ソフトウェア工学 87-- 3
(1992. 9. 29)

オブジェクトのモジュール化による複合オブジェクトの生成

宮本衛市　武村功司　渡辺慎哉

北海道大学 工学部 情報工学科
060 札幌市北区北 13 条西 8 丁目

あらまし

分散オブジェクト指向プログラミングは、分散システムの発展とともにプログラミングパラダイムの一つとして、今後ますます重要になってくる。そこでは不特定多数の計算機資源や人間が関与し、安全で、効率のいいソフトウェアが構築されなければならない。かつて、われわれはメッセージ集合に基づくオブジェクトの型機構を基礎とし、オブジェクト間におけるメッセージの送受機構として入力ビューと出力ビューを提案した。今回、これら入出力ビューをもとにしたオブジェクトの仕様を定義し、これに基づきオブジェクト群の接続構造を陽に記述することを提案し、分散プログラミングの枠組にも言及している。

和文キーワード

分散オブジェクト指向、型機構、オブジェクト接続構造、複合オブジェクト

Generation of Complex Objects Based upon Modularization of Distributed Objects

Eiichi MIYAMOTO　Kohji TAKEMURA　Shin-ya WATANABE

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

Abstract

Distributed object oriented programming is becoming more and more important as a programming paradigm along with the development of distributed systems. In such systems, many and unspecified computer resources and programmers take part in the construction of distributed software, so it must be implemented safely and efficiently. We have ever proposed input and output views as the mechanism to send and receive messages between objects based upon the type mechanism for objects by message set. We now propose to describe the linkage structure of objects explicitly based upon the specification of objects by input and output views, and refer to the framework of distributed programming.

英文 key words

Distributed Object Oriented, Type Mechanism, Object Linkage Structure

1 はじめに

分散オブジェクト指向プログラミングは、分散システムの普及発展とともに、プログラミングパラダイムの一つとして、今後ますます重要になってくるものと思われる。分散オブジェクト指向プログラミングはオブジェクト指向プログラミングのパラダイムを継承し、さらに分散環境でのプログラミングに適応したパラダイムの確立を目指して様々な角度から研究が行なわれている。

分散環境の特徴は計算機資源がネットワークを介して分散していることであり、そのため複数の計算機がその処理に関与するばかりではなく、複数の人間が同時に関与することも想定せざるをえない。しかも、関与する計算機および人間がいつも特定できるとは限らず、むしろ不特定多数の集団を一般的には想定しなければならない。このような分散環境に対して、情報隠蔽や部品化の点で優れているオブジェクト指向モデルが取り上げられ、分散システムに適応させた様々なモデルが提案されている [1]。

このような問題に対し、われわれは Kamui オブジェクトモデルを提案し、分散環境における、安全で、かつ柔軟なプログラミングの実現を目指してきた。一般的には、分散環境では単一の言語でプログラミングされるよりは、むしろ複数の言語でプログラミングされる事態を想定せざるをえない。メッセージで情報のやりとりを行なうオブジェクトに基づくプログラミングにとって、クラス階層に基づく型づけでは不十分であることを指摘し、われわれはメッセージの集合に基づく型機構を提案した [2]。

かつて、われわれはオブジェクト間の関係には 2 つの性質の異なる関係があることを指摘した [3]。1 つは主従の関係のある、いわば下請をするオブジェクト群とそれらを統括するオブジェクトの関係であり、他の 1 つがオブジェクト間に上下関係のない、いわば対等な関係である。前者の関係は階層的な関係であり、それらの外部からはこの統括するオブジェクトのみが見えていればよいが、後者はグラフ的な関係をもち、いずれかのオブジェクトに帰結させうるような関係ではない。後者のような関係を陽に記述するために、われわれはメッセージの送受の仕様をプロトコルに基づくビューで表現し、そのビューを入力ビューと出力ビューに分離して宣言すべきことを提案した。これにより、オブジェクトの生成、接続が、オブジェクトの中からばかりでなく、オブジェクトの外からでもできるようになった。

