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

辰本比呂記，東野輝夫，谷口健一
大阪大学基礎工学部情報科学科
(tatumoto,higashino,taniguchi)
@ics.es.osaka-u.ac.jp

安本慶一
滋賀大学経済学部情報管理学科
yasumoto@biwako.shiga-u.ac.jp

安倍広多，松浦敏雄
大阪市立大学学術情報総合センター
(k-abe,matsuura)@media.osaka-cu.ac.jp

概要　本稿では，QoS制御方針をシナリオとして与えることが可能なマルチメディアシステムの記述法を提案する．提案する手法では，マルチメディアシステムを，(1) 時間制約を含まない基本動作の実行系列，(2) ある品質を守るために満たすべき時間制約，(3) 負荷に応じてサービスの品質を調整するQoSシナリオ，の3つのモジュールからなる仕様として構成し，形式記述言語LOTOSのある時間拡張により記述する．これらのモジュールをLOTOSのもつマルチランデブ機構を用いて同期実行させることにより，システム全体を要求仕様どおりに動作させる．コンパイラを試作し，自動生成された目的コードにおいてQoS制御が正しく機能することを確かめた．

1　はじめに

複数ユーザ間で映像や音声データを実時間で交換するようなマルチメディアシステムを実現する際には，ユーザが要求する品質（遅延時間や画像の解像度など）にできるだけ沿うようにサービスを提供するための**QoS制御**が必要になる．

これまでに，エンドシステム（サーバやクライアント）における計算機資源（プロセッサやメモリ容量等）の調整機構や，利用ネットワーク帯域の予約に基づくエンドシステム間のネットワーク資源（バンド幅や遅延）の適切な割当て方式，各資源の負荷が変化しても品質を維持し続けるための方式など，様々なQoS制御機構が提案されてきた[2]．しかし，これら既存のQoS制御機構は対象とするネットワークアーキテクチャに依存している部分が多く，利用サイトによる利用可能資源の違いや資源の予約機構の有無等の違いにより，数あるQoS制御方針の中から，最も効果的なものを選択するための効果的な方法はあまり提案されていない．上記の問題点に対処するには，システム設計の際に，システムを構築するサイトの環境やシステムの性格に応じて制御方針（QoSシナリオ）を自由に記述・変更できるための機構およびシナリオ記述言語が望まれる．また，ラピッドプロトタイピングにより記述したQoSシナリオの効果を確認，評価するには，高速な処理系が求められる．

近年，システムの動作に求められる要求を形式的に記述し，その正しさを機械的に検証するための形式記述技法(FDT)が注目されている．文献[4]では，仕様記述言語Estelleに時相論理を拡張した言語を用いたマルチメディアシステム仕様の記述法，およびその実時間OSのマルチスレッド機構上への実装法を提案しているが，QoS制御を考慮していない．また，形式記述言語LOTOS[5]を，時間制約が記述できるよう拡張した言語を用いたマルチメディアシステムの記述法が提案されている[3, 7]が，動作の正しさの検証を主な目的としているため，実装時のQoS制御などについては考慮していない．

本稿では，各サービスに対するQoS制御方針をシナリオとして与えることが可能な，マルチメディアシステムの記述法を提案する．提案する手法では，マルチメディアシステムを，(1) 時間制約を含まない基本動作（I/Oや画面表示等）の実行系列（**主動作式**と呼ぶ），(2) 主動作式における各動作および動作間に対する時間制約により表されるシステムが満たすべき性質（**品質仕様**と呼ぶ），(3) 各時点で利用可能な資源の状況に応じて提供されるサービスの品質を調整するシナリオ（**QoSシナリオ**と呼ぶ），の3つのモジュールからなる動作仕様として，LOTOSのある時間拡張により記述する．これらをLOTOSのもつマルチランデブ機構を用いて同期させることによって，システム全体が要求どおりに動作するよう設定する．提案手法では，システムの満たすべき時間制約およびそのQoS制御シナリオを，システムの主動作とは別に設計・変更できるため，同様のシステムを利用目的や利用可能な資源が異なる環境で実装する際に有用である．

