記 号 処 理 50－2
（１９８９．３．１４）

ＥｕｓＬｉｓｐ
対象指向に基づくＬｉｓｐの実現と幾何モデラへの応用

松井俊浩　　稲葉雅幸
電子技術総合研究所　　東京大学工学部
matsui@etl.junet　　inaba@mech.t.u-tokyo.junet

ロボット、CAD などのための幾何モデラの実現を主たる目的として、単一継承型のオブジェクト指向に基づくCommonLispのサブセット、EusLisp を開発した。EusLisp では、コンスやシンボルなどの基本データ型もオブジェクトとして定義している。これらのデータ型にサブクラスを加えることで、システムの拡張が容易に行なえる。拡張されたデータ型に対しても効率よく基本関数を適用するための高速型判別法や、擬似的な多重継承を実現するためのメッセージフォワーディングの機能を実装している。幾何演算プリミティブを関数として用意し、幾何モデルの要素をオブジェクトとして定義することで、拡張性の高いソリッドモデラを簡潔に記述することができる。また、関数とシンボルの拡張で実現された他言語インタフェースを用いてライブラリをリンクすることにより、Lisp上のウィンドウシステムが容易に構築できる。

ＥｕｓＬｉｓｐ
An Object-Based Implementation of Lisp and Its Application to Geometric Modeling

Toshihiro MATSUI　and　Masayuki INABA
Electrotechnical Laboratory　　University of Tokyo
Umezono, Tsukuba-city, Ibaraki 305　　Hongo, Bunkyo-ku, Tokyo, 113
matsui%etl.jp@relay.cs.net

Aiming at the realization of a geometric modeler in Lisp for Robotics and CAD, an object-based subset of CommonLisp, EusLisp has been developed.
Since EusLisp defines all the data types except numbers in terms of objects, all the system facilities are highly extensible through the class inheritance. Based on EusLisp's object-orientation, a solid modeler which has capabilities for composing objects by set-operations and displaying hidden-line eliminated images in several window environments, has been built. The greatest advantage over the traditional modelers is the extensibility to the higher level applications in robotics field.

1．Euslisp の目的と特徴

高性能ワークステーション上のCommonLisp[1] によって、Lispの実用的応用分野が広がりつつある。ロボットの研究もそのような応用分野の１つである。

ロボットの研究では、作業・動作のプラニング、経路の探索、ワールドモデルの管理、環境の認識、マンマシン・インタフェースなど種々の要素技術とそれらを統合するシステム化技術が重要になる。これらの問題に共通的に必要な機能は、３次元物体の形状と振る舞いを記述するための幾何モデラである。幾何モデラには、頂点、稜線、面などの間の位相関係を記述する能力と、浮動小数演算を含む幾何的計算を効率よく処理する能力が要求される。さらに、幾何モデルを認識や作業計画に応用するには、幾何モデルは拡張を加え易くなければならず、多くのプログラムの統合を図る機能が必要になる。

これらの要求に応えるために、オブジェクト指向をベースにしたCommonLispのサブセット†、Euslisp を開発した。Euslisp は、①オブジェクト指向による高い拡張性[8] 、②幾何演算機能と幾何モデリング[9] 、③他言語インタフェースによる外部プログラムの統合とウィンドウ環境の実現[10]、などを特徴とする。特に、システムの基本部分をオブジェクトによって構成しているので、それらのサブクラスを定義する形で豊富な機能を統合することに成功している。また型の判別法を工夫することで、オブジェクト指向を導入することによる効率の劣化も最小限に抑えている。

本論文では、Euslisp のオブジェクト指向、メモリ管理を含む内部構造、幾何モデリング機能、他言語インタフェースを用いたウィンドウシステムについて述べる。

2．Euslisp のオブジェクト指向

2.1 オブジェクト指向とLispの型拡張

Lispは、古典的な記号・リスト処理言語から、豊富なデータ型を扱う言語に発展して来ており、今後も型追加の要求は続くものと思われる。システムの基本データ型に対してクラスの継承によるサブタイプ化が可能であれば、システムのインプリメントとアプリケーション開発の両方が容易になる[2] 。このような型拡張を、組み込み型とユーザ定義型の間でトランスペアレントに行なうために、Euslisp では数値以外のデータをすべてオブジェクトとして表現している。defstruct,CLOS[3] などと異なるのは、コンス、シンボル、文字列などのLispの組み込みデータ型もオブジェクトであり、ユーザが定義するオブジェクトと本質的に同一の構造を持つ点である。一方、数をオブジェクトとしなかったのは、数値演算の効率を重視したためである。

