計算機アーキテクチャ 77－9
（1989. 7. 14）

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

原田 康徳　浜田 昇　渡辺 慎哉　三谷 和史　宮本 衛市
北海道大学工学部

分散環境上の多くの資源を自由に使用し、複数のユーザで協調して利用したいという要求が高まっている。このような分散型アプリケーションを記述することは容易ではない。本研究は並列オブジェクト指向モデルKamuiに基づいたネットワークプロトコルKamuiProtocolについて報告する。KamuiProtocolは、異なったマシン、言語間の上で動作する。各プログラムは幾つかのKamuiオブジェクトとしてみなされ、それらが共通のKamuiProtocolによって通信しあう。

Distributed Application Construction Based on an Object-Based Concurrent Model

Yasunori Harada　Noboru Hamada　Sin'ya Watanabe
Kazufumi Mitani　Eiichi Miyamoto
Department of Information Engineering
Faculty of Engineering
Hokkaido University
kita-13jou,nishi-8tyoume,kita-ku,Sapporo 060,Japan

Recently,there have been increasing demands for using many resources by many users in a destributed environment. But,such distributed application programs are not easy to write. This article describes the network protocol 'KamuiProtocol' which is based on the object-based concurrent model Kamui. KamuiProtocol works on diffrent machines and programing languages. Programs based on the Kamui model are defined as a collection of Kamui Objects which communicate by KamuiProtocol.

1 はじめに

近年、多くのコンピュータが互いに接続されネットワークを構成することが増えてきた。この様なネットワークには多くの資源が接続され、多くのユーザがこれらを用いている。豊富なネットワーク資源を自由に使用したり、一つの資源を複数のユーザによって協調して利用という要求が、今後これまで以上に高まって来ることが予想される。

これらのコンピュータは一般に様々なCPU、OS、I/Oを持ち、また様々な言語がその上で動作している。したがって、こういった分散環境におけるプログラミングは、従来のそれと比較して難しい。そこで、分散環境全体をひとつに捉えた、プログラムの構築法が必要になってくる。

本論文では並列オブジェクト指向モデルに基づいて、分散型のアプリケーションを構築する方法について述べる。本構築法を用いることにより、様々な形態の協調問題を表現することができ、ネットワーク上の資源を有効に利用することができる。また、本構築法に基づいた幾つかの言語を紹介し、それらが協調して動作する例を示す。

2 研究の背景

この章では、研究の背景について述べる。

2.1 ネットワーク上の資源

ネットワーク上に存在する資源を共有の点から考察してみる。

(a) 計算資源

ネットワーク上の計算資源を有効に利用することにより、より高速な処理が可能である。また、特別な目的の計算のために専用のハードウェアを付加したものもある。例えば、Lispマシンは記号処理に有効なハードウェアを持っている。

(b) 記憶

幾つかのレベルがあるが、ネットワーク上の資源として有用なものは二次記憶である。NFS, RFS,TOPSは、ネットワークに接続されたマシンの二次記憶を共有するためのプログラムである。

(c) 入出力デバイス

スキャナのようなデータの取り込み装置や、プリンタような出力装置は高価な資源であるので、共有の利点がある。また、キーボード、マウス、ビットマップディスプレイ、スピーカ等の共有は良いユーザインタフェースの実現に寄与するであろう。

(d) ソフトウエア

ある特定の機種でしか動作しないソフトウェアを、他の環境から利用できるようにすることにより、ネットワーク上の重要な共有資源となる。

2.2 通信の分類

ネットワーク上で動作するプログラムは、プログラム間の通信を基にして組まれている。このコンピュータ間の通信を分類する。

2.2.1 1対1通信

1対1通信は、2つのプログラムの間で行う通信である。通常、この2つのプログラムの間で特定のプロトコルにより通信を行う。

(a) ターミナルエミュレータ

ホストコンピュータ上のプログラムとの1対1通信を行い、画面制御や特殊キーの為にESCシーケンスが用いられる。この中には、ホストコンピュータとのファイルの転送などができるものも有り、ここにも何らかのプロトコルが存在する。

