時間拡張LOTOSコンパイラの作成とマルチメディアアプリケーションへの応用

辰本 比呂記[†] 安倍広多[††] 安本慶一[†††]
東野輝夫[†] 松浦敏雄[††]
山口弘純[†] 谷口健一[†]

本論文では，形式記述言語 E-LOTOS の時間制御構文を従来の LOTOS に取り入れた言語（時間拡張 LOTOS と呼ぶ）で記述された実時間システム仕様を実時間スレッド機構を用いて実装するためのコンパイラを作成する．本コンパイラは，与えられた動作仕様をイベント列とその選択・反復から成る部分動作式に分割し，それぞれを我々が開発した実時間スレッド機構の 1 スレッドに割り当て，マルチランデブなどのスレッド間のインタラクションを共有変数領域を用いて実現する．イベントができるだけ時間制約内に実行されるよう，各スレッドは対応する処理にデッドラインを設定し，早いデッドラインを持つスレッドを優先してスケジュールする．また，時間制約付イベント間のマルチランデブを効率良く実行するための機構や，マルチメディアへの応用のため，動画や音声処理を時間拡張 LOTOS 仕様から利用するための機構を考案，実装した．動画再生アプリケーションの記述・実装実験より，記述工数の面で有利だが同期のオーバヘッドを要する制約指向スタイルの時間拡張 LOTOS 仕様から，本コンパイラにより実用的なスピードで動作する目的コードが生成可能なことを確認した．

Development of Realtime LOTOS Compiler and Its Application to Multimedia Systems

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

In this paper, we develop a compiler for realtime systems described in a subclass of E-LOTOS (called realtime LOTOS). The compiler first decomposes a given specification into multiple basic modules consisting of event sequences and their choices and iterations, then it maps each module to a realtime thread on our realtime thread library. To make the best effort for each thread to execute its timed events in time, all the running threads set their deadlines so that they are scheduled based on EDF (Earliest Deadline First) policy. We have designed and implemented mechanisms to efficiently schedule timed multiway synchronization among threads, and to utilize audio/video data in realtime LOTOS specifications for multimedia system development. Through some experiments, we have confirmed that from realtime LOTOS specifications in constraint oriented style (which introduces synchronization overhead in spite of its descriptive facility), the compiler can generate efficient programs practically usable.

1. はじめに

近年のネットワークの発展，普及にともない，動画や音声などの実時間性を持ったマルチメディア情報を QoS を保証しつつユーザに提供するためのフレームワークや制御機構がさかんに研究されている[13]．また，形式仕様記述言語を用いてマルチメディアアプリケーションを記述し，システムの検証や性能解析，さらにはプロトタイプの自動生成を行おうとする研究がいくつか提案されている．文献 5) では，ISO で標準化されている形式記述言語 Estelle に時相論理を組み込んだ言語を用いたマルチメディアシステムの仕様記述法や，その仕様を実時間 OS 上のマルチスレッド機構を用いて実装する手法が提案されている．

もう 1 つの ISO 標準 LOTOS[6] は，並行，割込み，

† 大阪大学大学院基礎工学研究科情報数理系専攻
Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University
†† 大阪市立大学学術情報総合センター
Media Center, Osaka City University
††† 滋賀大学経済学部情報管理学科
Faculty of Economics, Shiga University

およびマルチランデブ☆などの高度な制御機構を持つ．現在までに，イベントに時間制約（実行可能な時刻の範囲）が指定できるよう拡張された LOTOS を用いて，複雑な実時間/マルチメディアシステムを効率良く記述し，動作の正しさの検証や性能解析を行うための研究が行われてきた[4),8),9)]．これらの成果は，E-LOTOS[7)] として標準化作業が行われている．

本論文では，E-LOTOS の持つ時間を扱う構文およびいくつかのフロー制御構文を従来の LOTOS に取り入れた言語（**時間拡張 LOTOS** と呼ぶ）を対象としたコンパイラの作成とその応用について述べる．

時間拡張 LOTOS 仕様は，各プロセスの生成時刻があらかじめ予測できない，プロセスの動作が周期的であるとは限らない，などの特徴を持つため，各処理のデッドラインの動的設定・変更に適した EDF（Earliest Deadline First）スケジューリングを組み込んだ実時間スレッド機構 RT-PTL を開発し[2),3)]，これを用いて我々の LOTOS コンパイラ[17)] を時間拡張 LOTOS 仕様が扱えるよう拡張した．

作成したコンパイラは，文献 17) の手法に基づき，与えられた動作仕様をイベントの逐次処理・選択・その反復から成る部分動作式に分割し，各部分動作式を RT-PTL の 1 つのスレッドに割り当て，スレッド間の共有変数領域を用いて，マルチランデブなどの並列プロセス間におけるインタラクションを実現する．各スレッドはイベントの時間制約から実行開始時刻，デッドラインを計算，設定し，全体として早いデッドラインを設定したスレッドが先にスケジュールされるよう制御する．時間制約が指定されたイベント間のマルチランデブでは，それらのイベントが時間制約をすべて満たすようにスレッドを制御する機構や，複数のマルチランデブを時間制約に従って優先的にスケジュールする機構などを考案し，実装した．また，動画の各フレームや指定した時間分の音声データの読み込み，デコード，再生などを行うプリミティブを作成し，各プリミティブを時間拡張 LOTOS 仕様中にイベントとして記述し利用できる機構をコンパイラに実装した．

時間拡張 LOTOS では，システムをイベントとその実行順序のみを指定した主プロセスと，それらのイベントの実行時間間隔に対する制約（および代替処理）などを指定した別プロセスから構成し，関連するイベントを同期実行させるという制約指向記述スタイル[15)] が利用できる．本論文では，この方法で，典型的なマルチメディアアプリケーションの記述を試み，QoS 制御や複数メディア間の同期制御の追加，変更が容易であることを示す．また，制約指向スタイルでは，プロセス間のマルチランデブを実現するためのオーバヘッドが問題になりうるが，複数の動画を同時に再生するアプリケーションを，制約指向による時間拡張 LOTOS 仕様および実時間スレッドを用いた C 言語のプログラムとしてそれぞれ記述・実装し，得られたコードの実行効率を比較した結果，C コードと比べてそれほど遜色ない実行効率が達成できることを確認した．