分散アプリケーションが大規模になるにつれ、その構築に何人もの人間が携わるようになり、それを分散オブジェクト指向で構築する場合、分散オブジェクトの管理が極めて重要な問題として浮かび上がってきた。オブジェクトの接続関係をメッセージを通して行なうと、その実現形態に応じて様々なメッセージを用意しなければならず、またメッセージの送出はプログラムの中に埋め込まれていて、オブジェクト間の接続は実行時に形成されるため、接続関係の掌握が難しく、接続を誤る可能性が高まるのを避けられない。接続の誤りはデバッギングの難しい分散プログラミングでは致命的である。例えば、グループでまとまった仕事をするオブジェクト群は、1 つのメッセージをトリガとして、グループの生成と接続をする場合があるが、そのメッセージ名を見ただけで、どのような接続関係を構成することになるのかを理解するのは容易なことではない。このような事態を防ぐためには、オブジェクトの接続関係を宣言的に記述して、陽に見えるようにすることが必要である。本報告では、われわれの提案してきたオブジェクトの型機構がオブジェクトの接続に有効であることを示し、オブジェクト自身の振舞いとは別に、オブジェクト群の構成を記述する構造記述言語 Kamui Script を提案する。

一方、大規模なシステムを構築するときには、オブジェクトをまとめるモジュール化の仕組みが必要になる。その際、情報隠蔽の立場から、モジュールのあるなしにかかわらず、そこに含まれるオブジェクト群に対する外部仕様は変えるべきではない。さらに、分散処理の立場から、モジュール化により特定のオブジェクトへ処理の集中が起こって、システムの効率の低下をきたしてはならない。この観点から、われわれは分散プログラミングに対する新しいモジュール化の概念を提起する。

以下、2. では分散オブジェクトを構築する基礎となる型機構について述べ、Kamui オブジェ

クトのもつ枠組について明らかにする。3. では分散アプリケーションを分散環境で実行するため、オブジェクト群の生成、接続、更新などに関する構造記述の枠組について述べ、さらに分散ソフトウェアの開発へのアプローチについても言及する。4. では分散プログラムを実行する環境について概要を述べる。5. で本報告のまとめと、今後の課題について述べる。

2 分散オブジェクトの型機構

ネットワーク上に分散し、われわれの提唱する型機構を備えた分散オブジェクトをKamuiオブジェクトと呼んでいる。われわれはメッセージに基づくオブジェクトの型機構についてすでに提案しているが、そこではメッセージの集合をプロトコルと呼び、このプロトコルでオブジェクトにおける型に対応させる。また、オブジェクトがいくつかの仕事を抱えているときには、それぞれの仕事に対応してプロトコルをもち、そのプロトコルに対応したメッセージの送受機構をもつものとする。この機構をビューと呼び、これが複数あるときには多重ビューをもつという[4]。ビューにはメッセージを受け付ける入力ビューと、メッセージを送出する出力ビューがある。

2.1 入力ビュー

入力ビューはオブジェクトが抱えているそれぞれの仕事へのメッセージを受け付ける機構であり、受け付けるプロトコルと仮引数の対で表す。Kamuiオブジェクトでは、メッセージをキューに入れ、FIFO で処理していくので、この仮引数はビューを区別するための役目をもつだけで、オブジェクト内部で使われることはない。また、ビューがいくつかあるとき、各ビューごとにメッセージキューをもたせるか、1つにまとめてしまうかは、オブジェクトの実現いかんにかかっている。

2.2 出力ビュー

あるオブジェクトが他のオブジェクトへメッセージを送出するときには、相手のオブジェクトを指す内部変数を通して行なうが、オブジェクト間の接続をオブジェクトの外から宣言的に行なえるようにするため、この内部変数をビューに含めることにする。すなわち、出力ビューは送出メッセージを含むプロトコルと相手先のオブジェクトを指す内部変数の対で表す。もちろん、相手側には対応する入力ビューが備わっていなければならない。

通常、このような内部変数は情報隠蔽の立場から外部に公開されることはないが、Kamuiオブジェクトではオブジェクトの生成・接続をオブジェクトの振舞いとは別に、オブジェクトの外部で宣言的に行なうことを意図して、その存在をビューで示すことにしている。

