

オブジェクトと場に基づいたプログラム言語とその処理系

西尾 郁彦 渡邊 豊英 杉江 昇

名古屋大学 工学部 情報工学科

オブジェクト指向モデルは実世界でのさまざまな実体を表現するのに有用なモデルである。このパラダイムに基づいて、多くの情報システムが設計、構築され、その有効性が報告されている。しかし、このパラダイムでは、自律的な実体の動的な行動を十分にモデル化することができない。オブジェクト指向モデルでは、実体とそれが存在する環境、それらの間の関係を表現することができないためである。本論文では、これらの関係を表現可能なオブジェクト指向言語を提案し、その実装について報告する。

Design and Implementation of a Programming Language Based on Objects and Fields

Fumihiko Nishio Toyohide Watanabe Noboru Sugie

Department of Information Engineering,
School of Engineering, Nagoya University

The object-oriented model is very successful to represent various phenomena in our real world from an entity-category point of view. Many current topics have reported that this paradigm is very appropriate to design and implement information systems. However, it is not sufficient to model the dynamic actions of autonomous entities under this paradigm. This model is in short of representing the relationships between entities and the environments. In this paper, we propose an experimental object-oriented programming language to manage the relationships together, and also report the practical implementation.

1 はじめに

オブジェクト指向パラダイムは、実世界に存在する実体を表現し、これら実体間の相互作用により問題を解決する枠組みを提供する。この枠組みは情報システムを構築する上で非常に有用な手段であり、さまざまなシステムが構築されてきた。それにともない、オブジェクト指向の考えを採り入れたプログラム言語も数多く提案されている。これらオブジェクト指向言語により、実世界の事象を高い記述性を持って表現することが可能となる。

また、実世界のより自然なモデル化という視点から、個々の実体相互の並列性を扱うことにより、記述性の向上を図って並行オブジェクト指向言語、並列計算モデルを構成する研究も行なわれてきている [1]。

これらのオブジェクト指向に基づいたモデル、言語では、並列に活動する個々の実体とそれらの間での相互作用という2つの観点からモデル化が行なわれている。

しかし、このような考え方では実世界における種々の活動に比べて非常に単純な枠組みしか実現できない。分散AIの一分野として近年研究されているような、複数の自律的に活動している実体がお互いに協調しあって問題を解決していく分散協調型の枠組み [2, 3] は、実世界においても非常に一般的な実体の活動形態である。このような、実体が形成する集団、及び集団内での相互作用を表現する手段がオブジェクト指向パラダイムには欠けている。

本稿では、オブジェクトの形成する集団・組織、組織内の相互作用を表現するために「場」の概念を導入し、オブジェクトと場という2つの概念に基づいたプログラム言語について報告する。

2 モデル

ここでは、我々の言語が基づいているモデルについて説明する。2.1で従来のオブジェクト指向モデルについて検討を加える。2.2では場に基づいたモデル化の概要を述べ、2.3以降でモデルを構成する個々の要素について触れる。

2.1 オブジェクト指向モデル

オブジェクト指向パラダイムでは、オブジェクトという概念によって実世界におけるさまざまな実体を表現し、これらの間におけるメッセージ通信によってモデル化を行なう。しかし、オブジェクト指向パラダイムの枠組みが実世界の事象のモデル化に常に有用に利用できるわけではない。現実の世界では、個々の実体がその行動の目的・関連にしたがって集団を形成し、相互に作用して共通の目的を達成している。このような実体の活動をオブジェクト指向パラダイムによってモデル化する場合、以下のような制約がある。

