6U-7

並列オブジェクト指向言語DinnerBell
データフロー実装向きの分散インヘリタンスとその記述

三尾　晴彦　，　河野　真治　，　田中　英彦
（　東京大学　工学部　）

1．　はじめに

DinnerBellは、高並列を目指す並列オブジェクト指向言語であり、各プロセッサごとにクラスのコードが分散して配置される。　本稿は、こうしたコードが分散する環境下への、インヘリタンスの適用とその実装について述べる。

2．　DinnerBellのインヘリタンス

DinnerBellのインヘリタンスは、多重インヘリタンスである。　シンタクスは、以下の形をしている。

$class　クラス名　$inherit　スーパークラス名

スーパークラス名を複数列挙することで多重インヘリタンスが実現される。　メソッドの検索パスは、列挙されたスーパークラス名の左から右へ、depth-first に行われる。　一連のスーパークラスに同一セレクタのメソッドが複数存在したときのオーバーライドも、メソッド検索と同じ優先順位で行われる。

DinnerBellでは、スーパークラスの変数には、各クラスのmodularityを高めるため、サブクラスから直接アクセスすることを許していない。　サブクラスからの参照はすべてMessage送信による。　また、一連のスーパークラスに同一変数名の変数が複数存在するとき、それらはそれぞれ別の変数として扱われる。

DinnerBellへのインヘリタンスの適用上の問題は、高並列を目指すため、各プロセッサにクラスのコードが分散して配置されている点にある。　コード分散が行われる場合、あるクラスのスーパークラスのコードが、必ずしも同一プロセッサ上に存在するとは限らない所に難しさがある。　この問題は、インヘリタンスの実現上、次の2点につきつめる事ができる。　すなわち、

1．　スーパークラスのメソッドサーチの方法
2．　スーパークラスの内部変数の所有方法

である。

2－1．　メソッドサーチ

メソッドが各プロセッサに分散して存在するときにはどのメソッドがどのプロセッサにあるかを各プロセッサが知っている必要がある。

DinnerBellでは、各プロセッサごとに1つのメソッド・ハッシュ・テーブルをもち、あるプロセッサに属するすべてのクラスのメソッドと、そのスーパークラスのメソッドは、すべてそのプロセッサのメソッド・ハッシュ・テーブルに登録される。　こうすることで、メソッドサーチのためのプロセッサ間参照が不用となり、高速にメソッドサーチを行うことができる。

2－2．　スーパークラスの内部変数

スーパークラスの内部変数は、図1－1の様にサブクラスにまとめてしまうのが簡単ではある。　しかし、クラスが複数プロセッサに分散している場合には、図2の様な場合に問題が生じる。　メソッドのあるプロセッサと、その操作すべき変数とか分かれているのは、並列実行を効果的に行う妨げとなる。

DinnerBellでは、図1－2の様に、クラスのコードの分散にあわせて内部変数も分散させて所有することで、並列性を高めている。

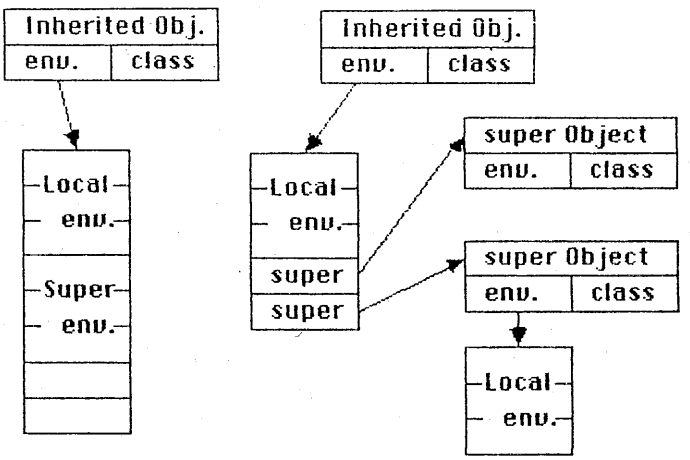

図1－1　　図1－2

図1　スーパークラスの変数の配置

Concurrent Object Oriented Language 'DinnerBell'　— Distributed Inheritance for Data-flow Machine

Haruhiko MIO , Shinji KONO , Hidehiko TANAKA
University of Tokyo

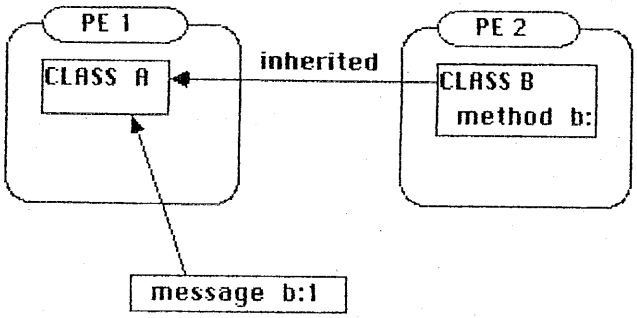

図２　クラスの分散

３．　インスタンスの生成

DinnerBellでは、サブクラスのインスタンス生成時には、スーパークラスのインスタンスを生成しない。（図３－１）　スーパークラスのインスタンスは、そのスーパークラスのメソッドが必要になった時点ではじめて生成され、サブクラスにリンクされる。（図３－２）　これにより、不必要なスーパークラスのインスタンスは生成されないことになる。

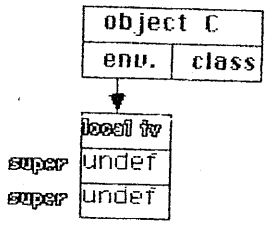

図３－１

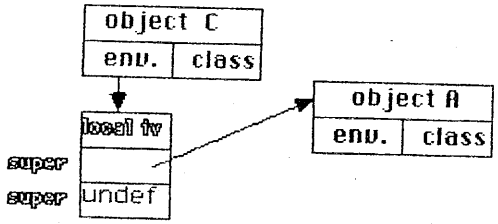

図３－２

図３　インスタンスの生成

４．　例題

例題　１　は単純インヘリタンスにおけるメソッドのオーバーライドの例である。 Swallow は canfly! に対して TRUE を返すが、Penguin は FALSE を返す。

例題　２　は多重インヘリタンスの例である。 class F におけるメソッド検索の順位は、class F→class D→class A→class B→class E→class C である。 従って、class F に do: メッセージが送られたときに起動するのは、class A のメソッドである。　また、class F からみて、 class B と class C の ~VALUE 変数は、別の変数である。

```
$class Animal [
        canMove! □ ^ TRUE ;
        canFly!  □ ^ FALSE
]

$class Bird $inherit Animal [
        canFly!  □ ^ TRUE
]

$class Swallow $inherit Bird

$class Penguin $inherit Bird [
        canFly!  □ ^ FALSE
]
```

図４　例題　１

```
$class A [
        do: A    □ ^ ( A +: 1)
]

$class B [
        write: ~VALUE    ;
        read!    □ ^ ~VALUE
]

$class C [
        put: ~VALUE    ;
        get!     □ ^ ~VALUE
]

$class D $inherit A B

$class E $inherit A C [
        do: A    □ ^ ( A -: 1)
]

$class F $inherit D E
```

図５　例題　２

５．　おわりに

現在、ＵＮＩＸ上でＣで書かれたインタープリタが動作している。

参考文献

[1] Yonezawa, Akinori ,
"On Object Oriented Programming",
Computer Software vol.1(1) pp.29-41 (1984)
[2] Weinreb, Daniel and Moon, David ,
"Flavors: Message Passing in the Lisp Machine",
A.I. Memo 602 (Nov. 1980)