2.3 オブジェクトの仕様

オブジェクトはいくつかの入力ビューと出力ビューを有しているので、それらを集めてオブジェクトの仕様と呼ぶ。図1に示すオブジェクトの仕様は次のように表す。

$$[Ax, By, -Cz, -Dw]$$

ここで、前2つは入力ビューを表し、後ろ2つが出力ビューを表す。プロトコルの前に付けた'$-$'は、それが出力ビューであることを示している。Ax および Byは、それぞれプロトコル A および Bに含まれるメッセージを受け付けることを示し、もし A および Bが同じプロトコルであるときには、x あるいは yにより入力ビューの区別をする。$-Cz$および$-Dw$は、それぞれ C および Dのプロトコルに含まれているメッセージを送出することを示し、送出先はそれぞれ変数 zおよび wが保持することを意味している。

Kamuiオブジェクトはクラスのインスタンスとして生成されるが、オブジェクトの接続は仕様により行なわれる。それゆえ、Kamuiオブジェクトは強い型検査機構のもとでメッセージの送受が行なわれることになる。この仕様をもとに、あるオブジェクトの出力ビューと他のオブジェクトの入力ビューの接続を厳密に、かつオブジェクトの外側で行なうことができる。そこで、われわれは次節で述べるように、オブジェクト自身の振舞いの記述から、オブジェクトの生成、接続などに関する記述を分離し、後者を別個に宣言することを考えている。

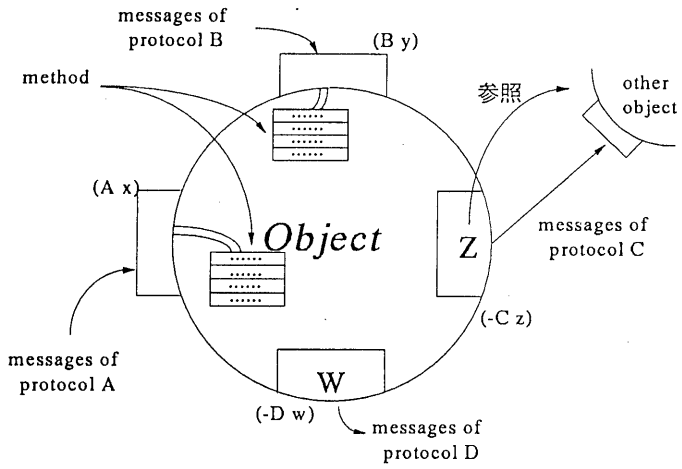


図 1: オブジェクトの概念図

3 オブジェクト群の構造記述

われわれは分散アプリケーションを実行するときに、まず基本的な、あるいは永続的なオブジェクトを生成し、それらを接続してからオブジェクト群に起動をかけるような計算機構を考えており、本節ではオブジェクトの実行環境の設定を記述するオブジェクト構造記述言語 Kamui Script について述べる。さらに、この言語を用いて、オブジェクトが稼働しているときにも、オブジェクト群の構成に介入するコマンドを発行することもできる。

3.1 基本操作

分散オブジェクトで分散アプリケーションを構成するためには、オブジェクトの生成、接続などの基本的な操作が必要である。また、アプリケーションの動的な変更を行なうためには、オブジェクトの交換や消滅、接続の変更などの操作が必要である。そこで、Kamui オブジェクトを操作するために、次のような基本操作を用意する。

(a) オブジェクトの生成

$$\&obj = new\ \langle CLASS \rangle\ at\ \langle DOMAIN \rangle$$
$$mode\ \langle "PERSISTENT" | "TEMPORARY" \rangle$$

クラス $\langle CLASS \rangle$ のオブジェクトをドメイン $\langle DOMAIN \rangle$ 上に生成する。ここで、ドメインとは開放型システムに対処するためにネットワーク内に設定する領域で、この領域内ではグローバルな名前づけができるものとする [5]。$\&obj$ のように、'&'で始まる変数は Kamui Script 上での変数であり、生成されたオブジェクトを指すポインタである。$mode$ "$PERSISTENT$" は永続的なオブジェクトとして生成することを意味し、生成されたオブジェクトはデータベースに登録される。$mode$ "$TEMPORARY$" は他のオブジェクトとの接続が絶たれたときには、ガーベジコレクションの対象となるような一時的なオブジェクトである。