(b) 外部機器の制御

外部機器との通信におけるプロトコルは、プリンタなどは多くの場合 ESC シーケンスが用いられているが、より強力な表現力を持った PostScript の様な言語が用いられる場合もある。プロッタの制御では、特別なグラフィック言語がプロトコルとして用いられている。

(c)talk

二人の人間が画面を通して互いに対話するプログラムである。これは、それぞれのユーザに対し一つのプロセスが対応し、互いにチャネル (ソケット) で結ばれている。各プロセスは標準入力と、プロセス間のチャネルの両方を監視しており、相手の入力をこちらの画面に出力し、こちらの入力を相手に送る。

(d) リモートプロシジャコール (RPC)

一つのプロセスが、別のマシン上で動作しているプロセス中のプロシジャを自分のプロセス中に存在するものとして呼び出す方法である。Matchmaker [4] は通信のプロトコルに相当する部分を自動的に生成する (スタブ生成系)。NFS は、ファイルの共有のためのプログラムであるが、RPC ライブラリを用いて記述されている。

2.2.2 クライアントサーバ方式

クライアントサーバ方式は一対多の通信である。クライアントの出す要求の処理をサーバが順番に行い、その返答をクライアントに返す。サーバは、一つの処理が終了すると、また次のクライアントからの要求待ちに入る。この方式はサーバがクライアントの要求を常に待って、その処理を実行する無限ループで実現される。

(a) ウィンドウシステム (X-window)

クライアントは描画に対する指令を RPC によってサーバに要求し、実際の描画はサーバが行う。サーバは、ウィンドウの重なりの管理、入力デバイスなどの処理を行い、各クライアントに関係のある情報をイベントとしてクライアントに送る。

(b) かな漢字変換 (WNN)

クライアントの要求に対して、漢字の変換などを行い、その返答を返す。

2.2.3 ブロードキャスト方式

それぞれのプロセスは互いに情報を送り、全員が平等に全体の処理を行う。この特徴は、中央に特別なプログラムを必要としない点である。全体が平等なためにプログラムは難しくなる。

(a)rwhod

マシンごとに存在する各デーモンは、そのマシンにログインしている人の情報を 1 分ごとに、ブロードキャスト通信で全部のデーモンに送る。これによって、ネットワーク全体にログインしているユーザがわかる。

2.3 プロトコルの統一化の必要性

以上のように分散環境では、様々な資源を利用し、様々な方式で通信を行っている。こういった利用がいっそう進められるためには、ネットワーク全体で統一のとれたアプリケーションの構築法が必要である。

3 KamuiProtocol

我々は、並列オブジェクト指向モデルを導入し、分散型アプリケーションの一般化を試みた。

3.1 Kamui モデルによる記述

並列オブジェクト指向モデル Kamui88 [1] を用いて分散型のアプリケーションを一般的に記述することを試みる。Kamui モデルは、actor [5] をベースにした並列計算モデルで、場を用い

てオブジェクトをグループ化するところに特徴がある。

Kamui では、オブジェクト間の情報のやりとりをイベントを用いて行う。イベントを送る方法は、オブジェクトに直接送る場合と、場に対して送る場合の2種類ある。場に送られたイベントは、その場に入っているオブジェクト全部にブロードキャストされる。場はオブジェクトの間接的な参照でもあり、全体的に柔軟なシステムが構築し易い。

3.2 KamuiProtocol の定義

KamuiProtocol は、ネットワーク間の異機種、異言語間の呼び出しを行うプロトコルである。そこで定義されているものは以下の通りである。

1. 並列に動作する単位がオブジェクトである。
2. オブジェクト間の通信には、send, call の2つがある。
3. 通信は、ネットワーク全体でユニークな意味を持っている event ID で行う。
4. event の引数、戻り値の型は、全体で統一された type ID で定義されている。

