記 号 処 理 57－6
（1990．11．22）

Kamui環境におけるオブジェクト記述言語について

浜田 昇　　原田康徳　　渡辺慎哉　　宮本衛市

北海道大学工学部

我々の研究室では、ネットワーク上の計算機資源の有効利用を目指して、並列オブジェクト指向計算モデルに基づく分散アプリケーション開発のための Kamui 環境を構築中である。本稿では Kamui 環境におけるオブジェクト記述言語の設計思想、言語仕様および環境内での言語プロセサの動作について述べる。

Object describing languages in the Kamui environment

Noboru Hamada　　Yasunori Harada
Sin-ya Watanabe　　Eiichi Miyamoto

Department of Information Engineering
Faculty of Engineering
Hokkdaido University
Nishi 8, Kita 13, Kita-ku, Sapporo 060, Japan

For the aim of effective use of computation resources on a network, we have been developing the Kamui environment which is based on the parallel object oriented computation model. In this paper, we outline a design philosophy of object describing languages, their specification, and the behavior of language processors in the environment.

1 はじめに

近年のハードウェア技術の発達により、高性能ワークステーションが比較的安価に入手できるようになり、複数のワークステーションがネットワークで結合されたLAN環境が身近になった。

このような背景のもと、ネットワーク上の様々な資源(メモリ、ディスク、CPUパワー、言語プロセッサ等のアプリケーションおよび特定のマシンに固有なハードウェア等)をネットワークを通して有効に利用したいという要求が高まっている。また、最近では複数の人々によるコンピュータを利用した協調作業(CSCW)のように、本質的に分散プログラムであるようなアプリケーションが求められている。

しかし、分散アプリケーションの構築は、分散問題に特有なメッセージの遅れやプログラム実行に必ずしも再現性があるとは限らないといった問題を含み、プログラムの作成は決して易しいものではない。

我々はこのような背景をもとに、ネットワーク資源の有効利用及び分散協調問題の表現を目的にして、並列オブジェクト指向モデルに基づいた分散環境Kamuiを構築中である[1]。Kamui環境は分散環境でプログラムを作成する際に、プログラマのワークベンチとなる。また、問題を複数の言語で記述する事を可能にし、ネットワーク上の資源の利用を容易にする。

本稿ではKamui環境について、特にオブジェクト記述言語の設計思想、言語仕様および環境内での言語プロセッサの動作を中心に述べる。

2 目的別言語の必要性

コンピュータ科学の歴史において各種の言語が考案され作成されてきたが、現在の状況を見ると、ただひとつの言語が全てを包含するには至っていない。これは優れた言語が存在しない事を示すものではない。プログラミングモデルの違いによって言語が使い分けられているのである。

対象とする問題が大規模になり、複数の問題領域にまたがる場合には、問題全体を単一の言語では記述しきれない。問題領域ごとに適切な言語を用いて記述し、それらを統合化する方法をとる方が自然で簡潔な問題記述ができる。

複数の言語による問題記述を可能にするために次のようなアプローチが考えられる。

1. 共通のオブジェクトコードによる解決

   複数の高級言語を用いて問題を記述し、コンパイラがプログラムを各言語に共通のフォーマットを持つオブジェクトコードにコンパイルしたあと、リンカによってひとつのプログラムにまとめあげる方法がある。従来から大型計算機上で行われてきた方法である。

2. マルチパラダイム言語による解決

   ひとつの言語のなかに複数の言語の要素を取り込んだマルチパラダイム言語がある。例えばTAOは、Lisp、Prologおよびオブジェクト指向プログラミングの要素を取り込んだ言語である。このようなマルチパラダイム言語においては、問題によって記述の方法を変える事ができる。また、大規模な問題を記述する場合には、問題の各部分をそれぞれ適切な記述スタイルを用いて書く事ができる。

3. オブジェクト指向による解決

   Cコンパイラ、Lispインタプリタ等の複数の言語プロセッサが同一のモデルに基づくオブジェクトを生成することによって、記述言語が異なっていても、生成されたプログラムの断片をオブジェクトにメッセージを送るという統一された方法で扱う事ができる。この方法はメッセージに基づいているため、先に挙げた2つの方法と比較して分散環境向きであると言える。よって、本研究においては、このアプローチをとった。

