ソフトウェア基礎論　37－13
プログラミング言語　27－13
（1990. 12. 14）

制約のある場に基づく並行オブジェクト指向計算モデル

浜田 昇　　原田康徳　　渡辺慎哉　　宮本衛市

北海道大学工学部

あらまし　従来のオブジェクト指向言語では、オブジェクト間の通信に関する制御（例えば共有資源の排他制御など）を、オブジェクトにオブジェクト本来の性質とともに記述していた。本報告では、オブジェクトの通信に関する制御の記述を、オブジェクトの記述から分離して、Kamui88 モデルにおける場に、場の制約として記述する計算モデルについて報告する。これによって、プログラムを簡潔に記述する事ができる。

A Concurrent Object Oriented Computation Model Based on Fields with Constraints

Noboru Hamada　　Yasunori Harada
Sin-ya Watanabe　　Eiichi Miyamoto

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

Abstract　In usual object oriented programming languages, a control discription of communication between objects(for example, mutual exclusion) are described in objects with their own scripts. In this paper, we propose a new computation model in which a control discription of communication between objects is separated from the object discription and it is described in a "field" of the Kamui88 model. By using this model, we are able to describe programs concisely.

1 はじめに

近年ハードウェアの発達により、マルチプロセッサ構成をとるワークステーションの出現等、プログラムを並列実行するためのハードウェアが実用の域に達しつつある。

しかし、ハードウェアが急激に発達しつつあるのに比べて、並行動作するソフトウェアの研究は遅れている。その理由として、並行プログラミングにおいては、プログラムの実行に必ずしも再現性があるとは限らないことなど、逐次集中型のプログラムに比べて、並行プログラミングが容易ではない事が挙げられる。

我々は、以前に並行プログラム記述性の向上を目的として、オブジェクトの通信に関する制御を集中して記述する並行オブジェクト指向計算モデルを提案した [1]。今回このモデルのあいまいだった部分を取り除き、モデルに拡張を施したので報告する。

2 並行プログラムの難しさ

この章では、並行プログラミングの難しさを明らかにするとともに、それについての解決策について述べる。

並行プログラミングにおいては、逐次型プログラミングにはなかった同期が必要となる。この同期動作は、プログラムの並列実行の単位であるオブジェクト同士が、メッセージを用いて連絡を取り合う事によってなされる。並列プログラミングにおいては、同期動作は重要なものであるが、従来の並行オブジェクト指向モデルにおいては、同期をとるためのメッセージと、オブジェクトに処理を依頼するためのメッセージとが、同じレベルの扱いを受けている。

例えば、共有資源であるプリンタの排他制御を考える。プリンタオブジェクトは文字をプリントすることが本来の性質であるが、排他制御に関するスクリプトも、文字をプリントするという本来の性質と同一のレベルで記述しなければならない。これは、排他制御も、プリントも、全てをメッセージで行うため、同一のレベルで記述せざるを得ないからである。

このように、現在のオブジェクト指向プログラミングでは、通信に関する制御記述が、プログラム中に分散して埋め込まれてしまっている。これが、プログラムをわかりにくいものにしているひとつの原因である。

本研究では、オブジェクトの本来の性質と、通信に関する制御の部分を明確に分離して、通信に関する制御の部分を、オブジェクトグループにおけるオブジェクト同士の制約として、オブジェクトとは別の場所にまとめて記述する事を提案する。

これによって、次のような利点が得られる。

- 簡潔なオブジェクトの記述

  オブジェクト間の制約の記述をオブジェクトから分離することにより、オブジェクトにはオブジェクト本来の性質のみを記述するだけでよいので、記述が簡潔になる。

- オブジェクトの独立性の向上

  オブジェクトの記述を変える事なく制約を変えるだけで、同じオブジェクトであっても異なる振舞いをさせることができる。これによりオブジェクトの部品としての独立性をより高めることができる。

- プログラムの見通しの向上

  複数のオブジェクトにまたがって記述されていた制約を、オブジェクトグループ毎に、1カ所にまとめて記述することによりプログラム全体の見通しが良くなる。

オブジェクトグループ内でのオブジェクト同士の通信を制御するためには、グループ内の全オブジェクトを一目で見渡す事のできる場所に通信に関する制御を集中して書く事が良い。本研究では、オブジェクトを統括するメタな存在として、Kamui88[3] における場を、制御を書く場所として用いる。

