マルチメディア通信と分散処理 79－12
（1996. 11. 15）

マルチランデブを用いた分散システムの記述とバス結合ネットワーク上での実行

後藤 和裕†, 安本 慶一††, 東野 輝夫†, 谷口 健一†

† 大阪大学基礎工学部情報工学科, 豊中市待兼山町 1-3, Tel: 06-850-6607
E-mail: {k-gotoh, higashino, taniguchi}@ics.es.osaka-u.ac.jp

†† 滋賀大学経済学部情報管理学科, 彦根市馬場 1-1-1, Tel: 0749-27-1060
E-mail: yasumoto@biwako.shiga-u.ac.jp

あらまし

本稿では，マルチランデブを用いた分散システムの記述法，マルチランデブを含む分散システムの仕様の効率良い実現法を提案する．まず，LOTOS 仕様の各プロセスを異なる計算機で実行することを想定し，プロセス間に選択や割込，マルチランデブが指定可能な LOTOS のある部分クラスを用いて，並列多重処理や資源の排他制御などの分散システムの動作を簡潔に記述できることを示す．このクラスで記述された分散システムの仕様は，以前に我々が提案したバス型ネットワークでの LOTOS 仕様の実現法を用いて，実行効率の良い目的プログラム群として実現される．数台の UNIX マシンで構成される Fast Ethernet 上で実験を行い，我々が提案した実現法の実装上の問題点の検討および性能評価を行った．同期するプロセスの組合せが一定かつその数が数個程度の場合，毎秒 900 回程度と，マルチランデブを高速に処理できることを確認した．

キーワード: LOTOS, マルチランデブ, 分散システム, バス型ネットワーク, 実装

Describing Distributed Systems using Multi-rendezvous and their Implementation on Bus-Networks

Kazuhiro Gotoh†, Keiichi Yasumoto††, Teruo Higashino† and Kenichi Taniguchi†

† Dept. of Information and Computer Sciences, Osaka University, Toyonaka, Osaka 560, Japan
Tel: +81-6-850-6607, E-mail: {k-gotoh, higashino, taniguchi}@ics.es.osaka-u.ac.jp

†† Dept. of Information Processing and Management, Shiga University, Banba 1-1-1, Hikone 522, Japan
Tel: +81-749-27-1060, E-mail: yasumoto@biwako.shiga-u.ac.jp

Abstract

In this paper, we propose a series of methods for describing distributed systems using multi-rendezvous and for implementing those descriptions. First, we assume processes in a LOTOS specification are executed at several remote nodes, and we define a sub-class of LOTOS where choice, disabling and multi-rendezvous operators can be specified among processes. We show that in such a class we can describe the specifications of distributed systems such as duplex systems and mutual exclusion systems simply. The specifications described in the class can be implemented as efficient object codes on bus-networks using our implementation method which was proposed in our previous paper. From our experimental results on Fast Ethernet with several UNIX workstations, we have found and solved several problems in implementing our method. We have also evaluated the performance of our implementation method. When the combination of synchronizing processes is fixed and its number is small, over 900 multi-rendezvouz have been processed per second.

key words: LOTOS, multi-rendezvous, distributed system, bus networks, implementation

1 はじめに

形式記述言語LOTOS [4] は，複数プロセス間の選択や割込，マルチランデブ（複数並行プロセスの間で同期しデータ交換を行うための機構）などの高度な機構を有する．分散システムを効率良く設計・開発するには，マルチランデブなどの機能を用いて，抽象度の高い仕様としてシステムの動作を記述し，仕様からなるべく実行効率の良い全ノードの目的プログラム群を自動生成できることが望ましい．

本稿では，LOTOSのあるサブクラスを用いた分散システムの記述法を与える．記述した分散システムは，文献 [3] で我々が提案した方法により，目的プログラム群として実現可能である．また，文献 [3] の実現法を，実際のネットワーク上に実装するため，実験を行い実装上の問題点の検討および性能評価を行う．

