6P−8

オブジェクト間の関係に基づいた記述モデル Distributed Object Connection

中田秀基, 小池 汎平, 田中 英彦
{nakada,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部*

1 はじめに

並列計算機のプロセッサ台数は年々増加している。これらの高並列マシンを活用するには、対象となる問題のもつ並列性を充分に抽出し記述することのできる言語が必要である。

本稿では、高並列な処理に向いた枠組として、オブジェクト間の関係に基づいた記述モデル DOC (Distributed Object Connection) を示し、その記述法と並列性に関して論じる。

2 DOC モデル

2.1 Actor

オブジェクト指向に基づくプログラミングでは、プログラミングの対象となる世界をオブジェクト単位に切り分けてとらえることで、対象の認識と記述を容易にする。オブジェクトはそれぞれ内部状態を持ち、対象世界全体の状態は、系全体に存在するオブジェクトの内部状態の総和として表現される。計算はオブジェクト間の相互作用であり、メッセージパッシングによって記述される。

Actor モデル [1] は、このようなオブジェクト指向の枠組に基づいた並列計算モデルで、、複数の並行するオブジェクトがメッセージパッシングで通信しあい計算を行なう。Actor では、オブジェクトが実行主体であるので、記述される計算の並列度は、同時に存在するオブジェクトの数である。

しかし、本来オブジェクトは記述のための単位であって、必ずしも実行の最小単位ではない。Actor ではこの二つを強制的に一致させるため、問題を記述する際に本来もっと小さくてすむはずの実行の単位が、オブジェクトの大きさに揃えられてしまう。このため、本来の問題の並列性を充分に生かした記述ができず、並列度が下がることが予想される。

2.2 DOC

以上のような考察に基づき、本稿ではオブジェクトにもたせる情報を最低限にし情報をオブジェクト間の関係としてもたせることで、オブジェクトによる抽象度の高い記述を許しながら並列度の高い実行を可能とするモデル DOC を示す。

従来のオブジェクト指向言語におけるオブジェクトは、実行制御の為の型情報と、内部状態をもっている。一般に、オブジェクトの内部状態は変数名とその変数にバインドされるオブジェクトの対であり、あるオブジェクトを主体として考えると、そのオブジェクトが、他のオブジェクトを何として認識しているかである。つまりオブジェクトの内部状態はそのオブジェクトを中心としたオブジェクト間の関係である (図1)。

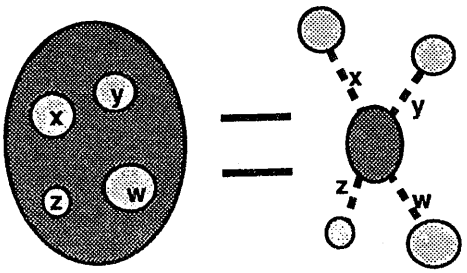

図 1: 内部状態=オブジェクト間の関係

従って、オブジェクト間の関係を陽に記述する方法があれば、オブジェクトの内部状態が記述できる必要はないと考えられる。

DOC では、オブジェクトには内部状態はなく型情報 (クラス) のみを持つ単なる ID とし、オブジェクト間の関係記述する方法としてコネクションを導入する。コネクションは、関係の意味を示すシンボルと、関係づけられる複数のオブジェクトのリストで表現される。DOC によって表現される対象世界の状態は、このコネクションの集合によって定められる。

計算は、コネクションによって表現された系の書き換えによって行われる。コネクションの書き換えはリダクションルールとして定義する。

ルールは条件が整うと自発的に発火し、存在するコネクションを取り除いたり新たなコネクションを加えたりする。この自発的発火がプログラムの実行に相当する。複数のコネクションを前提として発火するルールが記述できるので、コネクションのパターンによる発火なども記述できる。このため、問題のもつ並列度を損なうことなく自然な記述を行なうことができる。

それぞれのルールは発火の前提となるコネクションを共有しない限り、独立に発火するので、記述された並列性を充分に生かして実行することができる。

オブジェクトが内部状態を持たないため、オブジェクトの構造の定義は必要ない。このため、一般の言語で行なわれるような集中したクラス定義は行なわない。クラスの特徴は、そのクラスのオブジェクトに言及するすべてのルールに分散して存在する。これにより、複数のクラスに関する記述が自然に行なえる。

