北海道大学工学部研究報告
第151号（平成2年）

Bulletin of the Faculty of Engineering
Hokkaido University. No. 151 (1990)

オブジェクト指向による統合型分散プログラミング環境

原田　康徳　　渡辺　慎哉　　三谷　和史　　宮本　衛市

（平成2年4月6日受理）

An Integrated Distributed Programming Environment using Object Oriented Paradigm

Yasunori HARADA, Shin-ya WATANABE, Kazufumi MITIANI and Eiichi MIYAMOTO

(Received April 6, 1990)

Abstract

For programming in distributed systems, an integrated programming environment is required which compensates for some differences between programming languages and operating systems. We are constructing the Kamui environment, which is an integrated programming environment for distributed processing using object oriented paradigm. The main feature of the Kamui environment is that objects act independently (Kamui88 model) and it introduces 'message protocol' as type system for absorbing some differences between languages and operating systems. We present various ideas for object implementation in the Kamui environment including Kamui-C and Kamui-Shell which have been implemented.

1．はじめに

ネットワークの普及により，複数のコンピュータが互いに接続された分散システムが広まっている。それに伴い，分散された資源を有効に利用しようとする分散ソフトウェアが求められている。その中には，CSCW (Computer Supported Cooperative Work) DAI (Distributed AI) などのような最近の研究対象となっている分野のソフトウェアも含まれ，高度なプログラミングが要求されている。このようなソフトウェアは，従来のソフトウェアと比較してその複雑さが増し，開発過程の支援が不可欠である。

統合型プログラミング環境は，プログラミングの様々なレベルにおける支援を行なう。Smalltalk-88[1] Lispマシンにおける環境はその代表であり，プログラムの設計，作成，デバッグ，実行等を統合的に支援する。しかし，これらの大部分は単一の記述言語を基にしており，複数の言語による問題記述までも支援することは困難である。したがって開放分散な問題解決には不適当である。

そこで，我々は統合型分散プログラミング環境の構築のために分散オブジェクト指向パラダイ

情報工学科　言語情報工学講座

ムを導入する。分散オブジェクト指向パラダイムは，分散されたシステムの構築モデルとして有効であり，分散したソフト，ハードの両方の資源をそのままオブジェクトとしてとらえることにより，強力な部品化の機構となる。また，複数の記述言語によるオブジェクトの実現により，様々なレベルのプログラミングに対して，統一の取れた取り扱いができる。本論文では，現在我々が構築中である Kamui 環境について述べる。Kamui 環境は，分散オブジェクト指向パラダイムにより，様々の異なった言語や OS 上で動作しているオブジェクト（Kamui オブジェクト）の協調動作支援を目指している。

以下 2 章では，統合型環境と分散プログラミングにおいて生じている問題点とその解決法について述べる。3 章では，統合型分散プログラミング環境 Kamui 環境について述べる。4 章では，Kamui 環境におけるオブジェクトの型付けの機構について，5 章では，Kamui 環境の実行支援について述べている。6 章では，Kamui 環境の一つのオブジェクト記述言語である Kamui-C を紹介する。

2．統合型環境と分散プログラミング

本章では，従来の統合型プログラミング環境と分散プログラミングの両面からそれらの問題点を探り，それらを融合した新しい考えである統合型分散プログラミング環境について述べる。

2.1. 統合型プログラミング環境

統合型プログラミング環境は，Lisp マシンや Smalltalk-80 に代表されるように，プログラミングの設計，作成，デバッグ，実行等を全体にわたって支援する環境である。そこでは，複雑なプログラミングを極めて効率よく行なうことができる。そのような環境は，単一記述言語によるものがほとんどである。つまり，全てのプログラミングを単一の言語で行なうことにより，支援をそこに集中し統合された快適な開発環境を提供しようとするものである。従ってこの記述言語には，どのようなレベルのプログラミングであっても表現できる万能の表現力が要求される。このため，マルチパラダイム言語 TAO[2] などが登場している。