提案する記述法では，複数のプロセスが並行に動作してもよいが，各プロセスは1つの計算機で実行されるものとして記述する．各プロセスには，その計算機に固有な入出力イベントや内部イベントの他，他のプロセスと同期させたいイベントや任意の計算機に対するプロセス呼出しが含まれてもよいとする．分散システム全体の仕様は，これらのプロセスの内容と，それらの間の実行関係の指定により記述する．プロセス間の実行関係の指定には，同期オペレータ以外に，選択，非同期並列，割込，逐次オペレータを使用する．本記述法は，(1) 各計算機で動作させる内容を，LOTOS仕様の各プロセスとして他と独立に記述できること，(2) 複数の計算機間の同期やデータ交換，および排他制御をマルチランデブを用いて一度に指定できること，(3) 計算機間の動作の制約や依存関係を，各計算機の動作記述を変更することなく，別計算機で動作するプロセスとして新たに記述し追加できること，などの特徴を持つ．本稿では，以上の特徴を用いて，並列多重処理システム，資源の排他制御，クライアント・サーバシステムなどの分散システムが簡潔に記述できることを示す．

分散システムの記述は，文献 [3] の手法により，バス型ネットワーク上の計算機で動作する目的プログラム群として実現される．ノード間の1対1通信などを用いてマルチランデブを実現する方法 [1, 2, 5] では，選択的に実行される複数のマルチランデブの競合を解決するため，1回のマルチランデブの処理に分散システムの全ノードに応じたオーダのメッセージ交換が必要であった．一方，我々の手法 [3] では，バス型ネットワークのブロードキャストメッセージは全てのプロセスで同じ順序で受信されるという性質を利用して，先に同期条件の整ったランデブを各ノードで独立に決定するため，競合を解決するためのメッセージ交換が不要となり，マルチランデブを高速に処理できる．また，従来手法では考慮されていない，プロセスの動的生成やプロセス間の選択なども実現している．

文献 [3] の実現法をUNIXワークステーションで構成されたFast Ethernet上に実装するための予備実験を行った．OSのカーネルが管理するメッセージ受信バッファのオーバーフローなどの幾つかの実装上の問題点を解決することで，本手法の上記環境への実装が可能であることを確認した．また，実際に100Base-TXネットワーク上の数台のマシンの間で毎秒900回程度のマルチランデブが行えることを確認した．

2 分散システムの動作仕様の記述

2.1 実行環境と記述クラス

本稿では，分散システムを1つのLOTOS仕様として記述し，n 個の計算機からなるネットワーク上で実行することを想定している（LOTOSの詳細については，文献 [4] 参照）．

以下では，このような実行環境を考慮した，分散システム記述のための記述クラスを定める．

$M = \{M_1, M_2, ..., M_n\}$ を計算機の集合とする．L_j, S_j をそれぞれ，計算機 M_j に固有のイベント（内部イベントを含む）の集合，計算機間でマルチランデブにより同期するイベントの集合とする．M_j で実行可能な全てのイベントの集合を A_j とすると，$A_j = L_j \cup S_j$ である．

$P = \{p_1, p_2, ..., p_m\}$ を分散システムを記述したLOTOS仕様に含まれるプロセスの集合とする．P の各要素は，ある一つの計算機 M_j に割り当て実行することとし，M_j に割り当て可能なプロセスの集合を $P(M_j)$ で表す．

$p_i \in P(M_j)$ の場合，p_i の動作式を，イベントの集合 A_j とプロセス呼出しの集合 P，および，それらの間の実行順序の指定のための，順次（;），選択（[]），並列（|||），同期（|[G]|, ||），逐次（>>），割込（[>）などのオペレータを用いて，以下のクラスで記述する．

$B = B1 \mid B2 \mid B >> B$

$B1 =$ stop $\mid$ exit $\mid$ a;$B1 \mid B1[]B1 \mid B1|||B1 \mid$
$B1|[G]|B1 \mid B1||B1 \mid B1 >> B1 \mid B1[> B1 \mid$
p(ただし p は他計算機で動くプロセス呼び出しを含まない)

