並列処理プログラミング再考
―occamとJavaを比較して―
Reconsideration of Parallel Programming: occam vs. Java

喜家村　奨
Susumu Kiyamura

要　約

マイクロプロセッサの並列化が進み，インターネット上のP2P型アプリケーションが話題になる今日，メッセージ通信型の並列処理システムが注目を浴びつつある．本論文ではこのような背景から，並列処理プログラムのために必要なプログラミング言語の機能について並列処理言語occamとオブジェクト指向型言語のJavaを比較しながら考察する．また，並列処理特有の難しい問題の１つであるバッファリング問題についても考察する．

第１章　はじめに

最近のマイクロプロセッサユニット（MPU）の動向をみてみると，パーソナルコンピュータのMPUはマルチコア化が進み，ゲーム機ではセルコンピュータなどの疎結合型のMPUが搭載されようとしている．また，ソフトウエアに関してはプロセスモデルとしてサーバクライアントモデルにかわりP2P型のアプリケーションが注目されるようになってきている．このような状況から，ユーザレベルで柔軟な並列アプリケーションを記述できることが必要になってきている．

Javaは約10年前に開発された言語で，現在多くの環境で利用されている非常に有名なプログラミング言語である．Javaをはじめとするオブジェクト指向型言語で記述されたプログラムは動的に生成された複数のオブジェクトがメッセージを通信して処理を進めていくイメージで，あたかも並列に処理が進んでいくようであるが，少なくとも現在のJavaの実装では，マルチスレッド化しないJavaプログラムは処理の流れは１筋であり逐次的に処理される．つまり現在のJavaで並列プログラムを記述するためにはマルチスレッド化しなければならいないというのが現状である．

一方，occamはOxford大学のC.A.Rホーア氏によって考えられたCSP理論を基に，1980年代に英国Inmos社によって設計された言語である．occamは設計当初から並列処理をおこなうために作成された言語であり，Inmos社のトランスピュータという並列ＣＰＵ上で並列処理プログラムが

実行できた．さらにシリアルリンクと呼ばれる通信リンクで接続された複数トランスピュータ上で１台のトランスピュータで実行されていた並列処理プログラムが容易にマルチＣＰＵのプログラムとして実行することが可能であった．
本論文ではこの並列処理言語であるoccamのチャネルを用いたプロセス間通信とJavaのメソッド呼び出しによるオブジェクト間通信を比較することによって，並列プログラミングに必要なプログラミング言語の性質について考察する．また，並列プログラミング特有の難しさの１つであるバッファリングについても考察する．

第２章　準備

この章ではoccamという言語の特徴と本論文で提示するプログラムリストを理解する上で最低限必要なoccamの文法について紹介する．本文で利用するoccamのバージョンは文献[1]のoccam2であり，Javaは文献[4]のJava2の文法に準じている．また本論文に掲載しているプログラムリストは変数の定義など記述の一部を省略している．

第１節　occamの起源

occamはもともと並列処理を念頭に1980年代に開発された並列処理言語である．occamで記述できる並列処理モデルはCSPモデルと呼ばれ，プロセス間がチャネルと呼ばれる通信路によって接続されてコミュニケーションし全体の処理を進めていく．このCSPとはプロセス代数の１つで1980年代に英国，オックスフォード大学のC.A.R.Hoare氏によって，接続された複数のプロセスが同時並列に処理を進めるシステムを定義，検証するための考案された理論である．図2.1はCSPモデルの概念図である．この例のように複数のプロセスがチャネルを通じてコミュニケーションし（図のPとQ）さらにそのプロセスは階層的に並列構成できる（P1，P2およびP3）．

第２節　occamの文法

以下の節では，occamの基本的な文法について説明する．

第１項　チャネル

チャネルは単方向で１対の送信プロセスと受信プロセスの間に存在する通信リンクである．occam2からチャネルには型を指定することができ，例えばINT型のデータを通信するチャネルchの定義は次のように記述する．

```
CHAN OF INT ch：
```

第2項　プロセス

occamでは処理の実体をプロセスといいoccamのもっとも基本的なプロセスをプリミティブプロセスという．プリミティブプロセスには代入，入力および出力の3つのプリミティブプロセスがある．以下，プリミティブプロセスについて説明する．まず，代入プロセスは以下のように記述する．

```
a := 3
```

この例では変数aに値3を代入する．また入力プロセスは以下のように書く．

```
ch ? a
```

この例でchはoccamのチャネルである．aはチャネルから入力した値を受け取る変数である．出力プロセスは以下のように書く．

```
ch ! 3
```

この例ではチャネルchから値3を出力する．

occamにはこの3つのプリミティブプロセス以外の基本的なプロセスにはSKIPとSTOPがある．SKIPは即座に正常終了するプロセスで，STOPは終了しないプロセスを表す．occamのプロセスはこれらのプロセスを以下に示すコンストラクタで構成することによって記述される．以下コンストラクタについて説明する．

第3項　コンストラクタ

OccamにはSEQ，PAR，ALT，IFおよびWHILEの5つコンストラクタが用意されている．コンストラクタの有効な範囲をコンストラクションといい，各予約語（SEQなど）から2文字分字下げしてプロセスを記述する．ここでは代表的なSEQ，PARおよびALTコンストラクトについて説明する．

