6F-1

並列オブジェクト指向システム　ＯＲＡＧＡ DinnerBellのコンパイラ

立川江介　河野真治　神田陽治　田中英彦
東大　工学部

1.はじめに

ＯＲＡＧＡとはobect ORiented Architecture to Govern Abstractionの略である。ＯＲＡＧＡ計画は個人のプログラミング環境自体を支援する事によりソフトウェアの生産性を向上する事を目的とし，人間の持つ抽象化能力をコンピュータ利用全般で活かすためのコンピュータシステムの構成技術を開発する計画である。

ＯＲＡＧＡは次の４つのサブシステムより構成されている〔図１〕。

①NameMaster

名前ベースの管理をし，記号的プログラムを保証するためのツールである(1)。

②DinnerBell

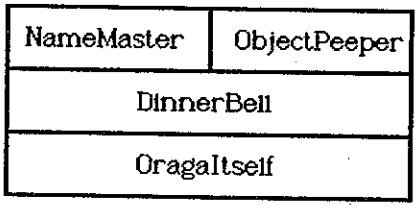
並列オブジェクト指向システム記述言語であり，言語の表現能力としてデータ及び処理フローの抽象化，並列処理の記述の能力を持つ。

③ObjectPeeper

オブジェクトやメッセージを観察するツールである。ＯＲＡＧＡシステムのユーザインタフェースとなる部分であり，DinnerBellのデバッギングを行うツールとなる(2)。

④Orgaltself

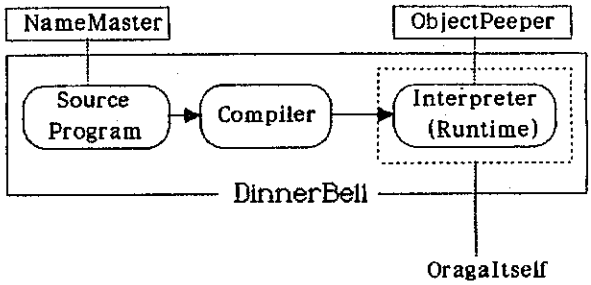
並列オブジェクト指向ネームベースアーキテクチャでDinnerBellを直接に実行する(3)。



〔図１〕ＯＲＡＧＡ構成

2.ＤｉｎｎｅｒＢｅｌｌ

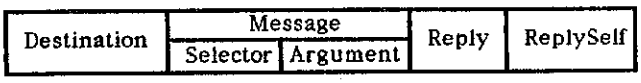
DinnerBellは並列オブジェクト指向言語であり，並列性をオブジジェクト指向に融合するために単一代入則を採用している。また２つのオブジェクト間で複数のメッセージを直接通信する場合はメッセージの到着順序は保証されている。我々はすでにDinnerBellの設計及び実装方法について提案した(4)(5)。本論文ではコンパイル過程を導入したDinnerBellの処理系をＣ言語を用いてVAX730上にインプリメントしたのでそれについて報告する。処理系の構成を図２に示す。



〔図２〕DinnerBell処理系の構成

3.DinnerBellのコンパイラ

コンパイラから出力される中間コードは，並列オブジェクト指向アーキテクチャを持つOragaltself で実行される事を考え，オブジェクトへメッセージを送る事を一つのコードに対応させており，これをコンテキストと呼ぶ。但しここでメッセージは一つのキーに一つの引数がついたものに制限し，複数のキーを持つメソッドはコンパイラが複数個のコンテキストに展開するようにした。これは，インタプリタで実行する場合を考えると，コンテキストはスケジューラのコンテキストプールへ頻繁に取り出し／追加を繰り返すため，スケジューラの管理が簡単となるように固定長としたためである。コンテキストの構成を図３に示す。コンテキスはメッセージの送り先オブジェクト（Destination），キー（Selector）と引数（Argument）により構成されるメッセージの本体及び結果を返す先（Reply），及びReplySelf より成る。ReplySelf については5.で説明する。