3 本報告で提案するモデル

Kamui88 は、複数のオブジェクトをグルーピングするために並列オブジェクト指向計算モデ

ルである。本研究では、このモデルをもとにして、オブジェクトの通信に関する制御を場に記述するためにいくつかの拡張を行った。本章では、本研究で提案するモデルについて詳しく説明する。

3.1 モデルの構成要素

ここでは、本モデルを構成する基本要素である、オブジェクト、場およびプロトコルのそれぞれについて説明する。

- プロトコル

  プロトコルは、イベントの集合として定義される [2]。オブジェクトOが、あるプロトコルPを知っているとは、OにPの全ての元 (イベント) に対応するスクリプトが存在する事を意味する。すなわち、プロトコルは、オブジェクトが理解できるイベントの集合である。この意味で、オブジェクトはプロトコルによって型付けされていると言える。ふつう、オブジェクトは複数のプロトコルを知っている。(多重ビュー)

  一般に、同じイベント名であっても、プロトコル名が異なれば、起動されるメソッドは異なる。本モデルにおいては、オブジェクトはメッセージを送信する際には、プロトコル名とイベント名の両方を指定する。すなわち、メッセージは、プロトコル名、イベント名、および引き数から構成される。

- オブジェクト

  オブジェクトはメッセージキュー、スクリプト、および内部状態からなる。オブジェクトは、キューからメッセージを一つ取り出して、それに対応するスクリプトを実行する動作を繰り返し行う。スクリプトの実行の際に、内部状態の変更、および、他のオブジェクトへのメッセージの送出が行われる。メッセージキューからメッセージを取り出し、スクリプトの実行を終了するまでは、アトミックアクションである。

- 場

  場は複数のオブジェクトをグルーピングするものである。場は、場の中にいるオブジェクトのリスト、メッセージの到着順序に関する制約、およびメッセージキューから構成される。

  オブジェクトは、プログラムの実行時に動的に場に出入りすることができる。また、一つのオブジェクトが同時に複数の場に属する事も可能である。場はプロトコルによって型づけされており、場の型と型の異なるオブジェクトは、場に入る事ができない。(実際のプログラムにおいては、実行時エラーになるが、オブジェクトと場の両方が型付けされているため、適当なツールを用いて型チェックを行う事により、実行前に誤りを検出可能である。)

  場には、オブジェクト同士の通信を制御する制約を記述する事ができる。この制約に基づいて、場は、オブジェクトグループの通信をつかさどる監視者のような働きをする。場の具体的な動作に関しては、3.2節および3.3節で詳しく説明する。

図1に本モデルの概念図を示す。

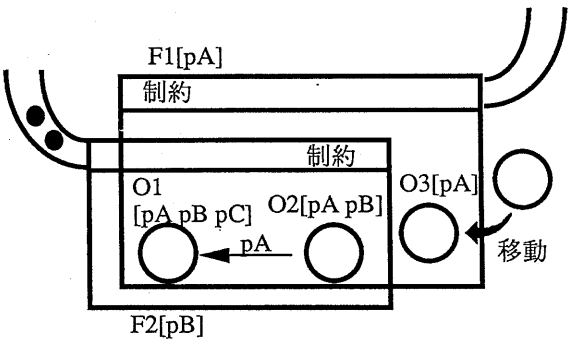

図1　本モデルの概念図

3.2 場による通信の制御

ここでは、場がオブジェクトグループの通信を制御する様子について述べる。

場には、場の内側にいるオブジェクトに到着するメッセージの順序、および回数に関する制約を書く事ができる。オブジェクトがメッセージを送信しようとする場合には、送信オブジェクトは、場によって制約を受ける。どのプロトコルを用いて通信を行うかによって、制約の効果が異なる。ここでは簡単のため、送受信オブジェクトおよび場は、同じプロトコルによって型付けされており、送信オブジェクトはこのプロトコルを用いてメッセージを送出するものとする。オブジェクトおよび場が複数のプロコトルによって型付けされている一般の場合については、次節で説明する。

送信しようとするメッセージが、場の制約に反していない場合には、メッセージおよび送信オブジェクトは、何も影響を受けず、メッセージは受信オブジェクトにそのまま到着する。一方、送信しようとするメッセージが、場の制約に反している場合には、メッセージおよび送信オブジェクトの振る舞いが影響を受けるが、その場合、送信オブジェクトが場の中にいるか、外にいるかによって、影響の受け方が異なる。

- 送信側のオブジェクトが場の外に居る場合には、送信オブジェクトは全く影響を受けずにメッセージの送信を終了する事ができる。しかし、送信されたメッセージは、場のメッセージキューに、制約が満たされるまで待たされる。
- 送信側のオブジェクトが場の中に居る場合には、場の制約が満たされるまで、送信オブジェクトのプログラム実行が、メッセージ送信の直前で中断される。場の制約が満たされた時点で、メッセージ送信から実行を再開する。