組み込み型の追加によってシステムの機能が拡張される例をいくつか上げよう（図１参照）。多バイト文字列は単純文字列の、ウィンドウはストリームの副型として定義できる。ストリームに対しては、ファイルと文字列を別クラスとし、さらにIPC 用にパイプ、ソケット、メッセージキュー等をストリームに加えることができる。シンボルは属性付きオブジェクトのサブクラスとなっている。これによって、属性リストの管理がシンボルから分離され、シンボル以外のオブジェクトも属性リストを利用できるようになる。コンスは、使用目的に応じて、alist,set,stack 等を派生させることができる。alist やset に対しては、assoc,member関数の:test 引数に応じたクラスを作成できる。これは、ハッシュ表がtest関数をmake-hash-table の時点で指定していることを考えれば何ら不自然ではない。クロージャはコンパイルドコードにレキシカルな環境の情報を加えたオブジェクトとして定義される。この他に４章、５章で述べるように、幾何モデル要素の階層化や他言語インタフェースのための拡張にもクラスの継承は有効に機能している。

defstruct によるユーザデータ型定義と異なり、Lispの組み込みデータ型を拡張できれば、Lispシステムの資産をフル活用でき、システム自体の拡張性が高まる。さらに、Lispの組み込み型が外部にオープンになっているため、Lispのインプリメントが容易になる。つまり、組み込み型を増やすにつれ、要素にアクセスする関数を多数用意しなければならなくなるが（symbol-value,symbo

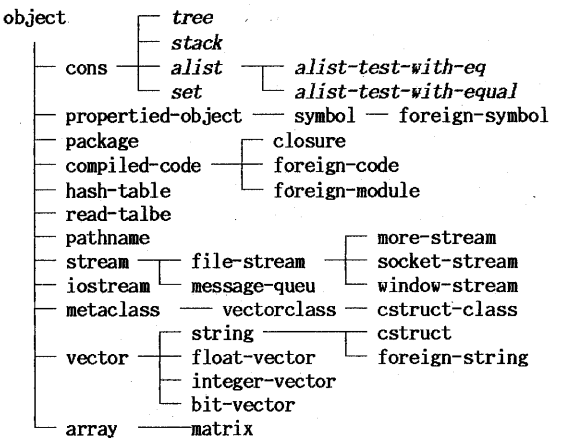

図１．基本クラスの階層構造（斜体は非組み込み型）

†CommonLispの機能でEuslisp に欠落しているのは、多値、無限エクステントを持つクロージャ、複素数、有理数、大数、deftype などである。

l-function,symbol-package などと、それらのsetfマクロ) Euslisp のアプローチではこれらはすべてEuslisp自身で記述可能である。

クラスの定義はdefclassマクロで、メソッド定義はdefmethod 特殊形式で行なう。メタクラスやクラスメソッドを定義することも可能である。インスタンスの作成はinstantiate 関数で、メッセージの送信はsendとsend-super関数で行なう。

2.2 クラスの階層に則った型判別法

継承による型の拡張を許す場合、基本型に対して定義された関数は、副型に対しても適用できるべきである。たとえば、ストリームのサブクラスであるsocket-stream に対してもread,print等の関数はそのまま適用可能でなければならない。そのために、すべての関数をメッセージ送信に置き換えたり、CommonLispのようにdeftypeによって任意の型のコンビネーションを図る方法は柔軟性に富むが、実行に探索を伴い、インライン展開が困難になるので効率上得策とは言えない。これに対し、型の階層を単一継承に限れば、次のような効率的な型判別が可能である。

オブジェクトには、元になったクラスを示す指標としてcid(class-id; 0..65535) を記録しておく。また、クラスには、自分のcid と共に自分のサブクラスのcid のうちで最大のもの (maxsubcid)をも記録しておく。オブジェクトxが、クラスCまたはCのサブクラスから生成されたことを検査するためには、xのcid がCのcid とmaxsubcid の間にあるかどうかを調べればよい。すなわち、C.cid ≦x.cid ≦C.maxsubcid 。

このような検査法を可能にするために、クラスCの順序付けられた直接のサブクラスC_1 ..C_n のcid,maxsubcid を以下のように定める (図2参照) 。