- 1) オブジェクト間の相互作用は1対1の直接通信を基本としている。すなわち、送信側は受信先を明示することによって通信をおこなう。これに対して、実際には1対1の他にも、1対多、多対1、多対多という通信形態も考えられる。特に、不特定多数の相手に対して作用を及ぼすような場合も多く、そのためにブロードキャスト型の通信形態の導入が必要である。さらにこの場合、作用の及ぶ範囲は関連を持ったオブジェクト群に対応し、ある特定の範囲に限定する必要がある。
- 2) メッセージの受信側は、到着順に受けとったメッセージを処理する。すなわち、受信側は送信側に従属的である。実際の世界では、実体は外部からのメッセージに対して自らの判断に基づいて行動する自律的な存在である。自律性を持ったオブジェクトの実現のためには、自律的な判断のもとでメッセージを受理、実行する機構が必要とされる。
- 3) オブジェクト間にはクラス階層によるISA関係が存在する。ISA関係はオブジェクトの性質・特性に基づいた分類であり、静的な関係である。これに対して、実際に実体として活動している状況では、他のオブジェクトとの間にさまざまな関係が生じる。さらに、この関係はオブジェクトの活動の中で動的に変化していく性質を持つ。
- 4) オブジェクト指向パラダイムにはオブジェクトが存在、活動している環境という概念がない。オブジェクトは他のオブジェクトとのメッセージ通信によって直接作用を及ぼしあうだけではなく、それが存在している環境から間接的な影響を受けて活動していると考えられる。そして、環境からの影響により、オブジェクトの行動に変化が起こる。

1)はオブジェクトの通信機能に関して、2)はオブジェクトのメッセージの処理に関して、3)はオブジェクトの集まりの扱いに関して、4)はオブジェクトの存在する場所に関しての問題点である。これらの問題点に対しては、オブジェクト自身に新たな機能を加える一方で、処理実体としてのオブジェクト以外に、オブジェクトの活動環境、オブジェクトの集団を表現するような構成要素を新たに導入する必要がある。

2.2 オブジェクトと場に基づいたモデル化

オブジェクトの活動環境を表現するために「場」の概念を導入する。場はオブジェクトの相互作用を定める環境でもある。オブジェクトはその活動の状況・目的に応じて適切な場に属する。場に対して送られたメッセージは場に属したすべてのオブジェクトに送られる。これにより、各オブジェクトは同じ場に属している他のオブ

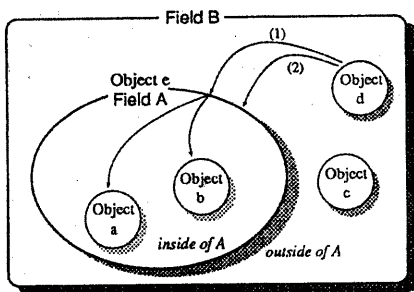


図 1: オブジェクトと場の階層構造

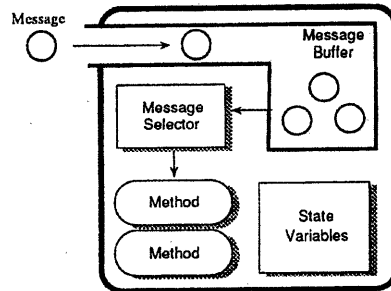


図 2: オブジェクトの内部構造

ジェットの動作を知ることができる。また、オブジェクトと場の間の関係は静的に固定されたものではなく、オブジェクトはある場から別の場へ任意に移動して活動することができる。場を介した不特定多数のオブジェクト間での相互作用は、場に属するオブジェクト間の協調的な動作の実現に有効である。オブジェクトが能動的に活動する要素であれば、場はある関連を持った複数のオブジェクトに共通した動作環境という受動的な要素といえる。

しかし、場をオブジェクト間の通信の媒体として働く受動的な要素として考えるだけでは十分なモデル化をおこなうことはできない。オブジェクトと場の区別は、多くの場合において不明確である。場を複数のオブジェクトに共通の環境と考えたとき、さらにこれをオブジェクトの集団という一つのオブジェクトととらえることも可能である。すなわち、オブジェクトと場の区別はモデル化する際の視点に大きく依存している。例えば、電車と乗客の二つの要素の間の関係を考えてみる。電車の中にいる人にとって電車は一種の場として考えることができる。車内アナウンスのような情報のやりとりは、車内という場に限定して伝播する。一方、車外の人から見れば電車は全体として一つのオブジェクトとしてとらえられる。車内に乗客がいるのか、乗客の間にどのような関係があるのかは車外の人から見た場合には重要ではない。したがって、さまざまな視点から柔軟にモデル化をすることのできる機構が必要となる。

