ソフトウェア工学 90－6
（１９９３．２．４）

ネットワークプログラミングのための並行オブジェクト指向言語の設計

小島 一人　　野呂昌満

南山大学情報管理学科

近年の計算機利用形態は分散処理が主流となり，ネットワークで接続された異なるホストの資源を利用して計算を行うことが一般的になってきた．このようなネットワークソフトウェアを作成する場合，資源をオブジェクトとしてとらえ，設計，実現することによりソフトウェアの信頼性，操作性が向上することが期待できる．しかしながら，これらの分散処理を行うソフトウェアはＣ言語などの既存の手続き指向言語やその拡張を用い，ＯＳの分散処理機能を使って実現されているのが現状である．一方，これらのソフトウェアをオブジェクト指向に基づいて実現するための並行オブジェクト指向言語が提案されてきたが，言語が複雑すぎるものや実用性に問題のあるものがほとんどである．
以上の問題点を解決するために，ネットワークプログラミングのためのオブジェクト指向計算モデルを提案し，そのモデルに基づいた並行オブジェクト指向プログラミング言語を設計する．

Design of Concurrent Object-Oriented Programming Language for Network Programming

Kazuhito Kojima　　Masami Noro

Department of Information Systems and Quantitative Sciences,
Nanzan University

A new object-oriented computation model for network programming is proposed. Based on the computation model, a concurrent object-oriented programming language is designed. Although the modern computing environment has moved towards networking, software on the distributed environment is still developed using the traditional non-concurrent programming languages, such as C, with the operating system's facility used for distributed processing. Since the nature of object-oriented computing deeply concerns with concurrency, it is natural to design and implement networking software in a concurrent object-oriented programming language. This idea gave rise to quite a few concurrent programming languages including object-oriented ones. Most of them, however, have problems with their run-time efficiency, and/or their complicated language design. We tried to design a simple and unified object-oriented programming language for networking to defeat these problems. The paper discusses design and implementation issues.

1 はじめに

ネットワークプログラミングを行うさい, 現状ではプログラム間の通信を記述する時にネットワークハードウェアや通信のための手順を意識しなければならない. ネットワーク透過性を言語で実現することで, ネットワークハードウェアを意識せずに分散処理ソフトウェアを作成することができる. 本論文では計算機上の資源をオブジェクトとしてとらえ, オブジェクト間通信としてネットワーク透過な通信を実現する言語を設計する.

言語を設計する方法には対象とする問題領域に適した計算モデルを規定し, そのモデルに基づく言語を設計するトップダウンの方法と, プログラミング言語として必要な機能を列挙することにより設計を行うボトムアップの方法とがある. 前者の方法には以下の利点があるので計算モデルに基づいて設計する方法をとる.

1. 対象とする問題領域に適した言語を設計できる.
2. また同じモデルに基づいた C 言語をもとにした言語, Lisp をもとにした言語など, 複数の言語を設計することができる.

言語を設計するさい, 直交する次元を定義し, その次元について議論することで矛盾のない, 完全な言語を設計できる. 本論文では分散処理のためのオブジェクト指向計算モデル λ を考案し, そのモデルに基づく並行オブジェクト指向プログラミング言語 y を C++[6] の文法をもとに Wegner[9] による直交した次元について議論し設計する.

オブジェクト指向計算モデルとして ABCM/1[12] があるが, λ は以下の点で ABCM/1 と異なる.

- クライアント型オブジェクトを考慮に入れた. ABCM/1 ではすべてのオブジェクトが, 外部からのメッセージによってのみ活性化され, その要求の処理後休眠状態になる受動的なサーバ型オブジェクトである. ネットワークプログラミングを考えると他からの要求を待つのではなく, 他のオブジェクトに対して要求を出すだけの 能動的なクライアント型オブジェクトを考える必要がある.
- 放送などオブジェクト間通信を強化した. ABCM/1 ではネットワークプログラミングで用いられる通信のうち, 放送など 1 対多のメッセージ通信がない.