*Discription Model Based on Connection between Objects
Hidemoto NAKADA, Hanpei KOIKE, Hidehiko TANAKA,
the University of Tokyo

3 DOC による記述と実行

DOC は、以下の要素で記述される。

- オブジェクト: 型のみを持つ、単なる ID
  プリミティブなオブジェクト (symbol, number 等) は値を 持つが、値の更新はできない。
- コネクション: オブジェクトとオブジェクトの関係を表現する
  関係を示すシンボルと、オブジェクトのリストで表現する。
  オブジェクトは、変数名と型を : で区切って書く。

```
(symbol Object1:type1 Object2:type2 ...)
```

- ルール: コネクションとコネクションの関係を記述する
  前提となるコネクションと結果となるコネクションを ":-" で 区切って書く。また、発火の条件をガードとして記述できる。

```
con1, con2,...
    | guard :-    conA, conB.
```

ガード には、Flat GHC と同様に、組込み述語だけが記述でき 発火を制限する。
ルールの前提となるコネクションに存在しないオブジェクトが 結果のコネクションで必要な場合は、ルールの発火時に 新たなオブジェクトとして生成する。

DOC においては、対象となる系の状態は、オブジェクトの中にではなく、系全体の空間に分散したコネクションとして表現される。計算は分散したコネクションの付け換えで行なわれる。

ルールの実行は以下のように行なわれる。

1. 系に存在するコネクションが、あるルールをみたすかどうか 調べる。
2. 満たすならば、そのルールを発火する。
   発火は、以下のように行なわれる。
   (a) ガードを評価する。真ならば以下に進む
   (b) 発火の前提となったコネクションを系から取り除く。
   (c) 発火の結果のコネクションを系に加える。

このプロセスは、すべてのルールに関して同時に行なうことが可能である。

4 記述の例

記述の例として素数を求めるプログラムを示す。
まず、以下のような初期状態を与える。

```
(member P0:primeMaker 2)
(member P0:primeMaker 3)
(member P0:primeMaker 4)
......
```

つまり目標までのすべての値 (素数の候補) を member コネクションで primeMaker に接続する (図 2A)。これに対して以下のルールを適用する。

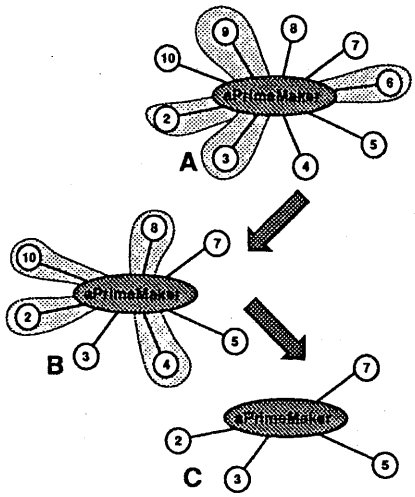

図 2: prime の実行

```
(member P:primeMaker N:number) ,
(member P:primeMaker M:number)
    | (M>N) & (M % N=0)
          :-  (member P N).
```

member コネクションは、素数かも知れない数を示している。このルールは、ある "素数かも知れない数" が別の "素数かも知れない数" で割り切れたら、その数は素数ではないので member コネクションを消去する、という意味である。

最後まで残った member コネクションによって primeMaker に接続している数が素数である (図 2C)。

図に示すように、同時に複数のルールが発火し計算が進行するため、高い並列性がでる。また、記述者がナイーブに認識した対象のモデルを、その並列性を損なわずに記述することができる。

5 おわりに

本稿では、状態をオブジェクト間にのみ持たせるモデル、DO についてのべ、その記述法を示した。

参考文献

[1] G.Agha: Actors: *A Model of Concurrent Computaion in Distriuted Systems*, The MIT Press, 1987.

[2] 中村克彦: "単位消去法 : 単位融合にもとづく論理プログラミングの計算方式", PROCEEDINGS OF THE LOGIC PROGRAMMING CONFERENCE '90, pp.55-63, 1990.

[3] 赤間清: "Object + Relation ≒ Program" 日本ソフトウェア科学会第 7 回大会論文集, pp.41-44, 1990.

[4] 中田, 小池, 田中: "Distributed Object Connection: オブジェクト間の関係に基づいた記述モデル" 情報処理学会研究報告 Vol.91,No.60,pp.27-36,1991.