3 Kamui言語の特徴

この章では、Kamui言語の特徴を計算モデルとグローバルIDの二つの側面から見ることに

する。

3.1 計算モデル

Kamui 環境では計算モデルとして、並列オブジェクト指向計算モデル Kamui88 [3] を採用した。採用の理由を以下に示す。

- ネットワーク上の資源は分散して存在しており、それぞれが並列に動作している。
- ネットワーク上で、資源とプログラムが通信し合う事が、オブジェクト指向におけるメッセージ通信とよく適合する。
- 並列オブジェクト指向モデルは、問題の記述においても、モデルとしてかなり広い範囲の問題をカバーできる。

Kamui 環境の全ての言語プロセッサは、このモデルに基づいたオブジェクトを生成する。また、実際のインプリメントに際して、以下の事柄を追加した。

1. オブジェクトが、並列実行の単位となる。
2. メッセージの送信には、返り値のない send 型と、返り値のある RPC 型の 2 種類がある。
3. ネットワーク上で用いられる型、およびメッセージの意味は、ネットワークでグローバルな ID によって定義されている。

3.2 3 種類のグローバルな ID

Kamui 環境を実現するのために、3 種類のグローバルな ID が存在する。それぞれオブジェクト ID 、プロトコル ID およびタイプ ID である。Kamui 言語が生成するオブジェクトは、これらの ID に基づいて動作する。以下にそれぞれの ID ついて説明する。

- オブジェクト ID

  全てのオブジェクトにはネットワークで固有な ID が付けられる。ID はマシン ID 、プロセス ID 、ローカルオブジェクト ID 等いくつかのフィールドにわかれており、ID がわかれば、オブジェクトがどこにいてもアクセス可能である。

- プロトコル ID

  Kamui 環境では、オブジェクトへのメッセージはプロトコルとイベント名からなる。つまり、メッセージはプロトコル ID とイベント ID からなる。プロトコルとはイベントの集合である。オブジェクトが、あるプロトコル P を受理可能であるとは、オブジェクトに P の全ての元 (イベント) に対応するメソッドが存在する事を言う。つまり Kamui では、オブジェクトは受理可能なイベントによって型付けされている。[2]

  プロトコルとイベントのそれぞれには、ネットワーク上で固有な ID が付けられる。プロトコル ID とイベント ID のそれぞれが決定されると、メッセージの引き数の個数と型、および返り値がある場合にはその型も一意に定まる。

- タイプ ID

  複数のプロセスが互いに通信しあうためには、メッセージを送信する際に、データを相手のデータ形式に変換しなくてはならない。例えば MatchMaker [4] は、インタフェースの仕様記述から、プロセス間のデータ変換規則を自動生成する。

  Kamui 環境では、データの変換を型 ID を用いて行う。Kamui 環境で用いられるデータ型には固有の ID が付けられている。同じ種類の型であっても、CPU や言語の違いによって異なるタイプ ID が割り当てられる。ただし、同じ種類の型同士は互いに変換可能である。

4 環境オブジェクト群と言語処理オブジェクト

3種類のIDに関する情報は環境オブジェクト群とよばれるデータベースに登録される。この章では、プログラム実行時にオブジェクトの動作支援をするデータベース群についてと、言語処理オブジェクトが環境オブジェクト群と協調動作する様子について述べる。

4.1 環境オブジェクト群

通常の言語ではグローバルな情報はヘッダファイルに記述するが、次のような欠点がある。

1. ヘッダファイルはある一つの言語に依存した形式にならざるを得ない。これは、複数言語間の連絡を目指すKamui環境にはそぐわない。
2. ファイルでは情報の動的な変化に対応できない。

そこで、Kamui環境ではグローバルな情報を納めるために、ヘッダファイルの代わりにデータベースを用いる。

以下にそれぞれのデータベースについて解説する。

- オブジェクトデータベース

  グローバルなオブジェクトは、名前とオブジェクトIDとの組みして、ここに登録される。オブジェクトデータベースもオブジェクトであるので、オブジェクトデータベースが他のデータベースに登録されることにより、データベースの階層が出来上がる。階層の一番上にあるデータベースには、特別なID番号として0が割り当てられる。ユーザとシステムは、このID番号が0のデータベースを基にして、名前によって他のオブジェクトのIDを知る。得られたオブジェクトがオブジェクトデータベースであった場合には、そこで階層が一段下がって再び検索が行われる。

- プロトコルデータベース

  プロトコルデータベースにはプロトコルIDと、そのプロトコルに属するイベントの名前とIDの組みが登録されている。また、イベントの引き数の個数及び型に関する情報もこのデータベースを参照する事により得る事ができる。ユーザによって新しいイベントが定義される度に、このデータベースに登録が行われる。