図 1 にこのようなオブジェクトと場の関係を示す。図 1 では場 A が場 B に含まれ階層構造をなしている。また、場 A は場 B の視点から見ればオブジェクト e としてとらえられる。前述の電車の例と対応させれば、場 A / オブジェクト e は電車に、オブジェクト a ~ d は人間に対応する。オブジェクト a, b は場 A という共通の環境下で活動している。一方、オブジェクト e は a, b が集まった、オブジェクト c, d と同様な一つのオブジェクトとして場内で活動することになる。この枠組みは場に存在する複数のオブジェクトから構成されるグループを一つ

のオブジェクトとしてとらえる手段を提供する。オブジェクト a, b の活動はオブジェクト e の活動として抽象化される。

次に、図 1 のようなオブジェクトと場による階層構造の中で、各要素の相互作用がどのようにおこなわれるかについて述べる。オブジェクト間での相互作用はオブジェクト指向パラダイムでのそれと同様に、1 対 1 の直接通信が基本となる。一方、オブジェクトと場の間では、あるオブジェクトから場に属したほかのオブジェクトへの場を介した 1 対多間接通信が行われる。これらに対して、場の外から場に対して送られてくるメッセージは、オブジェクトに対して送られたメッセージと考えられる場合もある。図 1 において、オブジェクト d (あるいは c) から e に対して送られたメッセージは場 A へのブロードキャストメッセージ (図 1(1))、オブジェクト e へ直接送られたメッセージ (図 1(2)) という解釈がなされる。

2.3 オブジェクト

オブジェクトは並行処理実体として自律的に処理を進める。オブジェクトの構造は図 2 に示すとおりで、次の要素から構成される。

- 状態変数… オブジェクトの状態を表す変数。
- 行動定義 (メソッド)… 受けとったメッセージに対するオブジェクトの振舞いを記述したプログラム。
- メッセージ・バッファ… 他のオブジェクトから送られてきたメッセージをいったん置いておくためのバッファ。非同期的に送られてくるメッセージのため、受理されていないメッセージを一時的にしておくためのバッファが必要である。
- メッセージ・セクタ… メッセージ・バッファからオブジェクトの状態にしたがって受理可能なメッセージを取り出し、適当なメソッドを起動する。

オブジェクトは Smalltalk でのオブジェクトのクラス定義と同様の形式でクラスとして定義される。ただし、定義されるすべてのオブジェクトはクラス Entity のサブクラスでなければならない。クラス Entity の詳細については 3.1 で述べる。オブジェクト定義の骨格は次のようになる。

```
Entity subclass: #ObjA
    instanceVariableNames: 'i1 i2'
    classVariableNames: 'c1 c2'
    poolDictionaries: 'p1 p2'!

! ObjA class methods !
    <definition of class methods>!!

! ObjA methods !
    <definition of instance methods>!!

! ObjA guards: #GuardA !
    <definition of guards>!!

! ObjA guards: #GuardB !
    <definition of guards>!!
```

インスタンスはクラスに対してクラス・メソッド create を呼ぶことで生成される。create によって生成されたオブジェクトは一つの独立なプロセスとして存在し、他のオブジェクトと並行に処理をすすめていくことができる。また、オブジェクトは destruct によって活動を停止し、消滅する。

メッセージ・セレクトはオブジェクトの状態等に従って、適当なメッセージを選択し、メソッドを起動する要素であるが、この部分もクラスを定義する際に一緒に定義される。これは、メソッドに対する一種のガード表現といえる。その記述形式は次に示すとおりで各メソッドに対して、そのメソッドを実行するために満たすべき条件を記述する。

```
! <Obj Class> guards: #<Guard Name> !
<method name>
    ~ <condition> !
```