☆ 複数並行モジュール間で指定された条件が成立するときに同期通信によりデータ交換を行う機構[6)]．

2. 時間拡張 LOTOS とその実現方針

2.1 時間拡張 LOTOS

本論文では，LOTOS に E-LOTOS で拡張された構文の一部を追加した言語（**時間拡張 LOTOS**）を定義し，それを用いてシステムを記述する．

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）．

時間拡張 LOTOS は**表 1** のような構文で定義され，従来の LOTOS に加え，時間に関する構文，繰返し，値の再代入が可能な変数などの機構を有する．

以下は，新たに追加された構文とその意味である．

- **loop** B **endloop** ··· 動作式 B を繰り返し実行
- **while** E **do** B **endwhile** ··· 式 E が真の間，動作式 B を繰り返し実行する
- **var** V{,V}* **in** B **endvar** ··· 動作式 B 中で使用する変数群（書換可能）を宣言する
- a@?t ··· イベント a の実行時刻を変数 t に取得
- a@?$t[p(t)]$ ··· イベント a は $p(t)$ を満たす時刻 t にのみ実行できる
- a@!T ··· イベント a は時刻 T にのみ実行できる
- **wait**(d) ··· d 単位時間待つ

表 1 時間拡張 LOTOS の記述のクラス
Table 1 Syntax of real-time LOTOS.

BHEXP ::= **hide** GATELIST **in** BHEXP | **let** LETDCL **in** BHEXP | **var** VARDCL **in** BHEXP **endvar**
| **while** *exp* **do** BHEXP **endwhile** | **loop** BHEXP **endloop** | BHEXP '[]' BHEXP | BHEXP '|||' BHEXP
| BHEXP '>>' BHEXP | BHEXP '[>' BHEXP | BHEXP '|'['[' GATELIST ']']'|' BHEXP | **exit** | **stop**
| **wait**'('*exp*')' | '['*boolexp*']'− > BHEXP | *proc* ['[' GATELIST ']'] ['('EXPLIST')'] | '?'*variable*':='*exp*
| *gate* IOLIST ['['*boolexp*']'] ';' BHEXP | *i*
LETDCL ::= *variable* ':' *sort* '=' *exp* | *variable* ':' *sort* '=' *exp*, LETDCL
VARDCL ::= *variable* {',' *variable*}* ':' *sort* | *variable* {',' *variable*}* ':' *sort*, VARDCL
IOLIST ::= {['@']'?'*variable* : *sort* | ['@']'!'*exp*}
GATELIST ::= *gate* | *gate* ',' GATELIST
EXPLIST ::= *exp* | *exp*, EXPLIST
' ' で囲まれた文字列は終端記号を表す．*gate*，*proc*，*variable* はそれぞれゲート名，プロセス名，変数名を表す．*exp* は ADT で記述された式である（ただし，整数，文字列，ブール，リスト型データに対する各種演算と C 言語で記述可能な任意の関数に限る[17]）．

時刻は各イベントが**アクティブ**☆になった時刻からの相対時刻であり，実数で表す（E-LOTOS では有理数あるいは離散数で表す[7]）．各イベントはその時間制約が満たされる時刻に，実行されることが可能である（実行されるかどうかは環境とのインタラクションによる）．すべてのプロセスにおける時間の進行速度は同じで，1.0 は 1 秒に相当する．プロセス呼び出しに要する時間は考慮しない☆☆．

時間制約に関する制限事項 時間拡張 LOTOS では，イベントの時間制約を表すガード式は，$C_1 \leq t \leq C_2$ のような 1 つの連続した時間範囲になるものに制限している．ただし，C_1，C_2 は変数を含む任意の式を許し，不等式の論理積も指定可能である．また，$a@?t[(C_1 \leq t \leq C_2)\ or\ (C_3 \leq t \leq C_4)]$ のような論理和の式は $a@?t_1[C_1 \leq t_1 \leq C2]$ [] $a@?t_2[C_3 \leq t_2 \leq C_4]$ として記述可能である．

時間拡張 LOTOS で新しく拡張された構文を用いた例を以下に示す．

```
loop
  var x,y:time in
    (a@?x:time; b@?y:time[x+y<10])
  [> wait(10)
  endvar
endloop
```

ここでは，イベント a が実行された時刻を変数 x に取り込むことによって，$a;b$ というイベント系列が，a が実行可能になってから 10 秒までに実行されなければならないことを指定している．また，10 秒までに，これらのイベントの実行が終わらなかった場合，wait(10) による割込みにより，イベントの実行はキャンセルされるよう指定されている．以上が繰り返し行われるが，変数 x，y はループ中何度も書き換えられるため，var 文で宣言されている．

2.2 時間拡張 LOTOS 仕様実現の基本方針

時間拡張 LOTOS 仕様実現の基本方針を以下に示す．

(1) 時間制約を満たさないイベントは実行しない．
(2) 並列処理が複数あるときは，できるだけ多くの処理が制約を満たせるよう制御する．
(3) 複数イベントが実行可能になった場合，時間制約上のデッドラインが最も早いものを優先実行する．

上記の (1) は時間拡張 LOTOS の意味定義に従うためであり，(2) はソフトリアルタイム方式での時間制約の充足率を高めるためである．(3) は，本論文で，時間制約上のデッドラインが遅いイベントは，デッドラインが早いイベントの代替処理と見なすためである．

3. 実時間スレッド機構

3.1 実時間処理のスケジューリング

時間拡張 LOTOS における並行プロセス群をソフトリアルタイムで実行するためのスケジューリング方式として，(1) テーブル参照方式（タスクをスケジュールする順序をあらかじめ決めておく方式），(2) 優先度制御方式〔周期の短いタスクに高い優先度を割り当てる RMS（Rate Monotonic）方式[10] や，デッドラインが早いタスクに高い優先度を与える EDF（Earliest Deadline First）方式など〕，(3) 動的計画法に基づく方式，(4) Best Effort 方式[11] などが考えられる．

