分散環境でのLOTOS仕様の実現とその評価

辰本 比呂記[†] 後藤 和裕[†] 安本 慶一[††]
東野 輝夫[†] 安倍 広多[†††]
松浦 敏雄[†††] 谷口 健一[†]

本論文では，プロセス間に選択や割込み，マルチランデブが指定可能なLOTOSのある部分クラスで記述された分散システムの仕様を実行効率の良い目的コード群として実現する手法を提案する．提案する手法では，LOTOS仕様における各プロセスはそれぞれ1つのノードに割り当てられる．1つのノードに割り当てられたプロセス群は我々の開発したLOTOSコンパイラによりマルチスレッド化目的コードに変換される．得られた目的コード群はそれぞれ割り当てられたノード上で実行されることにより互いにメッセージ交換を行いながら与えられた仕様を満たすよう協調して動作する．ネットワークを介したプロセス間のマルチランデブを効率良く実現するため，ブロードキャストを用いたアルゴリズムを考案し実装した．本手法に従って排他制御システムや多重化システムなどの分散システムを記述し，作成したコンパイラにより得られた目的コード群を実行した結果，ラピッドプロトタイピングなどの用途には十分高速に動作する目的コード群が生成できることを確認した．

Implementation of LOTOS Specifications on Distributed Environments and Its Evaluation

Hiroki Tatsumoto,[†] Kazuhiro Gotoh,[†] Keiichi Yasumoto,[††]
Teruo Higashino,[†] Kota Abe,[†††] Toshio Matsuura[†††]
and Kenichi Taniguchi[†]

In this paper, we propose an implementation method for specifications of distributed systems described in a subclass of LOTOS where operators such as choice and disabling can be used in combination with multi-rendezvous among remote processes. Each process in a LOTOS specification is assigned to a node. The processes assigned to a node are transformed into a multi-threaded object code by our existing LOTOS compiler. The set of object codes run on the corresponding nodes cooperating with each other by exchanging messages. Here, we assume that the broadcasted messages are received at all nodes in the same order, and propose a technique to implement multi-rendezvous among remote processes in the object codes running on different nodes. We have also developed a new LOTOS compiler which generates object codes running on Ethernet by extending our existing LOTOS compiler. From some experimental results, it is shown that the typical distributed systems such as duplication systems and mutual exclusion systems can be described simply and implemented efficiently.

1. はじめに

近年，計算機ネットワークの発展にともない，より大規模かつ複雑な通信プロトコルや分散システムが必要になってきている．複雑化・肥大化するこれらのシステムを開発する際には，仕様の誤解釈や誤りを回避し，また，仕様の正しさを機械的に検証できることが望ましい．このため，SDL，Estelle，LOTOSなどの形式記述技法が近年注目されている．仕様記述言語LOTOS[4]は，並行プロセス間の選択や割込みなどの強力な構文に加えてマルチランデブ（複数並列プロセス間で，指定されたゲートを介してイベントを同期実行しデータ交換を行う機構）と呼ばれる高度な同期通信プリミティブを持つ．マルチランデブを用いることで，システムの仕様を機能的に独立したモジュールの集合として記述することが可能になり（制約指向あるいは資源指向スタイル[9]と呼ばれる），システムの保守，機能変更などが容易になる．

† 大阪大学基礎工学部情報科学科
Department of Information and Computer Sciences, Osaka University
†† 滋賀大学経済学部情報管理学科
Faculty of Economics, Shiga University
††† 大阪市立大学学術情報総合センター
Media Center, Osaka City University

通信プロトコルや分散システムでは，実行効率の向上や応答時間の短縮を目的に，しばしば並列処理が用いられる．そのためには，通信チャネルのバッファ制御機構や並行モジュール間のタスク割当て機構など込み入ったモジュールの設計が必要となる．しかし，設計段階でのラピッドプロトタイピングでは，機能の変更が頻繁に行われることが予想されるため，これらのモジュールの設計を省略し，評価用のプロトタイプを自動生成できることが望ましい．LOTOSでは，複数並行プロセス間のマルチランデブと選択，並列，割込みなどのオペレータを組み合わせて用いることによって，複数の分散端末間の排他制御やタスク割当てなどを含む分散システム仕様の記述が簡潔に行える．また，負荷分散システムなどのLOTOS仕様から，複数ノードで協調動作する目的コード群を自動生成できれば，実際のネットワークや計算機の状況（負荷など）が動的に変化した場合のシステム全体の動的振舞いの解析およびそのフィードバックにかかる工数を大幅に削減できる．

これらの目的のためには，分散システムの動作仕様をLOTOSにより記述し，実行効率の良い目的コードを自動生成できることが望ましい．これまで，いくつかのLOTOSコンパイラが提案されているが[3),6),8),10)]，これらは分散ノード間のマルチランデブによる通信をサポートしていない．ネットワーク上の複数ノード間に指定されたマルチランデブを実現するための分散アルゴリズムがいくつか提案されている[2),7)]．文献2)，7)では，各ゲートで同期するプロセスの組合せはつねに同じであるが，各プロセスは異なる複数のゲートでのランデブのいずれかに参加できる，といった，あるサブクラスのマルチランデブを実現する分散アルゴリズムが提案されているが，これらの手法は1対1通信を想定しているために，どのゲートでのランデブを選択するかを，関係するプロセス間で合意するために最悪全ノード数 n に依存する $O(n\sqrt{n})$[2)] や $O(n \log n)$[7)] といった多大なメッセージ交換が必要である．