メッセージ・セレクトの記述は一つのクラス定義に対して複数定義することが可能であり、インスタンスの生成時にメッセージ・セレクトの名前を指定し、次のように任意のメッセージ・セレクトを持ったオブジェクトが生成される。

```
a := ObjA create: GuardA.
b := ObjA create: GuardB.
```

生成された a, b は同じ ObjA のインスタンスであるが、それぞれ GuardA, GuardB という異なるガードを持つため、メッセージを受理し、実行する方法もそれぞれに異なる。

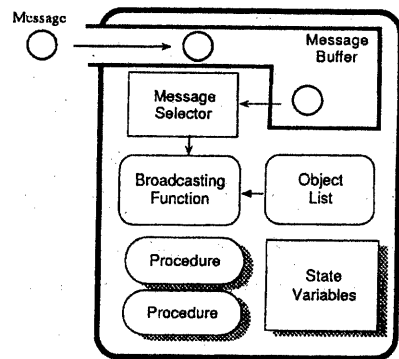


図 3: 場の内部構造

このように、同じクラスから生成されたオブジェクトでも、異なったメッセージの受理方法を持ったオブジェクトを生成することができる。この機構により、オブジェクトをより自律的なものとすることができる。

2.4 場

場はオブジェクトの活動する環境である。場は通常のオブジェクトとは異なる性質を持った、受動的な要素である。場の働きとしては、オブジェクト間のブロードキャスト通信の媒体、オブジェクトが処理実行時に参照することのできる場の状態の保持、オブジェクトの場への出入管理があげられる。場の構造を図 3 に示す。場は次の要素から構成される。

- 状態変数…場の状態を表した変数であり、オブジェクトはこの変数を参照・変更することにより、複数のオブジェクトに共通な場の状態を自身の行動に反映することができる。
- 手続き…オブジェクトが場の状態を参照・変更するために呼び出す手続き。
- メッセージ・バッファ…オブジェクトから送られてきたメッセージを一時的に置いておくためのバッファ。このメッセージは場に存在するすべてのオブジェクトに後述のブロードキャスト機能部によって送られる。
- メッセージ・セクタ…メッセージバッファからメッセージを取り出し、ブロードキャスト機能部へ渡す。
- ブロードキャスト機能部…場内のオブジェクトにメッセージを送る。

- オブジェクト・リスト…場に存在するオブジェクトを管理しているリスト。オブジェクトの場への出入りともない更新される。ブロードキャスト機能部はこのリストを参照してメッセージのブロードキャストを行なう。

場の定義はクラス Field のサブクラスとして定義される。定義の内容は、場の状態を表すインスタンス変数及びそれを参照・変更するためのメソッドを Smalltalk と同様の構文により記述する。オブジェクトの場合と同様に create、destruct により場を動的に生成・削除することができる。生成された場に対してオブジェクトは、enter、exit という操作により任意の場に参加・退出することができる。

2.5 オブジェクトとしての場

2.2で述べたように、オブジェクトと場の違いはそれをとらえる視点に依存している。つまり、場を受動的な要素としてではなく、能動的に活動するオブジェクトとして考えることができる。このようなオブジェクトを定義するために、すでに定義された場に対して、オブジェクトの性質を再定義する (図4)。図4のようなオブジェクトを表すクラスの定義は次のようになる。

```
Entity subclass: #ObjA
  overlay: #FieldA
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''.
```

このような機構により、オブジェクトと場の2つの視点の違いをプログラム上においても分離して、それぞれの視点から記述できる。図4の例では、オブジェクトbの視点から場AをFieldAとして定義し、次にオブジェクトcの視点からオブジェクトaをObjAとして定義することになる。このようにして、オブジェクトの集団を一つ高いレベルでの抽象的なオブジェクトとして扱うことにより、オブジェクト群を機能や処理の目的に応じて分類し、問題を階層的に表現できるという利点が得られる。