- タイプデータベース

  Kamui環境上で用いられる型はここに名前と型IDとの対として登録される。また、タイプデータベースにはネットワーク形式(ネットワーク上を流れているデータの形式)と言語内でローカルなデータ形式との変換規則が登録される。新しいマシンや言語がKamui環境に参加する度に、このデータベースに登録が行われる。

4.2 言語処理オブジェクトの動作

言語プロセッサオブジェクトから見た場合、環境オブジェクトはヘッダファイルのような役割を果たす。Kamui環境において言語プロセッサは、ヘッダファイルをインクルードするのではなく、データベースと交信することにより、必要な情報を得る。(図1)

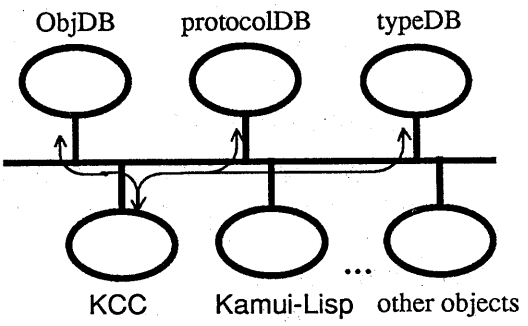

図1 言語プロセッサと
環境オブジェクトの協調動作

例えば、Kamui-Cのコンパイラkccは、メッセージを送信する構文に出会うと、プロトコルデータベースと通信することにより、プロトコ

ル名とイベント名からそれぞれのIDを得てメッセージを送信するコードを生成する。さらに、プロトコルを定義する構文に出会うと、プロトコルデータベースに新しいプロトコルを追加する。

また、Kamui-Lispは、タイプデータベースからデータ変換規則に相当するlambda式を取り出して、ネットワーク形式のデータを言語内でローカルな形式に変換する。

5 KamuiScript

この章では、Kamui環境においてユーザと環境との橋渡しとなるKamuiScriptについて述べる。

5.1 KamuiScriptとは

Kamui環境においては全てのネットワーク資源がオブジェクトとして扱われるので、これらにメッセージを送るという統一されたインタフェースで処理を進めることができる。KamuiScriptはKamui環境とユーザとの橋渡しをするための汎用のメッセージ送出言語であり、ユーザが環境と対話的にアクセスするために、インタプリタとして動作する。

ユーザは、KamuiScriptを通して既存のアプリケーションにメッセージを送出する事により、Kamui環境にそれらの機能を取り込むことができる。

つまり、KamuiScriptは、Kamui環境において、Unixでの Shellの役割を果たすものと見ることができる。

5.2 KamuiScriptの言語仕様

KamuiScriptは、デリゲーションベースのオブジェクト指向言語として設計されている。これは、インタプリタとして動作し、動的にインスタンス変数やメソッドを追加定義できるようにするためである。

KamuiScriptでは、メッセージ送出は次の構文を用いて行う。

```
<obj>.<message>
```

例えば、Kamui-Cコンパイラにクラス定義メッセージを送る場合には、

```
kcc.defclass <classname>
  [ <protocolnames> ] { <sentences> }
```

と書く。

特例として、KamuiScriptインタプリタ自身にメッセージを送る場合には、メッセージの送り先を省略する事ができる。

5.3 KamuiScriptの動作例

KamuiScriptインタプリタの動作を示すために、コンパイラに対するフロントエンドとしてKamuiScriptを用いた例を示す。図2に示したプログラムは、コマンド(文字列)を受け取って、そのコマンドをUnixのshの上で実行するオブジェクトを定義したものである。このプログラムは一見KamuiScriptのプログラムのようだが、実際には defclass等の文に出会う度に、KamuiScriptインタプリタによって、Kamui-Cコンパイラへメッセージが送り直される。冒頭のKamuiCという文は、KamuiScriptインタプリタ自身へのメッセージ送信であるが、このメソッドの中で、defclass等ほとんどのメッセージをKamui-Cコンパイラに再送信するように定義し直している。

この例のように、ユーザに対して、KamuiScriptの記述をなるべく見せないようにすることもできる。

```
KamuiC

include "str128.h"

defprotocol pSystem {
  send system(str128 str);
}
defclass cSystem [ pSystem ] {
}
defmethods cSystem pSystem {
  send system(str128 str) {
    system((char *)str);
  }
}
```

図2 KamuiScriptの記述例(つづく)

```
KamuiCMain {
  pSystem s;
  s = gDB.Get("sys");
  if (!s.object) {
    s = new cSystem;
    gDB.Add(s, "sys");
  }
  if (argc == 2) {
    s.system(argv[1]);
  }
}
```

