記 号 処 理 57－5
（1990．11．22）

分散型環境Kamuiについて

原田康徳　浜田 昇　渡辺慎哉　宮本衛市

北海道大学工学部

分散プログラミングを容易に安全に行なうために、分散型プログラミング環境Kamuiを提案する。Kamui環境では全ての分散資源をオブジェクトとして扱う。オブジェクトを仕様と実現とに分離することにより、より安全なプログラミングが行なえ、目的別に記述言語を選択することができる。本稿では、Kamui環境のゴールとその実現について述べ、例としてCSCW向けのtoolkitの作成を挙げる。また、Kamui環境の構築上で生じた問題点を述べ、開放分散へ向けて考察する。

Kamui – A Distributed Programming Environment

Yasunori Harada　Noboru Hamada
Sin-ya Watanabe　Eiichi Miyamoto

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

To use distributed programming easily and safely, we propose a distributed programming environment called Kamui. In the Kamui environment, all distributed resources are objects. Since we separate an object's specification and implementation, we gain safe programming and the free selection of object implementation languages. In this article, we describe the goal and implementation of the Kamui environment, and present an application example for constructing a toolkit for CSCW. We also discuss some problems that occurred when constructing the Kamui environment, and give consideration toward constructing open system.

1 はじめに

数多くのコンピュータがネットワークで接続され、互いに通信が行なえるようになってきている。それにより、様々なデータやハードウェア資源を共有し、資源の効率化と、取り扱いの容易さを得ることができる。例えば、ファイルシステムを共有し、ネットワーク上で共通なデータを一ヵ所に置いたり、プリンタなどのハードウェアを共有してネットワーク上で有効に使用している。

さらに、E-Mail により地理的に離れたユーザ間のコミュニケーションを可能にし、我々の、より複雑な仕事を支援している。一般的には GroupWare といったソフトウェアが、複数のユーザの協調問題解決を支援している。このような分散型のプログラムの必要性は年々高まっており、そのプログラムの複雑さも増してきている。

このようなの分散ソフトウェアの作成には多くの難しい点がある。並列プログラミングは直列なものよりも作成やデバッグが難しいと一般にいわれているが、分散プログラミングはそれ以上に難しい。そこには、並列プログラミングと同様に資源の相互排除や、プロセス間の同期の問題があり、それに加えて、分散プログラミング特有の通信の遅延、集中による効率の低下、距離の違いによる非決定性などの問題が存在する。これらが、分散プログラミングを難しくしている。

分散プログラミングを安全にかつ効率よく行なうために、分散プログラミング環境が必要である。このような環境が、高度なアプリケーション作成からの要求に答えるためには、以下の項目を満足する必要があろう。

1）柔軟で効率のよい安全な計算モデル
2）大規模なプログラミングに耐えるシステム
3）問題別の記述言語
4）OS, 機種に依存しない構造

1）のために、すべての資源を統一的に扱うことのできる、広いモデルを考える必要がある。2）の問題は、分散固有の問題ではなく、大規模プログラミングの技術が必要である。3）は、単一の強力な記述言語で全てを解決しようとするよりも、問題別に適した言語を用いて解決しようとするアプローチである。4）は、ネットワーク上の資源を有効に利用するために不可欠である。

我々は、分散プログラミングを支援する Kamui 環境を構築中である。Kamui 環境は、これらの項目を満足し、実働させることを目指したプログラミング環境である。Kamui 環境はその計算モデルとして並列オブジェクト指向モデルを導入し、オブジェクトのグループ化を考慮した動的及び静的な型機構を持ち、幾つかのレベルのオブジェクト記述言語を用意している。

本論文は、Kamui 環境の目的とその実現について述べている。2 章では、Kamui 環境の目指していることについて、3 章では、環境で動作するオブジェクトの定義を、4

章ではそのオブジェクトに付けられた型について述べる。5章と6章は具体的なKamui環境の実現について、7章はそこで行なわれているアプリケーションについて述べる。最後に、8章では今後の方向として、開放分散への発展について述べている。