$B2 = p \mid B2[]B2 \mid B2|||B2 \mid B2|[G]|B2 \mid$
$B2||B2 \mid B2[> B2$

ただし，$p := B \mid\mid\mid p$ のように，プロセスが自分自身を選択的，並列的に無限に呼び出す場合を除く．ここで，B1はイベントおよびローカルなプロセス呼出しだけからなるプロセス，B2は他計算機で動作するプロセスを含むプロセス呼び出しだけからなるプロセスを表している．また，p_i で使われている全イベントの集合 A_{p_i}，および同期ゲートリスト G は，$A_{p_i} \subseteq A_j$，$G \subseteq Gate(A_j)$ である（$Gate(A)$ はイベントの集合 A で使われているゲートの集合）．

B1, B2では，選択，並列，同期，逐次，割込のオペレータは，イベントおよびローカルなプロセス呼出しのみからなる動作式間，あるいは，プロセス呼出し間にしか設定できないよう制限している．これは，後の実装時に，各プロセス内で，同期可能なイベントの集合を求める処理，およびプロセス間で同期可能なイベントを求める処理を独立に行えるようにするためである．

同期オペレータ（|[G]|）を用いて，複数プロセス間でマルチランデブを行うよう指定できる．n 個のプロセス間の同期指定，

$$p_1|[G_1]|p_2|[G_2]|...|[G_{n-1}]|p_n$$

において，あるゲート $g \in G_1 \cap G_2 \cap \cdots \cap G_{n-1}$ のイベントを同期実行できるのは，全てのプロセスが g のイベントの実行を要求しており，かつ，どの2つのプロセス p_i, p_j においても，イベントの入出力値が表1の同期条件を満たす時に限る．

表 1: マルチランデブの実行条件

p_i	p_j	同期条件	作用
$a!E_i$	$a!E_j$	$val(E_i) = val(E_j)$	値の照合
$a!E_i$	$a?x:t$	$val(E_i) \in domain(t)$	値の代入
$a?x:t$	$a?y:u$	$t = u$	値の生成

2.2 分散システムの記述

2.1節の記述クラスのもとで，マルチランデブを用いることにより，計算機間の以下のような通信動作を簡潔に記述できる．

(1) 複数プロセスでゲート（ゲートからの入力）を共有する．

(2) プロセス間である条件が成立した時に，あるプロセス（群）の計算結果（データ）を他のプロセス群へ受け渡す．

(3) プロセス間である条件が成立した時に，複数のプロセスの間でイベントの同期により待ち合わせを行う．

(4) 他のプロセス群に指定されている実行関係に従い，それらのプロセスの生成・消滅などの制御を行う．

以下では，本記述法により，一般的な分散システムの動作を簡潔に記述できることを示す．

例 1. 並列多重処理システム

あるサービスの高信頼化のため，多重化を行う場合を考える．サービスを行うプロセスの誤り率が極めて低い場合には，例えば，3 つのプロセスに同じ処理を行わせた後，結果を比較し，多数決を取る方法が広く用いられる．

このようなシステムを LOTOS のマルチランデブを用いて記述してみよう．ただし，サービスの内容は，タスクの処理要求の受理（req?task），task の処理，処理結果の出力（line!result）の繰り返しとし，誤り検出（er!pid）を受けると，エラー状態 (入力を無視する) に入り，task の処理をしないものとする．サービス S は以下のように記述できる．

```
process S[req,line,er](pid):noexit:=
  (req?task; ProcessTask(task)
 >> accept in result line!result;
                    S[req,line,er](pid))
 [> er!pid; ErrorState[req]
  where
    process ErrorState[req]:noexit:=
      req?task; ErrorState[req]
    endproc
endproc
```

(上記において，ProcessTask(task) はタスク処理要求 task を処理し，結果 result を返すプロセスとする)

多重化システム全体としては，サービス S を異なる 3 つの計算機で並列に実行する（S1, S2, S3 とする）．これら 3 つのプロセスは，同じタスク要求を受け，並列に処理させるため，タスク要求入力用のゲート req を共有する．ゲートからの入力を 3 つのプロセスで共有するため，上記のマルチランデブの機能 (1) を用いて以下のように記述する．

```
S1 |[req]| S2 |[req]| S3
```

