)	静止画像型 (*img*)	クリックイベントの検出
getname	文字列型 (*name*)	マルチメディア型 (*MD*)	ファイル名の取得

表 4 標準手続き
Table 4 Standard procedures.

手続き名	引数の型	説明
settitle	整数型 (num), 文字列型 (name)	ウインドウタイトルの設定
delwindow	整数型 (num)	ウインドウの消去
setimage	整数型 (num), 静止画像型 (img)	表示されている静止画像の変更
resizewindow	整数型 (num), 整数型 (x,y)	ウインドウサイズの変更
movewindow	整数型 (num), 整数型 (x,y)	ウインドウの移動
setwinmax	整数型 (num)	ウインドウ最大化
setwinmin	整数型 (num)	ウインドウ最小化
setwinback	整数型 (num)	最小化されたウインドウの復元
setfgwindow	整数型 (num)	ウインドウを手前に表示

なお，マルチメディア型データの演算はファイル名に対して処理を行うものである．

2.4 標準手続きと標準関数

マルチメディア型の導入にともない新しく標準手続きと標準関数を導入した．SOL では，マルチメディアデータそのものを 1 つのデータとしてとらえており，マルチメディアデータの作成や加工などは想定していない．実装を完了している標準関数および標準手続きをそれぞれ**表 3**，**表 4** に示す．*num* はウインドウの番号，*bool* はイベントの有無，*MD* は対象を表すマルチメディア型データ，*img* は対象となる静止画像型データ，(x, y) は座標またはサイズ，*name* はファイル名またはタイトル名を表す．タイトル名はマルチメディアデータ表示時に生成されるウインドウのタイトルバーに表示される文字列を表す．

3. 内部処理系

SOL は翻訳系が S コードと呼ばれる中間コード[6] を生成し，通訳系が S コードの実行するコンパイラ・インタプリタ型のプログラミング言語である．従来の言語処理系[7] は OS に依存することはなかったが，マルチメディア拡張にともない，想定される利用環境と開発負荷を考慮し，処理系を Windows アプリケーションとして開発したことから，現在の処理系は OS に依存する（Microsoft 社 Visual C++6.0 を使用）．なお，静止画像型，動画像型，音声型はそれぞれ BMP，AVI，WAV 形式のファイルへの対応が完了している．

3.1 テーブル管理

マルチメディア型データの管理は，変数の型に応じて静止画像テーブル，動画像テーブル，音声テーブルを用いて行う．マルチメディア型データはそれぞれの型で管理する情報が異なっており，そのためデータの型ごとにテーブルを用意した．read() 手続きで読み込まれたデータへのポインタをこれらのテーブルに格納し，同時に管理上必要な情報を保持する．また，ディスプレイ上に表示するウインドウはウインドウテーブルによって管理する．データとウインドウは 1 対 1 に対応するものではなく，またウインドウごとにリソースなどを管理する必要があることから，データの管理テーブルとウインドウの管理テーブルは独立させた．静止画像テーブル，静止画像ウインドウテーブルはそれぞれ次のような情報を格納する．

静止画像テーブル

- ファイル名フィールド：静止画像のファイル名を格納する．静止画像の表示および集合演算，関係演算を行う場合にこのフィールドのファイル名を使用する．標準入力より入力された静止画像に対しては，入力時にユニークな名前を処理系が割り当て，その名前をこのフィールドに格納する．
- タイプフィールド：静止画像のフォーマットを格納する．ファイル形式の区別に使用する．
- リソースチェックフィールド：テーブルが指している静止画像のリソースを保持しておくべきかどうかの判定に使用する．画像テーブルの指す静止画像が利用（write()）されるたびにインクリメントされ，その画像が不要（delwindow() など）になるたびにデクリメントされる．この値が 0 になるとリソースの開放を行う．
- サイズフィールド：入力された静止画像のデフォルトの表示サイズを保持する．
- 静止画像ハンドルフィールド：静止画像をロードした後，そのハンドルを保持する．このフィールドを用いて描画やリソースの開放を行う．

静止画像ウインドウテーブルの宣言

- ウインドウハンドルフィールド：表示されるウインドウと 1 対 1 に対応するウインドウハンドルを保持する．このハンドルを用いてウインドウを特定し，指定された操作を行う．
- ポインタフィールド：静止画像テーブルへのポイ

ンタを格納し，表示する静止画像を特定する．対象が静止画像集合の場合には，その要素数に応じてポインタを保持する．

- 集合フィールド：ウインドウに表示する静止画像が集合であるかどうかを表す．
- 要素数フィールド：ウインドウに表示されている静止画像の数を保持する．通常は 1 である．集合を対象としている場合にその要素数を格納する．
- ウインドウサイズフィールド：表示サイズを保持する．このフィールドの値を変更することで静止画像の動的な表示サイズの変更ができる．
- ディスプレイフィールド：ウインドウに静止画像が表示されれば真，そうでなければ偽になる．メインスレッド中でウインドウにアクセスする場合にこのフィールドの値を用いて同期をとる．このフィールドが真になるまで，ウインドウにアクセスすることはできない．
- スレッドフィールド：ウインドウ生成時，プライオリティを最高位に設定しウインドウ作成を高速化する．また，スレッドが正常に終了したかどうか調べる場合に使用する．