このため，本論文では，ネットワークにおけるブロードキャスト機構の性質を利用したLOTOS仕様の効率良い分散実装法を提案する．

提案する手法では，同期するプロセスの組合せが動的に変化することを許した，一般のマルチランデブが指定可能なLOTOSを用いて分散システムの仕様を記述する．仕様には，複数の並行プロセスおよびその間の協調動作を記述できるが，個々のプロセスはいずれかのノードに割り当てられる必要がある．LOTOS仕様は，異なるノード上でメッセージ交換により協調動作する目的コード群として実現される．同一ノードに割り当てられたプロセス群は，我々が開発したLOTOSコンパイラ[10)]によってマルチスレッド化された目的コードに変換される．提案する手法では，全ノードはブロードキャストされたメッセージを同一の順序で受信するという仮定の下，受信メッセージの順序と内容から先に実行可能となったランデブを選択し実行する．

一般に，LOTOSでは，オペレータで指定されたプロセス間の関係（以後**実行関係**と呼ぶ．図3参照）がプロセスの生成，消滅によって動的に変化し，それにより，同期するプロセスの組合せも状況に応じて変わる．このような一般のマルチランデブを扱うためには，各目的コードは分散プロセス間の最新の実行関係を知る必要がある．そのために各目的コードは，LOTOSの構文木で表されるプロセス間の実行関係に関する情報を保持している．そして，つねに最新の実行関係を保持するために，あるノードでプロセスが生成・消滅した場合には，これを通知するメッセージがブロードキャストされ，すべてのノードで情報の更新が行われる．

予備実験から，イーサネット上でのUDPを用いたブロードキャストでは，メッセージの送信頻度が高い場合にはメッセージの消失やメッセージ順序の逆転が起こることが分かった．本論文では，これらの問題に対して，ネットワーク上に代理サーバを設置することで解決した．

実験から，いくつかの分散システム仕様が，提案する手法により効率良く実現されることを確認した．

2. 対象とするモデル

2.1 LOTOSの概要

LOTOSでは，システムの仕様をいくつかのプロセスからなる並行プロセスとして記述する．各プロセスの動作は**動作式**と呼ばれ，プロセス外部（環境）から観測可能なアクションである**イベント**，観測不可能な**内部イベント**，あるいはプロセス呼び出しの実行系列として定義される．ここでイベントは，環境との相互作用（データの入出力）であり，ゲートと呼ばれる作用点で発生する．イベント間の実行順序を指定するため，連接 $(a;B)$，選択 $(B1[]B2)$，同期並列 $(B1|[G]|B2)$，非同期並列 $(B1|||B2)$，割込み $(B1[>B2)$，逐次 $(B1>>B2)$ などのオペレータが任意の部分動作式間に指定される．特に，同期並列オペレータを使用することにより，複数のプロセスが指定されたゲート上のイベントを同時に実行しデータ交

表1 記述のクラス
Table 1 Syntax of our description language.

B ::= GB | LB | $B >> B$
GB::= $[boolean\ exp]- > GB$ | $GB[]GB$ | $GB|||GB$ | $GB|[G]|GB$ | $GB||GB$ | $GB[> GB$
| $GB >> GB$ | $p[G][E]$ $(*|node\ name|*)$ | $p'[G][E]$
LB::= $stop$ | $exit$ | $Action; LB$ | $[boolean\ exp]- > LB$ | $LB[]LB$ | $LB|||LB$ | $LB|[G]|LB$
| $LB||LB$ | $LB[> LB$ | $LB >> LB$ | $p'[G](E)$ | $hide\ G\ in\ LB$ | let 代入文 in
$Action$ ::= $gate$ | $gate\ IOList$ | $gate\ IOList\ [boolean\ exp]$ | i
$IOList$ ::= IO | $IO\ IOList$
IO ::= $?var : sort$ | $!exp$
G ::= $gate$ | $gate, G$
exp ::= (* ACT ONE で書かれた任意の式 *)
E ::= exp | exp, E

(i は内部イベントを表す．p' は同一ノードにおけるプロセスの呼び出し，$p(*|node_j|*)$ は他のノード $node_j$ に対するプロセスの呼び出しを表す．また，$p := B\ |||\ p$ のような無限プロセスを引き起こす記述は許さない．)

換を行う，といった動作を記述することができる（**マルチランデブ**と呼ばれる）．n 個のプロセスがゲートの集合 G についてマルチランデブするよう指定されている場合 $(p_1|[G]|\ldots|[G]|p_n)$，イベントを同期実行できるのは，これら n 個の全プロセスが $g \in G$ である同一ゲート g のイベントを実行可能であり，かつ，このうちの任意の2つのプロセス p_i, p_j のイベントの入出力値が次の生起条件を満たすときに限る（ただし，G に属さないイベントについては，$|||$ の場合と同様，各プロセスで独立に実行される）．

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

ある時点で，上記のマルチランデブの生起条件を満たすイベント組を実行可能なプロセスの組合せが複数ある場合は，それらの中から非決定的に1つのイベント組が選択され，実行されなければならない．たとえば，**図1**の例では，イベントb，c，aが (P,Q)，(Q,R)，(R,P) の組合せで実行可能であり，そのうちの1つがプロセス間の合意のもとで，一意に決定されなければならない．

2.2 クラスとプロセスのノードへの割当て

本節では，分散システムの仕様を記述するLOTOSのあるサブクラスおよび仕様の分散システムのノード群への割り当て方を定義する．

$M = \{node_1, \ldots, node_n\}$ を分散システムにおける全ノードの集合とする．また，L_j をローカル/内部ゲート（すなわち，$node_j$ 内のプロセス間や環境との同期，および，$node_j$ 内のプロセスの内部イベントのための作用点）の集合，S_j を他のノード上のプロセスと同期するためのゲートの集合，$A_j = L_j \cup S_j$ を

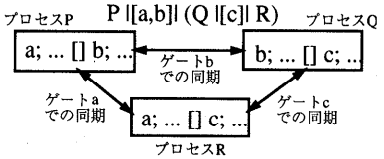


図1 相互排他的マルチランデブ
Fig. 1 Mutually exclusive multi-rendezvous.