SEQコンストラクトは予約語SEQに続くプロセスを逐次的に実行する．

```
SEQ
  key ? char
  screen ! char
```

この例ではチャネルkeyから入力し，次にその入力した値をscreenチャネルに出力する．

PARコンストラクトは以下に続くプロセスを並列に実行する．

```
PAR
  P（ch）
  Q（ch）
```

この例ではプロセスPとプロセスQが並列に実行される．

ALTコンストラクトは複数のチャネルからの入力を待ち，最初にレディになったチャネルから入力し，その入力に対応するプロセスを実行する．

```
ALT
  input1 ? packet
    output ! packet
  input2 ? packet
    output ! packet
```

この例では2つのチャネル（input1とinput2）のうちどちらか先にレディ状態になったチャネルから入力し，そしてその値をチャネルoutputから出力する（図2.2）．

第4項　プロセス間通信

occamのプロセス間通信はチャネルという通信路を用いたバッファを持たない完全同期通信である．例えばリスト2.1のようなプロセスAとプロセスBから構成される並列プロセスを考える．チャネルch上の通信はプロセスAの処理1が終了し，通信レディ状態になり，かつ，プロセスBの処理3が終了しプロセスBもレディ状態になったときに初めて成立する．そして，通信が終了した後，それぞれのプロセスは継続する処理（プロセスAは処理2，プロセスBは処理4）を実行する．このようにoccamのチャネル通信は同期通信であるため，通信遅延による矛盾が生じない

（送信プロセスが完了した時には相手の受信プロセスも完了している）.

```
1 PAR
2   SEQ                    ┐
3     …処理 1              │
4     ch ! 10              │ プロセスA
5     …処理 2              ┘
6   SEQ                    ┐
7     …処理 3              │
8     ch ? a               │ プロセスB
9     …処理 4              ┘
```

リスト2.1

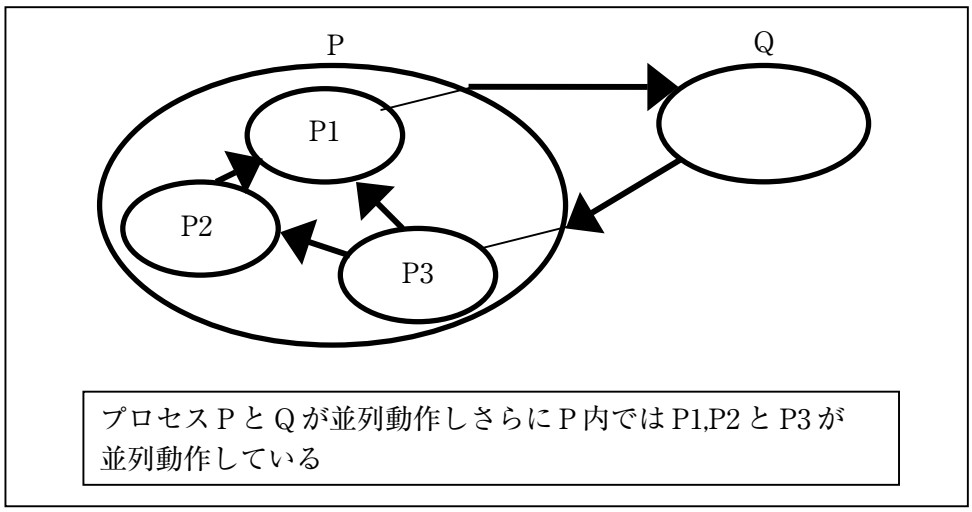

図2.1. CSPモデルの概念図

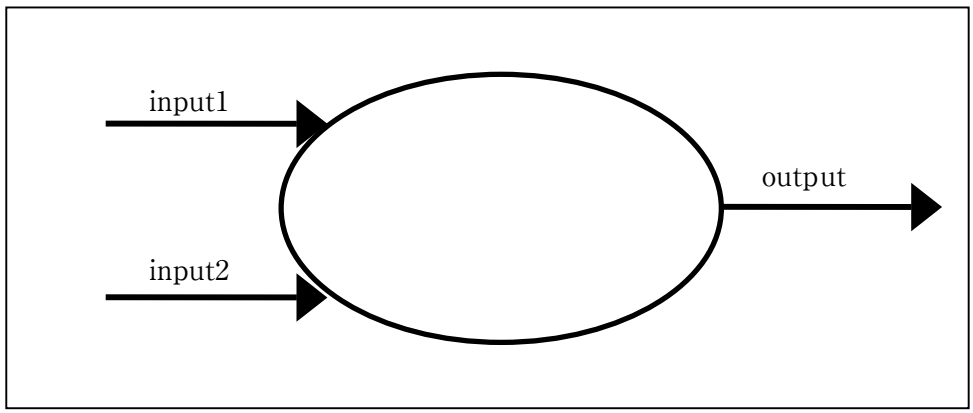

図2.2. ALTコンストラクションの例

第3章　occamとJavaの通信モデルの比較

並列処理プログラムの記述についてoccamとJavaを比較する場合，マルチスレッド化しないJavaプログラムは並列に処理されないため，比較するならマルチスレッド化したJavaプログラムとoccamを比較すべきだという考えもあろう．しかしオブジェクト指向で設計する場合，設計者は複数のオブジェクトが相互作用する並列処理モデルをイメージして設計する．よって，実装が逐次的に実行されようと，Javaはプログラム言語としてその並列処理モデルのための仕様を容易に実装できる機能がなければならない．よって，この章ではoccamによるCSPモデルとJavaによる（メソッド呼び出しによる）オブジェクト通信モデルについて比較し，並列処理プログラムの記述に関してプログラミング言語に必要な機能について考察する．