3.2 実行時スタックとテーブルの関係

マルチメディア型データは，実行時スタックと前述のテーブルを用いて管理する．静止画像を管理する場合の実行時スタックとテーブルの関係を**図 1** に示す．実行時スタックの値欄にはその構造上マルチメディア型データそのものを格納することはできない．そのため，図 1 中 (a) のように対応するテーブルへのポインタを格納することになる．これにより実行時スタックから変数に対応する静止画像を表すテーブルを特定することができる．また，ウインドウに対する操作は図 1 中 (b) のように直接静止画像ウインドウテーブルの ID を指定することで，対象となるウインドウを特定する．このとき，静止画像ウインドウテーブルには，そのウインドウに表示されている静止画像に対応する静止画像テーブルへのポインタを保持しているので，ウインドウ上の画像に関する情報を取得できる．

3.3 集合型，組型データの標準出力

マルチメディア型は基本型であるので集合や組の要素となりうる．これまで集合型のデータの標準出力は {*element1*, *element2*···*elementN*} の形式であった．これに対して，マルチメディア型データの集合の標準出力は要素そのものではなくそのファイル名を使用する．ただし，{*filename1*, *filename2*···*filenameN*} の表示に加えて，新しくウインドウを生成しそのウインドウ上にマルチメディアデータの表示を行う．静止画像データの集合は，静止画像の集合からなる画像を選択するような用途を考慮し，集合における静止画像の表示は 90 × 90 ピクセルで言語処理系のプロトタイプ実装を行った．静止画像型データの標準出力例を**図 2** に示す．

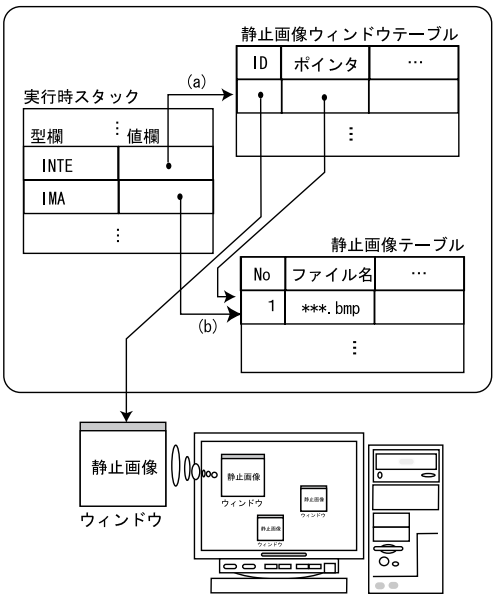

図 1 実行時スタックとテーブルの関係
Fig. 1 Relation between stack and tables.

図 2 静止画像型データの標準出力例
Fig. 2 Standard output example of images.

4. 通訳系の実装

4.1 静止画像型データの描画

静止画像の描画は，(1) 静止画像ウインドウテーブルの設定，(2) ウインドウの生成，(3) 生成したウインドウへの画像の描画の 3 つのステップで実現される．(1) 静止画像テーブルの設定では，まず対象とする静止画像が集合であれば静止画像ウインドウテーブルの集合フィールドに真，そうでなければ偽を格納する．次に，対象とする静止画像が集合である場合はその要素数からウインドウサイズを決定して，サイズフィールドに設定し，単一の静止画像の場合はその画像のデフォルトのサイズを設定する．それから，静止画像

```
CreateWindow( // オーバーラップウィンドウ，ポップアップウィンドウ，
              // 子ウィンドウを作成する．ウィンドウクラス，ウィン
              // ドウタイトル，ウィンドウの初期位置などを指定．
  "image",            // ウインドウクラス名へのポインタ
  window_name,        // ウインドウ名へのポインタ
  WS_OVERLAPPEDWINDOW,// ウインドウスタイル（枠，最大化ボタン，
                      // 最小化ボタン，タイトル，タイトルバー，
                      // サイズ変更境界を指定）
  CW_USEDEFAULT,      // X座標の指定（デフォルト値を使用）．
  CW_USEDEFAULT,      // Y座標の指定（デフォルト値を使用）．
  iwtab[*num].x,      // 高さを指定．
  iwtab[*num].y,      // 幅を指定．
  NULL,           // 親ウインドウ，オーナーウインドウの指定．
  NULL,           // 子ウインドウ，コントロールウインドウの指定．
  GetModuleHandle(NULL),
                  // ウィンドウに関連付けられたモジュールの指定．
  NULL            // インスタンスハンドルを指定．
);
```

図 3 CreateWindow() 関数呼び出し
Fig. 3 Call of CreateWindow().