$node_j$ 上の全ゲートの集合とする．

$P = \{p_1, p_2, \ldots, p_m\}$ をあるLOTOS仕様中で定義されている全プロセスの集合とする（p_1 は主動作式に相当する）．提案する手法では，各プロセス $p_k \in P$ に対するプロセス呼び出しをあるノード $node_j$ に割り当てることができる（$p_k(*|node_j|*)$ と指定する）．

各 $p_k \in P$ の動作式は，任意のイベント，P の要素に対するプロセス呼び出し，およびオペレータを用いて**表1**の構文で記述される．表1の生成規則から，各プロセス p_k について次の3つのタイプの動作式を記述することができる：(1) P のプロセス呼び出し（ノード割当てあり）とオペレータのみからなる**グローバル動作式**（GB），(2) A_j 中のイベントとローカルなプロセス呼び出し（ノード割当てなし）からなる**ローカル動作式**（LB），(3) 両タイプの動作式 GB と LB を $>>$ のみで合成した動作式．

ここで，$P(node_j) = \{p_{j,1}, \cdots, p_{j,l}\}$ をノード $node_j$ に割り当てられたプロセス呼び出しの集合とすると，各プロセス呼び出し $p_{j,k}$ の動作式中で用いられる実ゲートは A_j のものに限られるよう記述する必要がある．

表1では，ノード間の実行関係と各ノード内のローカルプロセス間の実行関係を GB と LB として分離して定義するようにしている．これは，各ノードの動

作仕様を別々に設計・開発し，後にそれらの間にマルチランデブによる協調動作を追加するといった，制約（資源）指向記述スタイルによるシステムの開発を容易にする．また，実装の際には，ノード間およびノード内におけるプロセス間の制御機構を分離して実装できるため，ノード内の制御機構に既存の単一ノード用のLOTOSコンパイラの技術を用いることができる．

2.3 マルチランデブを用いた分散システムの記述例

例1．並列多重化システム

あるサービスの高信頼化のために多重化を行う場合，サービスプロセスの誤り率がきわめて低いという条件下では，複数のプロセスに同じ処理をさせてその結果の多数決をとるという方法が一般的である．

ここではタスクの要求を $req?task$，またタスクを処理した後の結果の出力を $line!ProccssTask(task)$ で表すとすると，サービスプロセスは以下のように記述できる（ここで，$ProcessTask(task)$ は $task$ から結果を計算する抽象データ型の関数）．

$$S[req,line] := req?task;line!ProcessTask(task);\ S[req,line]$$

次に，*Checker* と呼ばれるもう1つのプロセスを記述する．*Checker* は多重化されたプロセス群から結果を集め，多数決に基づいてエラーを検出し，正しい結果を出力する．*Checker* はマルチランデブにより3つのプロセスから結果をそれぞれ x，y，z として受信する．もし，x，y，z のうち2つ以上の結果が等しければ（$Correct(x,y,z)$），*Checker* はその結果を出力し（$out!Major(x,y,z)$）処理を続ける．そうでない場合には，イベント er を実行し全プロセスを停止させる．*Checker* は次のように記述できる：

$$\begin{aligned}&Checker[ln1,ln2,ln3,out,er] :=\\ &\quad (ln1?x;exit\ |||\ ln2?y;exit\ |||\ ln3?z;exit)\\ &\quad >>\\ &\quad ([Correct(x,y,z)]->\\ &\qquad out!Major(x,y,z);Checker[ln1,ln2,ln3,out,er]\\ &\quad []\ [not\ Correct(x,y,z)]->er;stop)\end{aligned}$$

システム全体は次のように記述できる．非同期並列指定された3つの多重化プロセス S をノード1，2，3上に生成する．各プロセスは $ln1$，$ln2$，$ln3$ の3つのゲートを介して *Checker* に結果を送る．

$$\begin{aligned}&(\ (S[req,ln1](*|node_1|*)\\ &\quad |[req]|\ S[req,ln2](*|node_2|*)\\ &\quad |[req]|\ S[req,ln3](*|node_3|*))\\ &\quad [>\ Error[er](*|node_4|*)\ )\\ &|[ln1,ln2,ln3,er]|\\ &\quad Checker[ln1,ln2,ln3,out,er](*|node_4|*)\\ &where\\ &\quad Error[er] :=\ er;stop\end{aligned}$$

一般には S と *Checker* 間のタスク割当て機構が必要となるが，LOTOSではプロセス間のマルチランデブを用いて，これらの機構を簡潔に記述できている．

例2．排他制御システム

分散システムを設計する際には，あるノード上の資源を最大 k 個のユーザプロセスが同時に使用できるといった排他制御をしばしば扱う．各ユーザプロセスが $a!lock$ 実行後に資源 A を使用し，資源 A を解放後に $a!unlock$ を実行するようにすれば，資源 A に対する制約は以下のように記述できる．

$$\begin{aligned}&R[a](usr,max_A) :=\\ &\quad ([usr < max_A]-> a!lock;R[a](usr+1,max_A))\\ &\quad []\ a!unlock;R[a](usr-1,max_A)\end{aligned}$$

