計算機アーキテクチャ 77－10
（１９８９．７．１４）

並列オブジェクト指向モデルを用いた視覚的プログラミング環境の開発

浜田 昇　原田 康徳　渡辺 慎哉　三谷 和史　宮本 衛市

北海道大学工学部

Kamui88は、渡辺らの考案による並列オブジェクト指向計算モデルである。我々はこのモデルに基づくプログラミング言語 Kamui-Cとその処理系を開発中である。

本報告では、視覚的インタフェースを作成するツールとしてKamui-Cが有効であることを示す。その有効性を確かめるために、Kamui-Cの視覚的実行環境 Kamui-Kotanを Kamui-C自身を用いて作成した。

Kamui-Kotanは、オブジェクトが動作する様子を視覚的に表示し、プログラマとオブジェクト間の対話的なイベントのやりとりを可能にする。これはデバッグに威力を発揮するものと思われる。

Development of a visual programming environment using a concurrent object oriented calculation model

Noboru Hamada　Yasunori Harada　Sin'ya Watanabe　Kazufumi Mitani　Eiichi Miyamoto

Department of Information Engineering
Faculty of Engineering
Hokkaido University
kita-13jo, nishi-8tyoume, kita-ku, Sapporo 060, Japan

Kamui88, designed by Watanabe el al., is a model of concurrent object oriented calculation. We are developing a programming language Kamui-C, which is based on Kamui88, and its management system.

In this article, we show that Kamui-C is a useful tool to make a visual interface. We made a visual execution environment Kamui-Kotan with Kamui-C to prove its effectiveness.

Kamui-Kotan displays visualy how the objects behave, and allows a programmer to send any event to any object and field. We think this is powerful for debugging.

1. はじめに

近年、ビットマップ画面を備えたワークステーションの普及により、様々な視覚的インタフェースが作成され使用されている。その多くは、逐次型の言語を用いて作られているため不自然な表現を強いられたり、柔軟性に欠ける等の弱点がある。

現在、我々は並列オブジェクト指向言語Kamui-Cとその処理系を作成中である。本報告では、Kamui-Cを視覚的インタフェースを作成する道具として用いた場合の有効性について述べる。また、Kamui-Cの有効性確認のために、視覚的実行環境Kamui-KotanをKamui-C を用いて作成したので、それについても報告する。

2. 視覚的ユーザインタフェース

これまでの代表的な視覚的ユーザインタフェース作成ツールとその問題点について述べる。

2.1. Smalltalk-80

Smalltalk-80[2]は、代表的なオブジェクト指向言語の一つで、ビットマップディスプレイとマウスを備えたワークステーション上で動作する。

Smalltalk-80では、マウスやキーボードで絵やウインドウを操作するために、Model View Controllerという３つのオブジェクトの組みからなるＭＶＣモデルを用ている。このモデルでは、 Modelが表示されるオブジェクトであり、Viewが画面を操作するオブジェクト、Controllerがマウスやキーボードを監視するオブジェクトである。

Smalltalk-80は逐次型のオブジェクト指向言語であるため、各ＭＶＣモデルのControllerは、制御を獲得するために時分割でマウスとキーボードを占有し監視する。この方式では、マウスとキーボードは Contorllerの従属物であり、自立したオブジェクトではない。このため、マウスとキーボード以外の入力機器を接続しようとする際に困難が生じる。また、他のControllerに制御を渡すことをプログラム中に明示しなければならないため、プログラマに他のオブジェクトの振る舞いを意識させることになり、好ましくない。

2.2. X Toolkit

X Toolkit[3]はX Window System[4]のためのアプリケーション構築ツールである。X Toolkitは、C言語からX Window Systemへのより低いレベルのインタフェースであるXlibにかわって高いレベルで簡便なインタフェースを提供する。