単一言語の場合，例えば C 言語と Lisp のどちらが勝れているか，といったような議論がある。しかしこのような議論は，純粋な言語の能力だけでなく，個人の嗜好や過去の資産などにも大きく左右され，言語の優劣は一概には言えない。ましてや，言語の統一は甚だ困難である。

これに対して，問題の各部分を，それぞれの適用分野や個人の好みにあった言語を用いて記述し，それらを組み合わせて全体を構築する方法が考えられる。この場合，各部分問題に適した記述言語を自由に選択する。例えば，UNIX 上の Smalltalk では，Smalltalk と C 言語とのインタフェースが用意されており，それぞれの問題に適した言語を選ぶことができる。その場合，記述言語間の約束が必要であり，それを支援するための環境が必要である。

2.2. 分散プログラミング

分散ソフトウェアはネットワークに分散された様々な資源を有効に利用し，ハードウェア，ソフトウェアの区別なく，全ての資源を利用することを目標とする。このような分散ソフトウェアの例としては，ネットワーク上に接続されたディスクのファイルを共有する NFS や，Server-Client モデルに基づいて分散したインタフェースプログラムが記述できる X-Window System などが挙げられる。

分散ソフトウェアを構築する分散プログラミングのための，支援ツールは幾つかある。Sun

-RPC はリモートプロシジャコールを UNIX の socket 上で実現したライブラリーであり，NFS はこの上で組まれている。また，MatchMaker[3] は異なった記述言語間での RPC を行なう stub を生成する。しかし，これらは手続き的な言語を基にしたライブラリであり，プログラミング環境のレベルではない。すなわち，これらのツールの上では小さな局所的なアプリケーションしか構築できない。したがって，分散プログラミングを支援するプログラミング環境としては，従来型のプログラミングパラダイムの枠を超える必要がある。

2.3. オブジェクト指向分散プログラミング

オブジェクト指向パラダイムは，データとそれを操作するメソッドを一つの組みにして取り扱い，プログラムの部品化や独立性を強化する。そのために，このパラダイムは分散された資源を有効に利用するために，特に大規模な分散システムにおいて重要である。例えば Emelard[4] は分散プログラミングに適したオブジェクト指向言語である。しかし，分散システムを単一言語で捉えているので，前出のような問題点がある。

オブジェクト指向パラダイムの特徴の一つは，その内部記述と外部仕様とが完全に分離できることである。従って，オブジェクトの内部記述は幾つかの言語を用いることができる。換言すれば，オブジェクトの外部仕様の形式をを決めておけばよいのであって，一つのオブジェクト指向言語に統一する必要はない。すなわち，Emerald の場合は言語レベルの支援であったのに対して，さらに広い枠として環境レベルの支援によって，オブジェクト指向分散プログラミングを捉えようというのが我々の主張である。

また，OS レベルでオブジェクト指向分散プログラミングを考える試みもある。しかし，OS レベルの支援はその OS の中だけで閉じたものとなり，様々なシステムとの共存が図れない。したがって，分散された資源を真に有効利用しているとは言いがたい。むしろ，環境レベルで分散プログラミングを捉えることにより，様々な OS 上で動作しているオブジェクトを協調動作させることができるのである。

2.4. 統合型分散プログラミング環境

前節までの様々な方面からの議論をまとめると，統合型プログラミング環境を分散プログラミングの支援にまで拡張した，統合型分散プログラミング環境が必要であり，また，その実現には環境全体のプログラミングモデルとしてオブジェクト指向パラダイムが有効である，ということである。そこで本節では，その環境についてさらに深く考察する。

環境上で捉えることができるオブジェクトは，すべて統一的に操作できる必要がある。これによって，様々な支援ツールが統一的に作成できる。オブジェクトの統一的操作は，そのオブジェクトの大きさや寿命，位置などに依存しない。逆に，操作のレベルも，それぞれの効率を著しく低下させるものであってはならない。