前者の送信方法では、送信時に受信オブジェクトと同期をとることができなくて、メッセージの順序指定ができるだけであるが、送信オブジェクトは待たされないので、ただちに次の処理にとりかかることができる。一方、後者の送信方法では、並列性は多少犠牲になるものの、送信時に受信オブジェクトと同期をとることができる。

オブジェクトは、プログラムの実行中に自由に場に出入りできるので、送信オブジェクトが場を移動する事によって、これら2種類の送信方法を使い分ける事ができる。

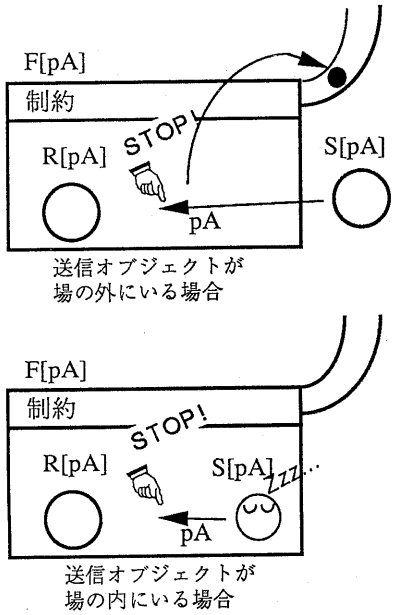

図2　メッセージが場の制約に反していた場合の動作

3.3　制約のスコープについて

ここでは、受信オブジェクトが複数の場に属している場合に問題となる制約のスコープについて説明する。

受信オブジェクトが複数の場に属している場合、送信オブジェクトおよび送信されるメッセージは、送信されるメッセージのプロトコルと、同じプロトコルを持つ場の制約を受ける。同じプロトコルを持つ場が複数ある場合には、それらの制約をすべて満たしたときのみ、全体の制約が満たされたとみなす。図3に具体例を示す。

図3-1において、受信オブジェクトRはF1(型pA)とF2(型pB)の両方の場に属しているが、送信オブジェクトSがプロトコルpBを用いて

メッセージを送信しているため、F2の制約のみを受ける。図3-2では、送信されたメッセージのプロトコルが、両方の場のプロトコルと等しいので、両方の場の制約を受ける。

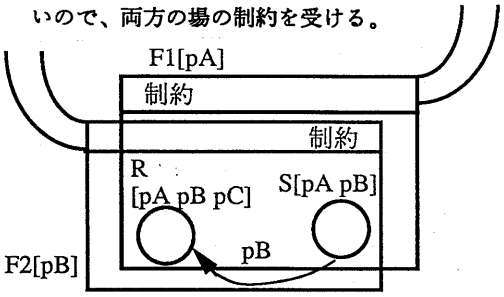

図3-1片方の場の制約しか受けない場合

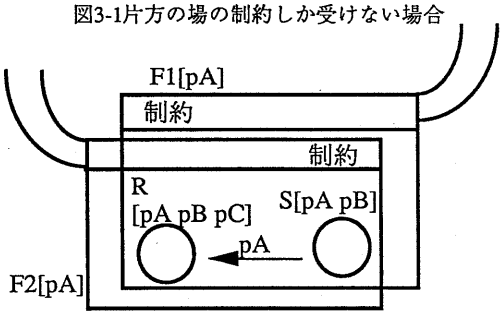

図3-2 両方の場の制約を受ける場合

図3　制約のスコープ

このように、プロトコルを変える事によって、あたかも送信経路を変えたかのような効果を得る事ができる。また、場の型と異なるプロトコルを用いる事によって、場の制約を受けない通信を行う事も可能である。図3において、プロトコルpCを用いて通信することが、これに相当する。

3.4　場のスクリプト

本モデルのシミュレータをLispを用いて制作中であるので、以下ではその表記法を用いて説明する。場は、以下の形式で記述する。

```
(deffiled <fieldname> <protocol>
  (eventorder <orderconstraintlist>)
  (constraint <genaralconstraint>) )
```

以下に、それぞれの要素について説明する。

- <protocol>

  場の型を指定する。場には、ここに示した型と同じ型を持つオブジェクトしか入る事ができない。また、ここで指定したプロトコルに属するイベントについてのみ、イベント順序の制約を書く事ができる。以下の説明では、プロトコルが指定されていることを前提とするため、メッセージの順序という語のかわりに、イベントの順序という語を用いる。

