ソフトウェア基礎論　38－3
プログラミング言語　28－3
（１９９１．３．８）

マルチメディアを対象とした オブジェクト指向言語「紋様」

小泉　忍　　小林　り恵　　山野　紘一

（株）日立製作所　システム開発研究所

帳票、文書、図形、画像、音声等を「オブジェクト」としてとらえたオブジェクト指向のメディア操作言語「**紋様**」を開発した。紋様は、データ抽象化を実現しオブジェクトの構造と操作の仕様を定義するテンプレート、操作の借用(delegation)による継承、オブジェクトの外部表現仕様であるメディア型、複数のオブジェクト間で相互の値および属性の直接的参照を行なうオブジェクトリンク、定義式に従いオブジェクトの値、属性、相互関係を調整するオブジェクト制約等の機能を持つ。特に、オブジェクト制約とオブジェクトリンク機能により、依存関係のある複数のオブジェクトを動的に結合した複合オブジェクトの作成・操作を容易にした点が特徴である。

MONYOH：OBJECT ORIENTED LANGUAGE FOR MULTI－MEDIA PROCESSINGS

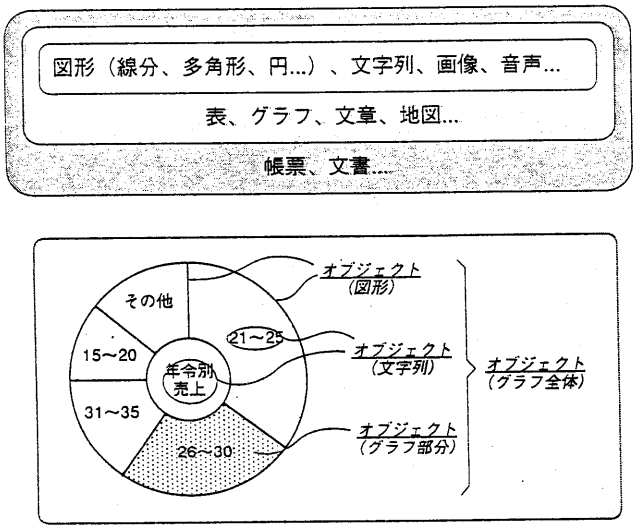
Shinobu Koizumi　Rie Kobayashi　Koichi Yamano

Systems Development Laboratory, Hitachi Ltd.

Monyoh is an object-oriented language being designed to operate multi-media objects such that documants, forms, figures, images or sounds. As an object-oriented language, data abstruction mechanism is performed by 'Template' which defines structure of a object combined with its operations. And inheratance facility is performed by method 'Delegation' which reuse other template's operations. Especially for creating and manipurating dinamically conbined objects easily, it has 'Object Link' facility which enables to directly access or refer to other object values and attributes and 'Object Constraint' facility which maintaines the values, attributs and inter-object relations consistent with given expression.

1．はじめに

マルチメディア処理では、対象がバラエティに富んでいるだけでなく、それぞれの操作自身も複雑である。このため、従来の数値データ処理等で一般的なサブルーチンパッケージでは、サブルーチンの数が多く、かつ対象と適合するサブルーチンの組み合わせやパラメータの適合性の検証に関して、利用者（プログラマ）に大きな負担を強いることが問題である。このような問題に対しては「オブジェクト指向」アプローチが有効である。メディア操作言語「**紋様(Monyoh:Media Operation Language System for Humanware)**」は、帳票、文書、図形、画像、音声等を「オブジェクト」としてとらえ、それらを統一的に取扱うことを主眼とした言語である。特に、図形、音声、画像等の要素からなる、文書、帳票等の複合オブジェクト(**第１図**)を取扱うための「オブジェクトリンク機能」「オブジェクト制約機能」を特徴としている。