(b) オブジェクトの除去

$$delete\ \langle OBJECT \rangle$$

$mode$ "$PERSISTENT$" で生成されたオブジェクト $\langle OBJECT \rangle$ を除去することを意味する。ただし、そのオブジェクトは他のオブジェクトとの接続を絶たれていることを確認する必

要があり、そのため永続的なオブジェクトの接続情報もデータベースに登録されているものとする。

(c) オブジェクトの接続

link ⟨*OUTPUTVIEW*⟩
with ⟨*INPUTVIEW*⟩

例えば、図2に示すようにオブジェクト間を接続させたいときには、

&obj1 : [−*Aa*, *Bb*],
&obj2 : [*Ap*, *Cq*],
&obj3 : [−*Ca*, −*Bb*]
link &obj1.a with &obj2.p
link &obj3.a with &obj2.q
link &onj3.b with &obj1.b

と記述すれば、出力ビューの変数に送信先の入力ビューが設定される。ここで、出力ビューと入力ビューのプロトコルの整合性が判定される。

(d) オブジェクトの接続の解除

unlink ⟨*OUTPUTVIEW*⟩
from ⟨*INPUTVIEW*⟩

図2で、*&obj3* と *&obj2* との接続を切り離したいときには、

unlink &obj3.a from &obj2.q

として、オブジェクトのビュー間の接続を解除する。

(e) オブジェクトの置換

replace ⟨*OBJECT1*⟩ *with* ⟨*OBJECT2*⟩

分散アプリケーションを実行中に、更新などのためオブジェクトの置換をするときのための操作で、上の例で

replace &obj3 with &obj4

とすれば、オブジェクト *&obj3* は *&obj4* に置き換えられる。ただし、置換される前後のオブジェクト間で、仕様が整合していなければならない。

この操作は *link* および *unlink* で実現できるのでアトミックな操作ではないが、使い勝手をよくするために基本操作に加えたものである。

3.2 複合オブジェクトの構築

アプリケーションが大規模になってくると、個々のオブジェクトの接続を行なっていたのでは記述量が増大し、オブジェクト間の接続形成も誤りやすくなり、一方オブジェクト群全体の構成が理解しにくくなる。そのためには、オブジェクト群のモジュール化が必要となる [6][7]。オブジェクトのモジュール化に当たっても、情報隠蔽の考えは重要であり、モジュールの仕様のみを公開し、モジュールを構成するオブジェクト群やそれらの接続に関するモジュール内部の情報は、モジュールの外部からは隠蔽しなければならない。さらに、分散オブジェクトをモジュール化することにより、特定のオブジェクトに処理の集中が起こってはならない。これらの点を考慮して、分散オブジェクトのモジュールを次のように定義する。

module ⟨*NAME*⟩ [⟨*SPECIFICATION*⟩]
{
  generation of objects;
  hold out internal views
    to specification views;
  linkage of internal views;
  initialization of objects;
}

上のモジュールの定義は複合オブジェクトを定義するクラスに相当し、必要に応じてモジュールオブジェクトを *new* で生成する。モジュール定義では、まずモジュール名 ⟨*NAME*⟩ と仕様 ⟨*SPECIFICATION*⟩ を宣言し、外部に公開する。次いで、モジュールの実現方法について記述するが、これは外部に対しては隠蔽される。まず、仕様のビューは内部のオブジェクトのビューを振り替えて実現する。すなわち、モジュールの仕様に現れるビューは内部のオブジェクトのビューが直接顔を出したもので、外部とのメッセージのやりとりは内部のオブジェクトが直接行ない、モジュール全体を仕切るオブジェクトはモジュール構成上からは要求しないものとする。そのようなオブジェクトを設置すると、そのオブジェクトにメッセージが集中し、分散システム全体の処理効率を低下させる原因となりかねないからである。続いて内部のオブジェクト間の接続を行ない、初期化のためのメッセー