試作した時間拡張LOTOS仕様のコンパイラを用いて，ソフトリアルタイム環境における実験を行った結果，シナリオに指定した通りのQoS制御を実現し，かつ高速に動作する目的コードが得られることを確認した．

2　時間拡張LOTOS

本節では，LOTOS[5]の概要および，E-LOTOS[6]の時間構文を追加したLOTOS^{T+}について述べる．

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生起する．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 $(B1|[gl]|B2)$，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なう，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 gl についてマルチランデブするよう指定されている場合 $(p_1|[gl]|...|[gl]|p_n)$，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 $g \in gl$ である同一ゲート g のイベントを実行可能であり，かつ，この内の任意の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時に限る．

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

時間制約を扱うための構文など，LOTOSを大幅に拡張したE-LOTOS[6]がISOにより標準化されつつある．本稿では，LOTOSにE-LOTOSで拡張された構文の一部を追加した LOTOS^{T+} を定義し，それを用いてシステムを記述する．LOTOS^{T+} は次のような構文で定義され，選択・並列・割込み・マルチランデブに加え，時間に関する構文，繰り返し，変数の取り扱い[1]などの機構を有する．

```
B       ::= hide G(,G)* in B | loop B endloop
          | var V(,V)* in B endvar | B '>>' B
          | B '[>' B | B '||' B | B '[[' G(,G)* ']]' B
          | B '|||' B | B '[]' B | B';' B | Action
          | exit | stop | wait'('E')' | Process
Action  ::= G [P(P)*] [@P] ['['E']'] | P:=E
P       ::= ?V[:T] | !E
Process ::= ProcessID ['['G(,G)*']'] [(E(,E)*)]
```

(G はゲート，$V[:T]$ は x:int のような変数宣言，E はADTで書かれた式を表す．$P := E$ は，?x:=3 のような変数への値の代入)

[1]LOTOSでは，変数への値の代入は一度きりであった

ただし，上記クラスの時間構文 $G@P'['E']'$ において，時間変数に関するガード式は，$C1 \leq t \leq C2$ のような一つの連続した時間範囲になるものに限る．

以下は新たに追加された構文とその意味である．

構文	意味
loop B endloop	B を繰り返し実行する
var $V(,V)^*$ **in B endvar**	B で使用する変数群（書換可能）の宣言
$a@?t$	イベント a が実行された時刻を変数 t に取得する
$a@?t[p(t)]$	イベント a は $p(t)$ を満たす時刻 t に実行できる
$a@!T$	イベント a は時刻 T に実行できる
wait(d)	d 単位時間待つ

(@オペレータが省略されたイベントは任意の時刻に実行可能であり，例えば，a?x:int は a?x:int@any と同じ)

時刻は各イベントがアクティブになってからの経過時刻である．本稿では，時間を表すソートは離散数であり，全てのプロセスにおいて時間の進行速度は同じ（1単位時間は1msに相当）に設定する[2]．時間指定が満たされない間はイベントは実行できない．プロセス呼出しは即座に実行されるものとする．

3 マルチメディアシステムの記述

本節では，時間拡張LOTOSを用いたマルチメディアシステムの記述法について述べる．

提案する手法では，QoS制御方針の変更に容易に対処できるようにするため，マルチメディアシステムを，(1) 主動作式 (Main)，(2) 品質仕様 (Restriction)，(3) QoSシナリオ (Scenario)，の3つの独立したモジュールからなる動作仕様として，LOTOS^{T+} により記述する．これらのモジュールをマルチランデブ機構を用いて次のように同期実行することによって，システム全体が要求仕様どおりに動作するよう設定する (ここで，Gatelist1, Gatelist2はそれぞれ，MainとRestriction，RestrictionとScenarioの間で同期させたいゲートの並びである)．

```
Main |[Gatelist1]| Restriction |[Gatelist2]| Scenario
```

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成るビデオ配信システム（図1）を例に，その記述を与える．各クライアントは，動画のサービスを好みの品質範囲で要求することができる．サーバには動画 (30fps) を記録したディスク装置が接続されており，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