第１図　紋様の対象世界

2．紋様の設計思想

グラフィカルユーザインタフェースの分野では、Ｘ－WindowにおけるWidgetなど、対話用の部品を提供するオブジェクト指向環境の利用が普及しつつある。これら部品を利用することにより、一般のアプリケーションプログラムのユーザインタフェース部分の生産性は劇的に向上するが、以下の点でまだ不十分と考えている。

（１）部品自身の記述の複雑さは、オブジェクト指向の「継承機能」等により軽減されている。しかし、例えば、部分的な変更時には、関連する全ての値の更新が明示的に必要であるなど、プログラミングの複雑さにまだ改善の余地がある。

（２）部品の提供する機能は簡単な対話処理関連に限定されているため、例えば動的に形状を変更するような場合の対応が困難である。このような場合にはアプリケーションとして従来のように複雑なプログラミングが必要であり、また、ユーザインタフェース部品とアプリケーションが完全に分離した上記アプローチに無理がある。

上記の（１）を解決する一つの方向としては、「制約プログラミング」の導入がある。即ち、従来のプログラミングでは、ある部分的な値の更新に対して、関連する他の値を明示的に変更する手続き的なプログラムが必要であったが、制約プログラミングでは、宣言的な「制約」（値や属性等の関係式）の記述を行ない、制約を満たす値の設定は処理系が行なう。このような制約プログラミング機能を持つオブジェクト指向言語には、ThingLab II[1)]（Washington大）、Coral[2)]（CMU）等がある。しかし、いずれもユーザインタフェース用の部品の提供に主眼をおいており、上記（２）の視点に欠けている。

紋様では、上記のユーザインタフェース部品の発想とは異なり、アプリケーション側からアプローチする。即ち、処理の対象

全体をオブジェクトとして捉え、その一側面として表示の形状等の記述を可能としている。このため、オブジェクトの範型であるテンプレート（一般のオブジェクト指向言語のクラスに相当）とは別に、オブジェクトの表現仕様を指定するメディア型を導入し、テンプレートの一部としてメディア型の指定を行なう方式を採っている。

さらに、アプリケーションの構成方法として、紋様を用いたオブジェクトライブラリ（テンプレート群）の作成と、オブジェクトライブラリの利用（インスタンスの生成／組み合わせ複合化／操作）との２つのフェーズを想定している。

ライブラリ利用者の立場からは「モジュラリティの高さ」が要求される。モジュラリティには、要素の独立性とそれらの結合性の２つの側面がある。紋様では、前者に対しては、テンプレートによるデータ抽象化機能と内部状態を自律的に決定する制約機能により、また、後者に対しては、オブジェクトリンク機能と外部状態に追随する制約機能により、対応を図っている。

一方、ライブラリ作成者の立場からは安全性よりも「自由度の高さ」や「実行性能の高さ」が要求される。これは、テンプレートの定義方法や定義内容に反映させた。例えば、継承の方法として、制限の多いクラス階層を用いずに借用を採用したこと、他のオブジェクトのスロットの値を参照する方法として、メッセージ通信によらない直接的な参照を可能にしたこと等がその表れである。

３．言語の概要

紋様の主要な機能を第２図に示す。

３．１　オブジェクト指向機能

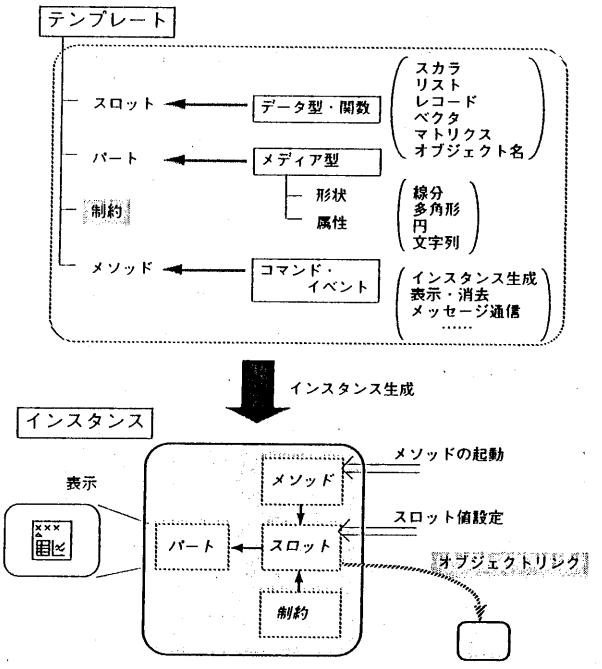
オブジェクト指向言語の要件には様々な議論があるが、紋様では「データ抽象化機能」＋「継承機能」がオブジェクト指向言語としての必要最小限と考え、これらを以下のように取扱っている。