時間拡張 LOTOS ではタスクが動的に生成されるため (1) の方式は使えず，ソフトリアルタイム環境ではイベントの処理に要する時間の予測が困難なため，予測時間を必要とする (3)，(4) の方式は適さない．また，時間拡張 LOTOS では非周期の処理も指定できるため，(2) のうち，周期依存の方式は利用できない．以上より EDF 方式が適当であると考え，我々が従来から

☆ 他プロセスや環境とのインタラクションにかかわらず当該プロセス内で構文上実行可能なこと．
☆☆ $a;P$ という動作式では，P の最初のイベントの時間制約は，a が実行された時刻を起点として計算される．

表 2 RT-PTL における実時間スレッドのデッドラインミス率
Table 2 Deadline-miss rates of threads.

i	0	1	2	3	4	5	6	7	8	9
ミス率（%）	1.0	1.6	1.5	1.6	1.7	1.8	1.9	2.1	2.0	2.1

開発してきた移植性に優れたマルチスレッド機構 PTL（Portable Thread Library）[1] に EDF スケジューリングを拡張した RT-PTL[2),3)] を開発した．

RT-PTL では，各スレッドは処理ブロックに対する開始時刻・デッドライン時刻を任意の時点で設定・変更できる．EDF では，開始時刻・デッドラインが設定されたスレッドの中から，開始時刻が到来しており，かつデッドラインが最も早いスレッドに CPU を割り当てる（デッドラインが設定されていないスレッドの優先度は最も低い）．スレッドは該当の処理ブロックの実行を終えると他のスレッドに CPU を譲る．

性能評価 実時間制御機構を持たない OS 上で，RT-PTL の使用によりどの程度時間を正確に守れるのかを調査するために，以下の実験を行った．

- 10 個のスレッド $T_i\,(0 \leq i \leq 9)$ を並列に実行．
- 各スレッドは 8 ミリ秒の処理を繰り返す．
- 各スレッド T_i に開始時刻 $S+i\times 10$ ミリ秒，デッドライン $S+i\times 10$ ミリ秒 +10 ミリ秒を設定．
- 各スレッドは処理終了ごとに 1 秒間スリープする．

FreeBSD2.2.7，PentiumII 333 MHz の PC で約 1 時間，デッドラインミスの状況を観測した．使用したマシンでは，sendmail，popper，NFS，WWW，Samba などのサーバが動いている．実験結果を**表 2** に示す．

表 2 によると，$T_i\,(0 \leq i \leq 9)$ がミスした場合，EDF の特性によりそれ以降のスレッド $T_j\,(i < j)$ もミスする確率が高くなるものの，全体のデッドラインミス率は 1〜2%の範囲に収まっており☆，動画や音声再生などのマルチメディアアプリケーションの実現には十分な精度であることが分かった．なお，上記のサーバプログラムを止めた状態（ただし，ypbind，nfsiod，routed，jserver などが稼働）で同じ実験を行ったところ，デッドラインミス率はすべてのスレッドで 0.7%以下となったことから，表 2 のデッドラインミスは NFS サーバなどの処理により，長時間プロセッサを奪われたためと考えられる．

3.2 I/O 処理の効率化

UNIX では，各ユーザプロセスは一度に複数のファイル I/O 処理を発行できない．そのため，一般に 1 つのユーザプロセス内にスレッド機構を組み込んだ場合，あるスレッドのファイル I/O 処理実行により他のスレッドまでブロックするという問題が生じる．そこで，動画や音声の再生など，頻繁にファイル I/O を行うシステムの効率化のために，スレッド機構におけるファイル I/O 処理の高速化方式を考案，実装した[3)]．考案手法では，ファイル I/O を行う専用のプロセスを別途設けることで，ファイル I/O を要求したプロセス（スレッドを実行しているプロセス）全体のブロックを回避する．プロセスの分離によるプロセス間通信のオーバヘッドを削減するため，UNIX の mmap 機構を利用してプロセス間でアドレス空間を共有する方法を考案した[3)]．実験の結果，I/O 要求でブロックする場合，オーバヘッド（Solaris2.6，Sun Ultra10 で 190 μsec）以外の時間を他のスレッドで使用できること，ファイル I/O 性能は 1 回の I/O のサイズが大きい場合，通常の方式とほぼ同等であることを確認した．これらの改良により，並行処理や実時間処理を要求するアプリケーションでのファイル I/O 処理による実行効率の低減が回避できる（詳細は，文献 3) 参照）．

4. 時間拡張 LOTOS 仕様のコンパイラ

本章では，まず，コンパイラ実現の際に必要な，スレッド機構上で時間拡張 LOTOS 仕様を実現するための基本的な制御機構について述べ，各スレッドへの時間パラメータの割り当て方，イベントやインタラクション実行のためのスレッドの時間制御について説明する．

4.1 スレッド間の実行制御の概要

本コンパイラは，入力仕様を文献 17) で提案した手法により，イベントの逐次・選択・繰返しからなる並列動作を含まない動作式（**単位動作式**）の集合へ分割し，各単位動作式を 1 つのスレッドに割り当てる．各スレッドは，**制御領域**と呼ばれる，LOTOS オペレータを節，各単位動作式を葉とする LOTOS 構文木を表す共有変数領域に基づいて実行制御される（**図 1**）．各スレッドは，対応する葉から根に向かって制御領域を参照することで，各スレッドとのインタラクションの情報を取得する．イベント実行時に LOTOS オペレータに対応する節に，どちら側の動作式が選択されたか（[] の場合），どのイベントで同期できるか（$|[gl]|$ の場合）などの情報を書き込み，これを基にスレッド間の実行制御を行う（詳細は，文献 17) 参照）．

4.2 時間パラメータの実時間スレッドへの割当て

スレッドの時間パラメータの設定は以下のように行う．

☆ スレッド間の開始時刻，デッドラインの間隔を 100 ミリ秒に設定した場合では，すべてのスレッドでミス率は 1%程度となった．

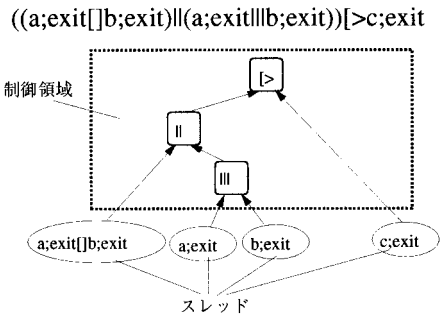


図 1 制御領域の構造
Fig. 1 Structure of the control area.