KamuiProtocol はオブジェクトの内側の記述については触れていないので、その部分は自由に設計できる (図1)。

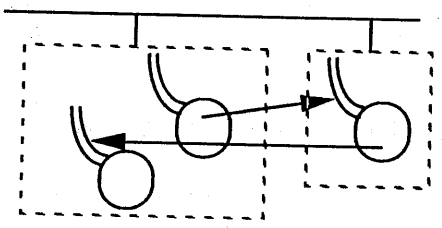

図1 ネットワーク上のKamuiオブジェクト

3.2.1 イベントのタイプ

最初のモデル Kamui88 ではイベントは非同期的に送られ、待ちは存在しない。送られてきたイベントキューに入れられ、その先頭のイベントから順に処理される。

KamuiProtocol では、これを拡張して返答付きのイベントを用意した。これは、通常の関数呼び出しと同じアナロジーで、SmallTalk [2] などのメッセージセンドと同様のものである。列挙すると、

(a) send

情報を非同期に送出

(b) call

情報を送出し、その返答を待つ (図2)

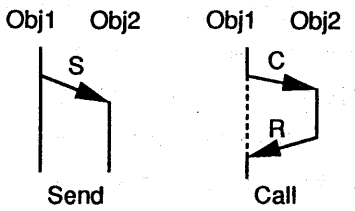

図2 イベントのタイプ

call の実行中のオブジェクトは、待ちに入っている。その時、他からのイベントの受理に工夫が必要である。ABCL [6] などでは、各メソッドの実行はアトミックアクションであり、この待ちの最中に速達モード以外のイベントを受けることができない。このために、逐次型の言語では一般的である再帰的な関数呼び出しが記述できない。

KamuiProtocol では色付きスレッドを導入し、

再帰的な call に対しては待たずにその処理を行えるようにした。ここでスレッドは並行に実行する単位である。send が起こる度にイベントの受け側は送り側とは異なった新しい色の付いたスレッドになる。一方 call は送り側と受け側の両方が同じ色のスレッドである。

call のイベントが送られてきた場合、そのスレッドが自分と同じ色であるならば、その call イベントを処理することができる。しかし、自分と異なった色のスレッドの場合は、それが再帰的なものではないので、現在のスレッドが終了するまでイベントの処理は待たされる (図3)。

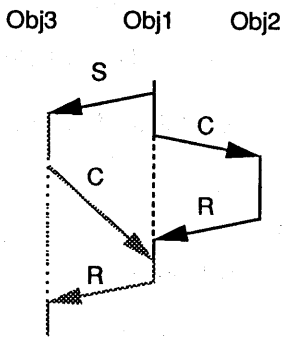

図3色付きスレッド

3.2.2 Kamui プロセス

並列に動作するオブジェクトが幾つか集まって Kamui プロセスとなる。UNIX などのマルチタスクの OS では、そこでのプロセスがこの Kamui プロセスである。PC などのシングルタスクの OS 上では、そこで動作するプログラムがプロセスとなる。これは、モデル的には必要のない概念であるが、実際のインプリメントにおいて必要である。

3.2.3 処理系

KamuiProtocol 上の処理系はコンパイラ型とインタプリタ型の両方が考えられる。コンパイラ自身も、KamuiProtocol での一つのオブジェクトであり、他のオブジェクトと通信しながらコンパイルを行う。

3.2.4 ネットワークでユニークな ID

KamuiProtocol にはネットワーク全体でユニークな3種類の ID がある。

(a)object 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。Object 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。

(b)event ID

情報の送出は event を用いる。event には、ネットワーク上で共通の ID が付けられている。ある名前 (意味) の event は、すべて共通の ID を持つ。さらに、各 event には、その引数の個数、型と、もし値があればその型とが一意に定義されている。