（１）データ抽象化機能

紋様では、「テンプレート」によりインスタンスの構造と操作の仕様を定義し、データ抽象化機能を実現している。ここで、インスタンスの構造とは、スロット（インスタンスの状態変数）の構成、および、表現仕様（⇒３．２メディア型）の指定である。また、操作の仕様とは、メソッド（スロットの更新、および他のメソッドやコマンドの適用手順）の定義である。テンプレートは、一般のオブジェクト指向言語における「クラス」に相当するが、紋様では、

テンプレート自身はインスタンスの範型を指定するものであってオブジェクトとして取扱わない。

なお、インスタンスの生成は、定義済のテンプレートに基づいて、生成コマンドにより行なう。インスタンスがオブジェクト



第２図　紋様の概要

として取扱われる。

（２）継承機能

紋様における継承機能は、メソッドの借用(delegation)により実現している。継承機能の目的の一つは「差分プログラミング」を実現することであり、一般のオブジェクト指向言語では、継承は「クラス階層」に基づき、親クラスからの変更点のみを再定義する方法をとっている。一方、紋様では、定義済みの任意のテンプレートのメソッドを新たに定義するテンプレートで引用することにより、メソッドの流用を可能としている。

３．２　メディア型機能

メディア型とは、インスタンスの外部表現仕様である。即ち、図形（線分、円、多角形、文字列...）画像、音声等の識別、それらの表現時の属性（線種、色、大きさ、位置等）、およびそれらの組み合わせである。紋様では、グラフィックプリミティブを既定のメディア型として提供している。また、既定のメディア型を部分として持つ、それらを組み合わせた新たなメディア型の定義が可能である。

メディア型という呼び方で分かるように、紋様では、オブジェクトの表示形状を宣言的に与えるのであって、手続き的な描画コマンドによってプログラムするのではない点に注意されたい。これに関連して、複合メディア方において図形の重なりにより描画の順序が問題となる場合があるが、紋様では、これを宣言順としている。

また、表示等の外部表現を制御するものとして、表示、消去の各コマンドを設けている。

３．３　オブジェクト制約とオブジェクトリンク

複数のインスタンスの組み合わせによるプログラミングを支援するため、紋様では「オブジェクト制約機能」と「オブジェクトリンク機能」を設けている。

従来一般に用いられているオブジェクト指向言語では、オブジェクトの個々の動作を逐次的・手続き的に記述する。このため、あるオブジェクト（参照側）が他のオブジェクト（被参照側）の値を参照している場合、これら言語を用いたプログラムでは、被参照側のオブジェクトの更新時には、被参照側から参照側に対して更新の通知メッセージを明示的に発行し、参照側で更新の通知メッセージを受理した時に、その更新に伴う値の再計算を行なうようプログラムする必要がある。

一方、紋様では、オブジェクト制約機能により、値の関係式／計算式を宣言的に与えておくことができ、上記のような更新に伴う再計算を明示的にプログラムしておく必要がない。即ち、従来の逐次的手続き型言語や上記のオブジェクト指向言語では、ある式は、その式を含む文に制御が渡った時に評価されるが、紋様では、制約式は明示的な評価の記述なしに自動的に評価され、また、参照項目の更新等により必要な再評価も自動的に実施される。ここでは、上記のオブジェクト指向言語で必要としていた通知メッセージの発行とその受理による再計算という逐次的な動作の記述は一切不要である。