イベントの連接とその選択　イベントの連接とその選択の形の動作式は並列動作を含まないため，1 スレッドで実現できる．2.2 節の方針に従い，選択肢が複数ある場合，時間制約上のデッドラインが最も早いものを優先して実行する．たとえば，$a?x:int@?t[1 \leq t \leq 4]$ [] $b@?s[2 \leq s \leq 3]$ の形の単位動作式が割り当てられたスレッドは，まず，時間制約の終了時刻が最も早い選択肢 b の終了時刻 3 を，デッドラインに設定する．この場合，この時点で実行可能なイベントは a，b の 2 つの候補があり，それぞれの実行可能性を前もって調べるため，開始時刻は 0 に設定する．自スレッドと排他的な関係にあるスレッドが未実行であり，かつイベントの実行条件（たとえば，ゲート a での変数 x への整数入力の到着）が満たされれば，開始時刻を 1 に再設定し，再スケジューリング後 a を実行する．また，時刻 3 までに b が実行されなければ，b は以後永久に実行不可能なため候補から除外し，2 番目に早い終了時刻 4 をデッドラインに再設定する．

周期処理　時間拡張 LOTOS では，周期的な処理（周期内に処理が終了しなかった場合の代替処理も含む）を以下のように記述可能である（ただし，周期は T_p，B_1，B_2 は動作式とする）．

$P := loop\ B_1\ [>\ wait(T_p); B_2\ endloop$
$where$
$B_1 := a_1@?t_1[t_1 < T_p];\ \cdots\ ; a_n@?t_n[t_n < T_p];$
$wait(T_p - \sum_{i=1}^{n} t_i)$

このようなプロセス P を実現するには，周期 T_p ごとに動作式 B_1 の処理を行い，処理が T_p 時間以内に終了しないときは，割込みにより動作式 B_2 の処理が行われる必要がある．この場合，B_1 と B_2 を並行して動作する別のスレッドに割り当て，B_1 に対応するスレッドには開始時刻，デッドラインとしてそれぞれ 0，T_p を，B_2 のスレッドには，それぞれ T_p，$T_p + \alpha$ を設定する（α は任意の正数）．

4.3 時間制御

4.3.1 イベントの中断機構

動画フレームのデコードなど，処理時間の長い処理を 1 つのイベントに対応づける場合，2.2 節の実現方針 (1) に従うためには，各処理の処理時間が前もって分かっていることが望ましい．ソフトリアルタイム環境では処理時間の予測が難しいため，ここでは，時間制約内であればイベントの実行を開始し，時間制約内に終わらない場合，途中で処理を打ちきることとした．中断されたイベントは実行されなかったものと見なす．以上の機構は，各スレッドが処理の途中でデッドラインをミスしていないかを調べ，ミスしていた場合，イベントの実行を中断，キャンセルすることで実現した．また，キャンセルによる副作用を回避するため，出力や変数入力は処理の最後に不可分に行うこととした．

4.3.2 時間制約付イベント間の同期

同期実行されるイベントの組を以降**ランデブ**と呼ぶ．2.2 節の実現方針 (3) に従うには，同期条件☆の判定に加えて，(i) 実行開始時刻に従った複数ランデブの優先度制御，(ii) 同期するスレッドどうしがタイミング良く待ち合わせるための制御，が必要となる．

(i) 複数ランデブ間の優先度制御

複数の排他的なランデブが存在する場合には，あるランデブが同期条件を満たすことが確認され，その実行開始時刻まで待っている間に相互排他的なランデブが実行されていないかを確認する必要がある．

図 2 の動作仕様では，(P,Q) または (P,R) のどちらか一方の組合せでのみランデブが成立する．この例では，P，Q，R のイベント a がアクティブになる時刻はそれぞれ 1，1，2 なので，P，Q は $4 \leq T \leq 5$ を満たす時刻 T にのみ a を同期実行可能である．現在時刻が 4 以前の場合，4 まで待ってからイベントの同期実行を行う必要があるが，この間に相互排他的で，かつ，より優先度の高い（開始時刻が早い）ランデブ (P,R) が実行可能になることがある．

このため，実行待ちのランデブを開始時刻が早い順に整列して登録するためのキューを導入した．上の例では，まず (P,Q) の組で時刻 [4, 5] にランデブ可能であることが分かるため，このランデブをキューに登録し，P，Q は時刻 4 まで他のスレッドに CPU を譲る．その後，時刻 2 以降に (P,R) 間で時刻 [3, 5] にランデブ可能なことが分かり，その開始時刻から (P,Q) 間のランデブより前に登録される．その後，時刻 3 以

☆ 入出力値の一致やガード式の真偽，同期するイベントの時間制約がすべて満たされる時刻が存在するかどうかなど

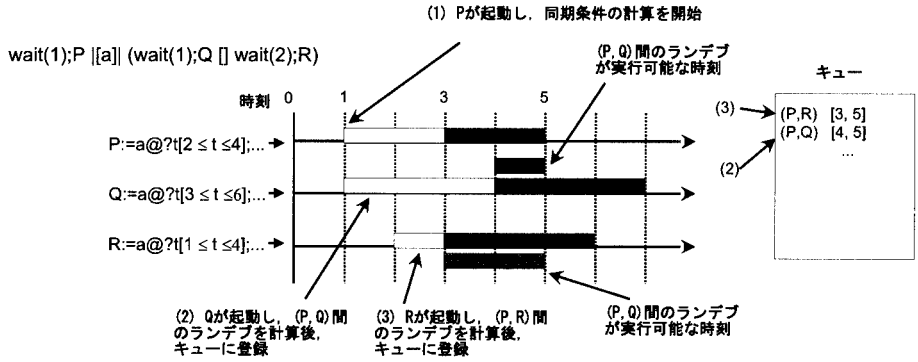


図 2 時間付マルチランデブの実行例
Fig. 2 Progress of the timed multi-way synchronization.