2 Kamui環境

Kamui環境が目指しているゴールを以下に述べる。

2.1 オブジェクトのネットワーク透過性

多くの分散システムが提供しているように[2]、オブジェクトのネットワーク透過性は重要な性質である。これは、分散した資源がどのような位置に配置されているのかがユーザから隠されていることである。この性質を満足するために、Kamui環境では環境中全体で動作するオブジェクトを一つのIDで指し示す。オブジェクトIDは環境全体でユニークである。この逆の例として、UNIXのプロセスは各マシンごとに閉じているために、ネットワーク上のUNIXマシン間でユニークではなく、ネットワーク透過性は持たない。それに対して、オブジェクトIDはマシンをまたがってユニークな番号が付けられる。最後の章で、このユニークなIDに付いての考察が行なわれる。

2.2 種々の言語によるオブジェクト記述

オブジェクト指向の特徴の一つに、仕様と実現との分離が挙げられる。その特徴を生かし、オブジェクトの実現を複数の言語によって行なう。これによって、問題向きの言語を用いてそれぞれの得意な分野のオブジェクトを実現することができる。オブジェクト指向モデルは広い範囲の記述言語を取り入れるのに十分な柔軟性を持っている。

高い実行速度や細かいシステムの制御などが要求される場合は、C言語をベースとした記述言語が用いられる。一方、システムのプランニングや柔軟な処理などが要求される場合には、lispをベースにした記述言語が用いられる。システムのプロトタイピングにはインタプリタ系の言語が、それが終了したオブジェクトにはコンパイラ系の言語が用いられる。

記述言語の選択は、すでに存在しているライブラリー的なオブジェクトの記述言語に捕らわれることなく、素直に問題に適した記述言語で行なえる。これは、環境の記述力やその発展性に大きく貢献する。

2.3 メッセージの構造化

独立に行なわれるプログラミングでは、メッセージ名の衝突が問題となる。これを解決するためにモジュールプログラミングと同様の手法を用いて、メッセージを構造化する。これは、オブジェクトの多重ビュー

[3] に基づいており、異なったビューの下ではメッセージはすべて異なったものとして扱われる。

2.4 オブジェクトの仕様記述

大規模なシステムが矛盾無く動作するためには、個々の部品の仕様の定義が厳密になされている必要がある。そのために Kamui 環境では、オブジェクトの仕様記述にも力を入れている。オブジェクトを自由な記述言語で実現するために、その仕様を統一した方法で記述する必要がある。これによって、異なった言語間での仕様の矛盾が検査できるようにする。

並列オブジェクトの特徴としてその自律性が挙げられる。これは、個々のオブジェクトが自発的にメッセージを外部に送出することを意味する。オブジェクトの仕様記述はこの性質に対応したものである必要がある。すなわち、各オブジェクトはどのようなメッセージを受けて、どのようなメッセージを出力するかの情報をまとめたものが仕様として与えられる。

3 Kamui オブジェクト

Kamui 環境で動作しているオブジェクトは全て Kamui オブジェクトである。この章では、Kamui オブジェクトとして必要な機能を述べる。Kamui オブジェクトは様々な言語で実現されるが、それらはここで述べた機能を満足するように作られる必要がある。

3.1 計算モデル

Kamui オブジェクトの基本となる計算モデルは Kamui88[4] である。これは、場を用いてオブジェクト間の協調動作を表現する Actor[1] をベースにした計算モデルである。Kamui88 では send 型のメッセージ送信のみを用意しているが、それを拡張して従来の直列オブジェクト指向モデルのメッセージ送信である call 型の通信も加えた。また、オブジェクト中の一つのメソッドはアトミックアクションであり、オブジェクトの内部に複数のスレッドは存在しない。そのため、call 型では相手からの返答メッセージが返るまでの間オブジェクトは待たされることとなる。その間にそのオブジェクトの他のメソッドが実行されることはない。

3.2 型の返答

オブジェクトには型が付けられる。プログラムの実行前にその誤りを検出する有効な手段として、静的な型検査が行なわれる。しかし、分散環境のように動的なシステムでは、静的な型検査だけでは満足な誤りの検出は行なえない。動的な型検査を行なうため、Kamui オブジェクトは自分の型についての問い合わせに答える必要がある。つまり、それぞれのオブジェクトが自分の型情報を持っており、それを動的に外部に返答する。オブジェクトの新しい参照が行なわれたときにこのような検査が行なわれる。型の実際の構造については次章で述べる。

4　オブジェクトの型とオブジェクトへの参照の型

Kamui 環境では、オブジェクトの型を2つのレベルに分けて考える [6]。一つは、オブジェクトそのものの型であり、もう一つはオブジェクトを参照する型である。参照の型は、オブジェクト全体を指すものではなく、オブジェクトのある性質の側面、つまり一部分を指す。