第1節　通信プロセス（オブジェクト）の記述

図3.1のようなプロセス（オブジェクト）Pの内部変数aの値をプロセスQの内部変数ｂにコピーするプログラムの記述についてoccamとJavaの記述の相違点について考える．

第1項　occamによる記述

occamで図3.1のプロセスを記述すると例えばリスト3.1のようになる．このリストを説明すると，まず1行目から4行目が出力プロセスPの定義である．Pはチャネルoutにaの値を出力する．5から8行目が入力プロセスQの定義でQはチャネルinから値を入力する．次に9行目で通信用チャンネルを宣言し10行目から12行目でプロセスPとQを並行動作させてチャネルch上の通信を成立しaの値をbにコピーしている．

```
1 PROC  P（CHAN  INT  out）
2    INT a:
3    out ! a
4 :
5 PROC Q(CHAN INT in)
6    INT b:
7    in ? b
8 :
9 CHAN OF INT ch:
10 PAR
11    P(ch)
12    Q(ch)
```

リスト3.1

第 2 項　Javaによる通信

一方，Javaのよる図3.1の実装には 2 つのパターンが考えられる．１つ目はリスト3.2に示すようにオブジェクト間の通信をQのメソッドとして実装する場合でPがQのメソッドsetvalueを呼び出し内部変数aの値をbにセットする．

```
1 class P { // オブジェクトPの定義
2    …
3    void start(Q q) {
4         …
5         q.setvalue(a) ;
6    }
7 }
8 Class Q {   //   オブジェクトQの定義
9    …
10    void    setvalue(int a) {
11         …
12         b    =    a ;
13    }
14 }
15 class sample {   //   メインプログラム
16    public static void main(String args[ ]) {
17         P p1= new P() ;
18         Q q1= new Q();
19         p1.start(q1) ;
20 }
```

リスト3.2

もう１つの実装は，Pのメソッドとして実装する場合でQがPのメソッドgetvalueを呼び出しaの値を取得してｂにセットする場合である．

```
1 class P { // オブジェクトPの定義
2    …
3    int getvalue() {
4         …
5         return a ;
```

```
6    }
7 }
8 Class Q {   //   オブジェクトQの定義
9    …
10    void    start(P p) {
11        …
12        b    =    p.getvalue();
13    }
14 }
15 class sample {   //   メインプログラム
16    public static void main(String args[ ]) {
17        P p1= new P() ;
18        Q q1= new Q();
19        q1.start(p1) ;
20 }
```

リスト3.3

どちらのパターンもPの変数aの値をQの変数bにコピーしていることには違いない．

第 3 項　通信処理の記述の比較

次にそれぞれの記述について考える．まず，occamの場合はPからQへaの値を通信するという動作に対してPとQが通信するためのチャネルを用意しなければならない．これは新たにチャネルを定義しなければならないという点が問題であるように思えるかもしれない．しかし，プロセス間の通信にチャネルを導入することは並列システムの構築のために有用な点が多い．このことについては以下の節で紹介する．

次にJavaの記述について考える．Javaではオブジェクト間通信をおこなう場合にチャネルを用意する必要はない．しかし上述のように，ある値を通信によって別のオブジェクトにコピーするときに 2 つの実装が考えられる．これは何を意味するのだろうか．

それは値を渡す場合も受け取る場合も常にメソッドを呼び出す側がその通信の主導権を持ち，他方のオブジェクトはメソッド呼び出しに対して待機していないといけないということである．すなわち通信において送信側または受信側のどちらかが主となる必要があり送信，受信はいつも対等ではないということである．

もし，いつでも通信の主導権をもつオブジェクトを一意に決定でき，かつそれは，そのシステムの実行されるあらゆる状況において不変であり，そのオブジェクトを他のシステムに移植する際

も通信に関する主従の関係が変わらないと言えないのなら，２つの実装パターンが存在することは考察に値するであろう．例えば図3.2のようにPとQを実装１ではP1とＱ１，実装２ではP2とＱ２と記述したとする．このとき，P1とQ2は（またはP２とQ1）結合できないということになる．再利用可能なオブジェクトを作成することは難しいことではあるが，この通信における主従の問題は本質的な問題のように思われる．