（max_A は同時に資源を使用できるユーザプロセスの数を表す）

資源 A を使用するプロセス $P_1,\cdots,P_n$ が A へのアクセス以外は独立に実行できるとすると，システム全体は次のように記述できる．

$$\begin{aligned}&(P_1[a](*|node_1|*)\ |||\ P_2[a](*|node_2|*)\\ &\quad |||\ \ldots\ |||\ P_n[a](*|node_n|*))\\ &|[a]|\ R[a](0,max_A)(*|node_1|*)\end{aligned}$$

上の仕様では，プロセス R 中のイベント $a!lock$ と $a!unlock$ は $P_1,\ldots,P_n$ のいずれかと同期し，変数 usr の値を増減させる．もし $usr+1$ が max_A を超えていれば $a!unlock$ のみが実行可能となり，この場合，資源 A を使用したいプロセスは他のプロセスが $a!unlock$ を実行するのを待ち続ける．

上記のシステムを，複数の種類の資源 A，B，C を扱えるように拡張することは容易である．これらの資源に対する制約のためのプロセスを追加するだけでよい．

$$\begin{aligned}&(P_1[a,b,c]\ |||\ P_2[a,b,c]\ |||\ \ldots\ |||\ P_n[a,b,c])\\ &|[a,b,c]|\\ &\quad (R[a](0,max_A)\ |||\ R[b](0,max_B)\ |||\ R[c](0,max_C))\end{aligned}$$

（ここで，プロセスのノード名は省いている）

以上のようにLOTOSではセマフォなどの概念を用いずに排他制御を簡潔に記述することができる．

2.4 マルチランデブの実装における仮定

仕様によっては相互排他的な複数のマルチランデブが同時に実行可能となることがある．図1はどの2つの排他的なランデブも1つのプロセスを共有している最悪のケースである．マルチランデブを正しく実装するためには，どのランデブを選択するかという合意のために，異なるノード上の分散プロセス間でのメッセージ交換が必要となる．

本論文では，一般のケースのマルチランデブを効率良く実装するために，ネットワークに対して次の性質を仮定する．

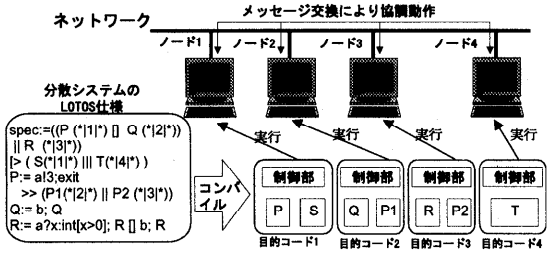


図2 分散システム実装の基本方針
Fig. 2 Basic policy to implement a distributed system.

表2 LOTOS仕様の例
Table 2 An example of LOTOS specification.

```
spec.:= (P (*|node1|*) [] Q (*|node2|*) || R (*|node3|*) )
        [> ( S (*|node1|*) ||| T (*|node4|*) )
where
    P:= (a!3;exit)
        >> ( P1 (*|node2|*) || P2 (*|node3|*) )
    Q:= b;Q
    R:= a?x:int[x>0]; R [] b;R
    S:= (S1 || S2 ) [] S3
    T:= d;stop
```

(1) 1回のブロードキャストにより全ノードに同じ情報を送信することができる．(2) メッセージは誤りなく有限時間内に各ノードに受信され，メッセージバッファに保管される．(3) ブロードキャストされたメッセージは全ノードに同じ順序で到着する．

3. アルゴリズム

3.1 実装の基本方針

提案する実装法では，表1のクラスで記述されたLOTOS仕様から分散システム内のノードに対応する目的コード群を生成する．生成された目的コード群は互いに協調して動作し，与えられたLOTOS仕様を実現する（図2）．

表2 のLOTOS仕様を例にあげると，$node_1$, $node_2$, $node_3$, $node_4$ に対応する4つの目的コードが生成され，各目的コードはそれぞれ $\{P,S\}$, $\{Q,P1\}$, $\{R,P2\}$, $\{T\}$ を個別に実行できる．これらの目的コードは実行を開始すると，まず初期動作式で実行可能なプロセス呼び出しを起動する．表2の例では，$node_1$ に対応する目的コードが実行されるとプロセス P と S が呼び出される．

プロセス間の初期の実行関係は，**図3** (i) の点線内部の領域に対応する．一般に，プロセス間の実行関係は，プロセスの生成や消滅により動的に変化する．表2の例では，(P,R) の組で $a!3$ が実行されると，プロセス Q は選択されなかったため終了し，プロセスPは $a!3$ の実行後，プロセス $P1, P2$ を呼び出す．その後，実行関係は図3の (i) から (ii) へと変化する．したがって，提案する手法では，実行中の全プロセス間の実行関係に関する最新の情報を各ノードごとに個別に保持することで，この関係に基づいたプロセス間のイベントの実行順序を実現する．

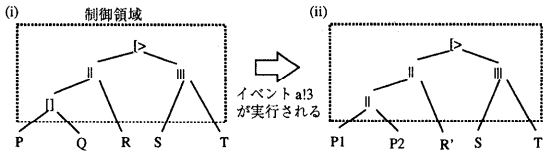


図3 実行関係の動的変化
Fig. 3 Executional dependence relation and its dynamic change.