(c)type 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 type が同一種類であるならば、それは互いに変換可能である。

これら3種類の ID により、異なったマシン、言語間での意味の統一を図ることができる。

3.2.5 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。

(a)object ID データベース

object は名前を付けることができ、それは Object ID とともにデータベースに登録しておく。データベースはそれ自身オブジェクトであるので、ここにデータベースの階層ができる。object 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 Object は、 ID 0 のオブジェクトのデータベースを用いて、名前により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。

(b)event ID データベース

コンパイラは、送出するイベントの名前を用いて、event ID データベースに問い合わせ、その ID を送出するコードや、必要な型変換の関数を通すコードを生成する。event の宣言では、既にその event が登録されていれば、それと型が一致するかどうか調べる。登録されていなければ、そのイベントの引数、値の型で登録を行う。

event ID データベースは、通常後からイベントが追加されるだけで、削除は行われない。つまり常に上位に互換である。イベントが追加される度に、データベースのマイナーバージョンが更新される。オブジェクトはそれがコンパイルされたときのデータベースのバージョンを知っているので、不慮の事故や、削除などで event ID データベースが自分のそれより古いものに戻った場合、各オブジェクトは警告、終了などの処理を行うことができる。イベントの削除は特別なツールを用いて行わなければならない。

(c)type ID 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。この場合、やはりバージョンの管理が必要である。

4 KamuiProtocolオブジェクトの例

オブジェクトの内部の動作の記述は、KamuiProtocol の定義の外側にある。ここでは、幾つかのオブジェクトを記述する例をあげる。

4.1 Kamui-C

オブジェクトの定義の為に C 言語を用いる。このために、以下の拡張が為された。

1. 単継承のオブジェクト指向言語。
2. 各オブジェクトは、並行に動作可能。
3. イベントの送出に拡張された文法を用いる。

イベントの送出は、

send の場合

```
[ object ← event(arg1, arg2, ...)];
```

call の場合

```
var = [ object.event(arg1, arg2, ...)];
```

の様に行う。場に対する操作は、C の通常の関数を用いて行われる。

この処理系は、C 言語のプリプロセッサとして動作し、C 言語のプログラムを生成する。それに Kamui の実行時ライブラリをリンクすることによって、実行可能となる。

イベントは、コンパイル時に event ID データベースをアクセスし、実際の ID に変換される。

4.2 Kamui-Lisp

Lisp を用いてオブジェクトの内部を記述することにより、オブジェクトに動的な性質を持たせることができる。Kamui-Lisp は、イベントの送

出の機能を関数で用意し、event ID と atom との対応表により、Lisp の世界との整合を取っている。

実際のオブジェクトの定義は、CLOS [3] などのオブジェクト指向ツールを用いることができる。

4.3 Kamui-Shell

UNIX の一つの特徴であるフィルタコマンドを、Kamui-Shell を用いて KamuiProtocol に参加させることができる。これにより、各フィルタは Kamui の一つのオブジェクトとみなすことができ、他のオブジェクトから利用できる (図4)。

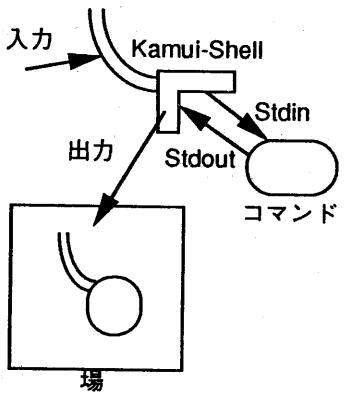

図 4 Kamui-Shell

標準入出力を用いている多くの言語 (例えば、lisp, prolog, awk, APL 等) もまた、この Kamui-Shell により KamuiProtocol に接続できる。Kamui-Shell は、文字単位と行単位の入出力のインタフェースを持っている。このコマンドに与える入力は、Kamui-Shell オブジェクトに直接 event を送り、コマンドの出力は、Kamui-Shell オブジェクトのインスタンス変数 stdout から指されている場に event として送られる。ここで場はパイプの役目をしているので、そこを通じて次のオブジェクトへ情報が伝えられる。