λ に基づいて y を設計するにあたって, これまでの並行プログラミング言語のオブジェクトの扱いについての問題点を解決した. 従来の並行プログラミング言語では扱われているオブジェクトが,

- 全て並行実行されるオブジェクト
- 並行処理単位と抽象データを表現する単位が別のオブジェクト

のどちらかであった. 全てのオブジェクトが並行に実行されるように言語を設計すると, オブジェクト間の通信の方法は単一の機構で行うことができるが, 実行時のオーバーヘッドが問題となる. また並行処理単位とデータ抽象を表現する単位を別のオブジェクトとして言語を設計すると, 複数のデータ抽象のための言語構成要素を持つことになり, 言語が複雑になる. y では, このような言語の複雑さを解消するために, 並行性を表現するオブジェクトと抽象データを表現するオブジェクトの記述をクラスによる記述に統一した. また並行処理単位としてのオブジェクトと, データ抽象を表現する単位としてのオブジェクトの 2 つのオブジェクトを扱うことによって, 実行時のオーバーヘッドの問題をなくした.

2 オブジェクト指向計算モデル λ

λ では, 並行に稼働する複数のオブジェクトがメッセージによって通信しあうことで計算を進める. 以下では, λ で扱うオブジェクトとオブジェクト間の通信について述べる.

2.1 オブジェクト

λ におけるオブジェクトは, つぎの構成要素をもつ.

- 局所記憶 (local memory)
  オブジェクトのメソッドだけがアクセス可能な記憶で, 局所記憶によってオブジェクトの状態が定義される.
- メソッド (method)
  局所記憶に対して演算を行うための操作で, メッセージによって起動され, インタフェースとして公開される.
- シナリオ (scenario)
  生成から終了までのオブジェクトの振舞いを記述する.

オブジェクトは生成されるとそのシナリオを実行する. オブジェクトはシナリオにしたがってメッセージを受理し, シナリオの実行の終了後オブジェクトは消滅する.

オブジェクトには次の 2 つの状態がある.

- 実行 (running)
  オブジェクトのコードを実行している状態.
- 待機 (waiting)

- 他のオブジェクトからのメッセージの到着を待つ状態.
- 送信したメッセージの結果を待つ状態.

オブジェクトは生成されると実行状態になる. 実行状態にあるオブジェクトは, メッセージの受理または送信したメッセージの結果の受信のときに待機状態になる. 他のオブジェクトからメッセージまたは結果を受理するときには, 受理可能なメッセージまたは結果が到着していても一旦待機状態になり, メッセージを受理する. 到着していない場合は, 受理可能なメッセージまたは結果が到着するまで待つ. メッセージを受理すると再び実行状態になる. また非同期通信の場合, メッセージを送信してもすぐには待機状態にならず, 結果を受信しようとするまでは実行状態にある.

2.2 オブジェクト間通信

λにおけるオブジェクト間通信の方法はメッセージの送受信による通信だけであり, 共有変数による通信は行わない. 共有変数を用いないことで安全性を高めることができる. 複数のオブジェクトの変数の共有は, 共有する変数をサーバオブジェクトとして実現することで代用できる.

ネットワークプログラミングでは並行プログラム同士が通信する時, 以下のことが保証されなければプログラムでそれを保証する必要がある. γではネットワークプログラミングの特性を考慮し, 言語でネットワークの透過性を実現するために, 以下の特徴をオブジェクト間通信に持たせた.

- 存在を知っているオブジェクトに対してのみメッセージを送信できる.
  存在を知らないオブジェクトに対しては, 放送を除きメッセージを送信することができない.
- メッセージは受信側が存在する限り有限時間内に到着する.
  メッセージが受信側に到着することを保証しないと, 送信者自身による受信の確認機構が必要となるため, 言語でメッセージの到着を保証する.
- メッセージは送信した順に到着する.
  メッセージの到着順が保存されないと, 副作用のあるメッセージ通信をする場合など, 送信者による到着順確認が必要となるため, メッセージの送信順を言語で保証する.