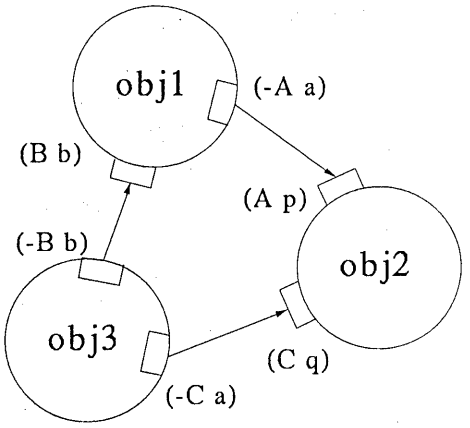


図 2: オブジェクトの接続例

ジをオブジェクトに送る。モジュールを定義するとき、すでに定義済みのモジュールを通常のクラスと同じように扱い、モジュールの入れ子構造を作ることができる。しかし、その入れ子構造は階層構造にはならない。上の定義で述べたように、モジュールを仕切るオブジェクトを要求していないので、モジュールが実現したときにはモジュールはフラットに展開されるからである。もちろん、記述上は抽象データ型に対応した情報隠蔽が保証されている。

複合オブジェクトの例として GUI ツールを Smalltalk の MVC モデルに対応したオブジェクト群で構成してみる。*model*, *view*, *controller* を図 3 (a) で示すようなクラスとすると、同図 (b) に示すモジュール MVC は次のように定義することができる。

```
module MVC [pCc]
{
  &m = new model;
  &v = new view;
  &c = new controller;
  c = &c.c;
  link &c.v with &v.v;
  link &m.mv with &v.mv;
  link &v.mw with &m.m;
  &v.update();
}
```

以後、定義されたモジュール MVC は、仕様 [*pCc*] をもつオブジェクトのクラスとして扱うことができる。

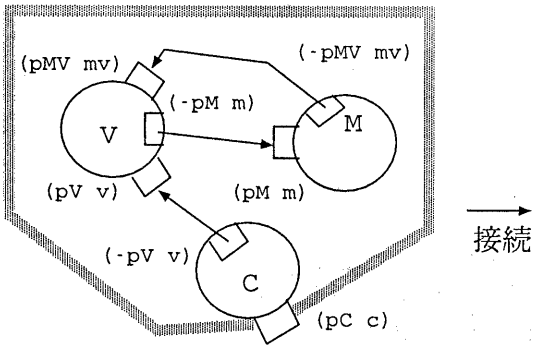
3.3 分散プログラミング

最近はオブジェクト指向によるソフトウェア開発に関する研究が盛んである。ソフトウェア開発をトップダウン的に考えると、基本的には 1 つの大きなオブジェクトをいくつかのより小さなオブジェクトに分割していくものであり、そのために元のオブジェクトの根となるオブジェクトと、そのオブジェクトの配下となるオブジェクト群とに細分化していく。もちろん、このプロセスはトップダウンアプローチに基づく典型的な方法論といえる。しかし、このようなオブジェクトの分割法は分散プログラミングを対象としたとき、根となるオブジェクトにメッセージが集中する可能性があり、分散システムの処理効率の点で問題となる。

そこで、われわれは分散ソフトウェアの解析・設計において、トップダウンに展開していくときにも、根になるオブジェクトは作らず、均一なオブジェクト群に分解すべきことを提唱する。そもそも、オブジェクトが大きければそれが内包している仕事も多様なはずであり、それらに対応したビューをもつはずである。それゆえ、図 4 に示すように、ビューに着目し、ビューを

```
defclass model [pM m,-pMV mv] {}
defclass view [pV v,pMV mv,-pM m] {}
defclass controler [pC c,-pV v] {}
```

(a)クラス定義



(b) モジュール MVC

図 3: MVC モデルの構成

含めたオブジェクトの分割を試みる。分割されたオブジェクト間には、入出力ビューを新設してメッセージの送受を行なわせる。必要なら純粋に内部的なオブジェクトを新設してもよい。これにより、分割前にもっていたオブジェクトの仕様を変えないで、いくつかのオブジェクト群に肩代わりさせることができる。ここで問題となるのは、分割前のオブジェクトが多重ビューをもっていて、各入力ビューからのメッセージの並行処理が考えられており、分割により入力ビューがいくつかのオブジェクトに分割されたような場合である。そのようなときには、オブジェクト間での交渉や送受信制御が必要になる場合もあろう [8]。