テーブルへのポインタの設定，ウインドウ番号の割り当てを行う．

（2）ウインドウの生成としては，まず Window クラスの定義および登録を行う．Window クラスを定義することで，利用できるウインドウの形状と機能が決定される．その後，ウインドウを生成する関数を呼び出す．このときの設定で表示されるウインドウの詳細な機能が決定される．この設定は CreateWindow() 関数で行い**図 3** のように設定している．またウインドウ生成後，OS（Windows）から送られてくるメッセージ（WM_PAINT，WM_CLOSE など）に対応した処理を行うためにメッセージループを作成する．メッセージループでは送られてきたメッセージをメッセージキューに格納し，メッセージの読み取りおよび必要な処理内容を OS に返す．OS はそのメッセージをウインドウ関数に送信し，メッセージに対応した処理が行われる．ウインドウはこれらのイベントを不定期に取得する必要があり，そのためウインドウごとにスレッドを生成し管理している．

（3）画像の描画は，まず静止画像データをウインドウに表示できるようにウインドウのデバイスコンテキストを取得することから始まる．次に，画像がウインドウに描画されるまでの間，データを保存するためにメモリデバイスコンテキストを取得する．これは画像表示前のイメージ構築に使用する．次に，read() でロードしていた画像データをメモリデバイスコンテキストに設定する．最後にデバイスコンテキスト間でコピーを行うことで画像の表示を行う．対象とする画像が集合の場合には，集合の要素ごとに表示位置を決定し，サイズを変更して表示する．また，ウインドウシステムではウインドウの重なりにより表示に欠損が生じる場合があり，OS からメッセージに応じた再描画処理を行う．

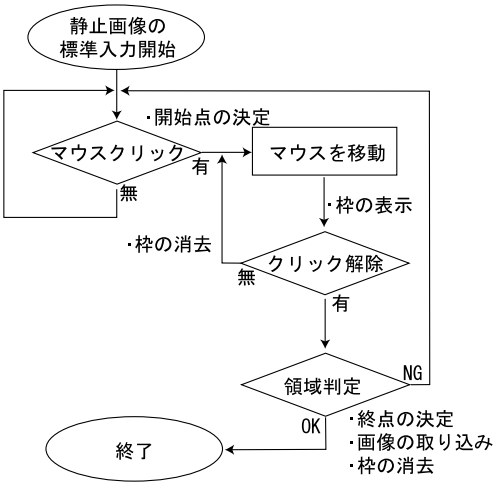

図 4 静止画像型データの標準入力手順
Fig. 4 Procedure of image type's data input.

4.2 静止画像型データの標準入力

マウスの範囲指定による静止画像型データの標準入力を**図 4** のように実現している．静止画像の標準入力は write() 手続きを呼び出すだけの簡素な記述で実現できる．また，入力操作も特殊なキーを使用することなく，マウスで範囲指定した領域を画像として読み込むことができる．このとき，開始点の指定ミスによる入力操作のやり直しや入力画像の選択（アイコン状態にあるウインドウの表示など）にマウスを利用できるようにするため，指定された範囲が 50×50 ピクセル以下のときにはそれを入力とは見なさず，開始点の指定からやり直すように実装した．また，SOL では変数に値が代入された時点で型が決定される不定型を実装しており，不定型変数でも静止画像型の標準入力が行えるように拡張した．

4.3 写像/対応のファイル入出力

SOL では写像/対応のファイル入出力を行うことができる．写像/対応は処理系内部ではネットワーク形式で構築される．このため従来のファイル入出力においては，写像/対応を始点データと終点データの組の集合として取り扱い，次の順次形式のデータ列に変換していた．

- 写像 → `{[始点 1, 終点 1],[始点 2, 終点 2],…}`
- 対応 → `{[始点 1,{終点 1a,…}],…}`

しかし，この方式では同一データが写像/対応内に複数回現れた場合，それらのコピーが複数個存在することになる．テキストデータのみを扱う場合には問題にはならないが，容量が大きいマルチメディアデータを扱うにあたっては改善する必要がある．そこで，写像/対応のファイル入出力においては，写像/対応の情

報を構成要素のデータとネットワーク構造情報に分離し，変換形式を次の順次形式に改良した．

番号　　写像の構成要素のデータ
：　　：　　（構成要素データ部）
{[始点番号，終点番号]，···}　（ネットワーク構造情報部）

この方式ではネットワーク構造情報部に同一のデータが複数個存在することはない．また，ネットワーク構造情報は番号（整数値）で示されているため，冗長性によるファイルサイズに関する問題は解消されたといえる．従来方式に対してネットワーク構造情報表の作成処理が追加されるが，従来方式も新方式も出力するデータ数に対して同じオーダの処理であり，処理時間の増加はたかだか定数項で表される．むしろ無駄なデータの出力処理が減るために画像データや音声データなどが写像/対応に含まれている場合は，ファイル出力に要する処理時間は短くなると考えられる．

5. プログラム記述能力の評価

現実社会の情報は実体や概念，状態などを表現する節点と，節点間の関係を表す枝から構成されるグラフ表現を用いて表すことができる．このような情報は，スキーマがあらかじめ限定されていれば，リレーショナルデータベースに代表されるように表形式でデータを管理することができるが，そうでない場合には表形式によるデータの管理は容易でない．特に，新しいスキーマの定義を許すような場合，表形式でデータを管理するのは現実的ではない．最近，半構造データ[10),11)]と呼ばれるデータが注目されているが，このようなデータも一種の半構造データの例と考えられる．SOL は集合や写像/対応などの機構を備えているため半構造データのようなデータ構造を効率的に扱うことができると考えられる．

そこで，マルチメディアデータを含むデータを構造化し，新しくスキーマを定義できるようなスキーマフリーの簡易データ管理アプリケーションを例題として考える．この例題に対して SOL と Delphi のプログラムを比較することで，SOL におけるプログラムの可読性と記述の効率性について評価を行う．ただし，プログラムの可読性および記述性の評価を行うことから，比較対象とした Delphi においても手続き型でプログラム作成する．