次に S1～S3 が独自に処理した結果を比較し，誤りを検出し，正しい結果を出力するプロセス Checker を用意する．S1～S3 と Checker 間の処理結果の受渡し，誤り検出の通知はそれぞれ，ゲート line, er を通して行われるものとする．また，S1～S3 は，イベント er により停止するものとする．以上は，マルチランデブの機能 (2) および LOTOS の割込オペレータ [>を用いて以下のように簡潔に記述できる．

```
((S1 |[req]| S2 |[req]| S3) [> er; stop)
     |[line,er]| Checker
```

Checker では，マルチランデブの機能 (2) を用いて，プロセスが 3 つとも稼働しているとき（state=1）には，S1～S3 からそれぞれ 1 回ずつ処理結果 x, y, z を受け取る．また，x, y, z を比較し，処理結果が異なるプロセスを故障したと判断し，そのプロセスをシステムから切り放すためのイベント er!pid を発行する．タスクを処理するプロセスが 2 つの状態で（state=2），お互いに異なる結果を返してきた時には，サービスを停止させるためのイベント er を実行する．

```
process Checker[line,out,er](state) : exit :=
  line?pid1?x; line?pid2?y[pid1 neq pid2];
   (
     [state eq 1]-> line?pid3?z
                    [pid3 neq pid1 and pid3 neq pid2];
     ([x eq y and y eq z]-> out!x;
                            Checker[line,out,er](1)
   [] [x eq y and y neq z]-> er!pid3; out!x;
                            Checker[line,out,er](2)
   [] [x eq z and z neq y]-> er!pid2; out!x;
                            Checker[line,out,er](2)
   [] [y eq z and z neq x]-> er!pid1; out!y;
                            Checker[line,out,er](2)
   [] [x neq y and y neq z and z neq x]-> er; stop
    )
  [] [state eq 2 and x eq y]-> out!x;
                            Checker[line,out,er](2)
  [] [state eq 2 and x neq y]-> er; stop
endproc
```

(ただし，上記では，変数の型名を省略している．pid1～pid3 はプロセスの ID，state はシステムの状態を表す整数変数であり，x, y, z は処理結果を記憶可能な変数とする．また，比較演算子 eq, neq がこれらの変数の各型ごとに定義されているものとし，論理演算 and も同様に定義済であるとする)

例 2. マルチメディア情報提供サービス

複数のデータストリームが同期して転送される必要のあるマルチメディア情報提供サービスなどは，マルチランデブの機能 (3) を用いて簡潔に記述できる．

動画の要求があった時に，それをゲート v を通じてクライアント C に提供するプロセスを SV とする．同様にゲート a を通じて C に音声データを提供するプロセスを SA，ゲート t を通じてテキスト情報を提供するプロセスを ST とする．

システム全体の構成は，マルチランデブの機能 (2) を利用して以下のように記述できる．

```
(SV ||| SA ||| ST) |[v,a,t]| C
```

しかし，これだと，SV, SA, ST は非同期に動作するので，3 つのプロセスが動作している計算機の負荷および C との間のリンクのバンド幅などにより，クライアント C で

再生される動画，音声，テキスト情報の関連はでたらめなものとなる．

そこで，動画と音声，テキスト情報の間で同期させるため，各プロセスに同期用のゲート s を用意し，マルチランデブの機能 (3) を用いて，待ち合わせを行わせる．同期を行うシステムの全体は以下のように記述できる．

```
(SV |[s]| SA |[s]| ST) |[v,a,t,s]| C
```

ただし，SV, SA, ST, C には一定の間隔，例えば SV では画像 30 フレーム毎に同期用のイベント s を発行するよう記述する．

```
process SV[v,s](video:file, f:integer): noexit:=
    ([(f mod 30) neq 0] -> v!Frame(video,f);
                           SV[v,s](video,f+1)
  [] [(f mod 30) eq 0] -> s; SV[v,s](video,f+1)
    )
endproc
```

(ここで，Frame(video,f) は動画 video の f フレーム目の画像データを表す)

ST のみ，SV, SA より少ない頻度で同期するようにするには，他の同期用ゲート s2 を導入し，SV, SA, ST, C において適当なタイミングで同期用イベント s2 を発行するようにすれば良い．