一方、ボトムアップにオブジェクトを集約してモジュール化するようなときには、3.2 で述べたような複合オブジェクトとして宣言し、外部に公開するビューのみを仕様として公開し、その他は隠蔽することができる。このとき、一般的には複合オブジェクトは多重ビューをもつオブジェクトとして扱うことができるが、それが抱えるオブジェクト群を統括するオブジェクトは特に要求しない。

このように、オブジェクトにプロトコルに基づくビューを考えることにより、オブジェクトの具体化・抽象化を安全に、かつオブジェクトの振舞いに特別な制約を課すことなく実現することができる。すなわち、ビューに基づいて定義された仕様をインタフェースとして、オブジェクトの内部と外部を明確に分離しているのであり、いわば構造的なソフトウェア設計法 [9] に分散オブジェクト指向の観点から与えたことに対応する。しかし、これはあくまで言語モデルとしての枠組であるので、どのようにしてオブジェクトの解析・集約を行なうべきかは、次元の違う方法論である。図 5 に、オブジェクトをトップダウンに設計していく例を示す。

4 実行環境

分散システムを構築していくときには、その包含しているオブジェクト群を管理しうる範囲を設定し、それをドメインと呼ぶことにする。したがって、ドメインの範囲内であれば、ドメイン内に存在するものはすべて、静的にも動的にも、ドメインにグローバルな名前を付与する

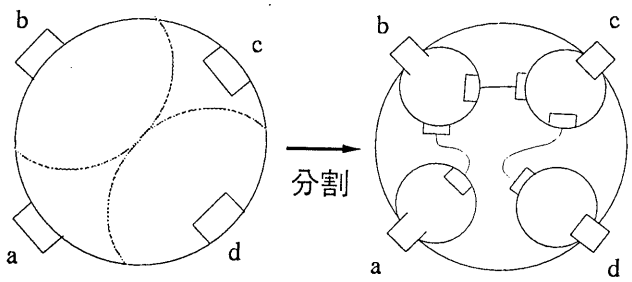


図 4: オブジェクトの分割

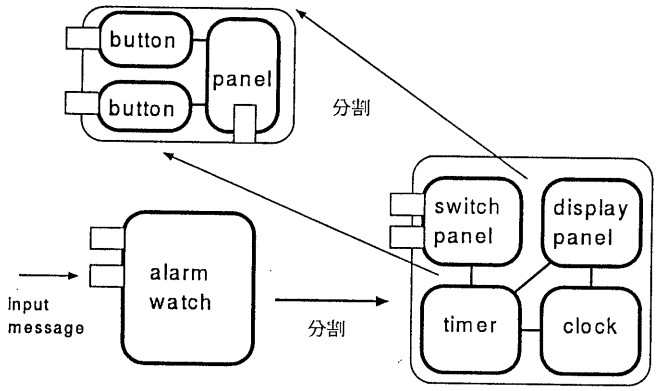


図 5: オブジェクトの設計

ことができるものとする。すなわち、ドメインには管理者モジュールを置き、ドメイン内のすべてのものを一元管理している状況を前提とする。分散アプリケーションはこのようないくつかのドメインにオブジェクトを分散配置させるが、ドメインの管理は Kamui 環境で行なっているので、アプリケーションレベルでは関知する必要はない。ドメインはオブジェクトのクラス定義をあらかじめコンパイルして保持しているものとする。

分散アプリケーションのユーザは、プロトコル定義やクラス定義を Kamui Shell に与えてコンパイルさせておき、Kamui Script を Kamui Shell に与えることにより、対話的に分散オブジェクトの生成、接続、更新などを行っていく。Kamui Shell は分散した資源を管理することのできるインタプリタで、与えられた Kamui Script に基づき、各ドメインと交渉し、指示されたオブジェクトをドメインに配置し、接続情報を管理する。図 6 に、分散プログラムの実行環境の概念図を示す。Kamui Shell はオブジェクトの管理情報を所持しているので、ユーザからの指示で不要になったオブジェクトを消滅させ、そのメモリ領域を回収する。

5 おわりに

本報告では、分散プログラミングの基礎となるオブジェクトの型機構について要約し、ビューに基づくオブジェクトの仕様について提唱した。そして、分散アプリケーションを構築するためには、オブジェクトの振舞いの記述とオブジェ