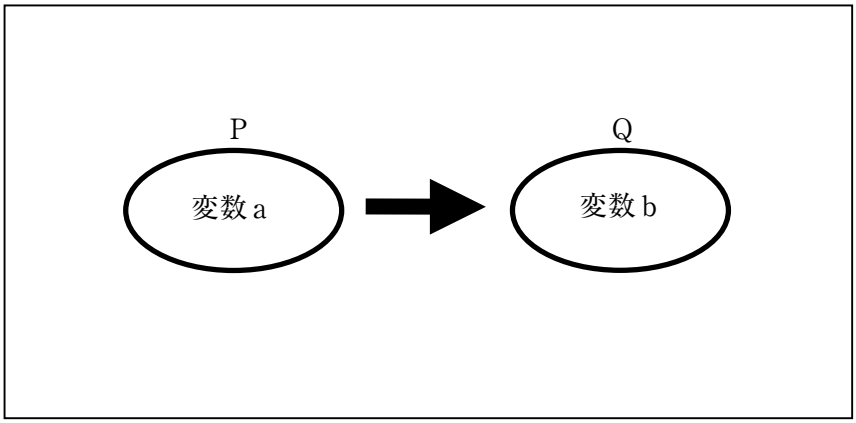

図3.1. 通信モデルの例

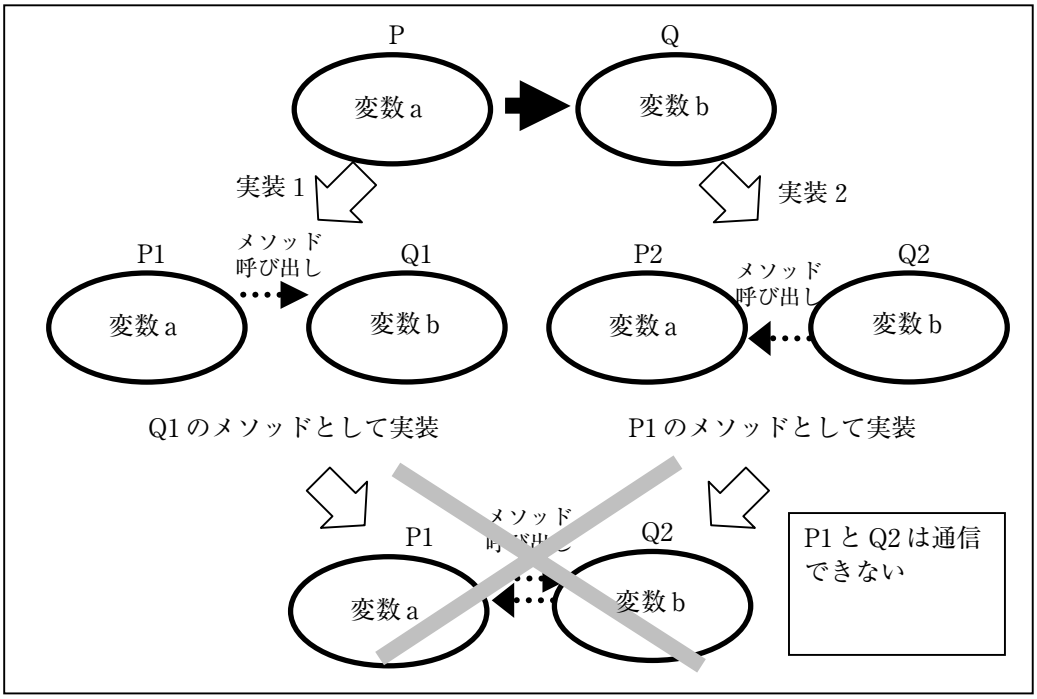

図3.2. 通信の主導権を持つオブジェクトを一意に決定できない場合の問題点

第2節　プロセス（オブジェクト）の独立性と移植性について

次にプロセス（オブジェクト）の独立性と移植性について考察する．図3.3のような計算プロセスがキャラクタ端末（キーボートとスクリーンを入出力装置とするコンピュータ）をユーザインターフェイスとして実行されていたとする．この計算オブジェクトをGUIを持つシステムに移植することを考える．Javaで記述されたプログラムの場合，キャラクタ端末を入出力装置として利用していた計算オブジェクトのプログラムコードには入出力装置との通信のためのメソッド呼び出しがオブジェクトを指定する形で記述されている．Javaではそれらのコードを GUIを利用するためのコードに変更しなければならない．つまり計算オブジェクトのプログラム内の変更が必要である．

一方，occamでこのシステムを記述した場合，計算オブジェクトと入出力装置とはチャネルだけで接続されているので計算オブジェクトの変更はほとんど必要ない．現在，オブジェクト指向においてもアスペクト指向などの考えを導入し，システムのオブジェクト間の依存性を低くしようという動きはある[1)]．しかし，そのような概念を導入しなければ依存性の高いプログラムが記述できてしまうこと，チャネルによる接続のほうが自然に独立性の高いプログラムが記述できることなどを考えるとチャネルによってプロセス同士が通信するoccamのほうが，プロセスの独立性，移植性の面で優位であると思われる．