各目的コードには**制御領域**と呼ばれる，プロセス間に指定されたオペレータの構文木と同じ構造を持つ作業領域を割り当て，初期実行関係を記憶させる．実行にともないあるプロセスが呼び出し/終了されたときには，そのプロセスは全ノードにその情報をブロードキャストし，各ノードではそれに従って制御領域の内容を更新し，最新の実行関係を保持する．

あるノードに割り当てられた各プロセス（$p(*|node_j|*)$ で与えられる）は，複数の並列に動作するローカル・サブプロセスとその間の選択，割込み，マルチランデブなどを含む．本論文では，文献10)の手法に基づいて，各プロセス内で実行可能かつ他のプロセスとの同期が必要なイベント（以後，**同期イベント**と呼ぶ）を他ノードと独立に計算し，それらの同期要求メッセージをブロードキャストする．そして，各プロセスは現在の実行関係と受信バッファ内のメッセージの順序から他のランデブより先に2.1節の同期条件を満たしたランデブから優先的に選択，実行する．

3.2 プロセス間の実行制御

各プロセスは同期イベントの集合 $\{a_1,\cdots,a_k\}$ を計算し（そのプロセスでは，同期イベントの集合内のいずれかのイベントが実行可能である），同期要求メッセージ $req(\{a_1,\cdots,a_k\})$ をブロードキャストする．現在の実行関係，および受信バッファ内のメッセージの内容と順序に基づいてランデブを選択するために，各ノード上の目的コード内にメッセージ・ハンドラと呼ばれる，全ノードからブロードキャストされたメッセージを処理し実行可能なイベントを決定するスレッドを新たに実装した．各プロセスはメッセージ・ハンドラが実行を許可したイベントのみ実行できる．**図4** は表2の例の $node_1$ の実行環境を表している．

ここで以下の点が実装上のポイントとなる：(1) プ

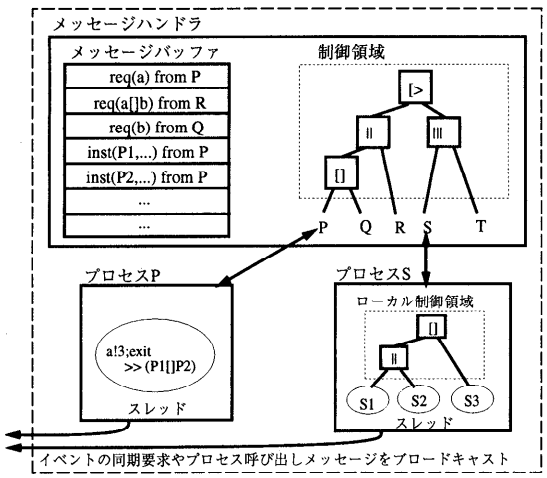


図4 各ノードの実行環境
Fig. 4 Environment of each node.

ロセス間の選択と割込みオペレータを正しく実現する，(2) 排他的なランデブが複数実行可能な場合に，そのうちの1つを全ノードでの合意のもとに選択する，(3) 実行関係の動的な変化を全ノード間で正しく扱う．

上記 (1) は次の方針に従って実装する：あるプロセスのインスタンス p がイベントを実行したい場合，p はこのイベントの同期要求メッセージにこのプロセスのインスタンスの識別子 $ID(p)$ を付けてブロードキャストする．各ノードのメッセージ・ハンドラがこのメッセージを処理する際，**制御パス**と呼ばれる，制御領域の p の位置（葉）から根までのパスを調べて，パス上の各オペレータの反対側の動作式が選択されていないかどうかをチェックする．すべてのオペレータに関して反対側の動作式が選択されていなければこのメッセージを受理し，'p' の側が選択されたことを示す情報を制御パス上の領域に格納する．あるオペレータにおいて，反対側のプロセスが選択されている場合は，メッセージを破棄し，それをブロードキャストしたプロセスを消滅させる．

(2) に関しては次のように実装する：各メッセージ・ハンドラが同期要求メッセージ m を処理する際，m の送信元プロセスの制御パスの葉に最寄りの $||[G]||$ オペレータに相当する領域に同期イベントについての情報を格納する．メッセージ・ハンドラが，この領域で同期条件を満たす適当な同期相手を見つけることができれば，未定義変数に適当な値を代入した後の新たなイベントの値を用いて同じ操作を制御パス中のより上位の $||[G]||$ オペレータに対して適用する．制御パス中により上位の $||[G]||$ がなくなれば，メッセージ・ハンドラはこのイベントの実行を許可する．メッセージの全順序が保証されているため，メッセージを到着順に逐次処理することで全ノードで一意にランデブを選択できる．

(3) は次のように実装する：あるノード上のプロセスでプロセス呼び出しが実行可能となると，このノードのメッセージ・ハンドラは呼び出されるプロセスの制御パスを計算し，この情報を含んだプロセス呼び出しの要求メッセージをブロードキャストする．各ノードのメッセージ・ハンドラは，このメッセージを処理したときに，制御領域にメッセージ中の制御パスを追加し，さらにプロセスの呼び出しが指定されているノードでは実際にプロセスを生成する．