オブジェクトは実行状態でメッセージを送信し, 次の時メッセージを受理する.

- 通常通信の場合, 待機状態の時受理する.
- 緊急通信の場合, 他の処理を中断し他のメッセージに優先して受理する.

メッセージを受理する際, オブジェクトは受信したメッセージから受理可能なメッセージを到着順に

- メッセージパターン
- オブジェクトの内部記憶の状態

によって選択して受理し, そのメッセージに対応したメソッドを実行する. 受理されなかったメッセージは破棄されることなく, 受理され処理されるのを待つ. メソッドの実行後, 必要に応じて結果をメッセージの送信者に返送する.

ネットワークプログラミングでの通信を考えた場合, [3] で定義される同期 / 非同期通信, 単方向 / 双方向通信の他, 割り込みを扱うための緊急通信, 同一のメッセージを複数のオブジェクトに送信するための 1 対多通信が必要である. λでは, そのようなネットワークプログラミングを行ううえで必要な同期 / 非同期, 単方向 / 双方向, 緊急 / 通常, 1 対 1/1 対多の 4 つの直交する次元で分類されたオブジェクト間通信を扱う.

以下ではそれぞれの通信方法について述べる.

同期通信 送信者がメッセージを送信後, 結果を受信するまで処理を待ち状態になる通信である. オブジェクト間の同期をとることも同期通信によって行われる.

非同期通信 送信者がメッセージを送信後, 結果の受信を待たず, 続く処理を開始できる通信である. 双方向通信の場合, 送信者はメッセージ送信後そのメッセージに対する結果を受信できるが, 結果が到着していない場合, 結果が返送されるのを待たされる.

双方向通信 メッセージの送信者が受信者からその結果を受けとる通信で, 送信者から受信者, 受信者から送信者の双方向の通信が行われる. 送信したメッセージに対応するのメソッドに戻り値がある場合, 双方向通信が行われる.

単方向通信 メッセージの送信者がメッセージに対する結果を受け取らない, もしくは送信者が受信者からの結果を無視する通信である. オブジェクトのメソッドの戻り値がない時, オブジェクト間の通信は送信者から受信者への単方向通信が行われる. 送信者が受信者からの結果を無視することによっても単方向通信が行われる. 同期通信では, 結果は無視されるが受信者がメッセージの受理し, メソッドを終了するまで送信者の処理は待ち状態になる.

1 対多通信 同一のメッセージを複数のオブジェクトに対して送信する通信である. 同一の処理を複数のオブジェクトに対して要求するソフトウェ

アを作成する場合, 同一のメッセージを複数のオブジェクトに送信することが必要となる. 双方向通信の場合, 結果の到着順に一度に1つの結果が受信される.

緊急通信 メッセージの受信者は実行中の命令を中断し, ほかの通常メッセージに優先してそのメッセージを処理させる割り込みのための通信である. プログラムの終了など, 実行中の処理を中断しオブジェクトの処理を終了させる際の通信など, 通常の処理に割り込み優先的に処理を実行されることが望ましいメッセージを送る時に使用することが考えられる.

```
screnario {
   select {
      when(queuelen < MAXQUEUE){
         accept op1();
         accept op2();
      }
      when(TRUE) {
         accept op();
      }
   }
}
```

図 1: シナリオの記述

3 並行オブジェクト指向言語 y

ここでは前節で述べたオブジェクト指向計算モデル λ に基づいてプログラミングを行うための並行オブジェクト指向プログラミング言語 y を設計する. 言語を設計するさい, 直交する次元を定義し, その直交する次元について議論し設計することによって言語を矛盾なく完全に設計することができる. 本論文では Wegner[9] の定義によるオブジェクト指向プログラミング言語における直交する次の設計次元について議論し, プログラミング言語 y を設計した

- オブジェクト
- クラス
- 継承
- データ抽象
- 型付け
- 並行性
- 永続性

以下ではそれぞれの設計次元について述べる.