サブクラスなし　C.maxsubcid = C.cid

サブクラスあり　C_1.cid=C.cid+1

C_{i+1}.cid=C_i.cid +1

C.maxsubcid= C_n.maxsubcid +1

この関係を保つため、継承木の下端以外に新しいクラスが登録されるときはcid の再割り付けが必要になる。この処理には全オブジェクトのcid の書き換えが含まれるので、オブジェクトの総量に比例した時間を要する。しかし、クラス定義は頻繁に起こるものではないので大きな問題とはならない。定義時にこのような手間をかけておけば、実行時には簡単なレンジチェック命令だけで型判別が可能となる。実際、クラスAがクラスBのサブクラスであるかどうかを調べるsubclassp,オブジェクトxがクラスAまたはそのサブクラスから導出されたかどうかを調べるderivedp関数はコンスタントタイムで実行される。

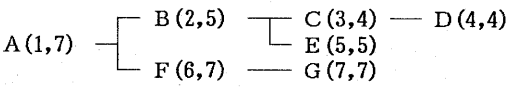

図2．クラスｉｄの割り付けの例
A(1,7) はクラス名＝A，cid=1，maxsubid=7

2.3 メッセージ・フォワーディング

Euslisp では、システムの根底にオブジェクト指向を据えている。オブジェクト指向と高い効率を発揮するLisp処理系を共存させるためには、前節で述べたような型の判別法が重要であるが、本手法を多重継承に拡張するのは困難であり、単一継承を採用せざるを得ない。Euslisp の前身となった、多重継承型オブジェクト指向プログラミング機能LEO[7] での経験では、①多重継承ではスーパクラスのどのメソッドが実行されるかを常に意識せねばならず、スーパクラスの組み合わせ方が難しい、②is-aの関係でなく、part-of の関係の方が柔軟な構成を取れる、などが明らかになっていた。このように、必ずしも多重継承がよいと言い切れるものではないが、ときに多重継承によれば簡潔に記述できる問題があるのも事実である。また、単一継承だけでは、vectorが sequence とarray の両方のサブタイプであるという構造を表せない。

Euslisp では、単一継承の簡潔性と高い効率を保ちつつ多重継承の機能を取り入れるため、メッセージフォワーディングの機能を実現している。これは、あるオブジェクトにメッセージが送られ、それがそのクラスで処理できないことがわかると、:forward指定されたスロットにメッセージを送り直す機能である。これは、サブクラス、スーパクラスの間で変数を共有しないような多重継承と等価な機能を提供する。図3の例ではpresident に送られた:telephone,:mailメッセージはsecretary に、:go-homeメッセージはchauffeur に自動再送信される。

```
(defclass person :super object
        :slots ((name :type string)
                (age :type :integer)))
(defclass president :super person
        :slots (secretary :type person
                    :forward (:telephone :mail))
               (chauffeur :type person
                    :forward (:go-home))))
```

図3．メッセージフォワーディングを用いた
クラス定義の例

3．ポインタ、オブジェクトの構造と記憶管理

3.1 ポインタの構造

ポインタは32ビットの長語で表現される（図4）。メモリセルを長語境界に置き、下位2ビットをタグ、上位30ビットをアドレスまたは数値とすることで、4Gbyteのアドレス空間が確保される。タグは、00が整数、01が浮動小数、10がポインタ、11は未使用を意味する。整数は$\pm 10^9$のレンジを持ち、浮動小数の計算機エプシロンは10^{-6}程度となる。数値の長精度形式はない。このポインタ表現には次の特徴がある。

①整数の加減算では、タグを無視してよい。

②ポインタとして解釈する時は-2のオフセットを加えればよく、タグをマスクする操作は不要である。

③ポインタが指すオブジェクトのクラスを表すためにオブジェクト側にタグを付ける必要がある。

④即値のレンジが広く、数値演算によってメモリを消費することがない。

3.2 オブジェクトの構造

オブジェクトは、図5に示すのような1語のヘッダとそれに続く変数・要素で構成されるセルによって表現される。ヘッダのフィールドは次のような意味を持つ。

bmgps: メモリセル管理情報

b: buddy-bit

m: memory-bit

g: ごみ集め用マーク

p,s:環状オブジェクトのプリント、コピー用マーク

elmt: ベクタ要素型

0: non-vector 1: bit 2: byte 3: char

4: integer 5: float 6: foreign 7: pointer

bix: buddy-index（バディセルの大きさの指標）

cid: class-id （オブジェクトの属するクラスid）

オブジェクトは、ベクタ型と非ベクタ型に分けられる。ベクタ型は要素の数が可変であり、インデックスでアクセスされる。非ベクタ型は要素（スロット）の数が固定（クラスが定める）であり、スロット名でアクセスされる。ベクタ型オブジェクトの場合は、ヘッダのelmtフィールドが要素の型を表し、文字、整数、浮動小数などに対しては効率的かつCなどと互換性のある表現がとられる。ヘッダの次の語には要素の数を示す数値が入る。