このオブジェクト制約機能が、紋様によるプログラムの記述性を飛躍的に向上させている一因となっている。

（１）オブジェクト制約機能

メソッドによらずにスロットの値を計算するメカニズムとして、紋様には、以下の２つのオブジェクト制約機能を設けている。

ａ）個々のスロットに付与された「スロット式」による制約

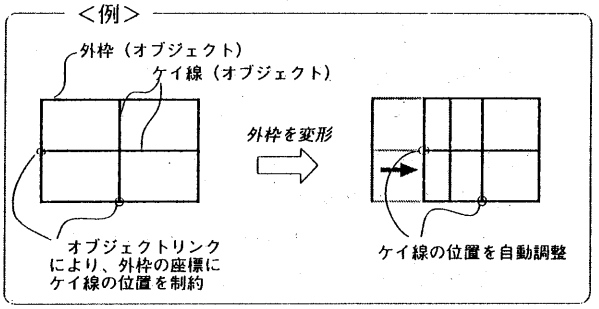
b）テンプレートに対して付与された「制約式」（＝複数スロット間の値の関係式）による制約

スロット式による制約では、スロットの値が常に式の値に一致するように式が再評価され、その値がスロットに代入される。また、制約式による制約では、制約式が成立するように、値が明示的に与えられていないスロットの値が調整される。いずれの場合にも、スロットの値の決定のタイミングは、プログラム上で直接指示するのではなく、処理系が必要に応じて計算を実施する。

（2）オブジェクトリンク

オブジェクトリンクとは、あるインスタンスから他のインスタンスのスロットの値を（メッセージ通信等を用いずに）直接的に参照する機能である。このため、参照先の他のインスタンス名を保持するための「オブジェクトスロット」と、オブジェクトスロットを介して、参照先のインスタンスのスロットを間接参照する機能を設けている。

上記オブジェクト制約とオブジェクトリンクとを併用することにより、依存関係のある複数のインスタンスを動的に結合することが簡単に実現できる。（**第３図**）



第３図　オブジェクト制約／オブジェクトリンク

３．４　その他の機能

紋様におけるその他の機能には、データ型、関数定義等がある。基本データ型としては、整数型、実数型、論理型、文字型、ビット型、文字列型、ビット列型があり、列挙型、直積型、リスト型等のユーザ定義機能も提供している。これらに対する算術演算、リスト操作等を組み込み関数として持つほか、座標変換に必要なベクタ・マトリクス型とその演算も既定のものとして提供している。

紋様では、インスタンス以外にユーザ定義可能なグローバル変数を持たない。従って、オブジェクトの外部における共通の演算に対しては関数型アプローチが適切であると考えている。

３．６　簡単なプログラムの例

ここで、**第３図**の図形に対する複数のプログラム例を示す。

第４図は、外枠、横線、縦線をそれぞれ別のオブジェクトとして生成し、それらを動的に組み合わせた例で、スロット式による制約を用いている。

この例では、外枠、横線、縦線のテンプレートをそれぞれ FRECT…１、 HLINE…９、VLINE…１７という名称で定義している。２、３、１０、１８はパラメタスロットの定義で、オブジェクト生成時に値を設定すべきスロットを示している。特に、１０、１８はオブジェクトスロットで、FRECT テンプレートから生成されたインスタンス名を保持する。また、**part**…４、１２、２０では表示形状を与えている。

local …１１、１９は、ローカルスロットの定義である。ここで与えた式がスロット式（この例では、中点の座標値を計算している）で、制約機能により、これらローカルスロットの値はスロット式に一致するよう制御される。

