記 号 処 理 57－1
（1990．11．22）

メタシステムに基づくオブジェクト間相互作用

繁田良則 原田康徳 渡辺慎哉 宮本衛市

北海道大学 工学部 情報工学科

並列オブジェクト指向モデルにおけるオブジェクト間の複雑な通信 (オブジェクト間相互作用) を、ユーザ定義の通信プリミティブとして扱うための枠組み (session) を提案する。session は通信を抽象化するための手段であり、同時にオブジェクトの外部インタフェースでもある。我々は、session を実現するためにメタ・アーキテクチャを導入した。本論文では、メタシステムにおける session の動作について述べ、最後に session の簡単な例を示す。

Interactions Between Objects Based on Meta-Objects

Yoshinori Shigeta Yasunori Harada Shin-ya Watanabe Eiichi Miyamoto

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

We propose a frame-work which deals with complex communication between objects in a concurrent object-oriented model. We use the word "*session*" to denote this frame work. With the *session* model, users can treat complex communication as system-defined primitive functions. A *session* is a method of abstraction of communication between objects. It is also an interface of an object. We introduce meta-architecture to represent a *session* in the concurrent object-oriented computation model. In this paper we explain the behavior of a *session* using a meta-system and show a simple example.

1 はじめに

ハードウェア技術の進歩によって、並列分散処理が実現可能となってきた。こうした中で、並列オブジェクト指向モデルが注目されている。オブジェクト指向モデルにおけるオブジェクトの高い独立性に着目して、これを並列実行の単位としたものが並列オブジェクト指向モデルである。

このモデルの基本構成要素は、オブジェクト指向モデルと同様、オブジェクトと通信 (メッセージ) である。このうち、オブジェクトに関しては ABCL/R[1] や MUSE[2] などにみられるように、メタ・アーキテクチャあるいはリフレクションの研究が進められている。一方、通信に関しては send 型通信[1]あるいは RPC 型通信[2]といった単純な通信プリミティブが用意されているに留まっている。

並列オブジェクト指向モデルでは、オブジェクトが独立に動作しているので、オブジェクト同士の交渉 (negotiation) のような複雑な通信 (相互作用) が必要となってくる。しかし、これまで提案されてきた並列オブジェクト指向モデルでは、プログラムの流れの中で単純な通信プリミティブを組み合わせ複雑な通信を表現することしかできなかった。このため、プログラムが煩雑になり、その可読性も低下することになる。これを解決するためには、複雑な通信を通信プリミティブとしてシステムが用意すれば良いが、ユーザの望むすべての通信プリミティブを用意することは不可能である。

そこで本研究では、メタ・アーキテクチャの柔軟な計算メカニズムに着目して、オブジェクト間で行われる複雑な通信 (オブジェクト間相互作用) をユーザによって定義可能な通信プリミティブとして扱うための手法について考察を行う。

また本論文では、通信の抽象化とオブジェクトの外部インターフェースの観点からオブジェクト間相互作用の考察を行った後、本研究で考えているメタ・システムの形態についての説明を行う。そして、並列オブジェクト指向モデルにユーザによって定義可能な通信プリミティブを導入し、その動作について説明した後、Time out RPC の例を示す。

[1]以降、非同期単方向通信を send と呼ぶ
[2]Remote procedure call

2 オブジェクト間相互作用

並列分散処理のモデルとして並列オブジェクト指向モデルを考えた場合、処理全体におけるオブジェクト間の通信の比重は非常に高くなる。オブジェクト間を何往復もするような複雑なメッセージのやり取りは、もはや通信と呼ぶよりもオブジェクト同士の交渉 (negotiation) あるいは相互作用 (interaction) と呼ぶ方がふさわしい。

2.1 通信の抽象化

これまで提案されてきた並列オブジェクト指向モデルでは、send 型通信と RPC 型通信を通信プリミティブとしたものが多い。send 型通信はメッセージ送信の後、処理を続行する一方向の通信、RPC 型通信はメッセージ送信の後、返値を受け取るまで処理を中断する一往復の通信である。この他にも、もう少し複雑なものとしては ABCM[4] の future 型通信などがあるが、これも基本的には一往復の通信である。複雑な相互作用をこれらの組み合わせとして表現することは可能であるが、通信のための処理と本来行うべき処理が混然一体となりプログラムの見通しが非常に悪くなる。システムが豊富な通信プリミティブを備えることによってこれに対処できるが、利用者が望むすべての通信プリミティブを用意することは不可能である。そこで、オブジェクト間を何往復もする複雑な相互作用ユーザがを通信プリミティブとして定義できる枠組みが必要となる。この枠組みを本研究では、session と呼んでいる。session を導入することで、通信のための処理とオブジェクト本来の処理を明確に分離でき、プログラムの可読性は格段に向上する。さらに、session は通信処理の再利用にも役立つ。session を組み合わせてより複雑な session を構成し、この session を用いてさらに複雑な session を構成することが比較的容易に行えるのである。