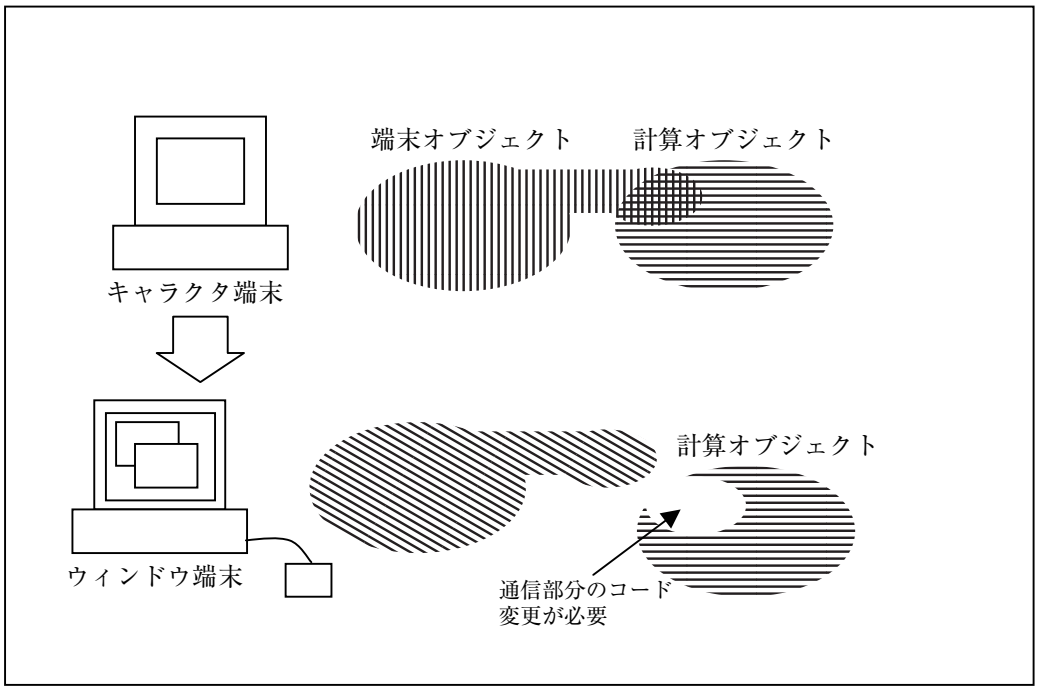

図3.3. 通信オブジェクトを指定するデメリット

第 3 節　プロセスの配置と並列システムの構造

occamではPARコンストラクション内のプロセスのみが並列に実行される．つまりPARコンストラクション内の記述をみれば，どのプロセスが並列に実行されているかが把握できる（実行時の状況によって変わるということはない）．そして配置されたプロセス間はチャネルによってのみ相互作用する．リスト3.4にその例を示す．この例ではまず，プロセスP1，P2，P3が並列動作する．そしてP 2 とP 3 が終了した後，P4とP5がP1と並列動作する．かつ，それぞれのプロセスは引数として与えられたチャネルを通してのみ相互作用する．

```
1 … PROC P1( )
2 … PROC P2( )
3 … PROC P3( )
4 … PROC P4( )
5 … PROC P5( )
6 CHAN OF INT p1p2, p1p4, p2p1, p2p3, p3p2, p4p1, p4p5, p5p4
7 PAR
8    P1(p1p2, p1p4, p2p1, p4p1)
9    SEQ
10       PAR
11          P2(p2p1, p3p2, p1p2, p2p3)
12          P3(p2p3, p3p2)
13       PAR
14          P4(p1p4, p5p4, p4p1, p4p5)
15          P5(p4p5, p5p4)
```

リスト3.4

プロセスの動的配置機能が並列プログラムのために必要かどうかは解決する問題によって異なるが，並列に動作するプロセスの振る舞いを把握することは極めて困難であることから，並列構成が把握しやすいことはシステムの構築，メンテナンスのために重要であると思われる．

第 4 章　並列プログラムの例

この章では簡単なP2P型の並列処理システムを例にoccamによる並列処理記述について紹介する．まず例となる並列システムの仕様および概念図（図4.1）を示す．

＜仕様＞

二人のプレーヤ（AさんとBさんとする）がn回じゃんけんをする．審判などはおらずお互い自分の勝った回数を数え，n回終わったら自分の勝った回数を宣言して終了する．それぞれのじゃんけんの手はプレーヤ自身が毎回決める（乱数などを用いて）．また，実際のじゃんけんと同じように自分の手を出しながら相手の手をみる．

この仕様をoccamで記述した例をリスト4.1に示す．

```
1 PROC player(CHAN OF INT in, out, VAL INT count, VAL [ ]BYTE name)
2   SEQ
3     …初期化
4     SEQ i=0 FOR count
5       SEQ
6         …自分の手を決める（変数handに代入）
7         PAR
8           out ! hand
9           in ? bhand
10        …勝敗の判定
11    …勝った回数の表示
12 ：
13 CHAN OF INT ch1,ch2
14 PAR
15   Player(ch1,ch2,3,”A”)
16   Player(ch2,ch1,3,”B”)
```

リスト4.1

この仕様の実装において，自分の手を出しながら相手の手を見るという処理をどのように実装するかが１つのポイントとなる．occamではリスト4.1のようにPARコンストラクタを用いて自分の手を相手に通信する処理と相手の手を見る（チャネルによって受け取る）処理を同時におこなうように記述できる．このようにoccamでは非常に細かいレベルでしかも階層的に並列処理を記述することが可能である．また，２人プレーヤは毎回それぞれの手をみるという通信で同期できるため矛盾なく最後まで処理が実行できる．