generate…２５、２９、３０により、テンプレートからインスタンスを生成する。ここで、横線 HL1、縦線 VL1 の生成時に、外枠 FR1 をパラメタスロットに設定することにより、オブジェクトリンクを実施している。

```
 1  template FRECT;
 2             Sx: ; Sy: ;
 3             Ex: ; Ey: ;
 4    part ,RECT, STARTx = Sx,
 5                STARTy = Sy,
 6                ENDx   = Ex,
 7                ENDy   = Ey;
 8  end FRECT;

 9  template HLINE;
10             FR: object FRECT;
11    local Yy: = (FR.Sy+FR.Ey)/2.0;
12    part ,LINE(1), P1x = FR.Sx,
13                   P1y = Yy,
14                   P2x = FR.Ex,
15                   P2y = Yy;
16  end FRECT;

17  template VLINE;
18             FR: object FRECT;
19    local Xx: = (FR.Sx+FR.Ex)/2.0;
20    part ,LINE(1), P1x = Xx,
21                   P1y = FR.Sy,
22                   P2x = Xx,
23                   P2y = FR.Ey;
24  end FRECT;

25  generate FR1, FRECT, Sx = 40.0,
26                       Sy = 40.0,
27                       Ex = 120.0,
28                       Ey = 100.0;

29  generate HL1, HLINE, FR = FR1;

30  generate VL1, VLINE, FR = FR1;
```

第４図　プログラム例１

第５図は、外枠、横線、縦線を一つのオブジェクトとして生成した例で、制約式による制約を用いている。

この例では、全体のテンプレートを FCROSS…３１という名称で定義している。ここで constraint…４８が制約部で、４９、５０で制約式を与えている。

ここでは、等式により制約を与えているため、スロット式による制約と機能的に同等であるが、制約式の場合には、不等式を与えてもよい。

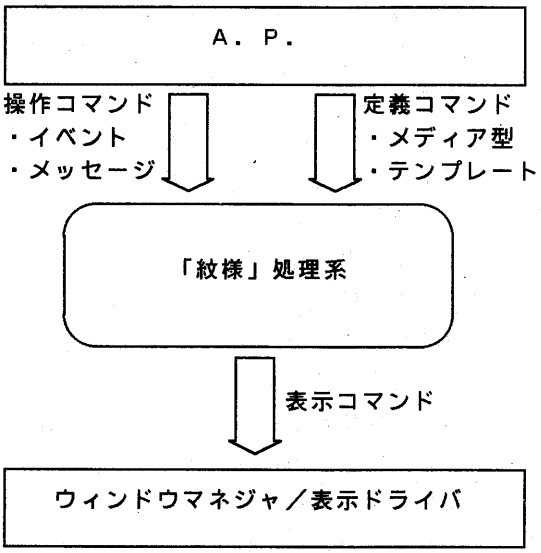
```
31  template FCROSS;
32             Sx: ; Sy: ;
33             Ex: ; Ey: ;
34    local Xx: ;
35    local Yy: ;
36    part ,RECT, STARTx = Sx,
37                STARTy = Sy,
38                ENDx   = Ex,
39                ENDy   = Ey;
40    part ,LINE(1), P1x = Xx,
41                   P1y = Sy,
42                   P2x = Xx,
43                   P2y = Ey;
44    part ,LINE(1), P1x = Sx,
45                   P1y = Yy,
46                   P2x = Ex,
47                   P2y = Yy;
48    constraint( Xx, Yy ) :-
49        Xx = ( Sx + Ex )/2.0,
50        Yy = ( Sy + Ey )/2.0;
51  end FCROSS;

52  generate FC1, FCROSS, Sx = 40.0,
53                        Sy = 40.0,
54                        Ex = 120.0,
55                        Ey = 100.0;
```

第５図　プログラム例２

４．処理系の概要

以下では、処理系の動作環境ならびに実装について述べる。

処理系の動作環境を**第６図**に示す。あるアプリケーションにおいては、そのすべてを紋様で記述することも不可能ではないが、一般的には、アプリケーションのうちマルチメディア関連の部分のみに紋様を利用することを想定している。また、対話処理のように、テンプレートやインスタンスを動的に編集することにも対応するため、独立した実行系として処理系を設計した。このようにすることにより、アプリケーション