sessionのような通信の抽象化は、問題が大規模になりオブジェクト間の通信が複雑になればなるほどその重要性は増す。

2.2 オブジェクトの外部インタフェース

sessionのもう一つの利点は、通信の抽象化とオブジェクトの外部インタフェースが同一視できることである。。これは、sessionがオブジェクト間の通信手順を定義したものであることから容易に理解できる。オブジェクトの外部インタフェースは、その部品化を考える上で非常に重要となる。外部インタフェースが、あるオブジェクトとあるオブジェクトが結合可能かどうかの基準となるからである。[6]では受信可能メッセージと送出メッセージをそのオブジェクトの外部インタフェースとする方法を提案している。sessionは送受信メッセージとその関係(通信手順)を含んでいるので、さらに強力な外部インタフェースとなる。

3 メタ・システム

本研究では、オブジェクト間相互作用の抽象化の方法であるsessionを実現するためにメタ・アーキテクチャを導入している。メタ・システムの柔軟な計算メカニズムを用いることで、従来はシステム定義であった通信プリミティブをユーザが定義することが可能となるからである。ここでは、本研究で考えているメタ・システムの形態とメタ・オブジェクトの役割について述べる。

3.1 メタ・システムとは

メタ・システムとはメタ・レベルが対象レベルの自己表現(self representation)を持っていて、メタ・レベルにおいて自己表現を操作できるモデルである。このメタ・システムをさらに発展させて、対象レベルからメタ・レベルの自己表現を操作することによって自分自身を動的に変更できるモデルをリフレクションという。両者は、自分自身の動的変更が可能かどうかという違いはあるものの、共にメタ・レベルにおいて自己表現を操作するという柔軟な計算メカニズムを提供している。

図3.1は送られてきた数をx倍するオブジェクトとそのメタ・オブジェクトの例である。メタ・オブジェクトはオブジェクトの自己表現を持っていて、これを操作することでオブジェクトの動作を変更することができる。例えば、xを加えるように動作を変えることができるのである。

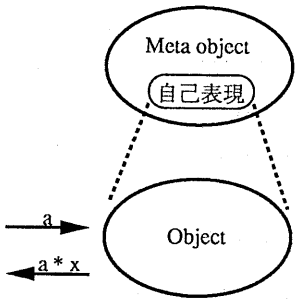

図3.1 メタ・システムの例

3.2 メタ・システムの形態

Ferber[3]は、オブジェクト指向モデルにおけるメタ・システムの形態として、以下の三つを提案している。

1. meta-class model
2. specific meta-object model
3. meta-communication model

meta-class modelは、オブジェクトのクラスをそのオブジェクトのメタ・オブジェクトとするモデルである。オブジェクトとクラスの関係は図3.2のようになる。

specific meta-object modelは、オブジェクトごとにメタ・オブジェクトが存在するモデルである。このモデルのメタ・オブジェクトはそのオブジェクトのクラスではなく、META-OBJECTというクラスあるいはそのサブ・クラスのインス

タンスである。(図 3.3) そしてメタ・オブジェクトもまたオブジェクトであるのでメタ・オブジェクトを持ち、このオブジェクトはメタ・メタ・オブジェクトと呼ばれる。同様にしてメタ・レベルの階層は無限に続く。これは infinite tower[5] と呼ばれている。