- eventorder節

  イベントが生じるべき順序を指定する。イベントの順序を指定するための基本要素として以下のものが用意されている。

  - seq・・・列挙された順序でイベントが生じるべきことを表す。
  - alt・・・列挙されたイベントまたは節のうち、どれか一つが生じることを表す。
  - para・・・列挙されたイベントまたは節が並列に生じることを示す。
  - ntimes・・・列挙されたイベントまたは節の0回以上の繰り返し

- constraint節

  ここには、一般的なlispの述語が書ける。述語がnilを返せば、制約が満たされなかったとみなし、それ以外の値を返したときには、制約が満たされたとみなす。例えば、aというイベントが生じた回数が、bというイベントの生じた回数よりも少ないという制約は次のように書く。

```
(constraint
  (< (num-ev a) (num-ev b) ) )
```

ここでnum-evはシステム組み込みの関数で、イベントの生じた回数を求めるものである。

4 応用例

4.1 108人の哲学者の食事問題

食卓の5人の哲学者の問題において、人数を増やした問題である。フォークの排他制御を行うために、フォークの両側の哲学者は、フォークとともにオブジェクトグループを構成し、場に入れられる。また、デッドロックにならないようにするために、哲学者は6人ずつ18のグループに分けられており、そのグループ毎に、グループ内の全員が、右手または左手のフォークを一斉に取り上げないような制約を設ける。

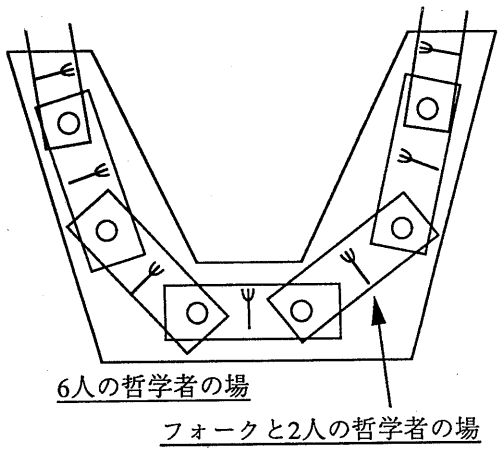

図4 108人の哲学者問題

哲学者はフォークとpForkプロトコルを用いて通信し、左右のフォークを、それぞれgetlおよびgetrというイベントで取り、putlおよびputrというイベントで置くとする。フォークの排他制御を行うためには、哲学者がフォークを取った後、置くまで次のイベントが起きないような制約を書けば良い。また、グループ内の全部の哲学者が、一斉に片側のフォークを取り上げないようにするためには、getrの生起回数とputrの生起回数の差が、哲学者によって取られている右側のフォークの数であるから、その数が6未満であるような制約を書けば良い。哲学者オブジェクトは、これら2種類の場に属し、両方の場の制約を受ける。

図4に問題の概念図を、図5に場の記述を示す。

```
;;; 哲学者2人とフォークの場
(deffield phil-fork-phil pFork
  (eventorder
    (ntimes (alt (seq getr putr)
                 (seq getl putl)))))

;;;6人の哲学者の場
(deffield six-phils pFork
  (constraint
    (and (< (- (num-ev getr)
               (num-ev putr)) 6)
         (< (- (num-ev getl)
               (num-ev putl)) 6))))
```

図5 108人の哲学者問題における場の記述

4.2 到着メッセージの順序指定

家を建てることを考える。発注者Hは、土台、壁、屋根の制作をそれぞれの専門業者S1、S2、S3に依頼する。依頼を受けた専門業者は、土台、壁、屋根を制作したあとで、それを現場に運ぶが、土台、壁、屋根の順序で到着しないと、組み立て業者Kは部品を正しく組み立てる事ができない。

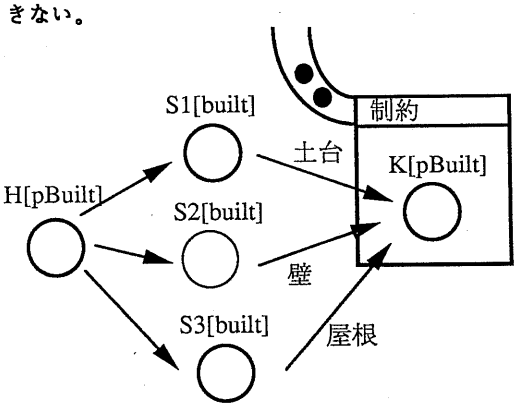

図6 到着メッセージの順序指定

部品の到着順序を場に記述する事によって、順序制御を行う事ができる。この場合、専門業

者S1、S2、およびS3を場の外におくと、場によって動作を中断されることがないので、部品を現場に配達した後、ただちに別の発注者からの仕事を受け付ける事ができる。

図6に問題の概念図を、図7に場の記述を示す。

```
(deffield Genba pBuilt
  (eventorder
    (seq dodai kabe yane)))
```