3.1 オブジェクト

y では並行処理単位であり λ のオブジェクトである並行オブジェクトと, 並行処理されないデータ抽象を実現する逐次オブジェクトの2つのオブジェクトを扱う. y で扱うオブジェクトをすべて並行オブジェクトとすると, 実行時のオーバーヘッドが問題となる. y ではこの問題を解決するために並行オブジェクトの他, 並行実行されないデータ抽象を実現するための逐次オブジェクトを扱う. 並行処理単位としてのオブジェクトとデータ抽象を表現するための単位とが別のオブジェクトとなることによる言語の複雑さをなくすため, 並行オブジェクトと逐次オブジェクトをクラスによって記述する同一の言語構成要素とした.

オブジェクトは λ によって定義された構成要素を持つが, そのメソッドにはインタフェースとして公開されている外部からのメッセージに対応して実行されるメソッドに加え, 局所メソッド, 初期化メソッド, 後始末メソッドを持つ. 逐次オブジェクトは並行実行されないため並行記述のためのシナリオは持たない.

3.1.1 並行オブジェクト

並行オブジェクトは λ のオブジェクトを言語で実現したものである. 並行オブジェクトは独立した並行処理単位として扱われ, 複数の並行オブジェクトが同時に動作することができる. プログラムは少なくとも1つの並行オブジェクト, メインオブジェクトからなり, プログラムはそのメインオブジェクトを生成することによって起動し, メインオブジェクトの消滅によって終了する.

並行オブジェクトはその生成後の振舞いを記述するシナリオを持つ. シナリオには, 他の並行オブジェクトとの同期, どのメッセージをどの条件の時受理するかが記述される. クラスにこのシナリオの記述がない場合, オブジェクトは言語によって規定されたシナリオを使用する. シナリオは scenario 文によってクラスに記述する (図1).

並行オブジェクトはメッセージバッファを持ち, 並行オブジェクトに対するメッセージは到着順にメッセージバッファに入れられる. 並行オブジェクトは accept 文の実行によってメッセージバッファに入れられたメッセージのうち, 受理可能なメッセージの最初のメッセージを受理する. メッセージを受理すると, 受理したメッセージに対応するメソッドを実行する. 受理されないメッセージは破棄されずメッセージバッファに残り, 受理されるのを待つ. 並行オブジェクトは, オブジェクトの外部で宣言されたその存在を知っている並行オブジェクトに対してメッセージを送信することができる.

安全性と永続性

並行オブジェクトは安全性と永続性の直行する次元で分類される. ネットワークソフトウェアには, パスワード情報など特定のユーザのみにアクセスを限定しなければならない資源を扱うものがある. このような機密データなどアクセス制限する必要のあるものをオブジェクトとして扱う際には, そのオブジェクトに対するメッセージを特定の認証されたオブジェクトからのみ受け付けるような機構と, オブジェクト間通信の不正な傍受を防ぐためにメッセージ通信の暗号化が必要である. またデータベースを扱う場合, データベースをファイルとして扱うのではなくオブジェクトとして扱うことが望まれる. このようなオブジェクトを扱う場合, プログラムが終了後もオブジェクトが存在するように実現することが望ましい. 以上の理由からオブジェクトの永続性と安全性が必要である.

以下では安全性, 永続性を持つ並行オブジェクト, 安全オブジェクト, 永続オブジェクトについて述べる.

安全オブジェクト

安全オブジェクトと呼ばれるオブジェクトは, オブジェクト間通信が安全に行われることを保証したオブジェクトである. 安全オブジェクトはオブジェクトが受理するメッセージの送り手を限定し, そのメッセージを暗号化することによって, オブジェクト間通信の安全性を保証する.