なお，以降の説明では A→B の関係があるときこの関係に着目して A を主体，B を対象，矢印（→）を関係と呼ぶことにする．

図 5 の初期構造を持つデータに対して，[主体，関係，対象] 中の必要な項目を指定することで表示，検索，追加，削除，およびファイル入出力を実現する．ただし，SOL における記述の効率性を検証するために，Delphi におけるプログラミングは記述を短くすることを前提に分かりやすさを重視して作成することとし，Delphi における記述の簡素化のために次の制限を設けるものとする．

(1)　主体および対象は静止画像または文字列
(2)　関係は文字列
(3)　主体は最大 25，主体あたりの対象数は最大 10

なお，SOL の例題プログラムにおいては，不定型変数を用いてデータを格納しており，制限 (1) および制限 (2) がない場合でも，プログラムの記述はほとんど変わらない．また，SOL の例題プログラムでは，主体と対象を集合として取り扱い，その関係は処理系が提供するデータ構造である対応を利用していることから制限 (3) は問題にならない．

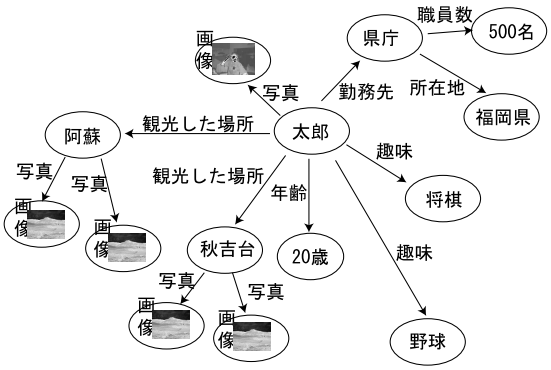

図 5　例題におけるデータの初期状態
Fig. 5　Initial state of data in example.

```
var menu : integer;
    S : setof anytype;
    LABEL : setof string;
  map f{LABEL} : S → S;
```

(a) SOL における宣言

```
type
  info=record
    sub:string;
    simg:TBitmap;
    styp:types;
    rel:array[1..10]of string;
    obj:array[1..10]of string;
    oimg:array[1..10]of TBitmap;
    otyp:array[1..10]of types;
    flag:array[1..10]of bool;
  end;

var
  data:array[1..25] of info;
  table:array[1..25] of bool;
```

(b) Delphiにおける宣言

図 6　変数宣言
Fig. 6　Declaration of variables.

5.1 変数の宣言

SOL においては不定型およびラベル付き対応の概念を利用するだけで，ラベル付き写像/対応における元像，ラベル，像をそれぞれ主体，関係，対象に対応させてデータの構造化を行うことができる．宣言は**図 6** (a)

```
        ⋮
open(fp,"aki1.bmp",input);
read(fp,image_data1);
close(fp);
open(fp,"aki2",input);
read(fp,image_data2);
close(fp);
defmap f{"写真"}*("秋吉台") =
  {image_data1,image_data2};
        ⋮
```

(a) SOLにおける初期化

```
        ⋮
table[3]:=true;
data[3].sub:='秋吉台';
data[3].styp := str_data;
data[3].rel[1]:='写真';
data[3].otyp[1]:=img_data;
data[3].oimg[1]:=TBitmap.Create;
data[3].oimg[1].LoadFromFile('aki1.bmp');
data[3].obj[1]:='yokoo.bmp';
data[3].flag[1]:=true;
data[3].rel[2]:='写真';
data[3].otyp[2]:=img_data;
data[3].oimg[2]:=TBitmap.Create;
data[3].oimg[2].LoadFromFile('aki2.bmp');
data[3].obj[2]:='1.bmp';
data[3].flag[2]:=true;
        ⋮
```

(b) Delphiにおける初期化

図 7 変数の初期化
Fig. 7 Initialization of variables.

のように記述するだけである．主体と対象を不定型のデータの集合 S，それらの関係を対応 f とした．このとき，それぞれの関係名を文字列の集合 $LABEL$ で表す．静止画像型は基本型であり，不定型や集合の取扱いにおいて制約がかからない．

一方，Delphi においても Variant 型や集合型が存在するが，静止画像を扱う場合には利用が困難である．そこで図 6 (b) に示すレコード型を使用して，データの構造化を行った．この info レコードの配列 *data* を主体の最大数用意し，新しい主体が登録されるごとにこれらの配列に格納する．関係や対象は主体ごとに保持する．また，配列 *table* で配列 *data* の主体格納状況を管理する．info レコード メンバ *flag* は関係の有無を保持する．

5.2 変数の初期化

初期化の記述を**図 7** に示す．SOL では静止画像の取扱いに必要な処理は処理系で行うために，特別な記述をする必要はない．一方，Delphi においては静止画像データを利用するには初期化のための記述が必要である．SOL では初期化の代わりにファイルのオープン/クローズの記述があるため記述量だけをみれば大きな違いはないが，ファイルのオープン/クローズは一般的な記述であり，マルチメディアデータの利用にともない新しい知識を必要としているわけではなく，作業効率は向上すると考えられる．