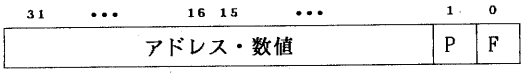

図4．ポインタの構造

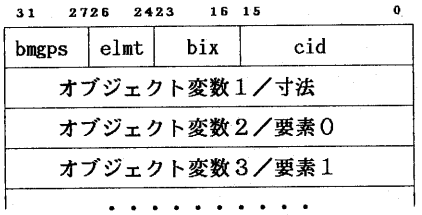

図5．オブジェクトの構造

3.3 メモリ管理

メモリの管理手法は、Lispの柔軟性と効率に大きく影響する。Euslisp では単一のヒープ上でのバディ法を用いたメモリ管理を行なっている[5]。バディ法を採用したのは、メモリの動的拡張を可能にするのと、コピーによる圧縮の必要性を除くためである。特に後者は、①コンパイルドコードが他のオブジェクトと同一のヒープに配置されること、②他言語プログラムとのリンクを図る上でセルの番地が変化することは容認できないこと、③ごみ集めの効率向上、④ごみ集めを抑制するためにユーザが消費したメモリを陽に返却する機能の実現、などの要請に基づいている。ただし、参照が広い領域に散らばるので、仮想記憶管理の負担は増加する。

バディ法の基本アルゴリズムは次の様になる。

①メモリが要求されると大きなセルを分割し、要求を満たす最小のセルを作成して返す、②ごみ集めによって不要なセルが見つかると、その前後のバディセルを検査しそれらも不要なセルであれば併合する、可能ならば併合を再帰的に繰り返し、より大きなセルにして回収する。

分割と併合が簡単に処理できるよう、バディ法ではセルの大きさを2のべき、フィボナッチ数などに制限する。前者をバイナリバディ、後者をフィボナッチバディと呼ぶ[4]。どちらが適当であるかは、要求されるメモリサイズの分布に依存する。Euslisp で幾何モデラを走らせた場合、図6に示すようなメモリの要求が生じている。これでわかるように、メモリ要求は3語（主にcons）、6語（主に3次元float-vector）のセルが圧倒的に多く、次のような考察に基づいてフィボナッチバディを採用している。

まずメモリ効率について考える。セルの大きさの種類に制限があるためにセルの内部に生ずる無効領域を内部ロス、セルが散在するために連続領域が取れなくなることによる無効領域を外部ロスと呼ぶ。フィボナッチバディはバイナリバディに比べて、外部ロスが大きいとされるが、要求・消費の多いconsセルのサイズが最小なので外部ロスは0になる。内部ロスは、セルの大きさの種類を増やすに従って減少する。フィボナッチの方がバイナ

buddy	size	free	total	total-size	wanted	wanted-size
1	3	53365	98568	295704	659833	1979499
2	6	2744	29116	174696	332272	1993632
3	9	167	8401	75609	37480	337320
4	15	67	254	3810	582	8730
5	24	29	42	1008	51	1224
6	39	59	68	2652	42	1638
7	63	49	52	3276	12	756
8	102	14	28	2856	58	5916
9	165	4	8	1320	12	1980
10	267	6	14	3738	14	3738
11	432	2	4	1728	8	3456
12	699	4	5	3495	8	5592
13	1131	1	7	7917	8	9048
14	1830	2	4	7320	5	9150
15	2961	2	13	38493	13	38493
16	4791	1	6	28746	6	28746
17	7752	0	5	38760	5	38760
18	12543	1	3	37629	2	25086
19	20295	0	1	20295	1	20295
20	32838	0	1	32838	1	32838
21	53133	0	0	0	0	0
22	85971	0	0	0	0	0

図６．幾何モデラ実行時のメモリ使用状況
(sys:memory-reportの出力)

リより多くのセルサイズを作れるので、PetersonとNorma[5]の解析では、バイナリ・バディの場合は約30% のロス、フィボナッチ・バディでは約22% の内部ロスが予測されている。さらに、consセルの割合が多い性質を利用して、consにぴったりの大きさのバディサイズを用意しておけばこのロスをさらに低減できる。実測では、コンパイルドコード（相対的に大きいセル）を除くと10% 程度のロスであることが確認された。これは、コピー法による記憶管理での50% のロスに比べて非常に効率的な値である。