永続オブジェクト

永続性を持つオブジェクトは生成されたブロックが終了後も存在し続ける. そのためオブジェクトの内部状態が失われず, 保持されることが必要である. 永続オブジェクトはプログラムの終了, またはシステムのクラッシュなど異常終了に際してもプログラムの内部状態を保持できるよう補助記憶に格納する. メソッドによって行われた内部状態に対する変更は commit 文の実行によって補助記憶にある内部状態に反映され, プログラムの終了によっても commit 文実行以前の変更は維持される.

3.1.2 逐次オブジェクト

逐次オブジェクトは並行処理単位にはならない, データ抽象のためのオブジェクトであり, 並行オブジェクトとの違いは, 並行実行されないことに関する違いである. 逐次オブジェクトは C++[6], Smalltalk[2] のオブジェクトと同じように, メッセージを受け取ることによって活性化される. また並行オブジェクトの局所記憶, または別の逐次オブジェクトの局所記憶として並行オブジェクトの内部にのみ存在し, 単独で存在することはない. 逐次オブジェクトがメッセージを送信することのできるオブジェクトは, 自身が存在する並行オブジェクト内の存在を知っている逐次オブジェクト, または自身が生成した逐次オブジェクトである. 逐次オブジェクトは並行処理単位ではないので, 並行オブジェクトに対してはメッセージを送信することができない.

3.1.3 オブジェクトの生成

オブジェクトは次の 2 つの方法によって生成される.

- 変数の宣言文の実行などで静的に変数とオブジェクトを結合する方法
- new 文による動的に変数とオブジェクトを結合する方法

全てのオブジェクトを new 文の実行によって生成することができるが, 初期化をすることができない. 変数の宣言文など生成し, 生成時に変数の初期化をできるようにすると記述性を向上できる. オブジェクトの生成は, 変数の宣言文を処理の流れが通過した時と, new 文が実行された時に行われる.

オブジェクトは生成後, それぞれのオブジェクトの内部記憶を初期化するために, C++ のコンストラクタに相当する初期化メソッドを実行する. 初期化メソッドはクラスに記述される. 生成されたオブジェクトは次の時消滅する.

- 生成された最も内側のブロックの終了
- 並行オブジェクトの場合, シナリオの終了

消滅に際してオブジェクトは C++ のディストラクタに相当する後始末メソッドを実行する.

並行オブジェクトは宣言時にオブジェクト名に '@' を続けることによって並行オブジェクトであることを示す. 動的な生成の場合, クラス名に '@' を続けることによって並行オブジェクトの生成を示す. オブジェクトの生成時にクラス名の前に secure, persistent を前置することによって, そのクラスのオブジェクトをそれぞれ安全オブジェクト, 永続オブジェクトとして生成することを示す.

逐次オブジェクトは並行実行されないため, そのオブジェクトのクラスに並行オブジェクトのシナリオの記述があっても, 逐次オブジェクトとして生成された時は無視される.

3.2 クラス

y はクラスによってオブジェクトを記述するオブジェクト指向言語である. プログラミング言語には, クラス

を使用せず，プロトタイプ(prototype, exampler)を使用する言語(ABCL/1[11]など)と，クラスによってオブジェクトの型紙を定義する言語(Smalltalkなど)がある．yでは型付けを行うため，オブジェクトをクラスによって記述し，クラスを型付けのための構文要素とする．

オブジェクト指向言語にはSmalltalkのように，統一的にすべてをオブジェクトとして扱う言語もあるが，yではクラスをオブジェクトとしては扱わない．これはクラスを定義する際に再定義する必要のなるものは，初期化メソッドと，後始末メソッドだけであり，メタクラスによってクラスのすべてを再定義する必要はないと考えたからである．

3.3 継承

yでは継承の方法として単一継承を用いる．オブジェクトがメソッドを共有または借用する方法として，他のクラスの局所記憶，メソッドを継承する方法(inheritance)と，メッセージを他のオブジェクトに委託する方法(delegation)がある．委託を行う場合，オブジェクトが実行時に動的に結合されるため，翻訳時に型検査することは困難である． yは強く型付けされた言語とするため，委託ではなく継承によって局所記憶，メソッドを共有する．