降になって，(P,R) 間のランデブがキューの先頭にあれば，実行される．実行の際，キュー内の他の排他的なランデブのエントリ（この場合，(P,Q) 間のランデブ）をすべて削除することによって，以後それらが実行されないようにする．

(ii) 同期のタイミングを合わせるための制御

本コンパイラでは，同期条件の判定をスレッド間の通信で実現するため，デッドラインが設定されているスレッドとされていないものの間の同期が，EDF ではスムーズに行われない場合が考えられる☆．

簡単のため，本コンパイラでは，あるスレッドで，同期指定されたイベントがアクティブになったときに，一時的に最高のスケジューリング優先度（すなわち，デッドラインが直近）を与え，同期条件の判定を最優先に行わせることで上記の問題を回避した．

また，時間制約が指定されていないイベントの繰返しからなるプロセスと，時間制約が指定されたイベントの繰返しからなるプロセスの間の同期で，かつ各周期で同期するイベントの組合せが特定可能な場合に限り，同期判定に至るまでの処理がタイミング良く行われるように，前者のプロセスを実現するスレッドに，後者のプロセスの時間制約に基づいたデッドラインを設定するようにした．

$P \ \ |[b_1, \cdots, b_k]| \ Q$ where
$P :=$ loop $B_1; b_1; \cdots; B_k; b_k$ endloop
$Q :=$ loop
$b_1@?t_1[t_1 \leq T_1]; \cdots; b_k@?t_k[t_k \leq T_k];$
endloop
（各 B_i はイベントの連接である）

たとえば，上記の仕様では，Q の各イベント b_i は，P のイベント系列 $B_i; b_i$ に対応するため，P に対応するスレッドは $B_i; b_i$ がアクティブになった時点でデッドラインを T_i に設定する．これにより，5 章で述べるような制約指向で記述した時間拡張 LOTOS 仕様において，ランデブが時間制約どおりに実行可能となる．

4.4 マルチメディア処理機能

作成したコンパイラをマルチメディアアプリケーションの開発に適用できるようにするため，動画再生プログラム xanim[12] のソースコードなどを利用し，動画のフレーム処理や指定した時間分の音声データの読み込み，再生などを行うプリミティブを作成した．動画は，コマとばしなどに対応できるよう，各フレームが他と独立に符号化されたモーション JPEG 形式を対象とした．また，ネットワーク上の異なるマシン間で UDP プロトコルによりデータを送受信するためのプリミティブも作成した．作成したプリミティブを**表 3** に示す．各プリミティブを，5 章で述べるように時間拡張 LOTOS 仕様中のイベントに対応づけることにより，コンパイラが生成する目的コードでそれらの機能が実現される．

5. マルチメディアアプリケーションの記述

本論文では，QoS 制御やマルチメディア情報を再生するタイミングなどを容易に変更できるようにするため，システムをイベントとその実行順序のみを記述した**主パート**と，主パート内のイベント間の実行時間間隔に対する制約や，制約が満たされなかった場合の代替処理などを記述した**タイミングパート**に分けて構成し，これらの間で $Main \ ||gatelist|| \ Timing$ のように，関連するイベントが同期実行されるよう指定する．これは制約指向記述スタイルと呼ばれ，システムの保守・運用に優れていることが知られている[15]．

5.1 動画配信システムの記述

以下，1 つのサーバとネットワークを介して接続された k 個のクライアントからなる動画配信システム

☆ デッドラインを設定したスレッドがつねに優先され，デッドラインを設定していないスレッドには CPU が割り当てられない．

表 3 マルチメディア処理用プリミティブ
Table 3 Primitives for multimedia processing.

プリミティブの名前	内容
OpenMovie	モーション JPEG ファイルのオープン
CloseMovie	モーション JPEG ファイルのクローズ
GetFrameRate	エンコード時のフレームレートの取得
CreateWindow	動画表示用のウィンドウの作成
ReadFrame	動画の次のコマの読み込み
Decode	JPEG 画像のデコード
DrawPic	ビットマップを指定のウィンドウに描画
OpenWav	wave ファイルのオープン
SendPacket	UDP により指定先へパケットを送信
RecvPacket	UDP により指定先からパケットを受信
…	…

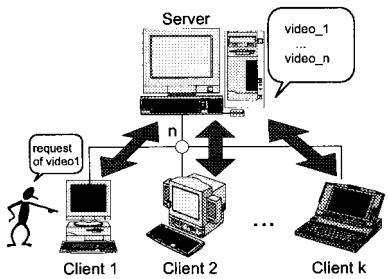


図 3 動画配信システム
Fig. 3 A video play-back system.

(**図 3**) を制約指向で記述する．各クライアントは，動画の配信サービスを好みの品質範囲で要求することができ，提供される品質はシステムの負荷に応じてその範囲内で動的に制御される．サーバのディスク装置にはあるフレームレートでエンコードされた動画ファイルが記憶されており，クライアントから要求された動画データを適当な通信速度で送信する．サーバは，k 個のクライアントとゲート n により通信する（図 3）．ここでは，紙面の都合で，品質として時間解像度（動画のフレームレート）のみを考慮する（色数やサイズを考慮した記述も可能[14]）．

5.1.1 主パートの記述

各クライアント（変数 id で識別する）がユーザからの要求 (u?req) により，新たなサービス（動画）の提供をサーバに依頼する (n!id!req) とする．要求が受け入れられると (n!id!Accepted)，サーバは要求された動画再生のサービスの提供を開始する．動画データがコマごとにサーバから送信される場合，クライアントの動作は，データの受信 (n!id?frame)，デコード (?pic:=Decode(frame))，表示 (v!DrawPic(pic)) の繰返し，あるいは，クライアントマシンの CPU 負荷が高い場合のコマ落とし（v!Skip）で表せる．一方サーバは，クライアントからの要求が認められると，接続ごとにサービスプロセスを生成する．各サービスプロセスは，1 コマのデータの読み込み（?data:=ReadFrame(fp)）と送信（n!id!data），あるいはネットワークの負荷による送信のスキップ（n!id!Skip），の繰返しで記述される（ただし，ゲート n におけるデータの送受信は，表 3 の SendPacket，RecvPacket を用いて実現するものとする）．