例 3. 資源の排他制御

ある計算機に存在する資源 a を P1～Pn のプロセスのうち高々k 個のみが同時に使用できると言った排他制御を行いたい場合がある．

資源 a の利用開始を a!lock, 使用終了を a!unlock で表し，資源 a の最大同時利用可能個数を maxusers とすると，資源 a の制約は以下のようにプロセスとして記述可能である．

```
process R[a](users,maxusers): noexit:=
  ([users leq maxusers]-> a!lock; R[a](users+1, maxusers))
[] a!unlock; R[a](users-1, maxusers)
endproc
```

資源 a を利用するプロセス P1～Pn は，資源 a の操作に関して以外は独立に動作可能とするなら，このようなシステムの全体は以下のように記述できる．

```
(P1 ||| P2 ||| ... ||| Pn) |[a]| R[a](0,k)
```

ここで，プロセス R[a] 中の a!lock, a!unlock は P1～Pn のどれかと同期し，資源の現在の利用者数 users を増減する．users が maxusers を越えた時には，a!unlock しか実行できず，a!lock を行いたいプロセスは資源が開放されるまで，待たされることになる．

このように資源の排他制御は，資源の制約を表すプロセスと利用者となる並列プロセス群の間を同期オペレータで結合することで，簡潔に記述することができる．同様に資源 b, c について最大利用者数 m, l を設けたい場合は，それぞれ制約を以下のように追加すれば良い．

```
(((P1 ||| P2 ||| ... ||| Pn)
|[a]| R[a](0,k)) |[b]| R[b](0,m)) |[c]| R[c](0,l)
```

P1～Pn が R[b], R[c] と資源 b, c に関して同期し，資源の利用を制約することができる．以上のように，制約指向スタイルで記述することによって，並列プロセス群の動作を制御することが可能になる（上記のマルチランデブの機能 (4) に相当）．

その他の応用

LOTOS の選択オペレータなどを，マルチランデブなどと組み合わせて用いることで，例 2 における，マルチメディアサービスを以下のように拡張することも可能である．

```
(SV |[s]| SA |[s]| ST) |[v,a,t,s]| Client

SV:= (SV1 [] SV5) >> SV
SA:= (SA3 [] SA4) >> SA
ST:= (ST2 [] ST6 [] ST7) >> ST
```

上の記述では，同じサーバプロセスを幾つかの計算機に分散配置し，計算機の負荷状況や通信リンクのバンド幅などの状況に応じて，サービスを提供するプロセスを切り替えるように拡張されている．

以上のことから，本記述法の特徴として，

- 各計算機で動作させる内容を，LOTOS 仕様の各プロセスとして，他と独立に記述できること，
- 複数の計算機間の同期やデータ交換，および排他制御を，マルチランデブを用いて一度に指定できること，
- 計算機間の動作の制約や依存関係を，各計算機の動作記述を変更することなく，別計算機で動作するプロセスとして新たに記述し，追加できること，

などが挙げられる．これらの特徴により，本記述法は，複雑な分散システムを簡潔に記述する方法の一つとして有効であると思われる．

3 動作仕様の実現

3.1 実装対象ネットワーク

本稿では，次の条件を満足するネットワーク上に 2.1節の LOTOS 仕様を実現する．

(1) 1 回のブロードキャストにより全プロセスに同じ情報を送信することが可能である．

(2) メッセージは誤りなく有限時間内に到着する．

(3) ブロードキャストしたメッセージの到着順序はどのプロセスでも同じである．

Ethernet などのバス型ネットワークはおおむね，この条件を満足するが，上記を満たさない場合の対処法については，4節で考察する．

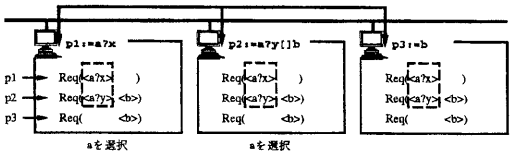
3.2 対象プロセス

以下では，ノード M_jで実行されるプロセスを p_jと記述する（ノード M_jで複数のプロセスが実行される場合は，$p_{j,1}, p_{j,2}, ...$とする）．本稿では，各ノードおよびプロセスについて以下を仮定している．

