2L－2

データの構造化を考慮したビジュアルプログラミングシステム

増田　健　加藤康之　満永　豊
ＮＴＴアクセス網研究所

１．はじめに

エンドユーザによるアプリケーションの開発、実行、改良を可能にするビジュアルプログラミングシステム（以下ＶＰＳ）は、処理の内容と業務知識との関連性が密接で、業務内容の変化に伴い様々な改良が必要となるデータベースアプリケーション（以下ＤＢＡＰ）の開発環境として有効であると考えられる。ＶＰＳにおけるプログラム記述形式には様々なものがあるが、データフロー型計算の考え方に基づくプログラム記述形式は、処理手順が陽に表現されるため理解しやすいという利点がある [1]。ただし、この記述形式をＤＢＡＰに適用する場合、レコード型データのようなデータの構造化の機能を追加する必要がある。

本研究では、特定のレコード型データをあらかじめ用意しておくのではなく、ユーザがＶＰＳ上でレコード形式を定義、変更できる汎用的な記述形式を目指している。このために、レコード型データを動的な内部表現であらわし、そのレコード形式の定義および変換を行う基本操作をＶＰＳ上で組み合わせられるようにしている。

２．システム構成

本ＶＰＳでは、処理の最小単位であるモジュールを組み合わせてプログラムを作成する。これは、モジュール間のデータの授受関係を表わすフロー図を、グラフィカルな図形編集によって作成、変更することで行う。ＶＰＳ上のモジュールは、次の２種類があり、構造化されたフローが記述できる。

1）あらかじめ準備されているオブジェクトプログラムをＶＰＳにロードしたもの（**基本モジュール**）
2）他の計算モジュールを使ってＶＰＳ上でフロー記述を行い作成したもの（**ユーザ定義モジュール**）

フロー図に対し、定められたメニューコマンドを選択すると、ＶＰＳは実行モードに変わり、フローに記述された処理が実行される。

本システムは、NeXTSTEP3.2J 上に実装されており、Objective-C 言語を用いて記述されている。

３．レコード型データの導入

レコード型データをＶＰＳに導入する際、ＶＰＳの適用範囲が限定されないように、ユーザが自由にレコード形式を定義できる機能が必要である。また、モジュールが入力データとして受け入れられるレコード形式は固定されているため、ＶＰＳ上で適切なレコード形式に変換する機能も必要である。

上記の観点から、本ＶＰＳではレコード型データの内部表現を動的なものにし、レコード形式の定義および管理をユーザがＶＰＳ上で行えるようにした。

３．１　レコード型データの内部表現

本システムでは実行時に動的に構成を変更できる内部表現として木を用いる。木の葉は、整数、実数、文字列等の基本データの値を格納し、木の途中に存在するノードは、レコードのフィールド名をキーとするハッシュテーブルになっている。ハッシュテーブルには、葉もしくはノードへのポインタが格納される。図１にレコード型データを表わす木の模式図を示す。この内部表現によると、図２に示すような、入れ子のレコード型データを表わすことも可能である。

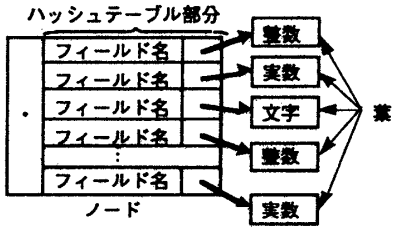

図１　レコード型データ

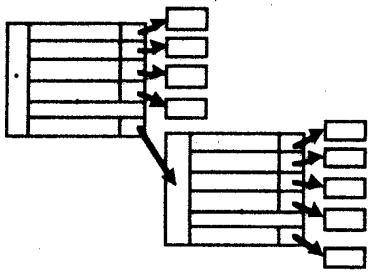

図２　入れ子のレコード型データ

A Visual Programming System Considering Data Structuralization.
Takeshi Masuda, Yasuyuki Kato, Yutaka Mitsunaga
NTT Access Network Systems Laboratories
Tokai-Mura, Naka-Gun, Ibaraki-Ken, 319-11, Japan

表1　基本変換

	基本変換名	機　能
基本変換	フィールド作成	指定された名前のフィールドを作成する。
	フィールド削除	指定された名前のフィールドを削除する。
	フィールド取得	指定された名前のフィールドを残し、他を削除する。
	レコードマージ	複数のレコード型データを併合する。
	フィールド名変更	指定されたフィールド名を変更する。
	フィールド名取得	フィールド名を値として取得する。