バディ法とBIBOP 法を比べると、バディ法ではすべてのページが平等で管理法が均質であり、異なった大きさのセルの間で融通がきくかわりに、メモリ割り当ての際の分割、ごみ集めの際の併合の手間が余分にかかる。ところが、ごみ集めの後で要求されるセルはconsであることが多く、せっかく大きな連続領域に併合されたセルも再び極限まで分割されることになる。この不合理を軽減するため、メモリの一定量（５割程度）は併合をしないで分割されるに任せておく。しばらくたつとその中はほとんどがconsで埋め尽くされ、consのための特別のページを用意しなくとも、効率的な割り当てが実現できる。

４．幾何モデリング

4.1 Lispと幾何モデラ

幾何モデラとは、基本素立体の集合演算による３次元物体の形状定義、移動・回転などの座標変換、属性の定義と問い合わせ、表示などの機能を持ったソフトウェアである。ロボットの環境認識、作業計画、動作の教示とシミュレーションなどで幾何モデルは重要な役割を演ずる[6]。従来、幾何モデラはFortran で実現される例が多かった。しかし、拡張性が低い、大きなモデルを扱えない、モデルデータへのアクセスが困難などの問題があり、ロボットシステムへの統合化が阻まれていた。

これに対して幾何モデラをオブジェクト指向型Lispで実現することの利点は次のようにまとめられる。

①モデル要素はオブジェクトで、位相関係はポインタによって明確に表現できる
②オブジェクトによってモデルの振舞いが抽象化される
③クラスの継承を用いることで属性の追加が可能
④モデルの保存・通信（外部表現への変換）が容易
⑤メモリ管理から解放され容量的制限が無い
⑥特別のコマンド言語を作成する必要がない。

Lispは数値計算には向かないとの意見もあるが、ベクタ、マトリクスなどの幾何データとその演算プリミティブを組み込み関数とし、コンパイラを利用すれば処理速度の点でも大きな問題とはならない。プリミティブとして定型化できない処理の大部分はポインタ操作であり、Lisp本来の機能によって簡潔に記述できる。

4.2 ベクタ・マトリクス演算と座標系

幾何計算ではベクタ・マトリクスの演算が多く登場するので、Euslisp では浮動小数の１次元配列をフロートベクタ、浮動小数の２次元配列をマトリクスとして定義し、それらの間の演算を行なう組み込み関数を多数用意している。

ベクタ演算関数には、和、差、内積、外積、スカラ３重積などがある。マトリクス関数には積、変換、転置、回転、等価回転軸の取り出し、LU分解による逆行列、行列式などがある。また、行列の線型計算については、５章で述べる他言語インタフェースによりlinpack 等の数値計算ライブラリをリンクすることができ、固有値、特異値分解などにも対応することができる。

原点位置と回転マトリクスの組合せで座標系を表すために、クラスcoordinates が用意されている。coordinates はワールド、ローカルあるいは任意の座標系で表現された移動、回転、変換、逆変換などを受け付ける。さらに、階層的に結合された座標系を表現するクラスとしてcascaded-coords がある。これによって多関節型のマニピュレータや部品と部品を取り付ける操作等を簡単にモデル化することができる。これらのcoordinates,cascaded-coords は、視点を表すviewing,物体を表すbodyなどのスーパクラスとして汎用的に用いられる。

4.3 オブジェクトによるモデル表現

物体は多面体で近似され、Brep(Boundary-representation) で表現される。edge,face,hole,body などのモデル要素は、図７のような継承構造を持つクラスによって階層的に定義される。vertex（頂点）は３次元フロートベクタで表される。オブジェクトのスロットの内容を図

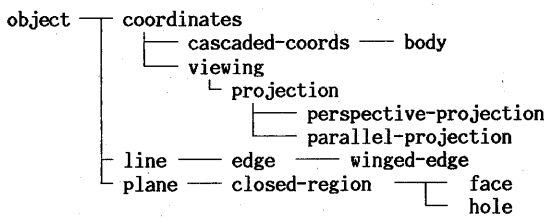

図7．幾何モデルのクラス構成

body	face,(hole)	edge
座標系	面方程式	始点
最小ボックス	エッジリスト	終点
面リスト	頂点リスト	右面
エッジリスト	凹凸フラグ	左面
頂点リスト	穴リスト	
凹凸フラグ		

図8．幾何モデルオブジェクトの構造

8に示す。モデル要素の参照関係は再帰的になる。

オブジェクトの状態に基づく計算法、状態の変更、機能名は同じだが対象物によって意味が異なるアルゴリズム（交点計算、点の内外判定など）はメソッドとして定義している。一方、n個の要素に対する対等な操作や新たな要素を生成する操作（干渉検査、点列からの凸包の生成など）は関数として実現している。