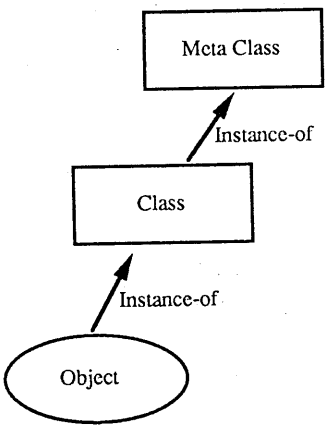

図 3.2 meta-class model

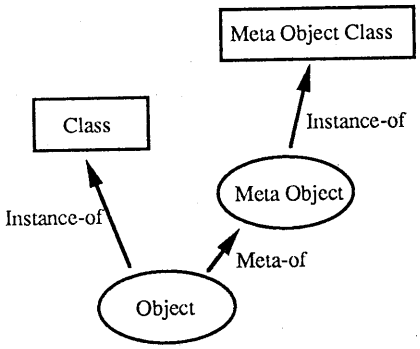

図 3.3 meta-object model

meta-communication model は、メッセージをオブジェクトとして具現化 (reification) するモデルである。メッセージ送信を MESSAGE クラスのインスタンス・オブジェクトとして具現化するので、MESSAGE クラスのサブクラスを定義することでメッセージ送信の機構を変更できる。

本研究では、オブジェクト間の相互作用を取り扱う。そのためにはオブジェクトごとに通信を管理する必要があるので specific meta-object model を採用する。ただし、メッセージの具現化 (reification) の手法は meta-communication model における手法に近い。また、specific meta-object model で問題となるメタ・レベルの無限階層は、メタ・オブジェクトを必要になった時点で生成する lazy creation[5] の手法で回避する。

3.3 メタ・オブジェクトの役割

並列オブジェクト指向モデルでは、オブジェクトは以下のものから構成される。

- 実行器
- メッセージ・キュー
- 内部状態
- スクリプト

これらすべてをメタ・オブジェクトの管理対象とすることも可能ではあるが、オブジェクト間の相互作用を取り扱うという立場からメッセージ・キューだけに注目する。つまり、メタ・オブジェクトは内部状態 (自己表現) としてオブジェクトのメッセージ・キューを持つ。そして、メタ・オブジェクトはこれを操作することでオブジェクトのメッセージの受信順序を制御する。

また、オブジェクトによるメッセージの送信についてもメタ・オブジェクトが管理し、オブジェクトの通信すべてをメタ・オブジェクトで司ることにする。こうすることで、通信 (相互作用) をオブジェクトから分離でき、通信の抽象化が可能となる。この場合、メタ・オブジェクトの定義が新しい通信プリミティブの定義となる。

4 ユーザ定義の通信プリミティブ: session

ここでは、ユーザ定義の通信プリミティブである session の概要とその動作について述べる。

4.1 sessionの概要

オブジェクト間の複雑な相互作用をユーザによって定義可能な通信プリミティブとして扱うための枠組みを session と呼ぶ。言葉を換えれば、session とはオブジェクト間相互作用の抽象化の枠組みと言える。session 導入の利点としては以下のようなものが考えられる。

- プログラム可読性の向上
- 相互作用 (通信) の部品化
- オブジェクトの外部インタフェースとの同一視

このうち相互作用の部品化とは、session を組み合わせて新しい session を順次定義できることを意味する。また、あるオブジェクトの外部との相互作用の抽象化がそのオブジェクトの外部インタフェースになることは容易に理解できる。

また、本研究では session を実現するためにメタ・アーキテクチャを導入した。これは、オブジェクトのメッセージ・キューをメタ・オブジェクトで管理する必要があるからである。

4.2 メタ・オブジェクトによる通信の管理

メタ・オブジェクトとオブジェクトの関係を図 4.1 に示す。

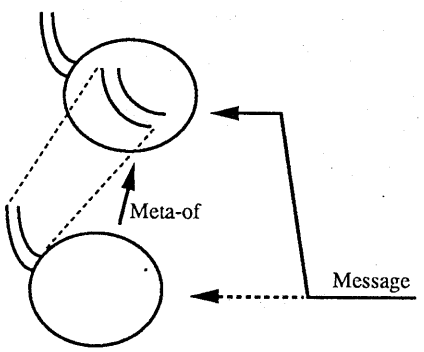

図 4.1 メタ・オブジェクトとオブジェクト

オブジェクトのメッセージ・キューがメタ・オブジェクトの内部状態として実現されており、オブジェクトの通信はすべてメタ・オブジェクトが司る。また、メタ・オブジェクトの通信はメタ・メタ・オブジェクトによって司られる。同様にして、メタ・レベルの無限階層が存在するのだが、実際には send 型通信を特別扱いすることと lazy creation の手法でこれを回避することができる。lazy creation については後で詳しく述べる。