第６図　紋様処理系の動作環境

は、「紋様」コマンドを用いて紋様処理系に依頼することで、必要な処理を実現できる。

「紋様」の機能を大別すると、定義機能（テンプレート、メディア型、関数の定義）と操作機能（インスタンス生成、メッセージ通信、表示等）に分類できる。前者による定義内容は、例えばテンプレート定義のように、実行中に変更がなく、かつ、インスタンスの生成のような操作にともなって複数回参照される。これらの実行の高速化と移行性・保守性の両立を狙って、仮想マシンを設定し、処理のトップレベルにおける１コマンド毎のコンパイル／即実行という方式で処理系を実現している。処理系の概要を**第７図**に示す。なお、以下では、紋様プログラムを仮想マシン語に変換する部分をコンパイラ、変換後の仮想マシン語を解釈実行する部分をインタプリタと呼ぶ。

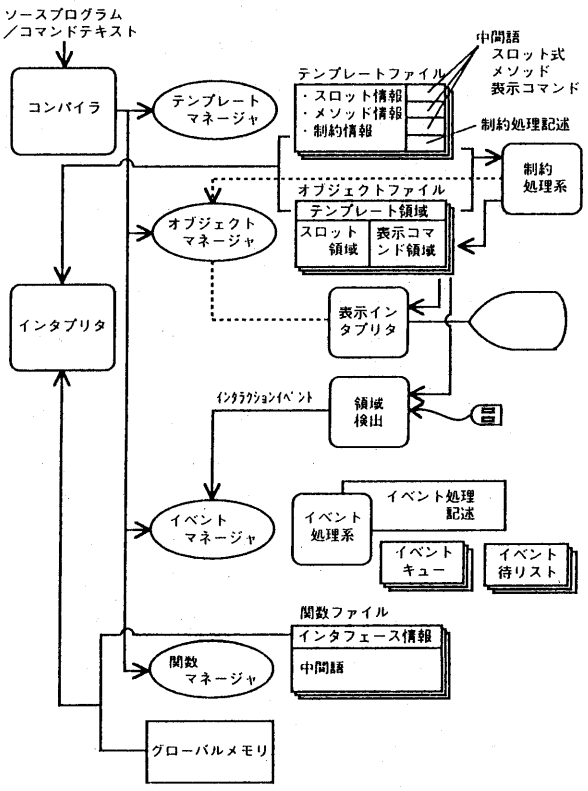
５．実装方式

５．１　コンパイラ

コンパイラでは、紋様プログラム構文解析を行ない、定義コマンドに対しては、定義内容を保持する内部テーブル類を生成し、操作コマンドや式に対しては、それを実行する仮想マシン語プログラムを生成する。内部テーブルには、テンプレートとメディア型の定義内容を保持するテンプレートファイル、定義された関数の定義内容を保持する関数ファイル、および、インスタンスの値を保持するオブジェクトファイルがあり、それぞれをマネージャと呼ぶプログラムが管理している。

（１）テンプレートマネージャ

テンプレートマネージャは、テンプレート定義、および、メディア型定義に対応してテンプレートファイルを作成管理する。テンプレートファイルには、インスタンスへのアクセスや値の設定に必要なすべての情報（下記）が設定されており、他のテンプレートのコンパイルやコマンドの実行時



第７図　紋様処理系の概要

に参照される。

1）スロット情報：スロット名、データ型、オブジェクトレコード（後述）内のアドレス（オフセット）およびメモリサイズ、スロットに個別に設定された制約スロット式を実行する仮想マシン語列。

2）メソッド情報：メソッド名、パラメタの仕様（パラメタ名、データ型、メモリオフセット）、メソッドを実行する仮想マシン語列／メディア型の場合オブジェクトレコード上に非ょ時コマンドを作成する仮想マシン語列。