2.6 通信形態の種類

オブジェクト間の通信はメッセージによって行なわれる。メッセージ通信の形式は大きく二つに分かれる。一つはオブジェクト間の直接通信、もう一つは場を介した間接通信である。

(i) 直接通信

直接通信には四つの形態がある。以下にこれらの通信形態を示す。

- 1) 非同期送信…送信側はメッセージ送信後、すぐに次の動作に移ることができる。

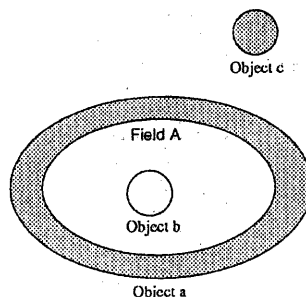


図4: 場に重ねられたオブジェクト

- 2) 同期送信…送信側はメッセージが受信側に受理されるのを確認するまで次の動作を待つ。
- 3) 非同期交信…送信側はメッセージ送信後すぐに次の動作に移る。戻り値が必要になった時点でまだ値が返ってきていない場合にはそこで待つ。
- 4) 同期交信…送信側はメッセージが受信側で受理・実行され、戻り値が返ってくるのを待つ。

オブジェクトはこれら四つの通信形態によるメッセージを、以下のような形式によって使用することができる。

1) 非同期送信

```
receiver <- message &.
```

2) 同期送信

```
receiver <- message .
```

3) 非同期交信

```
ret := receiver <- message &.
```

4) 同期交信

```
ret := receiver <- message .
```

(ii) 間接通信

オブジェクトは場に対してメッセージを送ることにより、その場に存在する不特定多数のオブジェクトへの間接通信を実現する。ブロードキャスト型間接通信は、会議やグループ活動などの協調的動作におけるグループ内での相互作用の実現を容易にする。間接通信によって送られるメッセージは、場内の各オブジェクトに非同期的に送られる。

オブジェクトは次のような形式で場に対してメッセージを送る。

```
field <<- message .
```

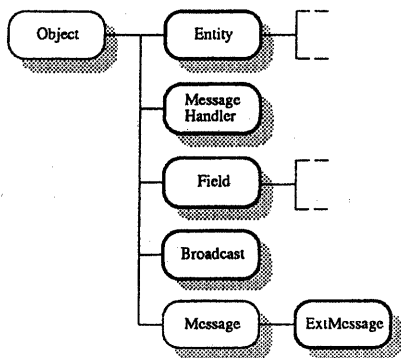


図 5: クラス階層

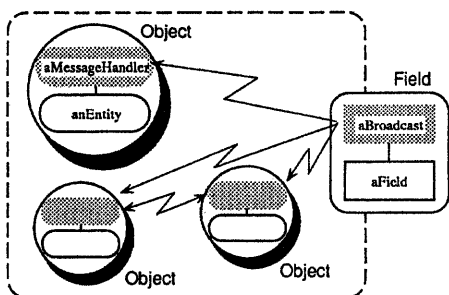


図 6: オブジェクトと場の実現

3 処理系の実装

処理系は Macintosh 上で動作する Smalltalk/V 上に実装した。まず、オブジェクトや場、メッセージ通信の制御を実現するために Smalltalk のクラス階層中に図 5 のようなクラスを作成した。さらに、2.6 で述べたさまざまな通信形態を Smalltalk の構文に変換する簡単なパーサを作成し、クラス定義の読み込み時にソースプログラムをこのパーサに通してからコンパイラに渡すことにした。

3.1 クラス Entity, MessageHandler

クラス Entity はすべてのオブジェクトのルートとなるクラスである。クラス MessageHandler はオブジェクト間の通信制御を行なう。オブジェクトはこの Entity と MessageHandler の各インスタンスから構成される (図 6 参照)。二つのインスタンスは、それぞれのインスタンス変数である myself でお互いを参照しあうようにして結びつけられている。