[2]E-LOTOSでは時間のソートに離散，有理数のいずれかを指定可

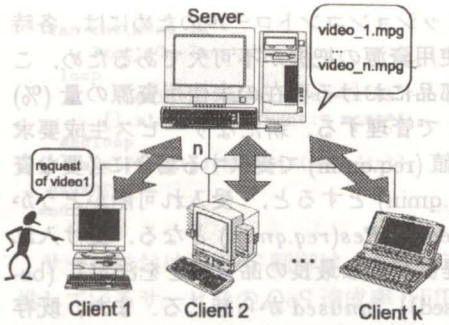

図 1: ビデオ配信システム

表 1: 時間・空間解像度に対する動画の品質値

	160 × 120	320 × 240	640 × 480
15 fps	1	3	5
30 fps	2	4	6

ントとゲート n により通信するものとする（図 1）．動画の時間解像度 (15〜30fps)，空間解像度 (160 × 120〜640 × 480pixels) の組に対応する**品質値**を表 1 に示す．

3.1 主動作式および品質仕様の記述

マルチメディアシステムを構成する各部品ごとに，主動作式とその品質仕様を記述する．図 1 のシステムでは，1 つのサーバを構成する部品と k 個のクライアントを構成する部品がある．各部品の主動作式，品質仕様には，提供可能な全てのサービスに対応する動作系列およびその守るべき時間制約を，それぞれ記述する．

例えば，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たら (n!id!Accepted!req!any)，サーバは要求された動画ファイルをオープンし (?fp:=Open(req.file))，サービスの提供を開始する（あるいは，なんらかの理由によりサービスの提供を拒否する (n!id!Rejected)）．圧縮されたデータが各フレームごとにサーバから送信されるとすると，クライアントの動作は，データの受け取り (n!id?frame)，デコード (?pic:=Decode(frame))，画面表示 (v!pic) の繰り返しで表すことができる．

以上より，クライアントの主動作式は LOTOS^{T+} のプロセス Client として次のように記述できる．

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

(ただし，Decode(frame) は，frame をデコードする関数とする)

ユーザのサービス要求が認められたら，サーバはクライアントに対するサービスプロセスを生成し，ユーザからの新たなサービス要求を待つ．各サービスプロセスにおいては，クライアントに対して，表 1 に示す 3 種類の解像度のいずれかのフレームデータを送信するか (それぞれ，n!id!F!data, n!id!H!Half(data), n!id!Q!Quarter(data) とする) あるいは，15fps の実現のため，送信をスキップする（n!id!Skip）．以上よりサーバの主動作式 Server は以下のように記述できる．

```
Server[n]: =
n?id?req;
( n!id!Accepted!req!any; ?fp:=Open(req.file);
  (Service[n](id,fp) ||| Server[n])
[] n!id!Rejected; Server[n] )
where
  Service[n](id,fp):=
  loop
    var data in
    ?data:=ReadFrame(fp);
    (  n!id!F!data [] a!id!H!Half(data)
    [] n!id!Q!Quarter(data) [] n!id!Skip )
    endvar
  endloop
```

(ReadFrame(fp) は，一フレームデータを読込む関数)

システム全体の主動作式は以下のように記述される．

```
Main[n,u1,...,uk,v1,...,vk]:=
    Server[n]
|[n]|
  ( Client[u1,n,v1](1) ||| ... ||| Client[uk,n,vk](k) )
```