(1) ノード M_jは，全プロセスからのブロードキャストメッセージを非同期に受信する．また，受信メッセージを保持するための十分なバッファbuf_jを所有する．

(2) $p_{j,i}$は，buf_jの内容を参照・変更できる．

(3) $p_{j,i}$は，各時点において他と同期させたいイベントの集合 $E_{j,i}$を計算できる（計算方法は文献 [6] 参照）．



$p1[a] \;||\; p2[a,b] \;||\; p3[b]$ $\quad |P(a)|=2, |P(b)|=2$

図 1: p1, p2, p3 の順にブロードキャストした時の動作例

3.3 バス型ネットワーク上での動作仕様の実現

マルチランデブを異なる計算機上のプロセス間で実現するには，(i) 各プロセス p_j がどのイベントを実行したいのかを他のプロセスへ通知すること，(ii) 同期条件を満たす全てのプロセスの組合せを計算すること，(iii) 複数存在する場合には，そのうちの一つを全てのプロセスで一意に選択すること，という 3 つの処理が必要である．

以下では，3.1節の仮定を利用し，上記の (ii)，(iii) を各プロセスが持っている情報から他と独立に処理できる仕組みを与える．

3.3.1 プロセス間に同期のみが指定されている場合

各ゲート g で同期するプロセスの集合を $P(g)$ で表す．まず，$p_1 \;||\; p_2 \;||\; ... \;||\; p_k$ のように，プロセスの間にすべてのゲートで同期するよう指定されている場合，すなわち，$P(g)$ が一定である場合を考える．全プロセス $\{p_1,...,p_n\}$ には，各ゲートについて $|P(g)|$ （P(g) の要素数）の情報があらかじめ与えられているとする．

実行したいイベントの集合の情報を $E=\{<g_1,v_1,e_1>,\cdots,<g_h,v_h,e_h>\}$ で表す．ここで，g は同期が行われるゲート名，v はイベントの入出力値のリスト，e はガード式である．また，E の同期要求メッセージを $Req(E)$ で表す．

以下に，各プロセス p_j の動作手順の概略を示す（詳細は，文献 [3] 参照）．

(1) p_j において他と同期可能なイベントの集合 E_j を求め，$Req(E_j)$ をブロードキャストする．

(2) buf_j の内容から同期可能なイベント a^+ を判定する．判定は，入出力値リスト v およびガード式 e が表 1 の同期条件を満たすメッセージが buf_j 内に $|P(g)|$ 個あるかどうか調べることによって行なう．

3.1節の仮定より，buf_j に格納されているメッセージの順序および内容はどのプロセスでも同じであるため，どのイベントを選択するかの判定は全てのプロセスで同じになる．

(3) a^+ が p_j の同期要求したイベントであった場合（a^+ が同期要求 $Req(E_j)$ に含まれる場合），a^+ を実行する．

1 回の同期の決定に必要なメッセージ数

従来の 1 対 1 通信 [2] や分散共有メモリ [5] を用いた方式では，それぞれのプロセスにおいてメッセージの到着順序に違いが生じるため，全てのプロセスからメッセージを受け取った後でないと，どのゲートでの同期を選択するのかを決定できない．よって，全プロセスの数 n に依存した $O(n\sqrt{n})$[2] や $O(n\log n)$[5] のメッセージ交換が必要となる．

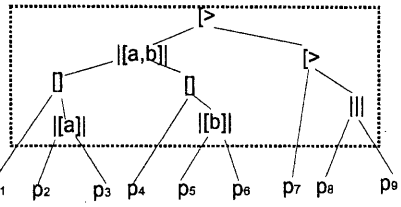


図 2: LOTOS 動作式の 2 分木表現

一方，本手法では，いずれかのゲートについてのメッセージがそろった時点で同期が可能になったことを判定できるため，各ゲート g_k における 1 回のマルチランデブを $|P(g_k)|=n_k$ 回のブロードキャストメッセージで実現できる．一般に，各 g_k について $n_k \ll n(1 \le k \le m)$ であり，バス結合型ネットワークでは，ブロードキャストによる 1 対多通信を，1 対 1 通信と同程度のコストで実現できるため，本手法では，マルチランデブの処理が大幅に高速化される．