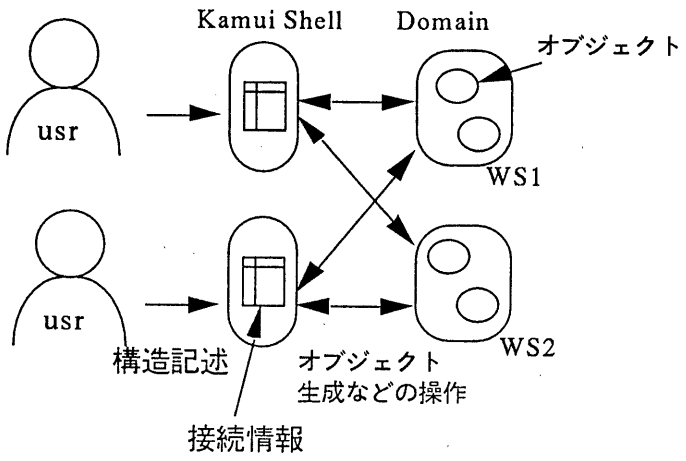


図 6: 分散オブジェクトの実行環境

クト間の構造の記述を分離したほうがよいことを述べ、後者の記述のあり方について検討した。そこからオブジェクト群のモジュール化による複合オブジェクトの考えを提唱し、分散プログラミングの基本的なアプローチについて論述した。

1. で、オブジェクト間の関係には、主従の関係と対等な関係に分類することができることを述べた。本報告は後者のオブジェクト間の関係に着目して、その構造記述やモジュール化について論述し、前者の関係はオブジェクトの振舞いの中で行なわれることを暗黙に想定していた。しかし、一般にオブジェクトはこれら両者の関係が絡みあって構成されるものと考えるべきであり、そのときには入れ子になった両者の関係を想定しなければならない。本報告では、オブジェクトの記述フェーズと実行指示フェーズを区別して扱っているが、両者の関係を考えたとき、それらを融合したシステムとしなければならない。永続オブジェクトと一時的オブジェクトの扱いなどとも関連し、その在り方について今後検討する必要がある。

また、分散プログラミングについては今回その枠組のみについて触れた。しかし、重要なことは、いかにしてそのような枠組の中で分散プログラミングを進めていくべきかの方法論を明らかにすることであり、今後の大きな課題である。

参考文献

[1] A.Black, N.Hutchison, E.Jul,H.Levy, and L.Carter: Distribution and Abstract Types in Emerald, IEEE TransSoft. Eng., Vol.SE-13, No.1, pp.65–76, Jan,1987.

[2] 原田康徳, 宮本衛市: オブジェクト分散プログラミングに適した新しい型付け機構, WOOC'90, 1990.

[3] 原田康徳, 浜田昇, 渡辺慎哉, 宮本衛市: 多言語分散環境のための型機構, コンピュータソフトウェア, Vol.9, No.2, pp.63–74, 1992.

[4] 本田康晃, 渡滋, 大沢英一, 所真理雄: 成長するオブジェクトのモデル, ソフトウェア科学会第 7 回全国大会論文集, 1990.

[5] 原田康徳, 渡辺慎哉, 宮本衛市: 開放系のための協調計算に基づく名前付けモデル, コンピューターソフトウェア, Vol.9, No.3, pp.36–49, 1992.

[6] J.Magee, J.Kramer, and M.Sloman: Constructing Distributed System in Conic, IEEE TransSoft. Eng., Vol.SE-15, No.6, pp.663–675, June,1989.

[7] 市川武彦, 魚井宏高, 野島光典, 萩原兼一, 首藤勝: 部品階層に基づくオブジェクト指向プログラミングのための開発環境, WOOC'92, 1992.

[8] 渡辺慎哉, 宮本衛市: 分散オブジェクトにおける到着順序に関するメッセージ群の構造化, コンピューターソフトウェア, Vol.9, No.4, pp.19–35, 1992.

[9] D.T.Ross: Structure Analysis (SA): A Language for communicating ideas, IEEE Trans. on Software Eng., Vol.SE-3, No.1, pp.16–34, 1977.