〔図３〕コンテキストの構成

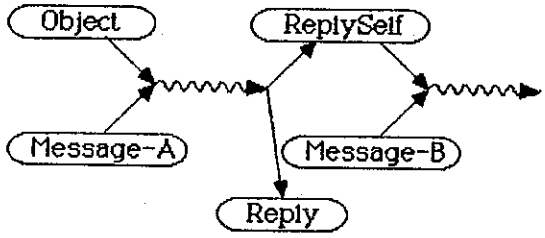
4.インタプリタの動作

インタプリタの実行は基本的にはコンテキストをスケジューラのコンテキストプールから取り出してきては発火し，新しいコンテキストを生成し，これを再びスケジューラのコンテキストプールに付け加える事を繰り返す

事である。今回作成したインタプリタではスケジーラのコンテキストプールとしてキューを用いる事により，擬似並列処理を行っている。またコンテキストにおけるメッセージの送り先オブジェクトの値がまだ束縛されていない場合，そのコンテキストはスケジューラのコンテキストプールから取り出され，送り先オブジェクトの値が確定するまで待ち状態に入る。スケジューラのコンテキストプールにコンテキストが一つも残っていない場合，実行は終了する。DinnerBellでは３種類の変数があり，変数のスコープが大きい順にそれぞれitBlock 変数，method 変数，statement 変数と呼ばれている(4)。このうちitBlock 変数はオブジェクトの生成とともに領域がとられ各メソッド毎に共通に使われ，method変数及びstatement変数はメソッドの発火毎に新しく領域がとられる事になる。

5.環境の共有

先にも述べたように一つのコンテキスト内には一つのキー及び一つの引数だけを持つように制限を加えたため，複数個の引数を持つメソッドを作る場合には，これをコンパイラにより複数個のコンテキストに展開するようにした。これら複数のコンテキストは元々は一つのメソッドを構成していたため，メッセージ送信は同一の環境のもとで行う必要がある。さらにここでは新しいコンテキストの生成を，メッセージが一つ来るたび毎に行うのではなく，メソッドを構成する複数のメッセージ全体に対して新しいコンテキストの生成は一回だけ行うようにすべきである。この機能を実現するためにコンテキストにReplySelf を加えた。複数の引数を持つメソッドの動作を図４に示す。



〔図４〕複数の引数を持つメソッドの処理

オブジェクトはメッセージＡを受け取ると返すべき値をコンテキストのReply に示されたところに束縛し，新しいコンテキストの生成を行う。さらにここでコンテキストのReplySelf に自分と環境を共有する別のオブジェクトを返す。このオブジェクトは，さらに別のメッセージＢを受け取っても新たに変数の領域を作り出す事は無く，自分のすでに持っている変数領域を用いる。さらにこのオブジェクトはメッセージを受け取っても新しいコンテキストを生成する事はない。この機構により複数の引数を持つメソッドを実現できるのである。実際の動作例を図５に挙げる。実行結果を評価する事は今後の課題として残されている。

6.参考文献

（1） 本大会 5G-2
（2） 本大会 6F-2
（3） 本大会 6F-3
（4） 情報処理学会第30回全国大会，3R-6 (1985)
（5） 情報処理学会第30回全国大会，3R-7 (1985)

```
$class AddTest {
    $u add:A and:B u$
        || sender#1 ret:(A +: B)
}


AddTest add:1 and:2.

     Destination        Message         Reply  Replyself
-------------------------------------------------------
At cycle 1
 *:     AddTest       add:       1      Reply      local1
  --     local2      eval!                 -           -
 *:      local1       and:       2        -           -
=>Fired/Reduced = 2/ 2
At cycle 2
 *:      local2      eval!                 -           -
  --          1         +:       2      Reply         -
=>Fired/Reduced = 1/ 1
At cycle 3
 #:           1         +:       2      Reply         -
=>Fired/Reduced = 1/ 1

3
```

〔図５〕動作例