3.3.2 プロセス間に選択や割込などが指定されている場合

(1) プロセス間の選択，非同期並列，割込，(2) 他のノードに対するプロセスの動的生成，などが仕様に指定されている場合には，各ゲート g で同期するプロセスの集合 $P(g)$ は動的に変化しうる．2 節に記述したような分散問題のほとんどは，このクラスを用いて簡潔に記述される．

2.1節の仮定より，各時点におけるプロセス群およびそれらの間の実行関係の状態（動作式）は，図 2のような，プロセスを葉とし，オペレータを節とする 2 分木で表せる．

各プロセスは，(1) 自プロセスと他プロセスとの間に指定されている実行関係および，(2) 関連するプロセスのイベントの実行要求とその受信順序が得られれば，どのプロセスの間でどのイベントを実行するのかを判定できる．プロセスの間の実行関係は，選択の実行によるプロセスの消滅や，プロセス呼出しなどにより刻々と変化するため，各プロセスで最新の情報を保持しておく必要がある．

2 分木の根から各プロセス p_j へ至る途中のオペレータの種類と接続位置の対の順序付きリストを，p_j のオペレータ情報と定義し，CID_j で表す．本手法では，プロセス間の最新の実行関係を各プロセスで保持するため，オペレータ情報を各イベントの同期要求メッセージに付加する．

プロセス間でイベントの実行順序を決定するための作業領域として，動作式の 2 分木からプロセスを除いた部分木の構造 (図 2の点線枠内) を持つ制御領域を各 p_j に用意する．制御領域内の各ノードには，オペレータの種類に応じて「どちら側のプロセスが選択されたか」「割込が発生したかどうか」を記憶するための領域を確保する．また，同期オペレータに対応するノードには，同期の相手を待っているプロセスのイベントの情報を記憶する領域を確保する．制御領域において，根ノードからプロセス p_j が接続される葉ノードに至る経路に相当する部分領域を p_j の制御パスと呼び，p_j のオペレータ情報 CID_j と関連づけ，$Path(CID_j)$ で表す．

表 2: メッセージ送信回数と受信率の関係

受信メッセージ数	P5-90	P5-133	P6-200	P6-200
2300	99.85	100.0	100.0	100.0
2500	99.98	100.0	98.32	99.93
3000	99.85	100.0	98.56	100.0
3500	98.50	100.0	98.27	100.0
4200	73.50	100.0	94.90	100.0
5900	40.01	100.0	97.87	98.34
6800	30.15	75.40	96.24	99.79
(mes./sec)				(%)

オペレータ情報 CID_j および制御パス $Path(CID_j)$ は，仕様を静的に解析することで求めることができるため，各プロセス p_j にあらかじめ与えておく．

オペレータ情報 CID_j と制御パス $Path(CID_j)$ を用いて，各プロセス p_j は，(1) オペレータ情報を付加した同期要求メッセージをブロードキャストする．次に，(2) 受信したメッセージから制御領域を構成あるいは更新し，制御領域の各項に書き込まれた情報に従って，受信したメッセージで要求されているイベントが実行可能かどうか判定する（詳細は，文献 [3, 6] 参照）．

4 性能評価および実装上の問題点

動作仕様を実装する対象のネットワークが，3.1節の仮定 (1)-(3) を満たすかどうかについて，各計算機から UDP/IP プロトコルを用いてメッセージのブロードキャストを行うことにより，調査を行った．実験は，イーサーネット (100Base-TX) で接続された 4 台の Gateway 2000 上 (CPU はそれぞれ Pentium 90MHz, Pentium 133MHz, Pentium Pro 200MHz × 2) で行ない，結果は 4 回計測した上での平均値を用いた．

最初に，3.1節の (2) に関して，どの程度のメッセージ送信頻度であればメッセージの消失を抑えることができるかについて調べた．メッセージの大きさを 16 byte とし，1 分間ブロードキャストした場合の実験結果を表 2に示す．送信するメッセージが 1 秒当たり 3500 個以下であれば，ほぼ 100%のメッセージを受信できることが分かった．このことから，同期するプロセスの数が 4 程度であれば，1 秒間に約 900 回程度のマルチランデブが処理できることになる．