以下に最も単純な meta object のスクリプトを示す。

```
(defMetaObj SimpleMetaObject
  (Slots
    (que nil)
    (oid nil))
  (Scripts
    (begin (obj)
      (setq oid obj))
    (HandleMessage (msg)
      (enque msg))))
```

Slots にはメタ・オブジェクトのインスタンス変数を記述する。que は管理しているオブジェクトのメッセージ・キュー、oid は管理しているオブジェクトの id を表している。Scripts の中の begin はメタ・オブジェクトの生成時に呼び出され、HandleMessage はオブジェクトにメッセージが到着したときに呼び出される。オブジェクトに届いたメッセージは、HandleMessage として具現化される。

4.3 send 型と session 型通信

オブジェクトの通信のすべてをメタ・オブジェクトが司ることにすると、通信はメタ・レベルの階層を無限に昇ることになる。そこで、send 型の通信だけは特別扱いし、メタ・オブジェクトではなく、システムが行うことにする。つまり、システムには二つの通信型

- send 型
- session 型

があり、それぞれ send 型はシステム定義の通信プリミティブとしてシステムが管理し、session 型はユーザー定義の通信プリミティブとしてメタ・オブジェクトが管理するのである。また、RPC 型は session 型で実現できるので、システム定義の通信プリミティブとはしない。

4.4 session の動作

session は通信プリミティブであるので、その動作はアトミックでなければならない。ここでいうアトミック動作とは、session が行われている間はオブジェクトに届くメッセージのうち session に関係するものだけが優先的に処理され、それ以外のメッセージは session の終了まで待たされることを指す。これは、session 状態中における予期せぬオブジェクトの内部状態の変更を防ぐためである。これがメタ・アーキテクチャを導入した理由の一つである。別なオブジェクトに複雑な通信を依頼するだけでは、session のアトミックな動作を保証することはできない。

また、メタ・レベルの無限階層を回避するためにメタ・オブジェクトは session 開始時に動的に生成され (lazy creation) 、session の終了時に消滅する。

図 4.2 に session の動作を示し、以下でこれについて説明する。

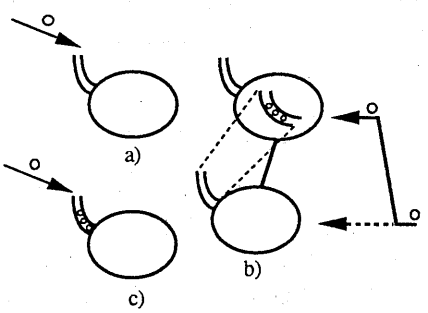

図 4.2 session の動作例

a)session 開始前 メッセージはオブジェクトのキューの最後に入る

b)session 状態 メタ・オブジェクトが生成される。このメタ・オブジェクトは内部状態としてオブジェクトのキューを持つ。オブジェクトに送られてきたメッセージは HandleMessage の引き数となり (具現化) メタ・オブジェクトに送られる。メタ・オブジェクトは具現化 (reification) されたメッセージを操作する。このとき、session に関係ないメッセージは単に保存される

c)session 終了 保存していたメッセージをオブジェクトに送り、メタ・オブジェクトは消滅する

4.5 session のネスティング

以下の二つの場合 session 中にさらに新しい session が開始される。

1. メタ・オブジェクトのスクリプトの中で session が呼び出される場合
2. メタ・オブジェクトによって呼び出されたオブジェクトのメソッド内で新たに session が開始された場合

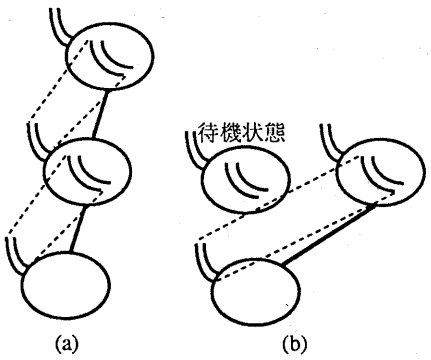

図 4.3 session の nesting

両者とも session のネスティング (nesting) が起こるのだが、その意味及び動作は異なる。

このときの動作はそれぞれ、

1. メタ・メタ・オブジェクトが生成される (図 4.3a)

2. 新しいメタ・オブジェクトが生成され、現在実行中のメタ・オブジェクトはこのメタ・オブジェクトが消滅するまで待機する (図 4.3b)