環境は，できるだけ多くの分散資源を環境内に取り込めることが必要である。資源としての重要度は，それがネットワーク上で特徴的な性質(例えば記号処理，グラフィックスが得意)を持っていること，それが数多く存在すること（例えばパーソナルコンピュータなど）などによって決まる。そのために，環境の枠組みはできるだけ広くしておく必要がある。

また，分散環境には多くのオブジェクトが混在するために，オブジェクトの分類が必要である。あるメッセージの列があった場合，それはある特定の種類のオブジェクトに対してのみ正常な動作が定義されている。どのようなオブジェクトがそのメッセージ列を正しく受理できるのか，で

きるだけ実行前にわかる必要がある。そのために，分散プログラミングに適したオブジェクトの型付けが必要である。

3．Kamui 環境

我々が現在構築中である Kamui 環境は，前章で述べたような様々に異なった言語や環境上で動作しているオブジェクトによる統合型分散プログラミング環境である。Kamui オブジェクトのプログラミングモデルは，並列計算モデル Kamui88[5] を基本にし，それに RPC と型を導入したものである。

Kamui 環境で動作する Kamui オブジェクトの定義を以下に述べる。

1）メソッドはアトミックアクションであり，オブジェクト中のメソッド間の並列性は無い。

2）メッセージキューを持っており，非同期的なメソッドの実行を行う。

3）Kamui 環境全体でユニークな ID が付いている。

4）自分の知っているメッセージプロトコルを答える。

5）メッセージプロトコルと名前の組によってメッセージを送る。

6）送られたメッセージをメッセージプロトコルと名前に分解し，対応したメソッドを実行する。

ここで，メッセージプロトコルとはオブジェクトに対する型の情報であり，くわしくは後で述べる。ここでの定義は Kamui 環境で動作できるオブジェクトとしての必要条件を述べたものであり，クラス，継承などのオブジェクトの実現に関することについては言及していない。したがってこの定義にのっとった様々なオブジェクトが Kamui オブジェクトとして考えられる。例として，以下に様々な Kamui オブジェクトを挙げる。

1）Kamui-C

Kamui-C[6] は C＋＋を基にしたコンパイル型の言語である。この言語で記述されたオブジェクトは，高速で効率のよい Kamui オブジェクトになる。

2）Kamui-Lisp

CommonLisp 上の言語であり，様々な動的な性質を持ったオブジェクトを実現するのに適している。

3）Kamui-Shell

UNIX のコマンドの標準入出力を Kamui オブジェクトのメッセージに変換する汎用のオブジェクトである。すなわち，UNIX の大部分のコマンドを Kamui オブジェクトとして扱うための変換オブジェクトである。

4）Kamui-HyperCard

HyperCard は Macintosh 上のカード型データベースである。その上で Kamui 環境とのインタフェースを実現し，外部の資源と協調動作させるものである。CSCW の基本的ツールになりうる。

5）Kamui-Term

PC 上の端末プログラムを Kamui 環境に参加させるものである。

このように，Kamui 環境は極めて広い範囲の分散資源をオブジェクトとして捉えることを目指している。

4．型付け機構

ここでは，Kamui 環境に取り入れられたオブジェクトの型付け機構メッセージプロトコルについて述べる。この型付け機構により，すべてのオブジェクトは似たような性質を持つもの同志分類され，オブジェクトを安全に使用できる。また，オブジェクトの部品としての性質が強調され，ボトムアップ的プログラミングの助けとなる。まず，その考え方の基となるオブジェクトの多重ビューについて述べ，次に，多重ビューをメッセージプロトコルを用いて定義する。

4.1. オブジェクトの多重ビュー

Meyer が述べているように[7]，一つのオブジェクトはそれが登場する場面によって異なった扱われ方をしている。例えば，自動車には，動いたりする性質と，空間的な物体としての性質，化学的な金属としての性質など様々な性質が存在し，それぞれに応じた扱われ方が対応している。これをオブジェクトの多重ビューと呼んでいる。