Entity ではオブジェクトが共通に持つ性質を定義している。具体的にはオブジェクトの生成命令 (create) を受

けると、Entity と MessageHandler のインスタンスを生成し、さらにオブジェクト自身を一つのプロセスとして活動させる。

MessageHandler では 2.6 で述べた通信形態を実現するための制御を行なっている。オブジェクト間でのメッセージは必ずこの部分を経由して送受信される。MessageHandler はメッセージ・バッファを表すインスタンス変数 buffer を持つ。オブジェクトの内部では MessageHandler のインスタンス (aMessageHandler) が独立なプロセスとして動作しており、buffer 中から適当なメッセージを取り出し、anEntity 中のメソッドを起動、実行するという動作を繰り返している。

3.2 クラス Field, Broadcast

すべての場はクラス Field のサブクラスとして定義される。クラス Broadcast は場に対して送られたメッセージをブロードキャストするための機能を持つ。オブジェクトの場合と同様に、場も Field 及び Broadcast の二つのインスタンスから構成される (図 6 参照)。

Field にはすべての場に共通な性質が定義される。具体的には場の生成命令により、Field 及び Broadcast のインスタンスを生成後、それぞれのインスタンス変数を互いに参照し合うように初期化し、場を作りあげる。Field のサブクラスとして場の状態を表す変数やメソッドを持ったクラスを定義することで任意の場を記述することができる。

Broadcast では場の機能の一つであるメッセージのブロードキャスト機能を実現している。このため、メッセージ・バッファ buffer、オブジェクト・リスト memberList というインスタンス変数を持つ。オブジェクトから送られてきたメッセージはいったん buffer に入れた後、memberList に登録されたオブジェクトに対してブロードキャストされる。オブジェクトの場への入出要求 (create, exit) も Broadcast が受けつけ、memberList を更新する。

3.3 メッセージ通信の実現

我々の実装した処理系におけるメッセージ通信は 2.6 で述べたような通信形態を持つため、Smalltalk でのメッセージ通信とは異なったものとなっている。メッセージ通信に関する処理はすべて MessageHandler, Broadcast によって行なわれている。図 7 に示すようにオブジェクトからのメッセージは以下のようなステップで送られる。

- (1) 送信するメッセージはいったん自分の MessageHandler へ送られる。この時、このメッセージは受け付けられない (対応するメソッドがない) ためエラーが起き、エラー処理のメソッド (doesNotUnderstand;) を呼び出す。

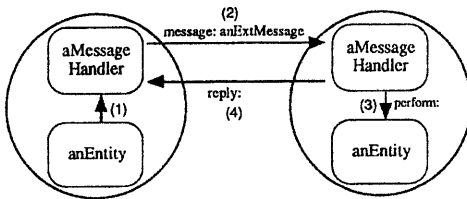


図 7: メッセージ通信の実現

が呼び出される。このメソッドを再定義することにより、以降の通信処理を実現する [4]。

- (2) メソッド `doesNotUnderstand:` 中ではメッセージに、通信形態、送信側の識別子の情報を付加し、送信先の `MessageHandler` へ送る。

受信側はメッセージを自身のメッセージ・バッファへ入れる。通信形態が非同期型であれば、この時点で送信側は次の動作に移ることができる。通信形態が同期型であれば送信先からの返答 (`reply:`) を待つ。

- (3) メッセージバッファ中のメッセージは順次、受理・実行される。実行は `perform:` によって、`anEntity` 中の対応するメソッドを起動することによって行なわれる。

- (4) 受理・実行されるメッセージの通信形態が同期型であればメッセージの送信元に返答メッセージを送る。

メッセージ通信を `MessageHandler` を介して行なうことによって、通信形態、送信側の識別子等の情報をメッセージに付加することができ、また、メッセージに関してさまざまな制御が可能となる。さらに、これらは `Entity` の側から何ら意識する必要はない。

3.4 例

簡単な例として、有限バッファ・オブジェクトの記述例を次に示す。

```

Entity subclass: #Buffer
  instanceVariableNames: 'buffer n
                          boundary'

  classVariableNames: ''
  poolDictionaries: '' !

! Buffer methods !
initialize
" initialize a finite buffer. "
  buffer := OrderedCollection new.
  n := 0.