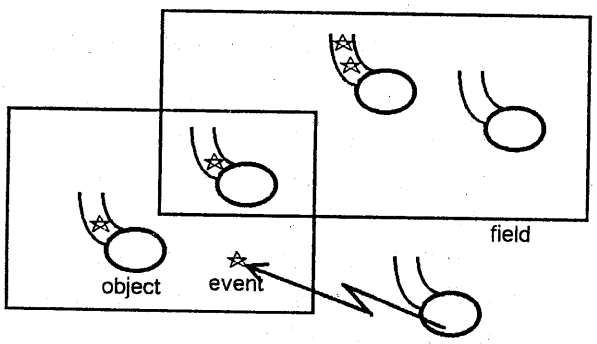
X Window Systemでは、サーバと呼ばれる特別なプロセスがディスプレイ、マウス、キーボード等の資源を一元的に管理している。このため、例えばマウスを２つ必要とするアプリケーションを作成する場合サーバを書き換えなくてはならず柔軟性に欠ける。

3. 並列計算モデルKamui88とそのモデルに基づく言語Kamui-C

Kamui88[1]は渡辺らの考案による並列オブジェクト指向計算モデルであり、Kamui-Cはそのモデルに基づき今回我々が開発した言語である。以下このモデルと言語の概要について紹介する。

3.1. 並列計算モデルKamui88

並列計算モデルKamui88は、任意個数のオブジェクトと場から構成される。（図１）



（図１）並列計算モデルKamui88

各オブジェクトは、それぞれ内部状態とイベントキューを持ち、イベントに反応して動作を行う。イベントに対するオブジェクトの動作はアトミックアクションである。送られてきたイベントはイベントキューに入り、実行されるのを待つ。

また、Kamui88では、場と呼ばれるものを用いてオブジェクトのグループ化とブロードキャストを実現することが特徴となっている。場に対してイベントが送られた場合、そのイベントはその場に属するオブジェクト全てにブロードキャストされる。このように、場を用いて複数のオブジェクトをひとまとまりに扱うことができる。

オブジェクトは動的に自由に場に出入りでき、また、ひとつのオブジェクトが複数の場に属することも可能である。

3.2. 並列オブジェクト指向言語言語Kamui-C

Kamui-CはKamui88モデルに基づくネットワーク透過な並列オブジェクト指向言語である。 Kamui-Cは、C言語を基にして拡張された言語である。

拡張された部分は、

- 単継承のクラスの定義
- イベントの送出
- 場へのオブジェクトの出入り
- 各オブジェクトの並列動作

である。

以下それぞれについて説明する。

3.2.1. クラス定義、継承

クラスの定義は、以下のように行う。

```
Class classname [superclassname]
        (instance-variables)
  eventname1(arg1, arg2, ... argn) {
    .....
  }
  eventname2(arg1, arg2, ... argn) {
    .....
  }
  ......
}
```

Kamui-Cの継承の方式は単継承である。次に、Kamui-Cのプログラムの例を示す。

```
Class parent[] (Int name, x) {
  init(Int n) {
    name = n;
    x = 0;
  }
  event() {
    printf("My name is %ld.\n", name);
    printf(" x is %ld.\n", x);
  }
}

Class child[parent] (Int y) {
  init(Int n) {
    [super . init(n)];
    y = 1;
  }
  event() {
    [super . event()];
    printf(" y is %ld.\n", y);
  }
}
```

クラスparentは、インスタンス変数name、xを持ち、イベントinitによりそれらの値をセットし、イベントeventにより、インスタンス変数の値をプリントする。

クラスchildは、クラスparentをスーパクラスとし、インスタンス変数name、x、yを持つが、そのうちnameとxはスーパクラスparentから継承したものである。クラスchildはイベントinitが来ると、まずスーパクラスのinitを呼び出し、 nameとxの値をセットする。その後、yの値をセットする。イベントeventが来るとスーパクラスのeventを呼び出して、 nameとxの値をプリントした後、yの値をプリントする。

3.2.2. オブジェクト及び場へのイベントの送出

ユーザインタフェースを構築する際には、ユーザとの対話をつかさどる部分はなるべく実時間で応答することが望ましい。実時間応答を実現するために、オブジェクト指向言語ではイベントに優先順位をつけることが必要である。このため、イベントの送出に、通常モード、飛び越しモード、リモートプロシジャコールの3種類の形式を用意した。これらは、それぞれ実行のされ方が少しずつ異なっている。おのおのについて次に説明する。

1)通常モード

送られたイベントが、イベントキューの最後に挿入される形式のイベントの送信方法を通常モードと呼び、次のように表される。

```
[receiver <- event(arg1 arg2 .... argn)];
```