3．2　レコード型データに対する変換操作

本研究では、レコード形式の変換処理を数種類の基本的変換操作の組み合わせと考え、まず、実用上十分と考えられる基本変換を整理した。その内容を表1に示す。本ＶＰＳでは、これらの基本変換を実行する基本モジュールを準備してあり、フロー図上で組み合わせることで、より複雑な変換操作を実現できる。

4．実施例

レコード型データを扱う処理として、実数配列の積和演算を取り上げ、本ＶＰＳ上で記述した例を示す。処理に対する入力は配列の要素数「imax」と値配列「vmat」および重み配列「wmat」とする。これらの入力データは一つのレコード型データにまとめて渡される。このレコード形式の模式図を図3に示す。配列は、インデックスをフィールド名とするレコード型データとして表現する。

積和を求める処理を記述したフロー図を図4に示す。最初に、「sum」というフィールドをレコード型データに追加し、配列のインデックスの初期値1で初期化する。ループモジュールは、「i」というフィールドを作成し、「i」を1から「imax」まで変えながらループ処理を実行する。ループ内では、「vmat」と「wmat」から配列の要素を取り出し、両者の積を計算し、続いてその結果と「sum」との和を計算する。得られた値で再び「sum」というフィールドを作成し、もとのレコード型データの「sum」と置き換える。

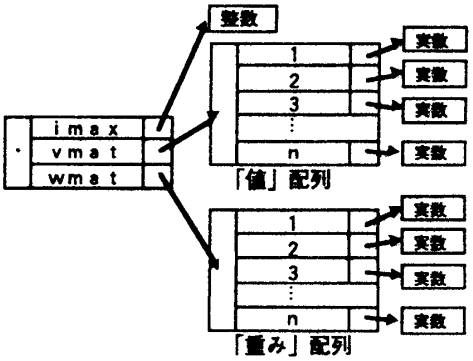

図3　入力されるレコード型データ

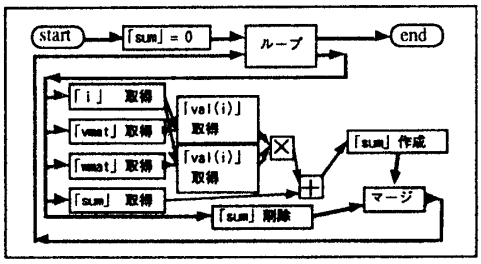

図4　積和演算フロー図

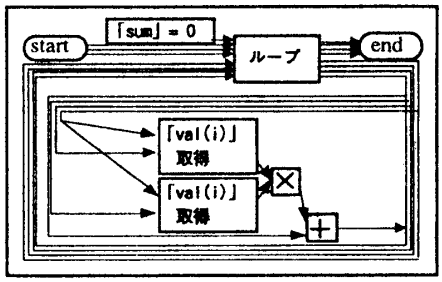

図5　レコード型データを使わないフロー図

5．考察

データフロー形式の記述で、ＤＢＡＰを開発する場合、複数のデータの流れが1つの実行の流れに属すケースが多くなる。例えば、図5は実施例と同じフローをレコード型データを使わずに記述した場合を示している。これに対し、図4はデータの構造化により、実行の流れが一本の配線で表現できるためフローが理解しやすくなる。

一方、実施例で示したフローをユーザ定義モジュールとして使用する場合、図3に示すレコード形式を含んでいれば、それを入力として受け入れることができることがわかる。このように、厳密にレコード形式が一致していなくてもモジュール間でデータをやりとりできるのは、レコード型データの内部表現としてフィールド名のハッシュテーブルを用いたためである。

さらに、本ＶＰＳでは、ユーザがフローの中でレコード形式の操作を行えるため、モジュールを呼び出す時点で、必要なフィールドを一時的に作成することも可能である。このように、既存のモジュールを変更することなく、様々なレコード形式に対応できるため、モジュールの再利用が行い易くなる。

6．おわりに

本ＶＰＳにより、ユーザは適用範囲の広いレコード型データ処理の記述が可能になったといえる。今後の課題としては、デバッグ機能および、レコード型の引き数チェック機能の実現などが挙げられる。

【参考文献】
[1] Nan C. Shu 著、西川博昭 訳、“ビジュアルプログラミング”, 日経BP社　1991.