以上について表2の例を用いて説明する．同期要求メッセージ $req(\{a!3\})$，$req(\{a?x : int[x > 0], b\})$，$req(\{b\})$ がそれぞれプロセス P，R，Q からこの順でブロードキャストされるとする（図4参照）．各メッセージ・ハンドラはプロセス P からの $req(\{a!3\})$ を処理すると，$a!3$ の情報を制御パス中の $||$ に対応する領域に登録する（$||$ はすべてのゲートでの同期を表すオペレータ）．次の R からのメッセージ $req(\{a?x : int[x > 0], b\})$ を処理したとき，P と R の間でランデブが $a!3$ で同期条件を満たし，かつ P も R も制御パス中で割り込まれていない．よって P と R の制御パスを更新した後，P と R でランデブ $a!3$ が選択，実行される（P 側が選択されたという情報が $[]$ にあたる領域に格納される）．その後，Q からの $req(\{b\})$ が処理されると，Q の反対側の動作式がすでに選択されていることを制御パス中の $[]$ の領域に格納された情報から知ることができる．よって Q は終了する．

一方，P, Q, R の順で同期要求がブロードキャストされた場合を考えると，R からの同期要求 $req(\{a?x : int[x > 0], b\})$ によって $a!3$ と b の2つの排他的なランデブが実行可能となる．このような場合にノード間で一意にランデブを選択できるようにするため，ここでは乱数に基づく手法[2),7)]を導入した．同期要求メッセージ中の各イベントに乱数を割り当て，その合計が最大のランデブが選択される．

$a!3$ が P で実行されると，P の動作式は $P1||P2$ に変化し，プロセス呼び出し要求メッセージ $inst$ $(P1, node_2, P1$ の制御パス) と $inst$ $(P2, node_3, P2$ の制御パス) がブロードキャストされる．

3.3 アルゴリズムの正しさ

ここでは，提案するアルゴリズムの正しさと効率について簡単に説明する．

プロセス間で実行可能なイベントは，現在のプロセ

ス間の実行関係および各プロセス中で実行したい同期イベントの集合に依存する．すべてのノードが実行関係に関する最新の情報を得ることができ，また，各ノードがこの関係と各プロセス中の同期イベントの集合に基づいて，あるプロセス（の組合せ）のあるイベント（ランデブ）を一意に求めることができれば，そのアルゴリズムは正しい．

本アルゴリズムでは，プロセス間に選択や割込みオペレータが指定されていれば，最も早く同期要求を送信したプロセスがノード間で一意に選択される．同期オペレータについては，メッセージを先に送信したプロセスの組合せが一意に選択される．提案する手法では，2.4節で説明したように受信メッセージの順序と内容が全ノードで等しく，全ノードは同一のアルゴリズムに基づいて動作する．これは，ローカルバッファ内のメッセージを逐次処理することで，各ノードはどのプロセスが選択あるいは生成されたかを知ることができ，よってつねに最新の実行関係が維持される．以上より本アルゴリズムは正しく動作する．

4. 実装と評価

4.1 イーサネット上での実装

動作式が表1の *LB* として記述される各ローカルプロセスは，我々が開発したLOTOSコンパイラ[10]によってマルチスレッド化目的コードとして実現される．並行サブプロセスを含む各プロセスは我々が開発した移植性に優れたマルチスレッド機構であるPTL[1]を使用して次のように実現される：(1) 各プロセスの動作式を**基本動作式**と呼ばれる，逐次的に実行可能な部分動作式群に分解し，(2) 各基本動作式を1つのスレッドにマッピングし，(3) 共有変数領域（**ローカル制御領域**と呼ばれる）を構築する．

広いクラスの動作式を実装するために，3.2節で説明されているのと同じ手法を用いてローカル制御領域を操作し，スレッド間の実行関係を制御する．ローカル制御領域中の各領域が複数の並行スレッドから同時にアクセスされるのを防ぐため，PTLの持つ排他制御機構を使用する．上記の手法により，本コンパイラは文献3)，6)などで提案されている他のコンパイラより実行効率の高い目的コードを生成することができる（詳細は文献10)参照）．

我々は3章のアルゴリズムに従って，イーサネット上の対応するノードで動作する目的コード群を生成するようLOTOSコンパイラを拡張した．

イーサーネット上でのアルゴリズムの実装

一般のLAN環境ではイーサネットが広く普及しているが，イーサネットは2.4節の仮定を完全には満たさない．そこで，まずどの範囲でイーサネットが前述の仮定を満たすかについて実験を行った．ここではファーストイーサネット（100BASE-TX）のネットワークで結合された複数のUNIXワークステーション（BSD/OS 2.1が動作するPentium 90からPentium Pro 200 MHz）上でUDPプロトコルを使用した．

実験から，以下の問題が起こることが分かった．(i) メッセージ交換の頻度が高い（本実験では，1秒あたり3000メッセージ以上）場合に，いくつかのノードで数パーセントのメッセージ消失がある．(ii) 複数のノードからメッセージが短い時間間隔でブロードキャストされると，送信側ノードでメッセージの到着順序が入れ替わることがある．

(i) はメッセージ処理の時間が不足するために遅い計算機のメッセージバッファがオーバフローすることが原因であり，(ii) はオペレーティングシステムのループバック機構の仕様が原因である．実験に使用した環境では，メッセージをブロードキャストしたノードに対してはメッセージが直接メッセージバッファに格納されてしまうため，他のノードよりも早いタイミングで受信してしまうことになる．