次にシステムが満たすべき性質を品質仕様として記述する．品質仕様には，各部品ごとに，提供可能なサービスの品質の全てを選択肢として記述し，現在の品質値 (q で表す) に応じて適切な一つが選択されるよう指定する．また，後に QoS シナリオを導入した場合に，シナリオから品質値を変更できるようにするためのイベント (r!id!Upgrade!q, r!id!Degrade!q) と，現品質値において時間制約が満たされているかどうかを通知するイベント (r!id!Success, r!id!Miss) も合わせて記述する (r は QoS シナリオとのインタフェース)．以上より，クライアントの品質仕様 QC およびサーバの各コネクションごとの品質仕様 QS はそれぞれ以下のように記述できる．

```
QC[v,r](id) :=
var q in
  n!id!Accepted?req?q0; ?q:=q0;
  loop
   r!id!Upgrade!q[q<req.qmax]; ?q:= q+1
[] r!id!Degrade!q[q>req.qmin]; ?q:= q-1
[] v!any@t;(
     [q==1 or q==3 or q==5]->
       ([t<=33]-> r!id!Success;wait(33-t)
```

```
    []  [t> 33]-> r!id!Miss)
   [] [q==2 or q==4 or q==6]->
       ([t<=66]-> r!id!Success;wait(66-t)
     []  [t> 66]-> r!id!Miss)
   )
  endloop
endvar

QS[n,r](id) :=
var q in
  n!id!Accepted?req?q0; ?q:=q0;
  loop
   r!id!Upgrade!q[q<req.qmax]; ?q:=q+1
[] r!id!Degrade!q[[q>req.qmin]; ?q:=q-1
[] ( [q==1 or q==3 or q==3]-> n!id!any@t;
                ([t<=33]-> r!Success; wait(33-t)
              [] [t> 33]-> r!Miss )
  [] [q==2 or q==4 or q==6]-> n!id!Skip@t; n!id!any@t2;
                ([t<=66)]-> r!Success; wait(66-t)
              [] [t> 66]-> r!Miss )
  endloop
endvar
```

システムの品質仕様の全体は以下のように記述される.

```
Restriction[n,r,v1,...,vk]:=
  ( QS[n,r](1) ||| ... ||| QS[n,r](k) )
|||
  ( QC[v1,r](1) ||| ... ||| QC[vk,r](k) )
```

3.2 QoS シナリオの記述

QoS シナリオには，各時点でユーザが要求した品質の範囲内でサービスが提供されるよう，計算機資源やネットワーク資源の各部品への資源の割り当てを調整するための QoS 制御を記述する.

一般に，マルチメディアシステムにおいては，(1) サービス要求時の資源割当てのための**アドミッションコントロール**，(2) サービス開始後，要求品質を維持するための **QoS 制御**の 2 つの異なる視点から制御される [2]. 以下それぞれについて，シナリオの記述法を与える.

アドミッションコントロール　アドミッションコントロールでは，新たに要求のあったサービスを提供可能かどうか調べ，可能なら資源の予約・確保・割当などを行い，サービスの提供を開始する.

アドミッションコントロールの手順は以下のようになる.

(1) 要求されたサービスを構成する部品の確認
(2) 全部品に対する割当資源の有無の確認
(3) 全部品に対する資源の確保
(4) サービスの開始

(1), (2) は，サービスを要求するユーザからサーバに向かうパス上の各部品について，利用可能な資源の量を計算し，その情報を巡回させることで実現可能である. ここでは，簡単のため，サーバのみについて考える.

正しいアドミッションコントロールのためには，各時点における未使用資源の把握が不可欠であるため，ここでは，ある部品における現在の未使用資源の量 (%) を変数 *unused* で管理する. 新たなサービス生成要求 req を最小品質値 (req.qmin) で提供する場合に必要な資源量を Res(req.qmin) とすると，受入れ可能かどうかの条件は $unused \geq Res(req.qmin)$ となる. 受け入れ可能な場合，提供可能な最良の品質値をを決定し (bestoffer(req,unused))，*unused* から減じる. また，既存サービスの品質値を上げ下げする場合も，*unused* の調整が必要である. 例えば，品質値を 1 増加させる場合 (r?id!Upgrade?q)，Res(q+1)-Res(q) 分の資源が新たに必要である. 一方，r?id!Degrade?q により，品質値を 1 減少させる場合，Res(q)-Res(q-1) 分の資源が *unused* に戻される.