サブクラス定義による属性追加は容易である。たとえば、ウィングを持ったエッジが必要であれば、edgeクラスの代わりにwinged-edge クラスを用いればよい。グローバル変数*edge-class*にwinged-edge をセットしておけば、以後生成されるエッジはwinged-edge のインスタンスになる。同様に面にcolor 属性を付加したければ、colored-faceをfaceのサブクラスとして定義すればよい。

4.4 形状定義と合成操作

形状は、素立体を元に和、差、積、反転、切断などの集合演算を施すことで定義される。これらの操作はクラスに定義された交点計算メソッドを用いた関数として実現されている。また、従来使用されてきたソリッドモデラとの互換性を保つために、SOLVER[6] の内部表現をEuslisp のオブジェクト構造に変換する方法も用意されている。

集合演算などの変形操作では、データ構造をトラバースしながら多くの交点計算、エッジセグメントの作成が実行される。従来の幾何モデラには計算の途中結果をデータ構造の中にマークとして残したり、ポインタを破壊的に置換するものがあるが、Lispでは途中結果をリストにつないでいく方法が取れるので、より安全で明確なアルゴリズムが実現できる。さらにcopy-object 関数により再帰的参照関係を保存したセルのコピー法が実現されているので、非可逆的なモデル操作における状態の保存・復帰が容易である。

4.5 属性計算

形状オブジェクトに対しては、干渉検査、凸包の生成（図9b）、断面図形の生成（図9c）などが実現されている。さらに体積、重心などのマスプロパティが計算できる。これらはいずれもロボットの動作シミュレーションや作業計画にとって重要である。たとえば重心と凸包を用いれば物体の安定姿勢が算出可能であり、環境モデリングの拘束条件に利用できる。

4.6 隠線処理、表示

最近のグラフィックワークステーションは非常に高速で隠面処理表示を行なうが、アプリケーションから表示される面の情報にアクセスすることはできない。視覚認識やスーパインポーズ表示のためにはソフトウェアによる隠線処理は依然として重要である。Euslisp の隠線処理ルーチンは結果としてimage オブジェクトを生成し、図形の領域、エッジ素片に対するシンボリックなアクセスが可能になる。image はSunview,Suncore,Xwindow 等を利用して表示される[10] (図11) 。

4.7 オブジェクトベース機能

ソリッドモデリングによって作成されるデータ構造は

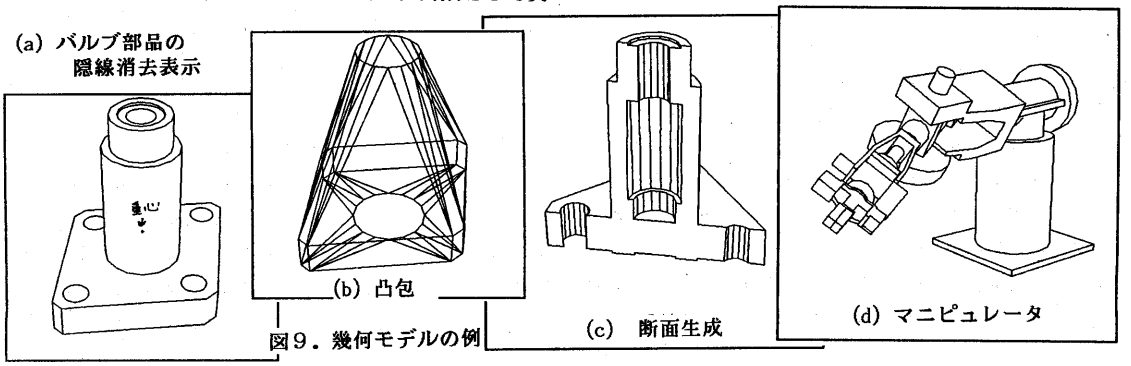

図9．幾何モデルの例

複雑な参照関係を持ち、作成に時間がかかるので保存の必要がある。グラフィックス標準ではメタファイルが検討されているが、拡張属性の入出力、再帰的参照関係の復元は厄介な問題である。Euslisp では、#Sフォーマットによるオブジェクトのread/print, #n=,#n# ラベルによる再帰参照の解決により、CommonLisp間で互換性のある外部表現との相互の変換が可能である。これによりモデル、図形の簡便なデータベース化が達成できる。さらに、計測、モデル生成、表示、計算などの作業を複数のEuslisp で分担し、socket-stream を通じてモデルを送受しあうような分散型のロボットモデルベースシステムの構築が可能となる。