イベントの大部分はこの通常モードで送られる。

2)飛び越しモード

イベントキューで処理されるのを待っているイベントを飛び越して、イベントキューの先頭にイベントを挿入する形式を飛び越しモードと呼ぶ。すなわち、飛び越しモードで送られたイベントは、通常モードで送られたイベントよりも優先順位が高く、オブジェクトが現在処理中のイベントの処理が終わり次第、処理される。

飛び越しモードでのイベントの送出は、次のように行う。

```
[receiver <<- event(arg1 arg2 .... argn)];
```

3)リモートプロシジャコール

このモードで送られたイベントはレシーバのイベントキューに入れられることなく直ちに実行され、イベントを処理した結果の返り値を持って戻ってくる。これは、Smalltalk-80におけるメッセージセンドと等価である。

Kamui-Cでは、おもにオブジェクトの生成時、および他のオブジェクトのインスタンス変数の参照時に使われる。また、イベントが処理されるのを待つので同期を取りたい場合に便利である。

リモートプロシジャコールは次のように行う。

```
result = [receiver . event(arg1 arg2 .... argn)];
```

3.2.3. 場への出入り

Kamui-Cではオブジェクトは動的に場に出入りすることができる。

これにより、プログラムの実行時の非決定性が増し、問題の表現力が増すが、これにより実行の再現性が低下するため、デバッグがしにくくなるという問題もある。

場への出入りは通常のC言語の関数によって実現される。

場に入る時： enterF(field, objectId);

場から出る時： exitF(field, objectId);

4. Kamui-Cを用いる利点

視覚的インタフェースを作成する際に、Kamui-Cの並列性および場を活用することにより、 Smalltalk-80等の逐次型オブジェクト指向言語で記述する場合に比べ、柔軟なインタフェースを作成することが可能である。以下にその理由を述べる。

4.1. 並列性による利点

Kamui-Cでは、各オブジェクトが並列に動作するため、マウスを自立したオブジェクトとして扱うことができる。これにより各ウィンドウはマウスやキーボードの動きを監視する必要がないため、Controllerを各ウインドウごとに用意しなくともよい。すなわち、マウスやキーボード自身が Controllerの役割を果たすことになる。

4.2. 場を用いることの利点

場はブロードキャストを実現するためのものである他に、間接参照を実現する手段としても有効である。特にウィンドウシステムのように、イベントのレシーバが不特定多数であり、かつ頻繁に変化するような場合に有用である。

前節で述べたように、Kamui-Cの並列性によりマウスとキーボードは自立的なオブジェクトになれるので、それら自身がControllerの役割を果たす。ここで、それらはイベントの送り先を知る必要があるが、場を用いることにより次のようにしてこの問題を解決することができる。

マウスおよびキーボードのために、それぞれ、場Mouse、場Keyboardを用意して、マウスやキーボードの状態が変化した時に、それらが、それぞれの場にイベントを出すことにする。マウスやキーボードについての情報を知りたいオブジェクトは、それぞれ場Mouseあるいは場Keyboardに入れば良い。この方法によると、マウスおよびキーボードは、それぞれ自分がイベントを出すべき場を知っているだけで良く、マウスおよびキーボードを参照するオブジェクトの数や名前を一切知らなくても良い。

Smalltalk-80では、マウスまたはキーボード以外の入力機器を接続しようとすると、 Controllerを書き換えなくてはならない。しかし、Kamui-Cの場合、入力機器は自立的なオブジェクトになることができるので、マウスのイベントに準じたイベントを場に出すことにより、マウスと同様に扱うことができる。

以上述べたように場をうまく利用することにより柔軟なインタフェースを作成することができる。

5. 視覚的実行環境Kamui-Kotan

Kamui-Cの有効性を確かめるためにKamui-Cの実行環境を試作した。

5.1. 視覚的実行環境の必要性

視覚的実行環境は次に挙げる点から必要である。

[1]起動用イベントの必要性

オブジェクトに計算を開始させるためには、なんらかの起動用のイベントを外部から送る必要がある。この際には、オブジェクトの名前を指定してキーボードから打ち込むよりも、オブジェクトが画面上に図で表され、それをマウスで指定する方がはるかにに分かりやすい。