5.3 表示，検索，追加

表示/検索および追加を行う関数をそれぞれ**図 8**，**図 9** に示す．SOL における不定型変数の表示は格納されているデータの型にかかわらず write() 手続きを呼び出すだけであり，マルチメディアデータを含む集合や写像/対応も同様に write() 手続きで表示できるため，表示手続きのプログラム記述は非常に簡潔である．また，検索に関してもデータの型を気にする必要はない．Delphi においては，すべてのレコードに対して主体および対象のデータ型を調べて適切な方法で処理する必要がある．また，Delphi では画像表示などで使用するフォームを用意しておく必要があり，フォームごとに再描画処理も必要になる．SOL ではこのような記述がまったく必要ない．

```
        ⋮
read(menu);
case menu of
1:begin /* 全リスト */
   writeln(f*);
  end;
2:begin /* 主体／対象リスト */
   writeln(S);
  end;
3:begin /* 関係リスト */
   writeln(LABEL);
  end;
4:begin /* 検索 */
   write("キーワードは
   (画像検索の場合は image)？");
   read(key);
   none ← true;
   if (key ≠ "image") then
     stmp ← S;
     while( stmp ≠ φ ) do
       tmpany ← getel( stmp );
        ⋮
```

(a) SOLにおける表示／検索手続き

```
for i:=1 to 25 do        ⋮
 begin
  for j:=1 to 10 do
  begin
   if data[i].flag[j]=true then
   begin
    nonzero:=true;
    writeln('[',data[i].sub,',',data[i].rel[j],
            ',',data[i].obj[j],']');
   if data[i].otyp[j]=img_data then
    begin
     num:=i;
     ele:=j;
     StretchBlt(Form2.Canvas.Handle,0,0,
            data[ele].oimg[num].Width,
            data[num].oimg[ele].Height,
            data[num].oimg[ele].Canvas.Handle,
            0,0,data[num].oimg[ele].Width,
            data[num].oimg[ele].Height,SRCCOPY);
     Form2.Width:=data[num].oimg[ele].Width+7;
     Form2.Height:=data[ele].oimg[num].Height+20;
     Form2.Show;
    end;
    if data[i].styp=img_data then
    begin
     num:=i;
     ele:=0;
     StretchBlt(Form3.Canvas.Handle,0,0,
            data[num].simg.Width,
            data[num].simg.Height,
            data[num].simg.Canvas.Handle,
            0,0,data[num].simg.Width,
            data[num].simg.Height,SRCCOPY);
     Form3.Width:=data[num].simg.Width+7;
     Form3.Height:=data[num].simg.Height+20;
     Form3.Show;
    end;
   end;                  ⋮
  end;
end;
```

(b)Delphiにおける表示／検索手続き

図 8 表示/検索手続き
Fig. 8 Procedure of display and search.

```
        ⋮
write("主体は
 （キーワードもしくはマウス
による範囲指定） ？ ");
read( sub_any );
write("関係は ？ ");
read(rel_str);
write("対象文字列は ？ ");
read(obj_any);
addmap f{rel_str}(sub_any)
          = obj_any;
        ⋮
```

(a)SOLにおける追加手続き

```
if x=1 then          ⋮
begin
 for i:=1 to 25 do
 begin
  if table[i]=false then
  begin
   flag:=true;
   table[i]:=true;
   write('主体は  1:文字列  2:静止画像？');
   readln(y);
   if y=1 then
   begin
    write('主体（文字列）を入力してください：');
    readln(str);
    data[i].sub:=str;
    data[i].styp:=str_data;
   end
   else
   begin
   write('主体（静止画像）のファイル名を
            入力してください：');
   readln(str);
   data[i].styp:=img_data;
   data[i].simg:=TBitmap.Create;
   data[i].simg.LoadFromFile(str);
   data[i].sub:=str;
  end;
 write('関係（文字列）を入力してください：');
 readln(str);
 data[i].rel[1]:=str;
 write('対象は  1:文字列  2:静止画像？');
 readln(z);
 if z=1 then
 begin
  write('対象（文字列）を入力してください：');
        ⋮
```

(b) Delphiにおける追加手続き

図 9 追加手続き
Fig. 9 Procedure of addition.

5.4 ファイルからの読み込み，ファイルへの保存

個々のデータのファイル保存は SOL，Dephi ともに簡単に実現することができるが，データ構造全体の保存を考えると，Delphi ではマルチメディアデータを

表 5 ソースプログラムの行数（コメント行と空行は除く）
Table 5 Total line number of source program.

項目	SOL	Delphi
合計	250	527
初期化	31	90
データの追加	28	114
データの削除	37	52
データの表示，検索	81	166
ファイル入出力	30	–
その他（メイン関数，宣言など）	43	105

含むレコードでデータ構造が表現されており，ファイル入出力の形式から検討する必要がある．一方，SOL ではマルチメディアデータを含む場合でも，ファイル名を指定して **write(fp,f*);** と記述するだけで，対応関係すべてをファイルに書き出すことができ，また **read(fp,f*);** と記述すればファイルから対応関係を読み込むことができる．なお，Delphi においてはファイル入出力の機能はプログラムとして実現していない．

5.5 可読性と記述の効率性

記述例から分かるように，SOL においてはマルチメディアデータを普通の変数のように扱うことができ，分かりやすくプログラムを記述することができる（可読性の向上）．全体としての記述量も**表 5** に示すように Delphi のおよそ半分で済んでおり，その記述性の高さ（記述における効率性）が明らかになった．