表 4 主パートの記述例
Table 4 An example of the main part.

```
Server[n]: =
  n?id?req;
  (  n!id!Accepted;
     (Service[n](id,req) ||| Server[n])
  [] n!id!Rejected; Server[n] )
where
  Service[n](id,req):=
   ?fp:=OpenMovie(req.file);
   loop
    var data in
      ?data:=ReadFrame(fp);
      (  n!id!data [] n!id!Skip )
    endvar
  endloop

Client[u,n,v](id) :=
 u?req; n!id!req;
 ( n!id!Rejected; Client[u,n,v](id)
[] n!id!Accepted;
   loop
     n!id?frame;
     ( ?pic:=Decode(frame); v!DrawPic(pic) [> v!Skip )
  [] n!id!Skip; v!Skip
   endloop )
```

以上より，サーバ，クライアントの主パートは**表 4** のように記述できる．

5.1.2 タイミングパートの記述

タイミングパートは，要求品質に対してシステムが満たすべき時間制約を記述した**品質定義パート**と，環境の動的変化などにより，提供品質をどのように切り換えるのかを指定する **QoS シナリオパート**からなる．

品質定義パート　ここでは，提供可能なサービスの品質のすべてを選択肢として記述し，現在の品質値（変数 q で表す）に応じて適切な 1 つが選択されるよう指定する．各選択肢には，指定の品質を満たすには，主パート中のどのイベントがどのような時間間隔で実行されるべきかを記述する．また，QoS シナリオパートから品質値 q を変更するイベント (r1!id!Upgrade,r1!id!Degrade) と現品質値において時間制約が満たされたかどうかを通知するイベント (r2!id!Success,r2!id!Miss) もあわせて記述する（r1, r2 は QoS シナリオパートとの通信のためのゲートとする）．サーバにおいて，品質値の値に応じて，送信するフレームレートを変更する場合の品質定義パート QS，クライアントにおいて，受信したコマデータの

表 5 品質定義パートの記述例
Table 5 An example of the quality definition part.

```
QS[n,r1,r2](id, q0) :=
  var q:int:=q0, P:time:=period(q0) in
   loop  (* period() は 1 フレームあたりの時間間隔 *)
     [q <req.qmax]-> r1!id!Upgrade;
            ?q:=q+1;  ?P:=period(q)
  [] [q >req.qmin]-> r1!id!Degrade;
            ?q:=q-1; ?P:=period(q)
  [] ( [q==1]->(n!id!any packet@?t1:time[t1<=P];
               r2!id!Success@?t2[t2<=P]; wait(P-t1-t2)
               [> wait(P); r2!id!Miss )
    [] [q==2]->(n!id!Skip@?t1:time[t1<=2*P];
               n!id!any packet@?t2:time[t1+t2<=2*P];
               r2!id!Success@?t3:time[t1+t2+t3<=2*P];
               wait(2*P-t1-t2-t3)
               [> wait(2*P); r2!id!Miss ) )
  ... (以下省略)
   endloop
  endvar

QC[v,r1,r2](id, q0) :=
  var q:int:=q0, P:time:=period(q0) in
   loop
     [q <req.qmax]-> r1!id!Upgrade;
              ?q:=q+1;  ?P:=period(q)
  [] [q >req.qmin]-> r1!id!Degrade;
              ?q:=q-1; ?P:=period(q)
  [] ( [q==1]->(v!any video_frame@?t1:time[t1<=P];
               r2!Success@?t2:time[t1+t2<=P];
                  wait(P-t1-t2)
               [> wait(P); r2!Miss )
    [] [q==2]->(v!Skip@?t1:time[t1<=2*P];
               v!any video_frame@?t2:time[t1+t2<=2*P];
               r2!Success@?t3:time[t1+t2+t3<=2*P];
               wait(2*P-t1-t2-t3);
               [> wait(2*P); r2!Miss ) )
  ... (以下省略)
   endloop
  endvar
```

デコード，表示処理を省略することにより 1/2, 1/3 のフレームレートを提供する場合の品質定義パート QC の記述例を**表 5** に示す．

QoS シナリオパート　過負荷時に，優先順位の低いサービスの品質を下げることによって，優先順位の高いサービスの品質を維持する方法が考えられる．このような QoS 制御を行うには，各サービスにおいて要求品質が保たれているかどうかを適当な時間間隔でチェックするためのモニタ機構が必要になる．ここでは，各サービス i に対し，適当な時間間隔 T の間に，周期処理が間に合わなかった場合の割合（周期ミス率）rate_i を計算するようにモニタ機構を記述する．たとえば，サーバにおける QoS 制御は，1〜k の各サービスに対応するモニタ機構から受け取った周期ミス率を基に，あるサービスの品質値を上下させることで実現できる．どのサービスの品質値を変更するのかをランダムに決める場合の QoS シナリオパートの記述例を**表 6** に示す．ここでは，サーバに対する QoS シナリオのみ与えたが，クライアントに対する QoS シナリオも同様の方法で与えることができる．また，QoS シナリオのみを変更することで，資源の予約などが可能な環境でのアドミッションコントロールを含んだ QoS 制御を行うことも可能である[14]．

表 6 サーバに対する Qos シナリオパートの記述例
Table 6 An example of the server's QoS scenario part.

```
QoSScenario[r1,r2,s]:=
  Monitor[r2,s] |[s]| QoSControl[r1,s]
where
  Monitor[r2,s](id) :=
   loop
    var cnt:=0, mcnt:=0 in
      loop
          r2!id!Success; cnt:=cnt+1
       [] r2!id!Miss; ?mcnt:= mcnt+1
      endloop
  [> wait(T); s!id!Result!(mcnt*100/cnt)
    endvar
   endloop
  QoSControl[r1,s] :=
   loop
    ( s!1!Result?rate_1:int; exit ||| ...
|||   s!k!Result?rate_k:int; exit )
>>([min(rate_1,...,rate_k) < 70]->
      (r1!1!Degrade [] ... [] r1!k!Degrade)
[] [rate_1==100 and ... and rate_k == 100]->
      (r1!1!Upgrade [] ... [] r1!k!Upgrade) )
   endloop
```

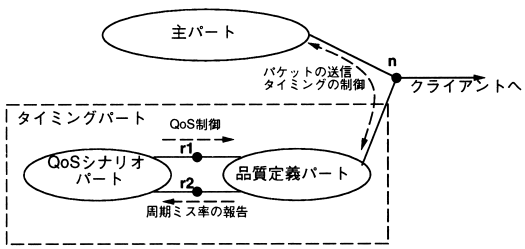


図 4 サーバにおける各記述パートの関係
Fig. 4 Organization of a server specification.