また，メッセージの送信回数を 1 秒当たり 2500 個とし，ブロードキャストするメッセージの大きさを変えて実験を行なった．受信率は，メッセージの大きさが 512 byte では 96.36%，1,024 byte では 93.21%となり，送信するメッセージの大きさが増加するにつれ，消失するメッセージが増加することがわかった．これは，メッセージの受信処理が追い付かずにメッセージの受信バッファがあふれてしまうためだと思われる．

次に，3.1節の (3) ブロードキャストしたメッセージの到着順序について調べた．UDP を用いてメッセージのブロードキャストをした場合，自分自身がブロードキャストしたメッセージの到着順序が他の計算機よりも早くなる時がある．これは自分自身へのメッセージはループバックにより直接受信バッファに書き込んでしまうためである．

実装時の解決策

(1) 受信バッファからのメッセージあふれについて

一般には，メッセージの再送機構が必要である．3.3節のマルチランデブの処理手順では，各プロセスはイベントの実行の度に，他と同期させたいイベントの集合についての同期要求を 1 回ブロードキャストする．従って，受信バッファに存在するメッセージ数は高々 N (全プロセスの数) 個であるため，各メッセージのサイズを縮小したり，適当な時間間隔で各プロセスのローカルバッファへ転送するなどの工夫を行うことで，大部分は解決可能であると思われる．

(2) メッセージの到着順序について

自分自身にブロードキャストしたメッセージの到着順序が早くなる場合については，メッセージを受け取った計算機が，送信者に，そのメッセージの到着順序を知らせる，という機構を実装することにより解決できると思われる．また，同期に関与しない計算機を 1 台用意し，同期に参加する計算機はその計算機にブロードキャストを依頼する方法も考えられる．OS での処理を変更することで解決する方法も検討中である．

5 おわりに

本稿では，LOTOS のマルチランデブを用いた分散システムの記述，および分散ノード間に指定されたマルチランデブを含む LOTOS 仕様の，バス型ネットワーク上への実現法を与えた．また，提案手法により，ノード間のマルチランデブを高速に実現できることを実験により示した．本手法は，従来あまり考慮されていなかった，各ゲートにおいて同期するプロセスの組合せが動的に変化するクラスの LOTOS 仕様を扱えるため，実際の分散システムを簡潔に記述し，効率良く実現できる．

今後，アルゴリズムの詳細をライブラリとして実現し，我々が開発している LOTOS コンパイラ [6] へ組み込みを行う．また，実際の分散システムの記述に対して目的プログラムを自動生成し，性能評価を行う予定である．

参考文献

[1] Bochmann, G. v., Gao, Q. and Wu, C.: "On the Distributed Implementation of LOTOS", *Proc. of 2nd Int. Conf. on Formal Description Techniques (FORTE'89)*, pp. 133 – 146 (1989).

[2] Cheng, Z., Huang, T., and Shiratori, N.: "A New Distributed Algorithm for Implementation of LOTOS Multi-Rendezvous", *Proc. of 7th Int. Conf. on Formal Description Techniques (FORTE'94)*, pp. 483 – 494 (1994).

[3] 後藤，安本，東野，谷口：バス結合型ネットワーク上でのマルチランデブの実現法，信学技報 SS96-7, pp. 17 – 24 (1996).

[4] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", *IS 8807* (1989).

[5] Naik, K.: "Distributed Implementation of Multi-rendezvous in LOTOS Using the Orthogonal Communication Structure in Linda", *Proc. of 15th Int. Conf. on Distributed Computing Systems (ICDCS'95)*, pp. 518 – 525 (1995).

[6] Yasumoto, K., Higashino, T., Abe, K., Matsuura, T. and Taniguchi, K.: "A LOTOS Compiler Generating Multi-threaded Object Codes", *Proc. of 8th Int. Conf. on Formal Description Techniques (FORTE'95)*, pp. 271 – 286 (1995).