また，Delphi においては，マルチメディアデータの表示などに API に関する専門的な知識が必要であるが，SOL においてはプログラミングの基本的な知識があればプログラムの作成が可能である．さらに，SOL では不定型や集合，写像/対応などをマルチメディアデータに対しても利用することができるので，マルチメディアデータを含むデータの構造化とその検索などの操作を行うプログラムをシンプルに記述できる．

6. Java との比較

5 章では，手続き型言語としての SOL の評価を行ったが，オブジェクト指向プログラミングによるクラスの再利用による開発効率の向上も無視できない．そこで，比較の対象をオブジェクト指向言語にまで広げる．ただし，SOL の機能をそのままクラスとして用意したのでは，構文上の違いからの記述の差異がみられる程度で，本質的には同じプログラムになると考えられるため，オブジェクト指向に基づくプログラミングを行い比較を行う．また，現在広く利用されている Java をオブジェクト指向言語の例として取り上げることにした．

作成したプログラムのクラスとそのメソッドを**図 10** に示す．マルチメディアデータ単体を格納する multimediaObject クラス用い，登録するマルチメディアデータはこのクラスのオブジェクトとして管理する．そのオブジェクトを managementObjects クラスで配列として管理することでマルチメディアデータの構造を実現した．このとき，主体，関係，対象は 3 つ組み

```
import java.util.*;
import java.io.File;
import java.awt.*;
import java.awt.event.*;

/* １つのユーザインターフェースのみあらかじめ提供 */
abstract class appFrame extends Frame implements ActionListener{
  void init(){
  /* ボタンの配置やテキストボックスなどのGUIを作成 */
  }
}

/* 例題の操作を行うアプリケーション */
public class multimedialManagement extends appFrame implements ActionListener{
  public static void main(String [] args){
   /* メイン関数（入出力処理，イベント処理）*/ }
  void initialize(){
   /* 初期データの登録 */ }
  public void paint(Graphics g){
   /* 描画処理 */ }
  public void actionPerformed(ActionEvent e){
   /* イベントの検出と対応する処理 */ }
}

/* マルチメディアデータの操作，管理を行うクラス */
class managementObjects{
  managementObjects(int smax,int rmax, int omax) {
   /* コンストラクタ */ }
  int getID(String target, String typ) {
   /* 既登録オブジェクトのIDを取得 */ }
  String dispOP(){
   /* 管理しているデータ全体を表示用 */ }
  boolean appendOP(String sub, String rel, String obj, String typS, String typO){
   /* データの登録 */ }
  boolean deleteOP(String sub, String rel, String obj, String typS, String typO){
   /* データの削除２（３つ組み） */ }
  boolean deleteOP(String subobj, String typ){
   /* データの削除１（主体／関係／対象の削除のいずれかが一致） */ }
  boolean searchOP(String sub, String rel, String obj, String typS, String typO){
   /* 検索１（３つ組み） */ }
  String searchOP(String sub, String rel, String typS){
   /* 検索２ （主体と対象を指定）*/ }
  void dispImage(String sub, String obj, String typS, String typO){
    /* 表示用のイメージオブジェクトの用意 */ }
  boolean fileCheck(String s, String o, String sTyp, String oTyp) {
    /* ファイルの存在を確認 */ }
}

/* マルチメディアデータ単体を格納するクラス */
  class multimediaObject{
    multimediaObject(){
     /* コンストラクタ */ }
    void newObject(String inputStr, String objTyp) {
     /* 新しいオブジェクトの登録 */ }
    String getObjectS(){
     /* 文字列オブジェクトを返す */ }
    String getAll() {
     /* すべてのオブジェクトの文字列またはファイル名を返す */ }
    Image getObject(){
     /* イメージオブジェクトを返す */ }
    public String getTyp(){
     /* オブジェクトタイプを返す */ }
    public boolean identify(String data, String typ){
     /* オブジェクトの存在を判定 */ }
  }

  /* ３つ組みの集合を提供するクラス */
class threeTupleSet{
  threeTupleSet(int rmax){
   /* コンストラクタ */ }
  int existElement(Integer x, Integer y, Integer z){
   /* 既登録の３つ組みのIDの取得 */ }
  int existElement(Integer x){
   /* 要素に着目データを含む既登録３つ組みのIDを取得 */
  boolean appElement(Integer x, Integer y, Integer z){
   /* ３つ組みの追加 */ }
  boolean delElement(Integer x, Integer y, Integer z){
   /* ３つ組みの削除 */ }
  boolean delElement(Integer x){
   /* １つでも要素が一致する３つ組の削除 */ }
  Integer getSub(int index){
   /* 指定した３つ組みの主体を取得 */ }
  Integer getRel(int index){
   /* 指定した３つ組みの主体を取得 */ }
  Integer getObj(int index){
   /* 指定した３つ組みの対象を取得 */ }
  int getElementNum() {
   /* 登録されている３つ組みの数を取得 */ }
}
```