さらに例をあげると，全てのオブジェクトに対してオブジェクトの生成，消滅等に関して一般的なオブジェクトの捉え方があり，これを一つのビューと見なすことができる。一方，量的な特定のクラスのオブジェクトには，'>'，'<' などの量を比較する操作が用意されており，別のビューとしてみることができる。個々のビューは，そこで用いられるメッセージの種類によって特徴付けられている。オブジェクトの前述の例では生成消滅に関しては copy, dispose といったメッセージが，量に関しては，'<'，'>'，max :, min : といったメッセージがそれぞれ対応している。

4.2. メッセージプロトコル

メッセージプロトコル（又は単にプロトコル）はオブジェクトを操作する時の手順を与えるものである。プロトコルとして，メッセージの順序の制約を付けた言語もあるが[8]，ここでは単純に幾つかのメッセージの集まりをもってメッセージプロトコルと定義する。なぜならば，プロトコルを導入した主なる目的はオブジェクトの型付けのためであり，メッセージ順序の制約の必要はあまりないからである。

メッセージプロトコルの定義を以下に与える。環境全体で用いられている全ての名前の集合を N とすると，メッセージプロトコル p は N の部分集合である。

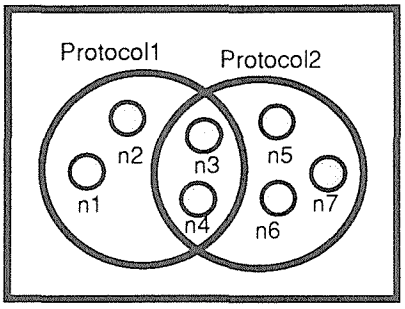
$$p \subset N.$$

メッセージはメッセージプロトコルと名前の対で表される。

$$mes = (p, n),\quad p \subset N,\quad n \in N.$$

この定義により，メッセージにスコープを与えたことになり，別のプロトコルに属しているメッセージに同じ名前を与えても混乱はない。

メッセージのスコープがない場合，幾つかのビューの間で要素が重複する場合があった。しかし，上記の定義ではメッセージの重複は存在しない。



名前の集合

	Prot1	Prot2
n1	m1	
n2	m2	
n3	m3	m5
n4	m4	m6
n5		m7
n6		m8
n7		m9

メッセージ表

図1　メッセージプロトコルの定義

4.3. オブジェクト

オブジェクトは，複数のビューを持っていることから，複数のメッセージプロトコルを持っていることになる。オブジェクトに送られたメッセージは，メッセージプロトコルと名前とに分解され，最初に，そのメッセージプロトコルを受け入れるかどうかの検査が行われる。その後，対応するメッセージプロトコルの表の中から，一致する名前の検索が行われ，必要なメソッドが得られる。

Kamuiオブジェクトは自分の知っているメッセージプロトコルを外部に対して答える必要がある。それによって，メッセージプロトコル単位でオブジェクトが受理可能なメッセージが分かり，実際のメッセージを送る前に検査をすることができる。

5．実行支援

ここでは，Kamui環境を稼働するにあたって，必要となる考え方とそのためのオブジェクト群について述べる。

5.1. ユニークなID

Kamui環境では環境全体でユニークな3種類のIDがある。

1）オブジェクトID

各オブジェクトにはネットワーク全体で固有のIDが付けられる。このIDから，オブジェクトがどのプロセスに属しているかがわかる。オブジェクトIDを知ることができれば，そのオブジェクトがどこに存在しようと自由にアクセスできる。

2）プロトコルID

メッセージの送出の際にそのベースとなるプロトコルを指定する。プロトコルは環境全体でグローバルであり，IDはユニークである。同じプロトコルである場合，それに属しているメッセージの引数と戻り値の型等は完全に同じであると見なされる。

3）タイプID

ネットワーク上で用いられているデータ型は，固有のIDが付けられている。このIDは，例えば異なるCPUや言語によってそれぞれ違う値を持っている。2つのタイプが同一種類であるならば，それは互いに変換可能である。