図 2 KamuiScript の記述例 (つづき)

6 オブジェクト記述

この章では Kamui 環境における主要なオブジェクト記述言語について述べる。

6.1 Kamui-C

Kamui-C は Kamui 環境における主要記述言語として、C++の拡張となるように設計されており、Kamui-C コンパイラは Kamui-C から C++へのトランスレータとして動作する。

コンパイルに際して必要な情報は、先に述べた環境オブジェクト群と通信することにより得る。

プログラム例として、図 3 に Kamui-C で記述したカウンターオブジェクトを示す。

```
defprotocol pCounter {
  send setvalue(int v);
  send inc();
  send dec();
  send print();
}

defclass cCounter[ pCounter ] {
  int value;
}

defmethods cCouter pCounter {
  send setvalue(int v) { value = v; }
  send inc() { value++; }
  send dec() { value--; }
  send print() {
    printf("My value is %d\n", value);
  }
}
```

図 3 Kamui-C で記述した Counter オブジェクト

6.2 Kamui-Lisp

Kamui-Lisp は、インタプリタとしての特性を生かし、KamuiScript とともにユーザと Kamui 環境との橋渡しとなる言語である。また、インタプリタベースの言語であるので、Kamui-Lisp で記述されたオブジェクトは動的な性質を持つ事ができる。

Kamui-Lisp は ELIS/TAO 上にインプリメントされており、Kamui 環境とは TCP/IP を用いて通信する。インプリメントに際して、Lisp のインタプリタには一切手を加えずに、TAO の関数のみで処理系を構築した。よって、Kamui-Lisp のプログラムのなかで Lisp の関数をそのまま使えるという利点がある。

Kamui-Lisp には、他の言語と通信するために、デフォルトの型変換規則がいくつかあるが、それ以外の型に関しては、タイプデータベースと通信する事により、変換規則に対応する lambda 式を得て、それを変換規則として用いる。

プログラム例として、図 4 に Kamui-Lisp で記述したカウンターオブジェクトを示す。

```
(defProtocol 'pCounter
  '(send setvalue (int))
  '(send inc ())
  '(send dec ())
  '(send print ()))

(defKamuiClass cCounter (pCounter)
  (value))

(defKamuiMethods (cCouter pCounter)
  (setvalue (v) (setq value v))
  (inc () (incf value))
  (dec () (decf value))
  (print ()
    (format
      t "My value is ~s~%" value)))
```

図 4 Kamui-Lisp で記述した Counter オブジェクト

6.3 Kamui-Shell

Kamui-Shell は、Unix の標準入出力をつかさどるオブジェクトである。Kamui-Shell 自身はオブジェクト記述言語ではないが、Kamui-Shell

を仲介オブジェクトとして用いる事により、標準入出力を用いるフィルタ、インタプリタ等のUnixのアプリケーションをKamuiオブジェクトとしてKamui環境に参加させることができる。

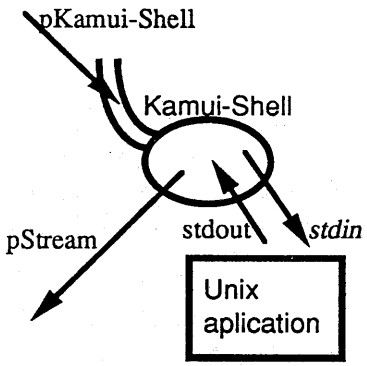

図5 Kamui-Shellの概念図

7 作成例

Kamui環境を用いてプログラムを作成した例を示す。

7.1 Lispマシンから Unixマシンを操作する

先ほどKamuiScriptの説明で示した、Unixのコマンドを実行するオブジェクトをELISから操作する例を示す。先ほどのプログラムをUnixマシンの上で実行したあとで次の関数を実行すると、コマンドrwhoがUnixマシンの上で実行される。

```
(defun system-demo (&aux s)
  ;; グローバル変数等のセット
  (kamuiinit)
  ;; オブジェクトデータベースから、
  ;;"sys"という名前のオブジェクトを得る
  (setq s
     (Kamui-Send
        *gdb* 'pObjectDB 'get "sys"))
  (Kamui-Send
      s 'pSystem 'system "rwho"))
```

7.2 ウィンドウコントローラ

Unixワークステーション上のウィンドウのコントローラとして、Lispで記述したコントローラを用いる。ウィンドウの上でマウスをクリックすると、それまでにウィンドウがクリックされた回数が表示される。プログラムの概要を図6に示す。