[2]場とオブジェクトの関係の視覚化

Kamui88の場合、オブジェクトは場に動的に出入りを繰り返すので、実行を一時中断して、プログラマがオブジェクトを場に任意に出入りさせることのできる機能が必要である。オブジェクトが場に

出入りする様子は視覚的表示すると分かりやすい。

[3]デバッグのための要請

デバッグのためには、実行を一時止めてプログラマが任意のオブジェクトあるいは場にイベントを自由に送れることが必要である。これは通常の手続き型言語において、デバッガで、プログラムを一ステップずつ実行させながら、その都度任意の変数の内容を参照することに対応する。このためにはイベントループを制御することが必要であるが、これはプロセス全体を操作することであるから、オブジェクトがその操作を行うことは好ましくない。これは実行環境でサポートすべき事柄である。

5.2. 視覚的実行環境Kamui-Kotanの機能

前節での考察により、次のような機能を備えた視覚的実行環境 Kamui-KotanをKamui-Cを用いて作成した。

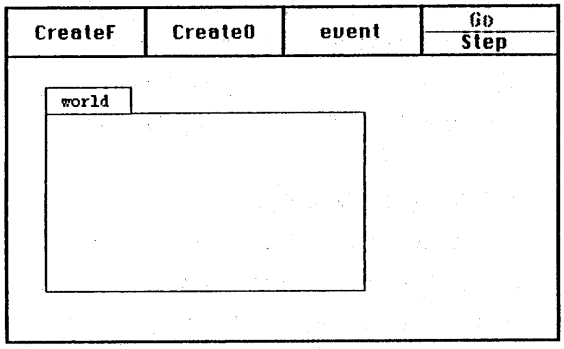
- 任意のオブジェクトあるいは場に任意のイベントを出す・オブジェクトの場への出入りを視覚的に表示する・任意の場に対してオブジェクトを出し入れする・1イベントごとにステップ実行する

5.3. Kamui-Kotanの動作の様子

Kamui-Kotan上でジャンケンを実行する場合を例にとって説明する。

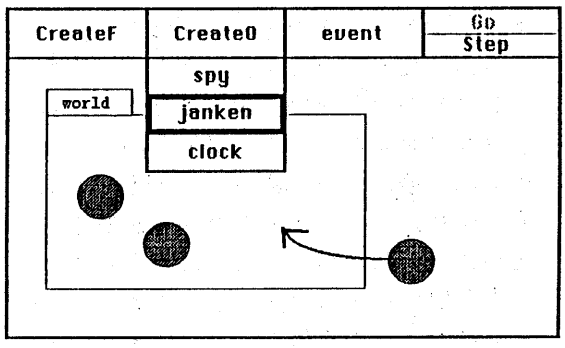
ジャンケンオブジェクトは、イベントinitを受け取ると互いにイベントを送りあってジャンケンを始め、負けると自動的に場から退出する。

まず始めに左上のCreateFボタンをクリックして場を生成する。場は、画面上では四角形で表される。（図2）



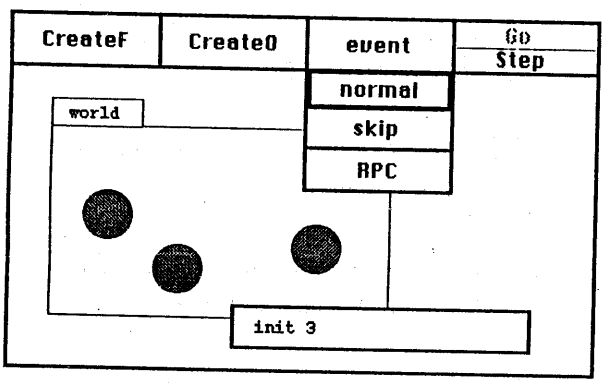
（図2）Kamui-Kotanの動作1

次にCreateOボタンをクリックして、生成するオブジェクトのメニューを表示させる。ジャンケンオブジェクトを生成するためにメニューの中からjankenを選ぶ。オブジェクトは画面上では円で表される。生成されたジャンケンオブジェクトをマウスでドラグして、先ほど生成した場の中に入れる。これをジャンケンさせたい人数が揃うまで繰り返す。（図3は、3人のジャンケンを示す。）（図3）