図7 到着メッセージの順序指定における場の記述

5 議論

5.1 デバッグの容易さ

本モデルでは、通信に関する制御を、オブジェクトグループ毎にまとめて記述するために、プログラムの誤りを発見する事が容易になる。

例えば、長い時間にわたって、プログラムの実行を中断しているオブジェクトや場に待たされているメッセージがある場合には、これらはプログラムの誤りである可能性が高い。本モデルにおいてこのような状況が発生するのは、受信オブジェクトが、同じ型を持つ複数の場に属していて、その制約が互いに矛盾している場合に多くみられる。この場合、場の制約を調べるだけで、容易にプログラムの誤りを発見する事ができる。

このように、本モデルにおいては、オブジェクトグループ毎に制約をまとめて記述するため、プログラム中にイベントの送出文が分散して存在している場合に比べて、デバッグを効率よく行うことができる。

5.2 オブジェクトの再利用性

今までのモデルでは、オブジェクトが特定の相手と通信を行う事など、オブジェクトが特定の状況で使われる事を仮定していた。本モデルでは、計算を

「オブジェクト本来の性質＋通信に関する制約」

としてとらえることにより、オブジェクトを通信の制御の業務から解放する。これにより、同じオブジェクトであっても、場を取り替える事によって異なる動作をさせることができるため、オブジェクトの再利用性が増す。

5.3 本モデルの分散環境へのインプリメンテーション

場の制約は、プログラム全体を制御するものではなく、オブジェクトグループ毎の制御を行うものなので、場の負荷はそれほど重たいものではない。最悪でも、オブジェクトグループ毎に、クライアントサーバモデルくらいの計算量であるが、ここでは、さらに場の負荷を軽減させるアルゴリズムについて考察する。

本モデルを分散環境にインプリメントする場合には、場の実体を作らず、場に関する情報をオブジェクトに持たせる事によって、場への負荷の集中を回避する事ができる。すなわち、各々のオブジェクトに、自分が属している場の制約、および場の現在の制約に関する情報 (各イベントの生起回数と、現在どこまで処理が進んでいるかといった情報) のコピーを持たせて、イベントが発生した時点で、各オブジェクトに、イベントが発生したことをブロードキャストすることにより、負荷の集中を和らげることができる。その場合、ブロードキャストが終了するよりも早く、別のノードで次のイベントが生起した場合に、そのノードにおいて、制約に関する情報の整合性が崩れるが、これは Time Warp [4] の手法を用いることによって解決する事ができる。

6 まとめと今後の課題

並行プログラミングにおける記述の簡潔さを高めるために、オブジェクト本来の性質と制御を分離するモデルを提案した。これによって、プログラムを簡潔に記述する事ができた。

今後の課題として、インスペクタの制作等、開発環境を整えていくことと、より大規模で実際的な問題への本モデルの適用性を調べる事が残されている。

また、分散協調問題への発展として、メタ計算を導入して、メタオブジェクトが場の制約を状況に応じて書き換えるようなモデルについても検討を加えて行くつもりである。

7 謝辞

本報告をまとめるにあたり、議論を通じて貴重な助言を与えて下さった北海道大学工学部情報工学科言語情報工学講座の諸氏に、心から感謝の意を表します。

参考文献

[1] 浜田 昇, 原田康徳, 渡辺慎哉, 宮本衛市 : 制約のある場に基づく計算モデル, 日本ソフトウェア科学会第 7 回大会論文集 (1990),pp89-92

[2] 原田康徳, 渡辺慎哉, 三谷和史, 宮本衛市 : イベントグループを導入した並行分散型オブジェクト指向言語 Kamui-C について, 日本ソフトウェア科学会第 6 回大会論文集 (1989),pp413-416

[3] 渡辺慎哉, 原田康徳, 三谷和史, 宮本衛市: 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

[4] Jefferson, D. and Sowizral, H.: Fast Concurrent Simulation Using the Time Warp Mechanism, *DISTRIBUTED SIMULATION 1985*, pp63-69, The Society for Computer Simulation