継承にはただ1つのクラスから継承する単一継承と，複数のクラスから継承する多重継承とがある．現実世界を表現しようとするとただ一つのクラスから継承するよりも多重継承を行う方が自然であるが，多重継承は単一継承よりもメソッドの結合が複雑になり，強い型付けをする場合，翻訳時の型検査も複雑になる．また並行オブジェクトを記述したクラスでは，複数のクラスから継承した時のシナリオの継承について現段階では解決していない．よってyでは単一継承を行うが，前述のように多重継承による利点もあるので，これについては今後議論したい．

サブクラスはスーパークラスより局所記憶とメソッド，シナリオを継承する．スーパークラスの局所変数に対するアクセスはスーパークラスのメソッドによってのみ可能である．シナリオを持つクラスのサブクラスは，そのクラスもシナリオを持つ並行オブジェクトのクラスとなり，サブクラスにシナリオの記述がない場合は，スーパークラスのシナリオが継承され使用される．

スーパークラスのシナリオの再利用によるシナリオの差分記述は，仮想関数(virtual function)によってのみ可能である．

3.4 データ抽象

yではクラスが型づけのための言語要素であるので，クラスによってデータ構造とそれに対する演算をするための操作を記述し，データ抽象を行う．クラスの記述は仕様部と実現部にわかれ，クラスの使用者は仕様部に記述されるメソッドによってのみ，オブジェクトの局所記憶にアクセスすることができ，オブジェクト内部のデータ構造とメソッドの実現の詳細は知ることができない．

クラス定義の構文はC++をもとにしたものであるが，C++とは異なり仕様部と実現部にわかれる．クラスの仕様部ではオブジェクトのインタフェースのみを記述し，局所記憶，メソッドの実現など，クラスの実現は実現部に記述する．オブジェクトは局所記憶，局所メソッドは公開せず，アクセスできるメソッドのみを公開するので，クラスの使用者にはインタフェースのみが公開されれば使用でき，局所記憶などオブジェクトの内部については知らせる必要がない． yではC++のように局所記憶に関する情報も公開するのではなく，クラスの記述を仕様部と実現部に分けることにより，クラスの使用者に公開する情報と公開しない情報を分離することができる．

クラスの仕様部ではそのオブジェクトのインタフェースのみを記述する．またクラスの継承も仕様部に記述する．

クラスの実現部ではそのオブジェクトの局所変数の宣言，オブジェクトの持つメソッドの実現，局所メソッドの宣言及び実現を記述する．オブジェクトの初期化メソッドと後始末メソッドはそれぞれC++のコンストラクタ，ディストラクタに対応し，C++同様，初期化メソッドの名前はクラス名と同じであり，後始末メソッドはクラス名の前に'~'をつけた名前となる．オブジェクトは生成されると初期化メソッドを実行し，消滅の際に後始末メソッドを実行する．

3.5 型付け

前述のようにyはクラスを型付けのための構文要素とする．言語にはSmalltalk-80, ABCL/1などの強い型付け機構を持たない言語がある．これらの言語はその使用目的をプロトタイプ用システムなど，変更が容易なシステムのためのプログラミングとしている．これに対して比較的大きな規模のプログラミングを行う際，強い型付け機構を持ち，プログラムの翻訳時に型検査できるプログラミング言語を使用することが望ましい．我々が設計をしたプログラミング言語yは，ネットワークプログラミングでの使用をその目的としているため，ソフトウェアの規模が大きくなることが予想される．よってyに強い型付け機構を持たせた．

3.6 並行性

λのオブジェクトは複数のオブジェクトが同時に実行できる，並行性を持つオブジェクトである．yではこのλのオブジェクトを並行オブジェクトとして実現し，オブジェクトをデータ抽象の単位としてだけではなく，並行処理単位として扱う．

yではλでのオブジェクト間の通信を以下の構文要素によって実現する．

同期通信 同期通信の構文は通常の関数呼び出しと同じである．

```
x = object1.op1(...);
```

非同期通信 メッセージの送信はsend文によって行い，receive文によって受信者からの結果を受信する．受信者からの結果が到着していない場合，結果を受信するまで待つ．

```
send object1.op1(...);
...
x = receive object1.op1(...);
```