以上のように記述したサーバの動作仕様における，主パート，品質定義パート，および QoS シナリオパート間のゲートの関係を**図 4** に示す．

5.2 メディア間の同期機構の記述

ビデオ会議システムなどでは，発話者の映像と音声をある程度の正確さで同期再生したい場合がある．今，T 時間ごとにビデオと音声（それぞれ，$n1$ コマ/秒，$n2$ フレーム/秒でエンコードされているものと見なす）を同期させる場合，その制約は**表 7** のように記述可能である．制約 MediaSync をタイミングパートに追加し，品質定義パートとゲート v，a に関して同期指定することで，主パートを変更することなくメディア間同期機構を追加できる．

6. 評　価

作成したコンパイラが生成する目的コードの実行効

表 7 メディア間同期の記述例
Table 7 An example of the inter-media synchronization.

```
MediaSync[v,a](T,n1,n2):=
 hide sync in
  var cnt1, cnt2:int in
   loop
    ?cnt1:= n1*T; ?cnt2:= n2*T;
    while ((cnt1>0) or (cnt2>0)) do
       [cnt1>0]-> v!any video_frame; ?cnt1:=cnt1-1
    [] [cnt2>0]-> a!any audio_frame; ?cnt2:=cnt2-1
    endwhile;
    sync
   endloop
  endvar
```

表 8 1 回の同期処理に要する時間
Table 8 Execution time for each synchronization.

同期するプロセス数	2	3	4	5
同期にかかる時間 (μs)	276.2	542.1	822.6	1095.0

率を調べるために，いくつかの仕様で実験を行った．実験に，RedHat Linux 5.2 が動作する PC（Pentium Pro 200 MHz，RAM 96 MByte）を使用した．

6.1 マルチランデブの実行効率

マルチランデブを含むイベント実行のオーバヘッドを調べるために，イベントを繰り返し実行するプロセスを複数並列に実行し同期させた場合の，1 回のイベントの同期実行にかかった時間を測定した．同期させるプロセス数を 2 から 5 まで変化させた場合の実験結果を**表 8** に示す（10 万回測った平均値）．

イベント a には処理を割り当てていないため，1 個のイベントの同期実行にかかる時間数百マイクロ秒のほぼすべてが同期実行のオーバヘッドであると考えられる．

6.2 動画再生プログラムの実装実験と評価

次に，比較的単純な動画再生プログラムを対象に，160×120 ピクセルのモーション JPEG でエンコードされた動画を使用して次の実験を行った．

- (a) 時間拡張 LOTOS と (b) C 言語+スレッド機構の両方で同じシステムを記述し，得られる目的コードの実行効率を比較する．
- 同時に再生する動画数，再生フレームレートを変えて実際に達成されたフレームレートを計測する．

(a) では，5 章で述べた制約指向記述スタイルに従い，主パートとタイミングパート（フレームレートを満たすための周期を指定）を同期させるように記述した．(b) では (a) と同様に RT-PTL を使用した．また，描画フレームの読み込み，デコード，表示などを行うためのプリミティブは (a)，(b) ともに 4.4 節のものを使用した．システムの負荷を監視するための QoS モニタは，3 秒おきに各周期の処理が時間内に終わらなかった場合の回数をカウントし，その頻度によってシステムの負荷を判断し QoS 制御を行う．この機構は (a) では時間制御部と同期指定された時間拡張 LOTOS のプロセスとして，(b) では独立して動作するスレッドとして記述した．実験結果を**表 9** に示す．

表 9 動画再生システムにおけるフレームレート
Table 9 Frame rates in a video play-back system.

	LOTOS			C 言語+スレッド機構		
	$n=1$	$n=2$	$n=3$	$n=1$	$n=2$	$n=3$
10 fps	10.01	10.03	10.04	10.00	10.00	10.00
15 fps	15.01	15.08	15.11	15.00	15.00	15.00
20 fps	20.01	19.99	16.16	20.00	20.00	19.38
25 fps	25.00	24.62	16.14	25.00	25.00	19.56
30 fps	29.97	24.60	16.15	30.00	29.55	19.63
40 fps	40.00	-	-	40.00	-	-
50 fps	49.88	-	-	49.84	-	-
60 fps	50.67	-	-	59.32	-	-
70 fps	-	-	-	59.01	-	-

実行効率　表 9 によると，再生する動画が 1 個の場合，(a) ではおよそ 50 fps まで，(b) では 60 fps まで指定したとおりのフレームレートが達成されており，これらの値から，(a) では，16%程度のプロセッサパワーが，(b) の場合より多く消費されている．(a) では，1 フレームの表示に 3 回の同期を行うため，表 8 から，その処理時間は 1.6 ミリ秒程度になる．動画フレームの読み込み，デコード，表示のための処理時間の和が 15 ミリ秒程度であることから，制約指向スタイルで記述したことによるプロセス間の同期のための処理に余分のプロセッサパワーの大半が消費されていることが分かる．同時に再生する動画の数を 2，3 にそれぞれ設定した場合，達成できたフレームレートは (a)，(b) ともに，動画数 1 の場合の約 1/2，1/3 になった．

記述量および変更の手間　システムの記述・変更にともなう手間を評価するために，(a)，(b) それぞれについてシステム全体の記述量および変更に必要な工数を調査した．動画数 3 の (a)，(b) のソースコードを比較したところ，絶対的な記述量はアクション（プロセス，関数，変数の宣言部を除く，代入文，選択文，繰返し文，プロセス/関数呼び出し）の数で見ると，(a) では (b) に比べて半分程度であった．この差は (b) における，スレッド間通信のための共有変数の排他制御や，構造体で表される時間変数間の演算のコードが (b) に比べ多いためと思われる．

次に，システムの変更にともなう手間を評価するため，動画 1 と動画 2 は 3 秒ごと，動画 2 と動画 3 は 20 秒ごとに同期をとりながら再生する機構を実装し，追加・変更したコードの量を計測した．**表 10** から，(b) ではアクション数でおよそ 1.5 倍程度に増加している

表 10 メディア間同期追加のための記述量
Table 10 Description size for inter-media synchronization.