```

```

  boundary := 10 !
get
" answer a value in buffer. "
  n := n - 1.
  ^ buffer removeFirst !
put: value
" put a value into buffer. "
  buffer addLast: value.
  ^ n := n + 1 !!

! Buffer guards: #BufGuard !
get
" check whether buffer has effective
  data. "
  ^ n > 0 !
put: value
" check whether buffer is full. "
  ^ n < boundary !!

```

クラス `Buffer` は `get`, `put` という二つのメソッドを持つ。これらのメソッドに対して、メソッドの実行条件を示すためにガード `BufGuard` が定義されている。例えば、メソッド `get` に対しては「メソッド `get` を実行するためにはバッファ中に一つ以上の値がなければならない。」という条件を表すガード `get` が定義されている。

`Buffer` のインスタンスを生成するには

```
b := Buffer create: BufGuard.
```

とする。生成された有限バッファ・オブジェクト `b` には次のような形式でメッセージを送る。

```
b <- put: 1234 &.
x := b <- get.
```

一つ目の例は、非同期的にメッセージ `put:` をバッファに送っている。次の例では同期的にメッセージ `get` を送り、返り値を `x` へ代入している。この時、返り値がバッファから返されるまで実行はブロックされる。

4 おわりに

本稿では、オブジェクトと場の概念に基づいたモデル化及び、それに基づいた処理系の実装について述べた。特徴として以下の点があげられる。

- オブジェクト間の直接通信に加えて、場を介したブロードキャスト型の間接通信をオブジェクト間の相互作用の基本的な通信形態として導入した。
- 互いに関連した複数のオブジェクトを場の概念により一つの集団として考えることができる。さらに、この集団を一個の活動実体として扱うための機構を用意した。

- オブジェクトと場の二つの要素を記述可能とし、それによって実世界の現象に対する記述能力を向上できる。

現在、処理系は非常に基本的な部分についてしか実現されておらず、今後は処理系の充実を行なわなければならない。具体的には、オブジェクトや場を Smalltalk の環境中に統合することなどが考えられる。

さらに、場がオブジェクトに及ぼす影響についてさらに考察を加える必要がある。例えば、会議に出席する人を考えてみる。この人は、ある会議では発表者であり、またある会議では司会者であったりと、出席する会議によってその役割、性質は異なったものに変化する。このように、同一のオブジェクトであっても、それが場の間を動的に移動することによって、自身の存在する場に応じてその行動に何らかの変化が起こるような場合がある。このような現象のモデル化を場との関連で検討し、場がオブジェクトの振舞いに対して影響を与えるような機構を考える必要がある。

謝辞

本研究を行なうにあたり貴重な御助言をして頂いた中京大学の福村晃夫教授、名古屋大学の稲垣康善教授、鳥脇純一郎教授に深く感謝するとともに、多くの討論をして頂いた杉江研究室の皆様に感謝いたします。

参考文献

- [1] 石川裕, 所真理雄: オブジェクト指向並行プログラミング言語, 情報処理, Vol.29, No.4, pp.325-333 (1988).
- [2] 丸一威雄, 市川正紀, 所真理雄: 自律的エージェントからなる組織計算モデルと分散協調問題解決への応用, 情報処理学会論文誌, Vol.31, No.12, pp.1768-1779 (1990).
- [3] 吉田紀彦, 檜崎修二: 場と一体化したプロセスの概念に基づく並列協調処理モデル Cellula, 情報処理学会論文誌, Vol.31, No.7, pp.1071-1079 (1990).
- [4] Geoffrey A. Pascoe: Encapsulators: A New Software Paradigm in Smalltalk-80, Proc. of OOP-SLA '86, pp.341-346 (1986).