図 10 作成したクラスのメソッド一覧
Fig. 10 The designed classes and their class methods.

```
/* ３つ組みの集合の機能を提供するクラス */
class threeTupleSet
{
  int MAXNUM_REL;
  Vector sub, rel, obj;
  threeTupleSet(int rmax){
    MAXNUM_REL = rmax;      sub = new Vector(3);
    rel = new Vector(3);    obj = new Vector(3);
  }
   /* 既登録の３つ組みのIDの取得 */
  int existElement(Integer x, Integer y, Integer z)
  {
   int i, indexTmp=0, ret=-1;
      while (indexTmp != -1){
      indexTmp=sub.indexOf(x,indexTmp);
      if (indexTmp!=-1){
        if ( (((Integer)rel.elementAt(indexTmp)).intValue()
          ==y.intValue())
             && (((Integer)obj.elementAt(indexTmp)).intValue()
               ==z.intValue()) ){
          ret = indexTmp;
          break;}
        indexTmp += 1;
      }
    }
    return ret;
  }
   /* 要素に着目データを含む既登録３つ組みのIDを取得 */
  int existElement(Integer x){
    int i, indexTmp=0, ret=-1;
    boolean notFound=true;
    indexTmp = sub.indexOf(x);
    if ( indexTmp!= -1)
      ret = indexTmp;
    indexTmp = rel.indexOf(x);
    if ( indexTmp!= -1)
      ret = indexTmp;
    indexTmp = obj.indexOf(x);
    if ( indexTmp!= -1)
      ret = indexTmp;
    return ret;
  }
   /* ３つ組みの追加 */
  boolean appElement(Integer x, Integer y, Integer z)
  {
    int i, id=0;
    boolean ret=true;
    if ( sub.size()==MAXNUM_REL-1 )
      ret = false;
    else{
      if ( existElement(x,y,z)==-1 ){
        sub.addElement(x);
        rel.addElement(y);
        obj.addElement(z);
      }
    }
    return ret;
  }
   /* ３つ組みの削除 */
  boolean delElement(Integer x, Integer y, Integer z)
  {
    int indexTmp;
    boolean ret=false;
    indexTmp = existElement(x,y,z);
    if ( indexTmp!=-1 ){
      sub.removeElementAt(indexTmp);
      rel.removeElementAt(indexTmp);
      obj.removeElementAt(indexTmp);
      ret = true;
    }
    return ret;
  }

  ...

}
```

図 11 threeTupleSet クラスのプログラム記述（抜粋）
Fig. 11 A part of the program list of threeTupleSet class.

```
/* マルチメディアデータの操作，管理を行うクラス */
class managementObjects
{
 int MAXNUM_SUB, MAXNUM_REL, MAXNUM_OBJ, MAXNUM;
  multimediaObject data[];
  public Image imgtest[];
  public int imgct;
  threeTupleSet relation;
  boolean dataF[];

   /* コンストラクタ */
  managementObjects(int smax,int rmax, int omax)
  {
    int i, j;
    MAXNUM_SUB = smax;
    MAXNUM_REL = rmax;
    MAXNUM_OBJ = omax;
    MAXNUM = MAXNUM_SUB + MAXNUM_OBJ + MAXNUM_REL;
    data = new multimediaObject[MAXNUM];
    imgtest = new Image[MAXNUM_SUB+MAXNUM_OBJ];
    dataF = new boolean[MAXNUM];
    for (i=0;i<MAXNUM;i++){
      data[i] = new multimediaObject();
      dataF[i] = false;
    }
    relation = new threeTupleSet(MAXNUM_REL);
  }

  ...

   /* 検索１（３つ組み） */
  boolean searchOP(String sub, String rel, String obj, String typS, String typO)
  {
    int sid, rid, oid;
    boolean ret;
    sid = getID(sub,typS);
    rid = getID(rel,"S");
    oid = getID(obj,typO);
    if ( (relation.existElement(new Integer(sid), new Integer(rid), new Integer(oid))) == -1)
      ret = false;
    else
      ret = true;
    dispImage(sub, obj, typS, typO);
    return ret;
  }

   /* 検索２ （主体と対象を指定）*/
  String searchOP(String sub, String rel, String typS)
  {
    int i, sid, rid;
    String tmp1, tmp2, tmp3, ret="";
    sid = getID(sub,typS);
    rid = getID(rel,"S");
    imgct=0;
    for (i=0;i<MAXNUM;i++){
      if (relation.existElement(new Integer(sid), new Integer(rid), new Integer(i))!=-1){
        tmp1 = data[sid].getAll()+" → ";
        tmp2 = data[rid].getAll()+" ⇒ ";
        tmp3 = data[i].getAll()+"¥n";
        ret = ret + tmp1 + tmp2 + tmp3;
        if (data[sid].getTyp()=="I"){
          imgtest[imgct]= Toolkit.getDefaultToolkit().getImage(data[i].getAll());
          imgct+=1;
        }
        if (data[i].getTyp()=="I"){
          imgtest[imgct]= Toolkit.getDefaultToolkit().getImage(data[i].getAll());
          imgct+=1;
        }
      }
    }
    return ret;
  }

 ...

}
```

図 12 managemnetObjects クラスのプログラム記述（抜粋）
Fig. 12 A part of the program list of the managementObjects class.