5.2. ID データベース

3種類の ID を管理するために，それに対応するデータベースがある。

1）オブジェクトデータベース

大域的に操作されるオブジェクトには名前を付けることができ，それはオブジェクトデータベースに登録される。そのオブジェクトを呼び出す場合は名前によってこのデータベースを検索する。データベースはそれ自身オブジェクトであるので，そのためにデータベースの階層ができる。オブジェクト ID が 0 のものは，特別なサーバオブジェクトでこれがデータベースの根となるものである。各オブジェクトは，ID 0 のオブジェクトのデータベースを用いて，他のオブジェクトの ID を知る。このオブジェクトが別のデータベースである場合は，さらに階層が一つ下がる。

2）プロトコルデータベース

コンパイラは，送出するメッセージのプロトコルと名前を用いて，それに対応するプロトコル ID を挿入し，必要な型変換の関数を通すルーチンを生成する。この場合，プロトコルデータベースによりその宣言と一致することを確認し，実際のプロトコル ID を得る。プロトコル宣言では，すでに宣言されているプロトコルとの衝突の有無などを調べ，データベースへの登録を行なう。

3）タイプデータベース

ネットワークに接続されているマシン上で動作している言語で用いられている，プリミティブな型を管理するデータベースである。ここには，同じ種類の型についての型変換の方法についても登録されている。新しいマシン，言語などをこの環境に追加する場合，このデータベースに登録が行われる。

これらのデータベースは，C 言語などで分割コンパイルを行なう際のヘッダファイルに相当する。データベースにすることにより，総合的な管理ができ，異言語間での情報交換も容易になる。

6. Kamui-C

ここでは，Kamui オブジェクトの記述言語の一つである Kamui-C について詳しく述べる。

6.1. Kamui-C の概要

Kamui-C は C++[9] をベースにした分散オブジェクト指向言語である。一つのプロセス中に複数個の Kamui オブジェクトが疑似並列で動作し，複数の Kamui-C によるプロセスがプロセス間通信により，Kamui オブジェクトの並行動作が可能になる。

一つのプロセス中には，Kamui オブジェクトと C++のオブジェクトとが共存している。C++のオブジェクトはそのプロセス中の局所的なオブジェクトであり，外部からは参照できない。そのため，C++のオブジェクトを参照する場合には直接オブジェクトにメッセージが送られる。これに対して，Kamui オブジェクトの場合には，そのオブジェクトの一つのビューを示すプロトコル型の変数を介して間接的にアクセスされる。この変数はオブジェクトがリモートな場合，スタブの役割をしており，プロセス間通信を実際に行っている。

Kamui オブジェクトがどのようなプロトコルを知っているかという情報は，一般に実行時にしかわからないので，Kamui-C は型検査に関して動的と静的な両方の性質を持っている。Kamui オブジェクトに送られたメッセージはプロトコルと名前とに分解できるが，名前には静的な検査が，プロトコルに対しては動的な検査が行われる。

6.2. プロトコル型

Kamui-Cにおけるプロトコル型は，そのプロトコルを知っているオブジェクト全体を意味している。すなわち，あるプロトコル型Pの属性が付いた変数Vには，そのプロトコルを知っているオブジェクトのみを代入でき，代入の検査は実行時に行われる。Vから指されているオブジェクトはすでにPを知っていることが保証されているので，それ以降Vを通じた処理は静的な検査で充分である。例えば，あるメッセージMの引数にプロトコル型Pが指定され，そこにオブジェクトAが引数として渡された場合を考える。Mを送る側は，プロトコルPを知っているAだけを送るので，Mを受けた側はP型としてAを動的な検査無しで使用できる。

6.3. 継承

Kamui-Cは単継承を行う。クラスにはインスタンス変数とメソッドの他に受け付けるプロトコルも宣言する。そのために，上位のクラスからプロトコルも継承する。例えば，Rootのクラス KObjectではPKamuiとPKamuiCの2つのプロトコルが記されている。プロトコルPKamuiにはKamuiオブジェクト全体に共通の操作(where：オブジェクトの位置を返すメッセージ等)が，プロトコルPKamuiCにはKamui-Cで記述されたオブジェクトに共通の操作(move：オブジェクトマイグレーション等)がまとめられている。