5．他言語インタフェースとウィンドウシステム

5.1 他言語インタフェース

UNIX上ではLispだけですべてのプログラムを開発するのは困難である。UNIXはC用のOSであり、UNIXに強く依存する部分はCで記述する方が簡潔に記述できるのと、バイナリで供給されるライブラリやパッケージを利用する必要があるからである。そこで、Euslisp はCやFortran で書かれたプログラムと双方向のリンケージを取る手段を実現している。

Euslisp のカーネルはCで記述されている。また、Euslisp コンパイラはLispソースをCに変換する。したがって基本的にEuslisp とCはよく親和する。Euslisp コンパイラの生成するCプログラムと、通常のCプログラムとの主な違いは、①関数のシンボルへの登録（初期化）、②引数と結果の型変換、の２点にある。①はdefforeignによってロード時に、②はfuncall によって実行時に解決される。

Euslisp がコンパイルしたプログラムはloadによって読み込まれ、compiled-code のインスタンスが作られると共にプログラム中の初期化ルーチンが実行されるのに対し、一般のCプログラムはload-foreignによって読み込まれ、単にforeign-moduleのインスタンスが作成される。foreign-moduleはcompiled-code のサブクラスであり、codeとしての性質の他に、モジュール中の外部シンボル表を属性に持っている。このシンボル表を参照して、defforeignマクロがLispからC関数へのエントリを付ける。defforeignには、外部関数のパラメタと結果の型を指定する。Lispのdefun がシンボルにcompiled-code オブジェクトを登録するのに対し、defforeignはシンボルにforeign-codeを登録する。foreign-codeはやはりcompiled-code のサブクラスであり、関数へのエントリ番地のほか、パラメタと結果の型が記憶される。

```
/* a C function in "sync.c"*/
float sync(x)
double x;
{ extern double sin();
  return(sin(x)/x);}

eus$ cc -c sync.c
eus$ (setf m (load-foreign "sync.o"))
eus$ (defforeign sync m "_sync" (:float) :float)
eus$ (sync pi)  -->0
eus$ (defforeign sprintf *eus-module* "_sprintf"
          (:string :string) :string)
eus$ (sprintf (make-string 20) "%d+%d=%d" 1 2 3)
     --> "1+2=3"
```

図10．Cプログラムのロード、定義、実行

funcall は、コードオブジェクトの型を見て、foreign-codeであれば、引数をLispのポインタからCの要求する型に変換する。実際は、Lispのポインタからは実行時に型の情報が抽出できるので、defforeignの引数の型指定は省略できる。Cから返される値の型は常に不明なので、結果型の情報は省略できない。型変換に当たって、Euslisp のstring,integer-vector,float-vectorの内容表現はCやFortran での表現に近いので、変換のオーバヘッドは僅かである。図10に、Cのプログラムをロードし、エントリを定義し、実行する例を示す。

CからEuslisp の関数を呼び出せるようにするためには、次のdefun-c-callableマクロを用いる。

```
(defun-c-callable  funcname
    (((param type))*)  result-type   . body)
```

defun-c-callableは、symbolのサブクラスであるforeign-symbolのインスタンスを作成する。foreign-symbolには、Lispの関数としての属性の他に、Cから呼ばれた場合にCからLispへの引数の変換を行なうルーチンに分岐する機械語列と、引数と結果の型の情報が格納されている。この機械語列の番地をCプログラムに知らせておけば、CからLisp関数を呼び出すことができる。defun-c-callableの機能は、ウィンドウサーバから送られて来るマウスのイベント処理などに必要となる。

LispからCのstructへアクセスするためにcstruct クラスがある。cstruct は、stringのサブクラスとしてdefcstructマクロによって定義される。さらにCがmalloc等で確保したメモリにLisp側からアクセスできるよう、stringのサブクラスにforeign-stringがある。foreign-stringはLispからはベクタとして扱われるが、実体はLispのメモリ空間外に取られることになる。

5.2 ライブラリのリンク

他言語インタフェースが威力を発揮するのは、linpac

k,eispack などの数値計算パッケージや、Xwindow,Sunview などのウィンドウシステムを利用する場合である。これらのライブラリは非常に多くの関数を定義しているので、Cでインタフェースを書き下すよりは、他言語インタフェースを利用する方がはるかに簡便である。ただし、いくつかの制約が加わる。

linpack は元来Fortran 用なので、パラメタの渡し方がcall-by-reference である。このため、整数、浮動小数の即値を渡す場合にはいったんベクタに値を格納しなければならない。また、Sunview はかなりの機能を関数ではなくマクロで供給しており、ユーザはマクロを使った関数を別に定義する必要がある。Xwindow にはマクロと関数の両方の定義があるので、このような手間は皆無である。単にXlibをリンクするだけで、400 近いXlib関数がLispから利用可能になる。この方法は、CLX のようなXのプロトコルをLispで実装する方法に比べて、Xlibの改変に強く、効率も劣化しないという利点がある。