3）制約情報：制約対象のスロットのリスト、制約式の中間語（後述）。

（2）オブジェクトマネージャ

オブジェクトマネージャは、インスタンスの内部データ構造であるオブジェクトレコードの生成・消滅・アクセス管理を行なう。オブジェクトレコードの集まり全体がオブジェクトファイルである。一つのオブジェクトレコードは、管理領域、スロット領域、表示コマンド領域からなる。

管理領域には、オブジェクトマネージャがテンプレートＩＤ，インスタンスＩＤ、およびメモリ管理ポインタを設定する。スロット領域には、実行中のスロットの値を保持する。また、表示コマンド領域には、メディア型に対応した表示コマンドが展開される。

この表示コマンドを表示インタプリタが実行することにより、当該インスタンスがディスプレイ上に表示される。また、各インスタンスに対するマウスのピッキングもこの表示コマンドに基づいて判定する。

スロット領域および表示コマンド領域には、スロット式やメソッドに対応する仮想マシン語の実行および制約処理によって値・表示コマンドが設定される。

（3）関数マネージャ

関数マネージャは、関数定義に対して、その関数と外部とのインタフェース情報、関数の処理内容の仮想マシン語情報を保持する関数ファイルを作成する。

トップレベルにおける関数評価では、関数の値はグローバルメモリに設定される。

（4）制約処理系

オブジェクトリンクにより、インスタンス間に依存関係が生じるが、あるインスタンスの値の再計算には、外部からの参照時点でその値を確定する「遅延評価方式」（後述）を採用し計算回数の最適化を図っている。制約処理には、スロット式による値の再計算と制約式による制約の解消の二つの側面があるが、いずれも、このインスタンスの値の再計算時に実施する。即ち、スロット式による制約は、遅延評価時点でスロット式を評価することにより、また、制約式による制約は、スロット式の評価後制約解消処理を実施することにより実現している。

制約解消処理は、方程式の解を求めるのと同様の処理であって、計算精度の維持のため、有理数領域（即ち、分数の形式）で計算を行う。このため、制約式に関しては、定義時の式をそのまま木構造中間語として保持し、その上で、定数伝播、線形方程式の求解、非線形方程式の線形化を適宜実施している。

（5）イベントマネージャ

イベントマネージャは、ユーザとの対話処理（キーボード／マウス入力とそれに対する応答処理）であるイベントの管理を行なう。イベント処理には、発生元（入力元）の名称と受け渡しするパラメタを定義するイベント宣言、ユーザやオブジェクトによって、キーボード、マウス等による入力によるイベント発生、がオブジェクトにおいて指定されたイベントの発生を待つイベント待ち等がある。イベント宣言に対しては、イベント種別の登録、イベント発生に対し

ては、イベントキューへの登録、イベント待ちに対しては、イベント待ちリストへの登録を行なう。イベントマネージャは、子プロセスとして動作し、イベントキューとイベント待ちリストとの照合とイベント待ちの解除を行なう。

5．2　遅延評価方式

紋様処理系における課題の一つは、上記オブジェクト制約機能におけるオブジェクトの値の再計算をいかに効率良く実現するかという点にある。

最も簡単な方法としては、処理系の内部で、更新通知メッセージの発行と更新通知メッセージの受理時における制約式の評価とを実施することであるが、以下のような問題がある。

（問題点１）例えば、あるオブジェクトに対して一連の変更を行なう場合、人間が作成するプログラムでは、その全ての更新が終了した後に更新通知メッセージの発行を指示できるが、インタプリタのような処理系ではこのような最適化は難しく、更新毎に通知メッセージを発行することになる。この場合、参照側では、制約式の評価を繰り返すことになるが、このうち最後の評価だけが有効で、それ以前のものは無駄となってしまう。

（問題点２）更新通知メッセージを送るための被参照側から参照側への逆リンクを処理系が生成・管理する必要があるが、この処理も面倒である。

以上のことから、紋様では「遅延評価方式」を開発した。

上記の（問題点２）を解消する手段としては、逆リンクを持たない、即ち、更新を特定のオブジェクトのみに通知するのではなく、ブロードキャストしてしまうことが考えられる。また、（問題点１）を解消する手段としては、被参照側先の更新時点で評価するのではなく、当該オブジェクト自身の値が必要となった時点で初めて評価する「遅延評価」を用いることが考えられる。