以上より，サーバにおけるアドミッションコントロール Admission は，以下のように記述できる.

```
Admission[n]:=
var req, unused in
?unused:=100%
loop
  n?id?req; ([Res(req.qmin) <= unused]->
      n!id!Accepted!req!bestoffer(req,unused);
      ?unused:= unused-Res(bestoffer(req,unused))
    [] [Res(req.min) >= unused]->
       n!id!Rejected
    )
[]  r?id!Upgrade?q[Res(q+1)-Res(q) <= unused];
      ?unused:=unused-(Res(q+1)-Res(q))
[]  r?id!Degrade?q;
      ?unused:=unused+(Res(q)-Res(q-1))
endloop
```

(bestoffer(req,unused) は，要求と未使用資源に対し最良の品質値を返す関数)

QoS 制御　ソフトリアルタイム環境では，他のアプリケーションによる資源の不足などにより，現在の品質を維持することが困難になる場合が生じる. ユーザが要求するサービスの品質に幅を持たせる場合，あらかじめ与えられたサービス間の優先順位に基づいて，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，適当な時間間隔で，各部品において要求品質が保たれているかどうかをチェックするためのモニタリング機構が必要になる.

モニタリング機構は，各部品から適当な時間間隔 T の間，時間制約内に終わらなかった処理の全体に対する割合（**QoS 達成率**と呼ぶ）を計算する. モニタリング機構は以下のように記述できる.

```
Monitor[n,r,s](id) :=
n!id!Accepted!any!any;
loop
```

```
  var cnt,mcnt in
      ?cnt:=0; ?mcnt:=0;
   loop
          r!id!Success; cnt:=cnt+1
       [] r!id!Miss; ?mcnt:= mcnt+1
    endloop
  [> wait(T); s!id!Result!(mcnt/cnt)
   endvar
  endloop
```

サーバにおけるQoS制御は，クライアント1〜kに提供しているサービスのQoS達成率 (r!i!Result?rate_i) を受け取り，これらをもとに，ある方針で品質値を上下させることで実現する．たとえば，あるサービスのQoS達成率が70%を切ったら，サーバの負荷が高いと考え，どれか一つのサービスの品質値を下げたり，逆に，全てのサービスでQoS達成率が100%であれば，未使用資源量の範囲内で，どれか一つのサービスの品質を上げる，などの方針が考えられる．以下にQoS制御の例を記述する．

```
  QoS[r,s] :=
  loop
    ( s!1!Result?rate_1;exit
|||  ...
|||   s!k!Result?rate_k;exit )
>>(
   [min(rate_1,...,rate_k) < 0.70]->
      (r!1!Degrade!any [] ... [] r!k!Degrade!any)
[] [rate_1==1.0 and ... and rate_k == 1.0]->
      (r!1!Upgrade!any [] ... [] r!k!Upgrade!any)
  )
  endloop
```

以上よりQoSシナリオの全体は，次のように記述される．

```
Scenario[n,r]:=
hide s in
    Admission[n]
|[n]|
  ( Monitor[n,r,s](1) ||| ... ||| Monitor[n,r,s](k) )
|[s]|
    QoS[r,s]
```

上記では，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，異なる方針に基づいたQoS制御法に変更することも可能である．

4 LOTOS^{T+} 仕様の実装

我々は，文献[8]において，LOTOSのある時間拡張で書かれた動作仕様の実装法を提案している．この時間拡張LOTOSでは，イベントおよびプロセス呼び出しに，プロセス生成時からの経過時間に基づいた時間制約を記述できる．例えば，`P:=a{1,3}; P{4,4}`という記述では，プロセスPの生成後，1〜3単位時間の間にイベントaが実行でき，時刻4にPの呼出しが実行される．各イベント系列は，我々が開発したリアルタイムスレッド機構であるRT-PTL[1]のスレッドに割当てられ，イベントに指定された時間制約から，スレッドの実行開始時刻，デッドラインを定め，スレッド間でEDF (Earliest Deadline First) によりスケジューリングしている．

文献[8]の時間拡張LOTOSでは，イベント，プロセス呼出しの実行可能時刻が，親プロセスの生成時刻からの経過時間として静的に決定できるのに対し，本稿で定義したLOTOS^{T+}では，イベントの時間制約が満足するかどうかはそのイベントがアクティブになってからの時刻により決まる[3]．

このため，LOTOS^{T+}の実装では，時間制約付イベント系列が割当てられたスレッドは，イベントが新たにアクティブになるたびに，その実行可能時刻を計算し，スレッドの実行開始時刻およびデッドラインとして設定する（実装法の詳細は文献[8]と同様である）．