ここでは，効率の点から，各ノードから送られたメッセージを代わりにブロードキャストする代理サーバを設置する方法を採用し，イーサネットで前述の仮定を満たせるようにした．各ノードはメッセージを直接はブロードキャストせず，サーバにブロードキャストを依頼する．独立したノード上にサーバを設置することで，分散システム中の全ノードはメッセージを同じ順序で受信することが保証される．また，サーバ側でブロードキャストするメッセージに連続な番号を付加することで，各ノードは容易にメッセージの消失を検知することができ，この場合，サーバにTCP/IPを用いて再送を要求する．

4.2 性能評価

本実装法を評価するために，いくつかの分散システムの仕様から得られる目的コード群の実行効率に関する実験を行った．実験では，異なるノード上のプロセス間で単位時間あたり同期実行できるイベントの数を測定した．各目的コードは4.1節と同じ環境で実行した．

基本性能

はじめに，次の仕様から得られる目的コード群を実

表 3 単純同期の実行効率（イベント/秒）
Table 3 Performance of simple rendezvous.

$k\backslash n$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
$k=1$	1101	993	818	647	571
$k=2$	807	663	534	426	367
$k=3$	654	481	370	319	257
$k=4$	519	519	290	237	190
$k=5$	428	428	225	184	139

表 4 排他制御システムの実行効率（回/秒）
Table 4 Performance of a mutual exclusion system.

$n=5$	$n=10$	$n=15$	$n=20$	$n=25$
374	374	362	358	343

行した．この仕様では各イベントが n 個のノード上のプロセス間で繰り返し同期実行される．実験結果を**表 3** に示す（ただし，図中の $n=1$ はプロセス間の同期が指定されていない場合である）．

$$P(*|node_1|*)||P(*|node_2|*)||...||P(*|node_n|*)$$
where
$$P := a_1; P[]a_2; P[]...[]a_k; P$$

表 3 によると，実行効率はプロセス数 n と選択イベント数 k の両方に依存していることが分かる．前者は各ランデブを構成するプロセス数が増加することにより，処理すべき同期要求メッセージの数が増えるためであり，後者は選択肢の増加により，同期条件をチェックすべきランデブの数が増えるためである．

分散システムの実行効率

次に，より複雑な分散システムの実行効率について実験を行った．

まず，2.3 節で述べた，n 個の並行分散プロセスが相互排他的に資源を使用する排他制御システムの実行効率を測定した．n の値を変化させて，単位時間あたりユーザプロセスが資源を使用した合計回数を測定した結果を**表 4** に示す．表 4 から実行効率はユーザプロセス数によらないことが分かる．これは，我々のアルゴリズムでは，n_k 個のプロセスからなるランデブを n_k 個のメッセージにより決定でき，全体のプロセス数 n に依存しないためである（ここで，選択されなかったランデブの同期要求メッセージは，要求を出したプロセスが消滅しない限りは，その後のランデブ決定のために使用される）．

最後に，アブラカダブラ・プロトコルに基づくファイル転送システムの実行効率について実験を行った．アブラカダブラプロトコルの LOTOS 仕様[5] には，あるノードがユーザの要求に従って，指定されたファイルを他のノードへ転送する動作および，他のノードからファイルを受信し，ユーザに掲示する動作が規定されている．このプロトコルではファイルは複数のデータパケットに分割され，各パケットはパケット消失の起こりうる通信路を介して転送される．実験では，送信ノード（*Sndr*）が通信路の振舞いを模倣するプロセス *Medium* と同期しながら 1 M バイトのファイルを n 個の受信ノード（*Rcvr*）にいっせいに配信するよう仕様を拡張した．ここで，各ノードに割り当てられるプロセス *Sndr* および *Rcvr* は，6 個の互いに同期する並行サブプロセスおよびいくつかの割込みプロセスを含む．

$$Medium[m_1,m_2](*|node_1|*)$$
$$|[m_1,m_2]|$$
$$(Sndr[a_1,m_1](*|node_2|*) \;|||\; Rcvr[a_2,m_2](*|node_3|*))$$
$$... \;|||\; Rcvr[a_2,m_2](*|node_{n+2}|*))$$

データパケット長を 256 から 1024 バイトまで，また n を 1 から 5 まで変化させて実験を行った結果を**表 5** に示す．

表 5 アブラカダブラプロトコルの実行効率（K バイト/秒）
Table 5 Efficiency in abracadabra protocol (Kbyte/sec).

パケット長	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
256 バイト	13.6	11.4	10.6	8.3	7.1
512 バイト	26.3	22.0	19.9	15.9	12.8
1024 バイト	49.5	44.1	23.6	24.4	16.6

表 5 では，パケット長が小さい場合には受信ノード数が増えるにつれて転送レートは緩やかに落ちている．対照的にパケット長が大きい場合は転送レートが急激に落ちる．これは，パケット長増加にともない，各ノードが短時間に受信するデータ量が増え，メッセージ処理のオーバヘッドが増加したためと考えられる．

5. おわりに

本稿では，複数の分散プロセス間における一般のマルチランデブを含む仕様をブロードキャストが可能なネットワーク上の複数ノードで実行できる目的コード群として実装する手法を与えた．実験により，1 秒間に数百回程度，マルチランデブによるノード間の通信が可能であり，複数ユーザ間による共有資源の排他制御システムなどの用途には，実用に耐えうる目的コードを生成可能であることが分かった．

本手法の適用例の 1 つとして，分散システムの動的な振舞いの可視化があげられる．文献 11) の可視化手法を用いることで，分散システムを構成するプロセスの動的な振舞いを，グラフィカルなアニメーションとしてリアルタイムでモニタすることができる．実際の