4.4 Kamui-HyperCard,Kamui-SuperCard

HyperCard は、Macintosh 上のカード型マルチメディアデータベースで、SuperCard はマルチウィンドに対応した HyperCard に上位互換のツールである。それらの拡張機能である XCMD, XFNC を用いることにより、KamuiProtocol に接続することができる。他のオブジェクトからは、Kamui-HyperCard は一つのオブジェクトとして見え、幾つかのイベントに反応する。

これにより、HyperCard から EWS の資源を利用したり、その逆に EWS から HyperCard を呼び出したりすることが可能となる。例えば、HyperCard で EWS 上のアプリケーションのマニュアルを作成し、アプリケーションからそのマニュアルの参照の event を送ることにより、動的な help 環境が実現できる。

4.5 KamuiProtocol の利点

KamuiProtocol を用いることにより、以下のような利点がある。

1) 参照透化性

オブジェクトはネットワークでユニークな ID で参照されるために、トランスペアレントな参照ができる。つまり、全てのオブジェクトはそれがどこに存在していても、対等にアクセスできる。

2) 動的な性質

C 言語やそれをベースにしたオブジェクト指向言語 (C++, Objective-C 等) は、動的にクラスを変更したり、現在動作しているプログラムに動的に新しいオブジェクトを参加させるなどの動的な性質はなかった。Kamui-C では、各プロセスが動的に他のプロセスと接続し互いの内部

データを送り合う、などのことができ、動的である。KamuiProtocol に参加するオブジェクトは具体的にどのようなベース言語で記述されていても構わない。Lisp は本来動的な言語であることから、Kamui-Lisp などで組まれたオブジェクトは、動的な性質を持っている。このオブジェクトが参加しているところは、動的なものとなる。

3) 対話的な性質

2) と同様の理由から、対話的な言語がこの KamuiProtocol に参加することにより、対話的な処理が可能となる。これは、対話的にイベントを発生させることで、別のプロセスのオブジェクトを動作させるものである。対話的な性質は、プログラムの開発時に特に効果的である。

4) 信頼性

各オブジェクトは、自分の内部状態を他のプロセスに送り自分の複製を作成することができる。高い信頼性が必要なオブジェクトは、頻繁に自分の複製を作成して、自分が壊れた時にその代行をしてもらうことができる。このようなことは、異なったマシン間であれば、更に高い信頼性を得ることができるであろう。

5 まとめ

現在、KamuiProtocol Ver1.0 は北大情報工学科の Ethernet 上で稼働中である。さらに、DomainTokenRing 、AppleTalk 、EWS とのシリアル回線などのネットワークへの移植が計画されている。Kamui-C などのコンパイラはすでに動作している。

課題としては、多くの機種への移植、多くの言語への対応などがあげられる。また、広い KamuiProtocol 上のアプリケーションの実現、特に分散型ユーザインターフェースへの応用などが今後の課題である。

謝辞

本研究を進めるにあたり、富士通国際研究所 松岡 雅裕 氏に貴重なご意見を頂いた。また、北大工学部 赤間 清助教授に多くの御助言を頂いた。ここに感謝致します。

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] Sonya E.Keene : *Object-Oriented Programming in Common Lisp (A Programmer's Guide to CLOS)* , Addison-Wesley, 1988.

[4] Michael B.Jones, Richard F.Rachid : *Mach and Matcmaker: Kernel and Language Support for Object-Oriented Distributed Systems* , OOPSLA'86 Proceedings , pp67-77, 1986.

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

[6] 米澤明憲, 柴山悦哉,Briot,J.P., 本田康晃, 高田敏弘: 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), pp.9-23.