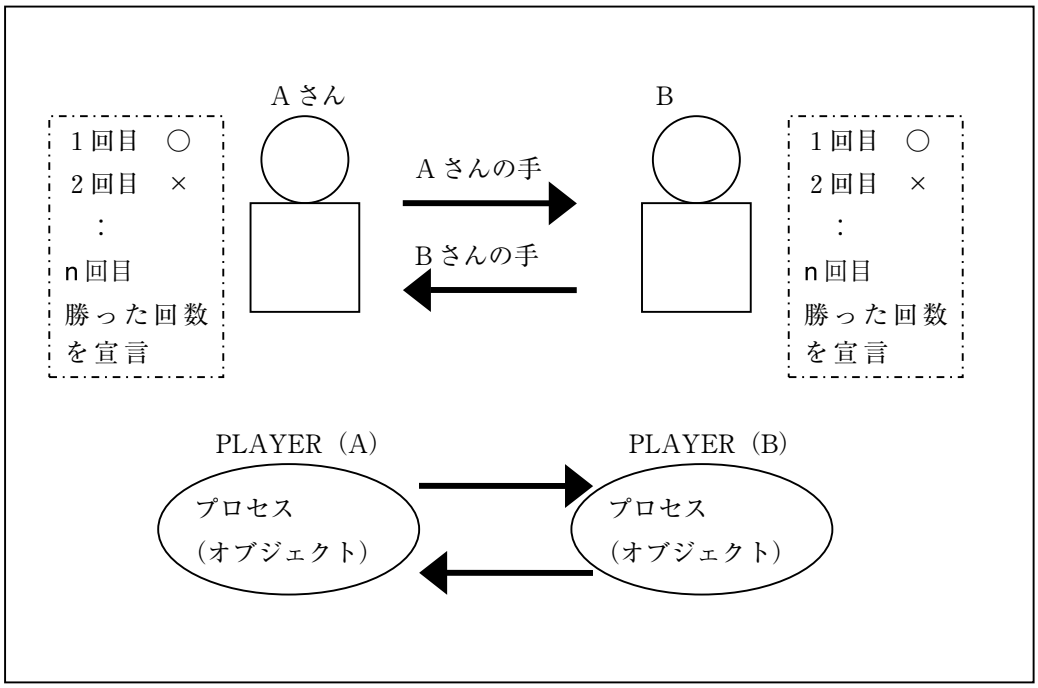

図4.1. じゃんけんシステムの概念図

第５章　並列処理プログラムにおけるバッファリングの問題

並列処理プログラムではプロセスのライブロック，デッドロック，インターリービングなどがよく問題になる．ここでは，それ以外の特にoccamのような同期通信型モデルにおけるバッファリングの問題について考察する．

occamにおける通信はバッファを介さない通信である．しかし，１つ１つの通信はバッファリングされなくても，バッファリングによってシステムの振る舞いが予期しない状態になる時がある．図5.1のようなシステムを考える．このシステムは２つのプロセスPとQから構成され，PとQおよびその環境（プロセスE）はそれぞれ図のようなチャネル構成で接続されている．この並列処理システムPとQの処理の内容がリスト5.1のようであるときのシステム全体の振舞いについて考える．

```
1 PROC P(CHAN OF INT  in1, in2, out1, out2)
2    WHILE TRUE
3       ALT
4          in1 ? a
5             out1 ! a
6          in2 ? a
7             out2 ! a
8 :
9 PROC Q(CHAN OF INT in1,in2,out1,out2)
10    WHILE TURE
11       ALT
12          in1 ? a
13             out1 ! a
14          in2 ? a
15             out2 ! a
16 :
17 CHAN OF INT chep,chpq,chqe, cheq,chqp,chpe:
18 PAR
19    P(chep, chqp, chpq, chpe)
20    Q(cheq, chpq, chqp, chqe)
21    … 環境プロセス
```

リスト5.1

今，環境であるプロセスEがリスト5.2のようだったとする．このときシステムが実行するチャネル通信の順番（トレース）[2] は例えば<chep.a,chqe.a>　となり，この場合はシステムはデッドロックする．

```
SEQ
   chep ! a
   cheq ! a
   chpe ? a
   chqe ? a
```

リスト5.2

次に図5.2のようにPとQの間にRを挿入したとする（例えばデバッグのためにPとQ間の通信をモニタリングするために）．この場合のトレースは例えば＜chep.a, cheq.a, chpr.a, chqp.a, chpe.a, chrq.a, chqe.a＞となり環境プロセスEは正常終了する．しかしこの場合も
＜chep.a, chpr.a, chrq.a＞と進むとデットロックする．つまりRを挿入することによってデッドロックしたりしなかったりするようになり，システムの挙動が不安定になってしまったのである．これはRがバッファプロセスとして作用したためである．occamでプログラムを作成するとこのような問題に直面しプログラムが思うように動かないことが多々ある．しかしそれはoccamが難しいのではなく，その柔軟な並列記述により，初めて並列処理の本質的な問題に直面したにすぎないと思う．このようなシステムの問題を解決するために（または，設計したシステムがデッドロックしないか検証するために）FDR2などのCSP理論を利用した検証ツールを利用することが望ましいであろう．