となる。

図 4.3a はメタ・レベル階層が深くなることを示しており、メタ・オブジェクトのメッセージ・キューがメタ・メタ・オブジェクトによって管理されることを意味している。また、図 4.3b はメタ・オブジェクトが切り替わり、新しい session が終了するまで古い session は待機させられることを示している。両者は新しく生成されたオブジェクトが管理対象とするメッセージ・キューの違いによって区別される。

4.6 例題

この節では、lisp に準じた記法を用いて時間切れ (time out) のある RPC 型通信を定義する。

TimeOutRPC の定義とその使用例を以下に示す。

```
;;TimeOutRPC の定義
(defsession TimeOutRPC
  (Slots
    (wait-que nil) ; 待ちキュー
    (timer-obj nil) ;Timer object id
    (src-obj nil)
            ;session を起動した obj id
    (ret-method nil) ; 成功時のメソッド
    (err-method nil)); 失敗時のメソッド
  (Scripts
     (begin (s-obj d-obj
             ret-m err-m args)
       (setq timer-obj (create-timer)
             src-obj s-obj
             ret-method ret-m
             err-method err-m)
       (send d-obj s-obj args))
       (send timer-obj s-obj 'start)
     (HandleMessage (msg)
       (case (from msg)
         (dest-obj (invoke src-obj
                           ret-method
                           msg)
                   (session-end))
         (timer-obj (invoke src-obj
                            err-method)
                    (session-end))
         (t (enque wait-que msg))))))

;; 使用例
(session 'TimeOutRPC
         'success-method 'fail-method)
```

図 4.3 TimeOutRPC の定義と使用例

この記述の基本的な動作を以下に示す。

1. 目的のオブジェクトにメッセージを送る

2. タイマ・オブジェクトを生成し、start メッセージを送る (タイマ・オブジェクトは一定時間後に time out メッセージを返す)

3. もし、目的のオブジェクトからの返答メッセージが time out メッセージよりも先にに到着すれば、成功時メソッドを起動し session を終了する

4. もし、タイマ・オブジェクトからの time out メッセージが返答メッセージよりも先に到着すれば、失敗時メソッドを起動し session を終了する

5. これ以外のメッセージが到着した場合には wait-que に保存する。保存されたメッセージは session 終了時にオブジェクトに渡される

invoke はオブジェクトのメソッドを起動する手続き、session-end は待ちキューのメッセージをオブジェクトに渡すなどの終了処理を行うものである。

5 おわりに

並列オブジェクト指向モデルにおいて問題となるオブジェクト間の複雑な相互作用を抽象化するための枠組みとしてsessionを提案した。また、sessionを実現するためにメタ・アーキテクチャの導入を図った。sessionによってユーザが新しい通信プリミティブを定義することが可能となり、プログラムの記述性および可読性の向上が図れた。また、通信の抽象化がオブジェクトの外部インターフェースとなることを述べた。今後、sessionを大規模な問題に適用しオブジェクトの外部インターフェースとしてのsessionについてさらなる考察を行っていく必要があると思われる。

謝辞

本研究を行うにあたり、御指導、御助言を頂きました北海道大学工学部 情報工学科 赤間清助教授、三谷和史助手に感謝致します。

参考文献

[1] Takuo Watanabe, Akinori Yonezawa: "Reflection in an Object-Oriented Concurrent Language", Proc. ACM Conference on OOPSLA, 1988, pp.306-315.

[2] 横手靖彦, 寺岡文男, 山田正樹, 手塚宏史, 所真理雄: "MUSE オペレーティングシステムの設計・実装、及びプロトタイプの評価", 日本ソフトウェア科学会第6回大会論文集, 1989, pp.297-300.

[3] Jacques Ferber: "Computational Reflection in Class based Object Oriented Languages", Proc. ACM Conference on OOPSLA, 1989, pp.317-326.

[4] 米澤明憲, 柴山悦哉,Briot,J.P., 本田康明, 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.

[5] B.C.Smith: "Reflection and Semantics in Lisp", Proc. ACM Symposium on Principles of ProgramingLanguage, 1989, pp.22-35.

[6] 原田康徳, 宮本衛市: "送出メッセージの型宣言機能に基づいたオブジェクトの部品化について", 日本ソフトウェア科学会第7回大会論文集, 1990, pp.249-252.