単方向通信 単方向通信はメッセージに対応するメソッドの戻り値がvoid型の時，または送信者がvoid型に強制的に型変換し，メソッドの戻り値を無視する時に行われる．

双方向通信 メッセージの送信者が受信者からその結果を受けとる通信である．オブジェクトのメソッドの戻り値がvoid型以外の型の場合，または戻り値がある場合にvoid型に強制的に型変換が行われない場合，双方向通信が行われる．

同報通信 複数のオブジェクトに対して同一のメッセージを送る通信であるが，yでは並行オブジェクトが生成したすべての並行オブジェクトに対して同一のメッセージを送る放送を実現する．allによってそのオブジェクトが生成したすべての並行オブジェクトを指定する．メッセージの結果は同期通信の場合は最初に到着した結果のみを受信できる．非同期通信の場合は一度のreceive文の実行につき到着順に1つの結果を受信する．

緊急通信 メッセージの受信者はほかの通常メッセージに優先してそのメッセージを処理する．

3.7 永続性

yにおける永続性は永続オブジェクトによって実現される．永続オブジェクトではオブジェクトの内部状態はcommit文によって補助記憶に格納され，プログラムの終了後も存在し続ける．

またスキーマ進化(schema evolution)の方法はクラスの変更前に生成されたオブジェクトにはクラスの変更が反映されないGemStone[4]の方法をとるが，これについては今後議論したい．

4 実行時環境

ここでは並行オブジェクト指向プログラミング言語yの実行時環境について述べる．

4.1 オブジェクト

並行オブジェクトをUNIX[1]のプロセスとして実現する．並行オブジェクトの実現方法としては，AMCL/M[12]の実現方法がある．ABCL/Mではノードプロセッサ上のスタックとしてオブジェクトが実現されており，1つのノードプロセッサ上では一時にはただ1つのオブジェクトが実行される．このような実現ではノードプロセッサでスケジューリングをする必要がある．並行オブジェクトをUNIXのプロセスとして実現することにより，オブジェクトのスケジューリングをOSに委ねることができ，容易に実現することができる．

4.2 メモリ管理

分散処理環境でのメモリ管理の方法として次の3つの方法が考えられる．

1. 1箇所で分散処理システム全体のメモリ管理を行う．
2. 各ノード毎にメモリ管理を行う．
3. 各プロセス毎にメモリ管理を行う．

yの実現では2の方法をとる．各プロセス毎にメモリ管理を行うと，メモリ管理ルーチンをプロセス毎に持つことになり，プログラムサイズが大きくなる．1つの並行オブジェクトを1つのプロセスとして実現するため，プログラム全体で使用するプロセスの数が多くなることが予想され，システム全体でのメモリの使用効率が悪くなる．

また各ノード毎にメモリを一括管理する方法では，メモリ管理のためのメモリサーバを各ノードに置き管理することにより，各プロセスの大きさを小さくでき，メモリ管理ルーチンを1箇所にまとめることができる．この方法ではメモリ管理のためのプロセス間通信が必要となる．メモリ管理をメモリサーバとして1つのプロセスで実現するのではなく，SunOS[2]の共有ライブラリとしてメモリ管理ルーチンを共有することで実現する．これによってプロセス間通信を使用せず，メモリ管理ルーチンを一箇所にまとめることができる．

[1] UNIXオペレーティングシステムはUNIX System Laboratoriesが開発，ライセンスしている．

[2] SunOSはSUN Microsystems社の商標である．

メモリ管理方法は分散システム全体のメモリを一箇所で管理する方法も考えられるが, メモリ管理のためにネットワークをとおして通信をする必要がある. 同一ノード内のプロセス間通信と, ネットワークで接続されたノード上のプロセス間通信を考えた場合, 通信費用は後者の通信の方が格段に大きい. 以上の点から各ノード毎にメモリ管理を行う方法をとった.