5.3 ウィンドウシステム・インタフェース

Euslisp 上にXwindow,Sunview,環境を構築した。Xでは,drawable,window,pixmap,graphic-context などをクラスとして定義している。Sunview では window,frame, canvas, panel, tty, button, slider,menu など18個のクラスを定義している。各々のオブジェクトには、マウスのクリックやメニュの選択によって非同期に実行されるメソッドが、defun-c-callableによって登録される。この非同期処理によって１つのEuslisp プロセスに複数のウィンドウから並列に指令を出すことができ、良好なユーザインタフェースが構成できることが確認できた。図11は、Sunview を用いたロボットビジョンシステムのユーザインタフェースの表示である。

これらのクラスおよびテクトロ端末用のview-surfaceクラスを適当に選ぶことで、上位のソフトウェアに手を入れることなく、多くの描画用ハード・ソフトに対応できるようになっている。

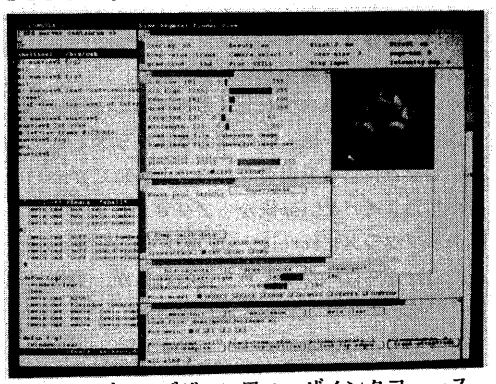

図11. ロボットビジョン用ユーザインタフェース

6．まとめ

ロボットプログラミングの核となる言語、Euslisp のオブジェクト指向に基づくインプリメンテーション、幾何モデラ、他言語インタフェースを用いたウィンドウシステムとの結合について述べた。クラスの継承により、システムの機能を拡張する形で、多くの機能を簡単に実現することができた。

Lispとしての性能はインタプリタではKCL より数10%高速であり、コンパイルドコードではKCL と同程度である。幾何計算の関数が多数用意されているので、幾何モデラとしての効率はSOLVER等と遜色ない。幾何モデラ部分のプログラム量は千数百行であり、１人月程度で開発できた。モデルのデータ量に制約がない点、モデルの情報にアクセスできる点により、オブジェクト指向による高い拡張性を有する点などから、高次のロボットプログラミングに応用できると考えられる。EUSlisp は現在sun3,sun4,News,vax/ultrix などで稼働中であり、ソースコード付きで公開されている。

謝辞　本研究を支援下さった、電子技術総合研究所、弓場敏嗣知能システム部長、柿倉正義自律システム研究室長、高瀬国克行動知能研究室長に感謝致します。また貴重なご意見をお寄せ下さった電総研、東京大学、神戸大学、三洋電機のEuslisp ユーザの皆様に感謝致します。

参考文献

[1] Steel Jr.,G.L.: "Common Lisp the Language," Digital-Press,(1984).
[2] Lang,K.J. and B.A. Pearlmutter: "Oaklisp: An Object-Oriented Scheme with First Class Types" OOPSLA,(1986)
[3] Bobrow,D.G.,et al: "CommonLoops: Merging Lisp and Object-Oriented Programming,"OOPSLA,(1986)
[4] Cranson,B. and R.Thomas:"A Simplified Recombination Scheme for the Fibonacci Buddy System," CACM, vol.18, no.6, (1975).
[5] Peterson,J.L. and T.A.Norman:"Buddy Systems," CACM, vol.20, no.6, (1977).
[6] Koshikawa,Shirai:"A 3-D Modeler for Vision Research," Proc. of Int. Conf. Advanced Robots, (1985).
[7] 松井、他：「Lisp上の対象指向型プログラミング機能LEO 」、電総研彙報、vol.49,no.7,(1985).
[8] 松井、塚本：「Euslisp:対象指向による型拡張性を有するLispの実現」、情報処理学会第35回全国大会(1987).
[9] 松井、稲葉：「対象指向型Lisp:Euslispを用いたロボット用幾何モデリングシステム」、情報処理学会第37回全国大会、(1988).
[10]稲葉、松井：「対象指向型Lisp:EUSlispの他言語インタフェースとウィンドウシステムへの応用」、情報処理学会第37回全国大会、(1988).