（図3）Kamui-Kotanの動作2

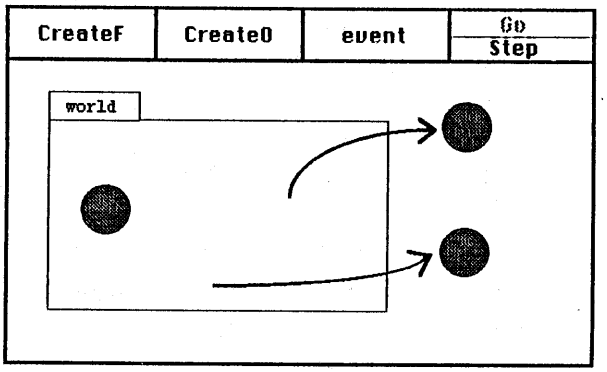
次にeventボタンをクリックしてイベントのメニューを表示させる。通常モード、飛び越しモード、リモートプロシジャコールの3種類のイベントの送信方法のうち、通常モードで送信することにする。送出するイベントinitをタイプして、引数には、今3人のジャンケンなので3を指定する。次にイベントの送り先をマウスでクリックすると、そこにイベントが送られキューに入る。ジャンケンの場合、先ほど生成した場を指定する。（図4）



（図4）Kamui-Kotanの動作3

今、Stepボタンがアクティブに、Goボタンがインアクティブになっているが、これは、イベントが一つ処理されるたびに実行が止まり、プログラマが介入できる状態になっていることを示す。ここでStepボタンを押すとイベントが一つ処理される。Goボタンを押すと、イベントは連続的に処理されるようになる。

ステップモードでは、イベントの送受のたびにオブジェクトはフラッシュする。また、ジャンケンに負けたオブジェクトが場から退出する様子も、見ることができる。（図5）



（図5）Kamui-Kotanの動作4

5.4. インプリメンテーション

視覚的実行環境Kamui-Kotanの実現方法について述べる。

5.4.1. 各オブジェクトの並列動作

Kamui-Cはコンパイラ型の言語であり、Kamui-Cのコンパイラは、Kamui-Cのソースファイルを C言語に翻訳した後、それに実行時ライブラリをリンクして実行形式にする。

実行時ライブラリの働きにより、各オブジェクトは並列に動作しているようにシミュレートされる。

5.4.2. 監視用オブジェクト

オブジェクトに送られるイベントに応じて、画面を操作するために、各オブジェクトに、そのオブジェクトの監視用のオブジェクトを付加する。つまり、監視用のオブジェクトは、ＭＶＣモデルにおける Viewに相当する。

この場合、監視されるオブジェクトに送られるイベントおよび送出されるイベントは、全て監視用オブジェクトを通して送受されるようになる。このとき、監視用オブジェクトには、特別なイベントeventが、監視されるオブジェクトに送られるべきイベントを引数にとって送られる。

コンパイルするときにオプションを指定することにより、コンパイラが監視用のオブジェクトを、全てのオブジェクトに自動的に付加するようにした。

5.4.3. X Window Systemとのインタフェース

現在の所、ウィンドウの書き直しなど、Kamui-Kotanの画面操作に関係する部分は、多くをX Window System[3] (以下X)に依存しているため、 Kamui-Cの実行時ライブラリは、 Xのイベントキューと Kamui-Cのイベントキューの両方を見ている。 X側で発生したイベントは、Kamui-Cのイベントに直されて、Kamui-Cのイベントキューに入ることになっている。一度Kamui-Cのイベントに直す理由は、将来Kamui-Cでウィンドウシステムを実現することを予定しているからである。

6. 今後の課題

この視覚的実行環境Kamui-Kotanは、将来的にはKamui-Cの標準実行環境とする予定であり、さらに多様なデバッグ用の機能や、ソースを書き込める機能なども盛り込みたい。

また、Kamui-Cを用いた柔軟なウインドウシステムの構築も今後の課題の一つである。

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]渡辺,原田,三谷,宮本: 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,No1(1989),pp.41-55.

[2]A.Goldberg , D.Robson : *Smalltalk-80: The Language and its Implementation*, Addison-Wesley,1983.

[3]X Toolkit documentaion, X Version 11 Release 2 documentaion

[4]Installation Guide and Release Notes, X Version 11 Release 2 documentaion