4.3 オブジェクト間通信

並行オブジェクト間の通信はバーチャルサーキットによるプロセス間通信として実現する. λのオブジェクト間通信ではメッセージの到着と到着順を保証する. これを実現するために到着順, 信頼性を保証したバーチャルサーキットによる通信を使用する. また放送もデータグラムによる通信での実現ではなく, 到着順, 信頼性を保証するためバーチャルサーキットによる通信を使用して実現する.

ネットワークソフトウェアでは, 到着順の保証, 信頼性を犠牲にしてもコストの小さい通信を行う必要のあるものがある. これらを実現するために, データグラム通信による実現を考える必要がある. これは今後の課題としたい.

5 おわりに

以上のように, ネットワークプログラミングのための並行オブジェクト指向プログラミング言語 y を設計してきたが, 言語設計上議論すべき点として次の問題点がある.

- 安全オブジェクトでの送信者の認証
- 1 対多通信
- 例外処理
- オブジェクトの遠隔生成
- オブジェクトの移送 (migration)
- スキーマ進化 (schema evolution)

まず安全オブジェクトにおける送信者の認証は, 安全オブジェクトの特徴としてメッセージ通信の安全性とともにあげられるが, 現段階ではメッセージ送信者を限定する方法について解決していない.

また 1 対多通信には, 受信者を名前で指定しない全てのオブジェクトに対してメッセージを送信する放送 (broadcasting) と, 特定の複数のオブジェクトに対してメッセージを送信する同報通信 (multicasting) がある. 現在, y には放送のみが実現されているが, 全てのオブジェクトに対して送信するのではなく, 特定のオブジェクトのグループに対して送信する同報通信も必要である. したがって放送の他に同報通信も実現する必要がある.

例外処理については現段階では例外を処理するためのハンドラについて, その検索方法と指定の方法について決定していない. また緊急通信との関係, 処理中の割り込みの禁止についても議論する必要がある.

y はその使用目的として分散処理のためのソフトウェアを作成することとしているため, オブジェクトの移送について実現する必要があるが, これについても現段階では解決しておらず, 今後議論したい.

参考文献

[1] Gehani, N., Roome, W.D. *The Concurrent C Programming Language*, Sillicon Press, 1989

[2] Goldberg, A., Robson, D. *Smalltalk-80 The Language*, Addison-Wesley, 1989

[3] 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 29, 4 (Apr, 1988), 325–333

[4] Maier, D. et al. Development of an Object-Oriented DBMS, *Proceeding of the ACM Conference on OOPSLA*, (Sep, 1986), 472–482

[5] Noro, M., Harada K. Design and Implementation of the Object-Oriented Concurrent Programming Language A-MODEL/PL, *WGSE, 76-3* (Dec, 1990)

[6] Stroustrup, B. *The C++ Programming Language Second Edition*, Addison-Wesley, 1991

[7] 高田 敏弘, 米澤 明憲 : 並列オブジェクト指向言語の分散環境における実現, コンピュータソフトウェア 6, 1 (Jan, 1989), 17–29

[8] Bal, H.E., Steiner, J.G., Tanenbaum, A.S. Programming Languages for Distributed Computing Systems, *ACM Computing Surveys 21*, 3 (Sep, 1989), 261–322

[9] Wegner, P. Concepts and Paradigms of Object-Oriented Programming, *OOPS Messenger 1*, 1 (Aug, 1990), 7–87

[10] 米澤 明憲, 柴山 悦哉, J.-P.Briot, 本田 康晃, 高田 敏弘: オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1, コンピュータソフトウェア 3, 3 (Jul, 1986), 9–23

[11] Yonezawa, A., Tokoro, M. *Object-Oriented Concurrent Programming*, MIT Press, 1987

[12] Yonezawa, A. *ABCL: An Object-Oriented Concurrent System*, MIT Press, 1990