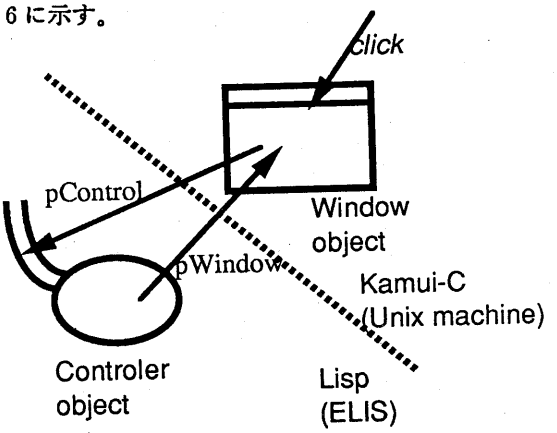

図6 プログラムの概要

以下にKamui-Lispで記述したコントローラのプログラムを示す。

```
(defKamuiclass cControl (pControl)
  (manager window (name 'elis) x y w h
   status clickX clickY (clicknum 0)))

(defKamuiMethods (cControl pControl)
  (down (mx my) nil)
  (up (mx my)
    ;; 前に書いた文字を消す
    (Kamui-Send window
      'PWindow 'text 0 20
      (princ-to-string clicknum) 0)
    ;; 文字を書く
    (Kamui-Send window
      'PWindow 'text 0 20
      (princ-to-string
         (incf clicknum)) 1))
  (cancel () nil)
  (move (nx ny)
    (setq x nx) (setq y ny))
  (size (nw nh)
    (setq w nw) (setq h nh))
  (show () ;; ウィンドウ表示
    (Kamui-Send window
      'PWindow 'text x (+ y h) name 1)
    (Kamui-Send window
      'PWindow 'rect x y w h 1))
```

```
  (hide () ;; ウィンドウを隠す
    (Kamui-Send window
      'PWindow 'text x y name 0)
    (Kamui-Send window
      'PWindow 'rect x y w h 0))
  (getName () name)
  (setName (nam) (setq name nam))
  (getMgr () manager)
  (setMgr (mgr) (setq manager mgr))
  (getWindow () window)
  (setWindow (win) (setq window win)))

;; メインルーチン
(defun KWinDemo (&aux c w)
  ;; グローバル変数等のセット
  (kamuiinit)
  ;; オブジェクトデータベースから、
  ;;"window"という名前のオブジェクトを得る
  (setq w
       (Kamui-Send *gdb*
                   'pObjectDB 'get
                   "window"))
  (setq c (create-instance cControl))
  ;; コントローラを貼る
  (Kamui-Send w 'pCW 'addControl c)
  ;; コントローラにウィンドウを通知
  (Kamui-Send c 'pControl 'setWindow w))
```

8 おわりに

Kamui 環境について、主として Kamui 環境のオブジェクト記述言語について、その言語仕様および環境内での言語プロセッサの動作に付いて述べた。

Kamui 環境の各種の言語プロセッサは、環境オブジェクト群と協調動作し、同一のモデルに基づくオブジェクトを生成する。これにより、異なる言語で記述されたプログラムの断片を、オブジェクトにメッセージを送るという統一された方法で扱うことができ、様々な言語によるアプリケーションの作成が可能になる。

現在 Kamui 環境の第 2 版を作成中であり、簡単なデモプログラムを実行させる事ができる。今後はウィンドウシステム等、ある程度の規模の問題を記述して、システムの評価を行う予定である。

9 謝辞

本研究を進めるにあたり、北海道大学工学部 赤間 清助教授、三谷 和史助手、ならびに言語情報工学講座の皆様に数多くの貴重な助言を頂きました。ここに感謝の意を表します。

参考文献

[1] 原田康徳, 浜田 昇, 渡辺慎哉, 三谷和史, 宮本衛市: 並列オブジェクト指向モデルに基づく分散型アプリケーション構築法, 情報処理学会研究報告 89-ARC-77(1989),pp67-74

[2] 原田康徳, 渡辺慎哉, 三谷和史, 宮本衛市 : イベントグループを導入した並行分散型オブジェクト指向言語 Kamui-C について, 日本ソフトウェア科学会第 6 回大会論文集 (1989),pp413-416

[3] 渡辺慎哉, 原田康徳, 三谷和史, 宮本衛市: 場とイベントによる並列計算モデル - Kamui88, コンピュータソフトウェア, vol.6 No.1(1989),pp.41-55

[4] Michael B. Jones, Richard F. Rashid: Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems, *ACM OOPSLA '86 Conference Proceedings(1986),pp.67-77*