6.4. 構文

以下にKamui-Cの構文を簡単に述べる。

6.4.1. プロトコル宣言

プロトコルの宣言は以下の通りである。

```
%protocol 〈Protocol-Name〉
 {
   〈Type〉 〈Mes-Name〉 (〈Args〉) ;
   〈Type〉 〈Mes-Name〉 (〈Args〉) ;
    ..... } ;
```

ここで，〈Type〉にはC言語の通常の型の他に，sendという型が使用できる。sendは，計算モデルKamui88と同じメッセージの送出を非同期的に行うもので，ABCL/1[10]での過去型に相当するものである。この呼び出しによって処理の並行性が生じる。他の型の呼び出しはすべて同期的に行われるRPCであり，相手先のメソッドの実行が終了するまで待たされる。

〈Protocol-Name〉はKamui-Cでの型としての宣言でもあり，通常の型と同様に扱われる。処理系は，この宣言によってメッセージプロトコルDBへの登録を行う。

6.4.2. クラス宣言

クラスの宣言は以下の通りである。

```
〈Class-Defs〉 : =
   '%class' 〈Class-Name〉
   '[' 〈Protocol-Name〉 * ']'
```

```
    ' : ' ⟨Super-Class⟩ '{'
        ⟨Instance-Variables⟩
        '— — —'
        ⟨Protocol-Body⟩ *
    '}' ' ; '.

⟨Protocol-Body⟩ : =
    ⟨Protocol-Name⟩
    '{' ⟨Message-Body⟩ * '}' ' ; '.
⟨Message-Body⟩ : =
    ⟨Type⟩ ⟨Message-Name⟩
    '(' ⟨Arguments⟩ ')'
    '{' ⟨Method-Body⟩ '}'.
```

ここで，Instance-Variables 及び Method-Body は C＋＋のプログラムを直接記述する。

6.5. 他の分散プログラミングシステムとの比較

window system の幾つかは server-client モデルを用いた分散プログラミングである。GMW[11] では，stub 生成系によって C 言語と G 言語との通信部分を生成し，server-client 通信を行っている。そのほかに NCS，NFS などの RPC を基にした分散処理システムでも server-client モデルを用いており，重要な分散処理システムのモデルである。Kamui オブジェクトのモデルは並行オブジェクトであるため，server-client モデルを含んでいる。しかし，実際の Kamui-C の実行系では，複数の Kamui オブジェクトがプロセス中で共存しているので，このプロセス間の繋がりが問題となる。

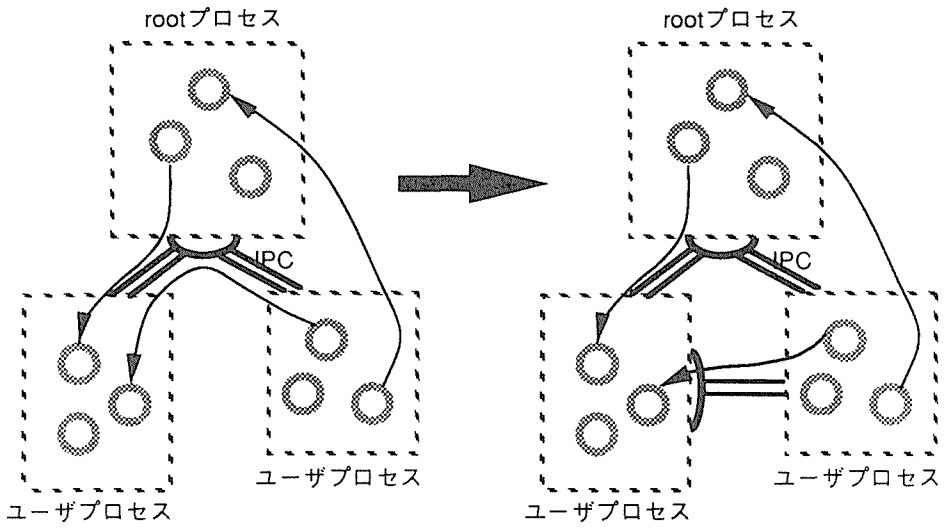


図2　プロセス間接続の変化

Kamui 環境の起動は，初めに，Kamui 環境の中心となるオブジェクトのプロセス(root プロセス)を起動し，次にユーザプログラムに相当するオブジェクト群のプロセスを root プロセスに接続する。これは，server-client モデルによるプロセス間通信で実現される。全てのオブジェクトはこの root プロセスを介して通信が行われる。