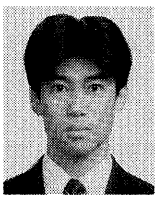
ネットワークや計算機の負荷に応じて，使用する計算機資源を調整するようなシステム（QoSシステムなど）の動作解析に，本手法は適用可能と考えられる．本手法により得られる目的コード群を異なる複数ノードで実行することにより，ネットワークおよび特定の計算機の負荷を実際に変化させ，その影響を調べることができる．

今後の課題は，時間拡張LOTOS（e.g. E-LOTOS）を扱えるように本コンパイラを拡張することである．

参考文献

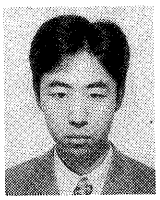
1) 安倍広多，松浦敏雄，谷口健一：BSD UNIX上での移植性に優れた軽量プロセス機構の実現，情報処理学会論文誌，Vol.36, No.2, pp.296–303 (1995).
2) Cheng, Z., Huang, T. and Shiratori, N.: A New Distributed Algorithm for Implementation of LOTOS Multi-Rendezvous, *Proc. 7th Int. Conf. on Formal Description Techniques*, pp.493–504 (1994).
3) Dubuis, E.: An Algorithm for Translating LOTOS Behavior Expressions into Automata and Ports, *Proc. 2nd Int. Conf. on Formal Description Techniques*, pp.163–177 (1990).
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) ISO/IEC/TR 10167: Information Technology – Open Systems Interconnection – Guidelines for the application of Estelle, LOTOS and SDL (1991).
6) Manas, J.A., Salvachia, J.: $\Lambda\beta$: A Virtual LOTOS Machine, *Proc. 4th Int. Conf. on Formal Description Techniques*, pp.445–460 (1991).
7) Naik, K.: Distributed Implementation of Multi-rendezvous in LOTOS Using the Orthogonal Communication Structure in Linda, *Proc. 15th Int. Conf. on Distributed Computing Systems*, pp.518–525 (1995).
8) 野村眞吾，瀧塚孝志，長谷川亨：LOTOSによる仕様からの自動実装方式，信学論，Vol.J76-D-I, No.11, pp.575–584 (1993).
9) Vissers, C.A., Scollo, G. and Sinderen, M.v.: Architecture and Specification Style in Formal Descriptions of Distributed Systems, *Proc. 8th Int. Symp. on Protocol Specification, Testing, and Verification*, pp.189–204 (1988).
10) 安本慶一，安倍広多，後藤和裕，東野輝夫，松浦敏雄，谷口健一：マルチスレッド化目的コードを生成するLOTOSコンパイラの実現，情報処理学会論文誌，Vol.39, No.2, pp.283–292 (1998).
11) 安本慶一，東野輝夫，松浦敏雄，谷口健一：マルチランデブを用いたLOTOS仕様の実行の可視化，情報処理学会論文誌，Vol.37, No.5, pp.687–697 (1996).

(平成10年5月7日受付)
(平成10年10月2日採録)



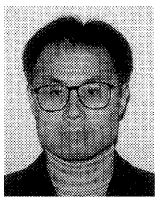
辰本比呂記

平成9年大阪大学基礎工学部情報工学科卒業．現在同大学大学院博士前期課程在学中．並行処理系，マルチメディアシステムの実現法に興味を持つ．



後藤　和裕

平成9年大阪大学大学院基礎工学研究科（情報工専攻）博士前期課程修了．同年ソニー（株）入社．並行言語処理系，通信システムの実装法等の研究に従事．

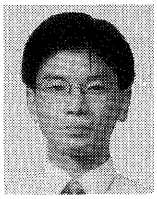


安本　慶一（正会員）

平成3年大阪大学基礎工学部情報工学科卒業．平成7年同大学大学院博士後期課程退学後，滋賀大学経済学部助手．現在，同大学助教授．工学博士．平成9年モントリオール大学客員研究員．通信プロトコルや分散システムの形式仕様記述・実装法に関する研究に従事．

東野　輝夫（正会員）

昭和54年大阪大学基礎工学部情報工学科卒業．昭和59年同大学大学院博士課程修了．同年同大学助手．平成2，6年モントリオール大学客員研究員．現在，同大学大学院基礎工学研究科助教授，工学博士．分散システム，通信プロトコル等の研究に従事．電子情報通信学会，ACM各会員．IEEE Senior Member．



安倍　広多（正会員）

平成4年大阪大学基礎工学部情報工学科卒業．平成6年同大大学院博士前期課程修了．同年NTT入社．平成8年大阪市立大学助手．マルチスレッド機構の実装，マルチメディアアプリケーションの設計等に興味を持つ．電子情報通信学会会員．

松浦　敏雄（正会員）

昭和50年大阪大学基礎工学部情報工学科卒業．昭54年同大大学院基礎工学研究科（情報工専攻）博士後期課程退学後，同大助手．平4年同大情報処理教育センター助教授，平7年大阪市立大学教授．工学博士．ユーザインタフェース，マルチメディア，情報教育等に興味を持つ．ACM，IEEE，電子情報通信学会等会員．

谷口　健一（正会員）

昭和40年大阪大学工学部電子工学科卒業．昭和45年同大大学院博士課程修了．同年同大助手．現在，同大大学院基礎工学研究科教授．工学博士．この間，計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法および支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法等に関する研究に従事．