紋様で採用した遅延評価方式は上記の組み合わせをベースにしている。即ち、ブロードキャストの時点で即オブジェクトの評価を実施しないこと前提とすると、オブジェクトの値の評価を実施しようとする時点で、以前の評価時点からその時点までになんらかの更新が発生したか分かれば良いので、ブロードキャストの方法として、グローバルな更新フラグまたは更新カウンタを用いることができる。また、オブジェクトの参照時点で更新フラグをチェックし、更新が発生していたときにのみ再評価を実施することにより、評価回数の最小化が保証される。以下に概要を述べる。

今、任意のオブジェクトの更新によりカウントアップするグローバルな更新カウンタと、各オブジェクトに対してその値の評価時点における更新カウンタのコピーとが設定されているものとする。各オブジェクトにおいて、その値の再評価が必要が否かは、基本的に更新カウンタとそのコピーとの比較をすることにより判断できる。即ち、それらが一致していれば、更新が発生していないので再評価の必要がない。

しかし、あるオブジェクトの参照先とは無関係のオブジェクトの更新によっても更新カウンタはカウントアップされるので、グローバルな更新カウンタのみによる判定では無駄な再評価が行なわれる。

そこで被参照側（参照先）オブジェクトが本当に更新されているか否かの判定が必要となるが、これは被参照側の更新カウンタと参照側（参照元）のそのコピーとの比較によって判定できる。即ち、グローバルな更新カウンタの値が不一致であっても、参照先のオブジェクトの値が最新のもので

あって、かつ、参照関係のあるオブジェクト間のローカルな更新カウンタが一致していれば参照元の再評価が不要である。

5．3　仮想マシンとデータ形式

仮想マシンは、演算用スタックを持ち、四則演算をこのスタック上で行なう、スタックマシンアーキテクチャを採用した。

仮想マシン語には、固定長データに対する四則演算／比較演算等の演算命令、可変長データに対する生成／追加／結合／比較等の操作関数がある。

仮想マシンで取扱うデータ型は、整数型や実数型等の固定長データ、文字列型やリスト型の可変長データ、およびこれらを要素とする構造型データである。

固定長データと構造型データは、ベースアドレスと固定オフセットを用いたアドレッシング方法によりアクセス高速化を図っている。可変長データへのアクセスには、各データに固有のデータ情報テーブルを介して間接的に行なう。

可変長データの更新時には、新たなデータ領域を作成するため、更新以前のデータ領域は無効となる。また、関数等のローカルな可変長データのデータ領域も、その関数の評価後には無効となる。これら無効領域に対しては、ガーベジコレクションを実施している。

また、表示インタプリタやオブジェクトマネージャの起動を実行時に行なう必要があるため、外部の実機械語プログラムの汎用呼び出し機能も備えている。

以上から分かるように、可変長データの処理を除けば、既存の実マシンのコード生成を行なうよう改造するのは容易である。

6．終わりに

帳票、文書、図形、画像、音声等を取扱うことを主眼とし、処理対象を部品化しそれらを有機的に結合することによって、プログラミングを支援する「オブジェクトリンク機能」「オブジェクト制約機能」を特徴としたメディア操作言語「紋様」の言語の概要と処理系の概要について述べた。

＜参考文献＞

1）Maloney, John H.; Borning, Alan; Freeman-Benson, Bjorn N.: "Constraint Technology for User-Interface Construction in ThingLabⅡ", Proceedings of the OOPSLA'89 (Oct. 1989) pp.381-388

2）Szekeiy, Pedro A.; Myers, Brad A.: "A User Interface Toolkit Based on Graphical Objects and Constraint", Proceedings of the OOPSLA'88 (Sept. 1988) pp.36-45