集合の threeTupleSet クラスを用いてデータの構造化を行った．記述例として，threeTupleSet クラスのプログラム記述（抜粋）と multimediaObjects クラスの記述（抜粋）をそれぞれ**図 11**，**図 12** に示す．

このような方法でプログラムの記述を行うと，5 章の例題は 569 行のプログラム記述になる．比較的汎用性があると考えられる multimediaObject クラスと threeTupleSet クラスをあらかじめ提供するとすれば，その記述は 422 行で済む．この時点では，まだ SOL のプログラム記述量の方が幾分少ないが，提供するクラスを機能的に拡張することで，Java によるプログラムの記述量はさらに削減できるため，一概に SOL の記述量の方が少ないとはいえない．しかし，本例題のような規模が大きくないプログラムであれば，SOL でも十分に効率良くプログラムを記述できると考えられる．

7. おわりに

本論文では，マルチメディアプログラミングツールとしての集合指向言語 SOL の拡張について述べた．拡張に際してはマルチメディアデータの構造化および表示や検索などを考慮し，マルチメディア型を基本型として導入した．これにより，集合や写像/対応にマルチ

メディアデータを利用することができ，また手続きの拡張とあわせて，マルチメディアデータの構造化と表示や検索などの操作が容易に実現できるようになった．また，スキーマフリーのデータ管理アプリケーションのプログラム記述を Delphi と比較することによって，SOL の可読性の良さと記述における効率性が確認できた．SOL のプログラムは記述量だけをみても Delphi の半分程度であり，可読性を考慮するとそれ以上の効果があるといえる．また，オブジェクト指向言語との比較も行い，SOL の記述性について言及した．しかし，オブジェクト指向言語は保守性と拡張性において優れており，アプリケーションの大規模化が進むことを考えれば，SOL においても発展的な検討が必要かもしれない．

5 章において，スキーマフリーのデータ管理アプリケーションを例として示したが，このほかにも 1 章で述べた WWW 上のホームページ自動作成システムへの応用などが考えられる．例題で示したマルチメディアデータの管理手法を拡張すれば，利用者が保持するマルチメディア情報をデータ構造として実現することも容易であり，また構造化したデータに対して，追加や削除などを対話的な操作として簡単に行える．このようにして実現したデータ構造を HTML 形式や XML 形式に変換する手法を考案することで，ホームページの自動生成や保守に役立つことが期待される．

なお，SOL のプログラム開発環境（エディタ，言語処理系，入門書，サンプルプログラム）のダウンロードができるホームページを次の URL に用意している．

`http://www.cs.comp.kyutech.ac.jp/~sol/sol_project/src/index.html`

謝辞 本論文に有益なご助言をいただいた九州工業大学大学院工学研究科の小出洋講師に深く感謝いたします．

参 考 文 献

1) Abiteboul, S., Buneman, P. and Susiu, D.: *Data on the Web—From Relations to Semistructured Data and XML*, p.258, Morgan Kanfmann, San Francisco (2000).
2) エンタープライズアプリケーション開発ガイド—Microsoft Visual Studio6.0 Enterprise Edition, マイクロソフト (1998).
3) Borland International: Delphi3.0 ユーザーズガイド，p.559, ボーランド (1997).
4) Yokoo, N. and Shigematsu, Y.: Development and Extension of Set Oriented Language (SOL), *Proc. 2nd International Conference on Parallel and Distributed Computing and Networks*, pp.64–69 (1998).
5) 横尾徳保，重松保弘：WWW を利用した集合指向言語 SOL 遠隔実行環境の構築，情報処理学会論文誌，Vol.41, No.9, pp.2661–2664 (2000).
6) 横尾徳保，重松保弘：集合指向言語 SOL のマルチメディアデータへの対応，情報処理学会第 31 回 PRO 研究会発表資料，p.14 (2000).
7) 迫江義彦，重松保弘：不定型とラベル付き写像記法の導入による集合指向言語 SOL の仕様拡張とその評価，情報処理学会論文誌，Vol.38, No.8, pp.1662–1665 (1997).
8) Schwartz, J.T., Dewar, R.B.K., Dubinsky, E. and Schonberg, E.: *Programming with Sets—An Introduction to SETL*, p.493, Springer-Verlag, New York (1986).
9) Baxter, N., Dubinsky, E. and Levin, G.: *Learning Discrete Mathematics with ISETL*, p.416, Springer-Verlag, New York (1988).
10) 田幡 勝，有次正義：半構造データモデルによる画像処理履歴の管理，情報処理学会論文誌，Vol.41, No.SIG1, pp.64–75 (2000).
11) 田島敬史：半構造データのためのデータモデルと操作言語，情報処理学会論文誌，Vol.40, No.SIG3, pp.152–170 (1999).

(平成 13 年 1 月 31 日受付)
(平成 14 年 3 月 14 日採録)

横尾 徳保（正会員）

1972 年生まれ．1995 年九州工業大学情報工学科卒業．1997 年同大学大学院情報工学研究科博士前期課程修了．同年，九州工業大学大学院工学研究科助手．コンパイラ技術，計算機ネットワークに関する研究に従事．

重松 保弘（正会員）

1947 年生まれ．1970 年九州工業大学工学部電子工学科卒業．1972 年九州工業大学工学部助手．その後，九州工業大学工学部講師，九州工業大学工学部助教授を経て 1995 年より九州工業大学大学院工学研究科教授．工学博士．計算機言語設計とコンパイラ技術，計算機ネットワークに関する研究に従事．著書「UNIX プログラミング入門講座」等．電子情報通信学会会員．