処理が進み，2つのユーザプロセス中のオブジェクト同志の通信が増えた場合，その両プロセス間に専用の通信のポートを設ける必要が出てくる。そのため，今度はどちらかのプロセスが server になって相手との接続を行なうことになる。Kamui-C の実行時ライブラリは，この server と client の両方に相当する機能を持っており，プロセスはいつでも server や client になることができる。

さらに新しく起動するプロセスは，通信のコストを考えて，一番都合の良いプロセスに接続される。幾つかのプロセスが起動すると，もはや server, client の区別は無くなり，極めて動的なプロセス間接続が実現されている。したがって最初に起動した root プロセスも特別な存在ではなくなり，それを切り離しても全体の動作に影響しない。

このように，Kamui の実行環境は client-server モデルを拡張した，柔軟な並行オブジェクトのモデルを実現している。

7. おわりに

Kamui 環境は，並列オブジェクト指向モデルに基づいた，統合型分散プログラミング環境である。Kamui 環境により，分散プログラミングを統一がとれた形で容易に行なうことができ，プログラムの再利用性も向上する。現在，Kamui-C と Kamui-Shell が動作しており，Lisp, Smalltalk, Pal, Laplas などの言語や HyperCard, Emacs などの環境の上にも Kamui 環境を実現中である。現在の実現は，ネットワークでの実行の効率をそれほど考慮していない。環境の実行効率の向上は，実用システムへ向けて必要な課題である。また，統合型分散プログラミング環境として，さらに様々なレベルの支援を今後探って行く必要がある。

CSCW のアプリケーションなどを構築する場合，ウインドウシステムやツールキットなどのライブラリが重要となる。それらは，設計の段階で分散を考慮しなければならないため，Kamui 環境でプログラミングするのに適している。今後の課題として，Kamui 環境でこのように大規模なプログラミングを行なうことが挙げられる。

最後に，本研究を進めるにあたって北海道大学工学部情報工学科　赤間助教授に貴重な御意見を頂いた。ここに感謝致します。

参考文献

1) A. Goldberg, D. Robson: Smalltalk-80: The Language and its Implementation, Addison-Wesley, Reading, MA, 1983.

2) 大里，竹内，奥乃：TAO におけるオブジェクト指向計算，オブジェクト指向　解説と WOOC'85 からの論文，共立出版，1985.

3) M. B. Jones, R. F. Rashid : Mach and Matchmaker : Kernel and Language Support for Object-Oriented Distributed Systems, OOPSLA'86 Proceedings, September, 1986.

4) A. Black, N. Hutchinson, E. Jul, H. Levy, L. Carter : Distribution and Abstract ypes in Emerald, IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, January 1987.

5）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.
6）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.
7）B. Meyer : Harnessing Multiple Inheritance, JOURNAL OF OBJECT-ORIENTED Programing, VOL. 1, No. 4, NOV./DEC., 1988.
8）, J. van den Bos, C. Laffra: PROCOL A Parallel Object Language with Protocols, OOPSLA'89 Proceedings, October 1989.
9）B. Stroustrup : The C++ Programming Language. Addison-Wesley, 1986.
10）米澤，柴山，Briot, J. P.，本田，高田：オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1，コンピュータソフトウェア，Vol. 3, No. 3, 1986.
11）大谷，角野，児島，萩谷，服部，劉：GMW ウインドウシステム上のアプリケーション構築について，コンピュータソフトウェア，Vol. 7，No.1, 1990.