この考え方の基本となるのは、オブジェクトの多重ビューである。オブジェクトが複数の目的に用いられる場合に、それぞれの目的に対応して付けられるものがビューである。目的別に付けられたビューの間の違いはオブジェクトを操作するメッセージの種類の違いに表れる。すなわち、それぞれのビューに対して幾つかのメッセージが対応する。

4.1　オブジェクトの参照の型

オブジェクトをある局面で使用する場合、一つの目的によって操作する場合がほとんどで、複数の目的を組み合わせて使用するのはまれである。オブジェクトは一つの参照を通じて使用されるので、参照がオブジェクトの使用の目的に対応している。つまり、参照の型はその目的に対応したビューである。Kamui 環境ではこの型をメッセージプロトコル [5] と呼ぶ。

メッセージプロトコルは、メッセージの集合である。正確には、メッセージプロトコルとその一つの要素によって通信のメッセージが決定される。同一のメッセージプロトコルにおけるメッセージの意味は一意である。逆に、メッセージプロトコルが異なればメッセージの形が同じものでも、異なった意味を持つメッセージとして扱われる。

これによって、メッセージの意味がメッセージプロトコル内で閉じることになる。これは、モジュールプログラミングのモジュール名と関数名の関係に似ており、大規模なプログラミングに役立つ。

4.2　オブジェクト自身の型

オブジェクト自身の型は、オブジェクトの仕様を表わす。オブジェクトが自律的な性質を持っていることから、それが送出するメッセージの種類についても仕様として与える。このような情報は、外部のオブジェクトを指すインスタンス変数へ付けられた型にある。つまり、仕様としてそのオブジェクトが外部にどのような種類のオブジェクトを参照してるのかを示すことになる。

オブジェクトが参照しているすべてのオブジェクトに関する情報を外部に公開する必要はない。大まかに言えば、自分と同じかそれ以上の粒度のオブジェクトについて公開するのが妥当であろう。なぜなら、自分より粒度の小さいオブジェクトは、大部分自分の下請け的な動作をすると考えられるからである。下請けオブジェクトは、オブジェクトの実現に大きく依存するので、その構成を公開すべきではない。逆に、自分と対等かそれ以上のオブジェクトへの参照は、それらが独立した部品としての存在

であるので、カプセル化を妨げずに、部品の仕様として有益な情報になる。

4.3 型検査

送信されるメッセージは、レシーバオブジェクトの参照の型と、メッセージの形によって決定される。ある参照の型が決まった場合、そこから指されているオブジェクトに送られるメッセージが、正しくその参照の型のものであるかどうかは、静的な型検査によって行なわれる。一方、あるオブジェクトがその参照として正しいかどうかは、参照が起きたときにそのオブジェクトへ直接問い合わせる動的な検査によって調べられる。

5 環境オブジェクト群

Kamui環境の構築には幾つかの環境オブジェクトが用いられる。これらは、Kamuiのオブジェクト記述言語によって書かれたもので、通常のKamuiオブジェクトと同じものである。これらのオブジェクトに対するくわしい説明は文献[7]で述べられているので、ここでは、その概要について述べる。

プロトコルデータベース 通信の基本となるメッセージプロトコルを登録しているデータベース。

タイプデータベース 通信の基本となるプリミティブな型が登録されているデータベース。

オブジェクトデータベース 大域的で特別なオブジェクトとそれに付けられた特別な名前とを登録してあるデータベース。

6 オブジェクト記述言語

オブジェクトを記述する言語は各種用意されている。これらの言語の処理系は、それ自身がKamuiオブジェクトである場合とそうでない場合とがある。言語についてのくわしい議論は[7]で行なわれる。

Kamui-C C++をベースとしたオブジェクト記述言語。

Kamui-Lisp Lispをベースとしたオブジェクト記述言語。

KamuiScript Kamui環境における汎用のスクリプト言語。

その他、Kamui環境の記述言語としてSmalltalk-80、HyperCardなどの言語にも発展する予定である。

7 評価

Kamui環境の評価のために、ある程度規模の大きいプログラミングを行なう必要がある。その一つとして、現在進めているKamui-Toolkitを紹介する。

Kamui-Toolkitは、CSCWのプラットフォームとなるツールキットである。各マシンのマウスが分散資源として扱われ、ネットワーク透過に全てのマウスが見える。ボ