	LOTOS	C 言語
元のアクション数	53	101
元のサイズ（バイト）	2048	5672
追加・変更したアクション数	12	55
増加サイズ（バイト）	518	2420

のに対して，(a) では 2 割程度の増加にとどまっていることが分かる．これは，5 章で述べたように，(a) では，主パートおよびタイミングパートにすでに記述されたプロセスの動作を変更することなく，別モジュールとして新たな制約を追加できるのに対し，(b) では描画を行うスレッド間で同期をとる機構，およびどのスレッド間で同期しなければならないかを決定する機構などを新たに実装する必要があったためである．この差は同期関係が複雑になればさらに大きくなると考えられる．

7. おわりに

本論文では，時間拡張 LOTOS を定義し，その動作仕様を実時間スレッド機構を用いて実装する手法を提案した．さらに，典型的なマルチメディアアプリケーションを例に，資源の動的割当てを指定した QoS 制御やメディア同期などが簡潔かつ効果的に記述できることを示した．試作したコンパイラを用いた実験により，C 言語により作成したプログラムと遜色ない効率で動作する目的コードが得られることを確認した．以上より，提案手法は，QoS 制御などを含むアプリケーションの開発に十分適用可能と思われる．

近年，動画や音声，テキストなどの情報をどのようなタイミングで再生するかを記述するための言語 SMIL[16] が注目を集めている．SMIL では，複数メディアの同期再生や，システム資源が不足する場合の代替メディアを指定可能であるが，このようなメディアの再生順序/タイミングを記述したシナリオを時間拡張 LOTOS で記述し，本論文で記述した制御機構と同期させることによって，本コンパイラをシナリオの実行系として利用することも可能である．

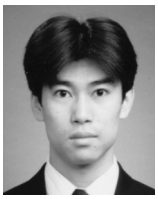
ATM などの下位層を模倣するプロセスの記述・追加による QoS 制御手法のシミュレーションや，分散環境でのマルチランデブを利用した多人数参加アプリケーションの実装・評価などが今後の課題である．

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) 安倍広多，松浦敏雄，安本慶一，東野輝夫，谷口健一：UNIX 上で周期スレッドを実現するユーザレベルスレッドライブラリの実現法，信学技報，CPSY-97-24, pp.49–54 (1997).
3) Abe, K., Matsuura, T., Yasumoto, K. and Higashino, T.: Design and Implementation of an efficient I/O Method for a Real-time User Level Thread Library, *Proc. IEEE 5th Int. Workshop on Real-Time Computing Systems and Applications* (*RTCSA'98*), pp.117–120 (1998).
4) Courtiat, J.-P. and Oliveira, R.C.: RT-LOTOS and its application to multimedia protocol specification and validation, *Proc. IEEE Int. Conf. on Multimedia Networking*, pp.31–45 (1995).
5) Fischer, S.: Implementation of multimedia systems based on a realtime extension of Estelle, *Proc. 9th Int. Conf. on Formal Description Techniques* (*FORTE'96*), pp.310–326 (1996).
6) ISO: Information Processing System, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, ISO 8807 (1989).
7) ISO: Final Committee Draft 15437 on Enhancements to LOTOS, ISO/IEC JTC1/ SC21/ WG7 (1998).
8) Lakas, A., Blair, G. and Chetwynd, A.: Specification and Verification of Real-Time Properties using LOTOS and SQTL, *Proc. 8th Int. Workshop on Software Specification and Design*, pp.75–84 (1996).
9) Leonard, L. and Leduc, G.: An Introduction to ET-LOTOS for the description of time sensitive systems, *Computer Networks and ISDN systems*, Vol.29, No.3, pp.271–292 (1997).
10) Liu, C.L. and Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
11) Locke, C.D.: Best-effort decision making for real-time scheduling, PhD thesis, Carnegie Mellon Univ, Pittsburgh, PA (1985).
12) Podlipec, M.: The XAnim Home Page, http://xanim.va.pubnix.com/home.html
13) Steinmetz, R. and Wolf, L.C.: Quality of Service: Where are We?, *Proc. IFIP 5th Int.*

Workshop on Quality of Service (*IWQOS'97*), pp.210–221 (1997).
14) 辰本比呂記，安本慶一，東野輝夫，安倍広多，松浦敏雄，谷口健一：時間拡張 LOTOS を用いたマルチメディアシステムの記述とその実現，情報処理学会マルチメディア通信と分散処理（DPS）ワークショップ論文集，pp.133–138 (1998).
15) Vissers, C.A., Scollo, G. and Sinderen, M.v.: Architecture and Specification Style in Formal Descriptions of Distributed Systems, *Proc. 8th Int. Conf. on Protocol Specification, Testing, and Verification* (*PSTV'88*), pp.189–204 (1988).
16) W3C: Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, http://www.w3c.org/TR/REC-smil/
17) 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.566–575 (1998).

(平成 11 年 5 月 10 日受付)
(平成 11 年 10 月 7 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．平成 11 年同大学大学院博士前期課程修了．現在松下電器に勤務．リアルタイム OS 向けマルチメディア処理チップの開発等に従事．

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 年同大学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．

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，大阪大学大学院基礎工学研究科教授，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.

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．昭和 54 年同大学大学院基礎工学研究科（情報工専攻）博士後期課程退学後，同大学助手．平成 4 年同大学情報処理教育センター助教授，平成 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．

山口　弘純（正会員）

平成 6 年大阪大学基礎工学部情報工学科卒業．平成 10 年同大学大学院博士後期課程修了．工学博士．同年オタワ大学客員研究員．平成 11 年大阪大学大学院基礎工学研究科助手．現在に至る．分散システムの設計法等の研究に従事．

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 年同大学大学院博士課程修了．同年同大学助手．現在，同大学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．