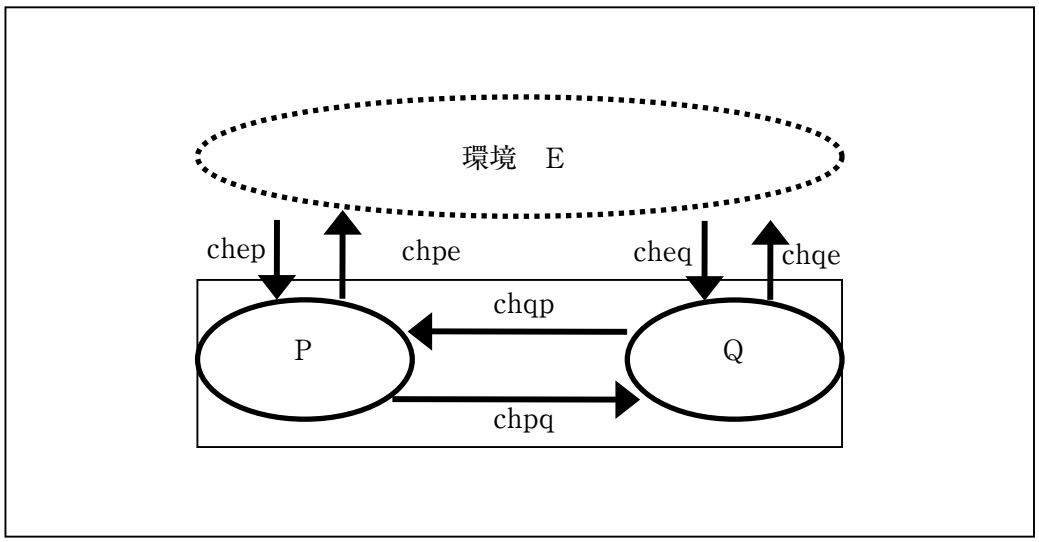

図5.1. プロセスP, Qと環境（E）の構成

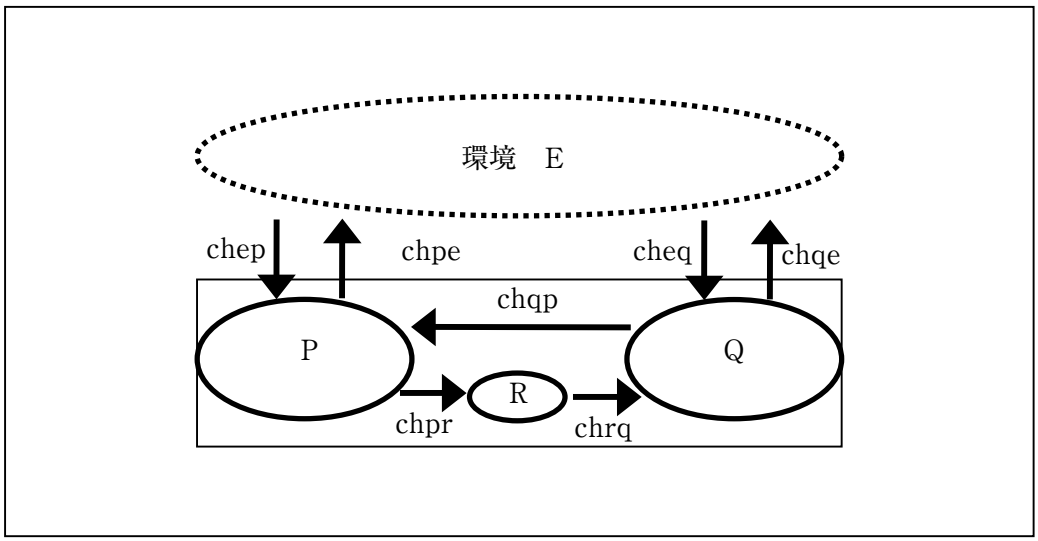

図5.2. プロセスP, Qと環境（E）にRを追加した構成

第 6 章　CSPモデルに関する動向

この章では本論文で紹介したCSP関連の動向について紹介する．

第 1 節　並列プロセスの検証ツール（FDR2）

CSP理論に基づく検証ツールとして，イギリスのFormal Systemsで開発された検証ツールFDR2がある．FDR2によってCSPで記述した並列プロセスがデッドロックするかどうかや記述した並列プロセスが仕様を満たしているかなどの検証をおこなうことができる．FDR2の関する情報としては文献[7]およびURL[3]が参考になる．

第 2 節　Occamの現在（KROC）

現在occamの代表的な処理環境としてはイギリスのKENT大学を中心に開発されたKROCがある．KROCとは（Kent Retargeted occam Compiler）の略で，Inmos社のトランスピュータ用として利用されていたoccamを他のCPU環境に移植するプロジェクトで，現在，Linuxなどの環境でoccamを利用できる（URL[1]）．利用するよりは，まずoccamについて学習したいという場合は文献[1，2，5，6，8]などが参考になるであろう．

第 3 節　Java環境におけるCSPモデルの実現（JCSP）

JCSPはCSPモデルにおけるチャネルとプロセスをJavaプログラムで実現するライブラリである（URL[2]）．例えば 3 章の図3.1で示すプロセスPとQをJCSPを用いてJavaで記述するとリスト6.1の

ようになり, 非常に容易にJavaでCSPモデルを記述できる. JCSPの特徴を挙げると次のようになる.