タン、メニューなども分散オブジェクトとして実現され、自由に他のマシンのツールを操作できる。

Toolkit の作成は、速度の要求される部分と、柔軟性が必要な部分とがある。オブジェクトの描画に関する部分や、イベントの処理に関する部分は、速度が要求されるのでKamui-Cで記述される。インタフェースのユーザカスタマイズに関する部分は柔軟性が要求されるので、Kamui-Lisp で記述される。

このように、Kamui 環境を用いると、問題向けの様々な記述言語を用いて、大規模なシステムを容易に構築できる。

8 開放分散への発展

これまでの議論は、分散プログラミングを支援することが前提となっている。しかし、実際に Kamui 環境の実現に際して幾つかの問題点が生じてきた。

1）プロトコルを定義した時間が異なることにより、様々なバージョンのプロトコルが存在する。

2）環境の扱う範囲が大規模になるにつれて、ユニークなオブジェクト ID を環境全体で保つことが難しい。

3）単一なオブジェクトのモデルで対応できない問題がある。

4）新しいメッセージの形式が欲しい場合がある。

時間と距離の空間全部に対して大域的に統一がとれた情報を扱うことが Kamui 環境の前提であった。そのため、1）、2）の解決は極めて困難であろう。しかし、実際の運用では局所的に矛盾が無ければ動作可能であるから、大域的なものに対するこのような前提は必ずしも必要では無い。3）、4）を解決する為に、例えば現在考えられる全ての問題を取り込むことができるように、Kamui 環境の定義を広くしたとしよう。つまり、複数のオブジェクトモデルに対応し、未来型やタプルスペース等考えられるものを全部導入するのである。しかし、これは現時点での解決であり、将来にどのような拡張が必要になるか全く予想がつかない。そのために、常に環境の扱う範囲が変化し、定義も変わる。

これを根本から解決するためには、環境自体に開放分散のモデルを導入しなければならない。開放分散では、局所的には ID や計算モデルが統一されていても、大域的にはそのような仮定が存在しない。また、現在の Kamui 環境の採用した方法は一つの例と見なされ、いつでも別の例として別の環境を考えることができる。

このような見方が必要になった理由としては、Kamui 環境の目指したものが実際にはかなり開放分散に近かったからと思われる。Kamui 環境の目標を正確に実現しようとすると、Kamui 環境自体がいつでも発展できるものである必要があるし、それを新しい Kamui 環境として呼び直すことが必要である。

現在、我々は Kamui 環境を開放分散な

モデルにするために、その目的、仮定、実現等を修正中である。開放分散の立場からは、現在の Kamui 環境は一つのインスタンスであり、したがって、他の分散プログラミング環境も容易に取り込むことができる。また、計算モデルも Kamui88 は一つの例であり、他の並列計算モデルとの計算を容易に取り込むことができる。さらに、異なったバージョンのクラスやメッセージが同時に存在することができるので、独立して作成された2つのプログラムを（例えば同じ名前のクラスやメッセージがそれぞれで違う意味で用いている）矛盾無くつなげることができるのである。

参考文献

[1] Atkinson, R. and Hewitt, C. E.: Synchronization in Actor Systems, *4th SIGPLAN-SIGACT Symp. on Princ. of Prog. Lang.*, 1977, pp.267–280.

[2] A.Black, etl : Object Structure in Emerald System, OOPSLA'86 Proceedings , September 1986.

[3] J. Shilling , P. Sweeney : Three Steps to Views: Extending the Object-Oriented Paradigm, *OOPSLA '89.*

[4] 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 , No.1 (1989) , pp . 41-55.

[5] 原田, 渡辺, 三谷, 宮本: イベントグループを導入した並行分散オブジェクト指向言語 Kamui-C について, ソフトウェア科学会第 6 回全国大会論文集,1989.

[6] 原田, 宮本: 送出メッセージの型宣言機構に基づいたオブジェクトの部品化について（君はどんな言葉を話すの?）, ソフトウェア科学会第 7 回全国大会論文集,1990.

[7] 浜田, 原田, 渡辺, 宮本: Kamui 環境におけるオブジェクト記述について, 情報処理学会記号処理研究会報告 57,1990.