- 多彩なチャネルタイプ（１対１，多対１，１対多チャンネル）のサポート
- 高速なタスク（プロセス）スイッチング（文献[3]より）
- 並列の動作する部分をプロセスオブジェクトとして実装することでCSP理論を用いてJavaの並列プロセスの検証が容易になる
- シングルCPUのプログラムをEthernetにチャネルをマッピングすることによって比較的容易にマルチCPUプログラムの変更できる（JCSPネットワークエディション）

```
1 import com.quitckstone.jcsp.lang.*
2 public class P implements CSProsess {
3   private ChannelOutput out ;
4   public P(ChannelInput out) {
5     this.in = out ;
6   }
7   public void run() {
8     out.write(new integer (a)) ;
9   }
10 }
11 public class Q implements CSProsess {
12   private ChannelInput in ;
13   public Q(ChannelInput in) {
14     this.in = in ;
15   }
16   public void run() {
17     Integer b = (Integer)in.read();
18   }
19 }
20 public class SampleProgram {
21   public static void main(String[] args) {
22     One2OneChannel ch = new One2OneChannel() ;
23     new Parallel ( {new P(ch), new Q (ch)} ).run();
24   }
25 }
```

リスト6.1

第７章　おわりに

本論文ではoccamとJavaを比較することによって，並列処理を記述するためにどのような機能がプログラミング言語に必要なのかを考察した．

３章ではoccamのチャネル通信とJavaのメッソド呼び出しによるメッセージ通信を比較しoccamのチャネル通信は送信側と受信側が対等であること，チャネルによって接続されたプロセスは移植性や独立性が高いことなどを述べた．４章では簡単なP2P型の並列処理システムを例に並列処理プログラムに同期などの機能が必要であることを述べ，それをoccamで容易に実装できることを示した．

汎用性が重視されるプログラミング言語で同期通信の機能を実現することは非常にコストがかかることであり，かつ，現在作成されている多くのプログラムでは同期などの並列処理のためのメカニズムは必要ないかもしれない．しかし，マルチCPU環境が利用可能になったとき，今のメソッド呼び出しによって通信しているJavaプログラムを各CPUにオブジェクトを分散配置して並列実行したとしてもインターリービングの問題などによって正常に動作しないプログラムが出てくる可能性がある．

Javaは現在，非常にメジャーなプログラミング言語であり，多くのコストをかけて開発，普及活動が行われている．しかし，Javaも万能ではないと筆者は考える．CSP理論のような形式的手法によるシステムの設計，検証，そしてそれをoccamのようなシンプルで非常に柔軟に並列処理を記述できる言語で実装する．このような設計法もある．Javaをはじめとするオブジェクト指向的アプローチが並列処理システム構築のための唯一の選択肢なのか再考してみてもいいのではないだろうか．

謝　辞

本論文の執筆にあたり細部にまで目を通していただき多くの貴重なコメントをいただいた奈良先端科学技術大学院大学 情報科学研究科の関浩之教授に感謝します．また，今までヨーロッパの本分野の動向やCSP，JCSP関連の多くの有益な情報を提供していただいた松井和人氏（所属確認中）に感謝します．彼の学術的活動が世の中に広く認められることを期待します．また今まで著者の研究に関わっていただいた帝塚山学院大学 人間文化学部の山本正樹教授をはじめとする全ての研究者に感謝します．そしてこの研究の機会を与えてくださった帝塚山学院人間文化学部に感謝します．

最後にいつも笑顔でサポートしてくれた愛する妻と子に感謝します．

参考文献

[1] INMOS Limited occam2 Reference Manual, Prentice Hall International(U.K.), 1988

[2] Occamとトランスピュータ, 尾内理紀夫, 共立出版社, 1986

[3] Javaによるプロセス指向と並列処理プログラミングの方法 初級編, 松井和人, 2003,（JCSPネットワークエディション付属マニュアル）

[4] Java言語仕様 第2版, James Gosling, Bill Joy, Guy Steele, Gilad Bracha, 村上雅章訳, 株式会社ピアソン・エデュケーション, 2000

[5] Transputer/occamによる系列プログラミング入門, Ronald S. Cok著, 梅尾博司監訳,　共立出版株式会社, 1993

[6] 並列処理言語 Occam2, John Galletly, 山本正樹 監修, 日刊工業新聞社, 1991

[7] The Theory and Practice of Concurrency, Bill Roscoe, Prentice Hall, 1998

[8] ホーアCSPモデルの理論, C.A.R. Hoare著, 吉田信博訳, 丸善株式会社, 1992

[9] ポストオブジェクト指向, 日経BYTE, 日経BP社, May 2005, pp.26-49

URL

[1] KROCに関するページ　http://www.cs.kent.ac.uk/projects/ofa/kroc/

[2] JCSPに関するページ　http://www.cs.kent.ac.uk/projects/ofa/jcsp/

[3] FDR2に関するページ　http://www.fsel.com/index.html

注

1）複数のクラスにわたって横断的に共通の処理（例えばシステムのログ出力など）が必要になるとき，それをひとまとめにしたものをアスペクトと呼ぶ．ログ出力のような動作環境に依存する部分をアスペクトとすることによって依存性の低いオブジェクトを設計できる．（文献[9]）．

2）トレースとは発生するイベントの系列のことでoccamではチャネル通信がイベントとなる．チャネルchに値aが通信されるイベントをCSPではch.aと記述するため本論文ではその記述に準じる．