5 実験および考察

試作したLOTOS^{T+}仕様のコンパイラを用いて，(1) EDFスケジューリングの効果，(2) QoS制御の効果について調べた．実験には，BSD/OS3.1が稼働するPCマシン (PemtiumII, 300MHz) を使用した．

EDFの効果　EDFによるスケジューリングの効果を調べるため，異なる時間制約を持つ複数のイベントを並列実行した時の，時間制約が満たされない割合を測定した．実験では，以下のように，処理時間Tを要するイベントa_iを繰り返し実行するプロセスP_iを周期1000msで10個並列実行した (必要な処理量の合計は10*T)．また，各P_iにおけるイベントの時間制約を$a_i@t[t \leq i*100]$のように設定した．

$P_1 \,|||\cdots|||\, P_{10}$
where
$P_i :=$ *loop*
$a_i@t[t \leq i*10]; wait(100-t)$ [> $wait(100)$
endloop

イベントの処理時間Tを75ms〜90msで変化させた場合の実験結果を表2に示す．

表2によれば，CPUの利用率が75%までは(T=75ms)，イベント実行における時間制約は100%満たされた．また，利用率が80%を越えると，EDFの性質上，最もデッドラインの近いイベントのデッドラインミスがそれ以降

[3] 例えば，$a;b@?t[t \leq 3]$では，イベントbはaの実行後3秒以内に実行可能であり，aが実行されて初めてその実行可能時刻が決まる

表 2: EDF におけるデッドラインミス率

T (ms)	75 以下	76〜80	81〜85	90
ミス率 (%)	0	10	90	100

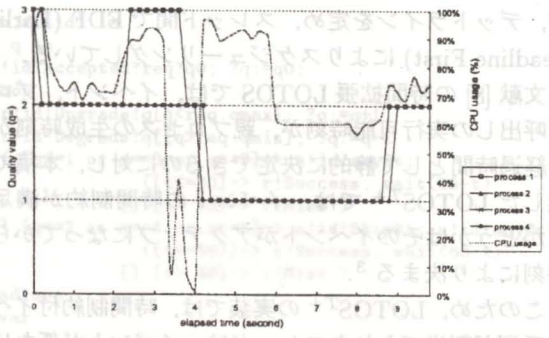

図 2: QoS 制御時の品質値の移り変わり

のイベントのデッドラインミスを誘発し，急激にミス率が高くなった．過負荷時には，QoS 制御により CPU の使用率が 75%以下になるよう調整することが望ましい．

QoS 制御の実行例　最後に，QoS 制御用のシナリオを追加した次のようなシステムを 3 節の方法に従って記述し実験を行った．

(i) 4 つのプロセスを周期 33ms で並列実行．
(ii) 各プロセスは，各周期で，(1) フレームの読込み，(2) フレームのデコード，(3) デコードされた画像の表示，を順に実行する (各処理は 2.5ms 要する)．
(iii) 品質値 q の値に応じて，QoS 制御を行う．q=3 の時は 30fps, q=2 の時は 15fps, q=3 の時は 10fps になるよう，(ii) の (2), (3) の処理を省略する．
(iv) あるプロセスにおいて，連続 10 周期にわたって，フレームの処理が時間内 (q=3 の時 33ms, q=2 の時 66ms, q=1 の時 100ms) に終われなかった場合，あるプロセスの品質を 1 減らす．
(v) 全てのプロセスでフレームの処理を 30 周期以上連続で時間内に終えれた場合，どれか一つのプロセスの品質を 1 増加させる．

実験では，最初全てのプロセスには最高品質 3 を与えた (各プロセスでは品質値が 1 まで下がっても良いとする)．システムを実行後，3 秒後に，計算機全体に人為的に負荷をかけた場合の，経過時間に対する，プロセスの品質値の移り変わりを図 2 に示す (図中の CPU 利用率は，当該システムが使用している CPU 時間の全体に対する割合)．

6 おわりに

本稿では，E-LOTOS[6] の時間構文を取り入れた LOTOS^{T+} による，マルチメディアシステムの記述法を与えた．提案する手法では，LOTOS の持つマルチランデブ機構を利用して，システムを制約指向スタイル [9] で記述することにより，資源の動的な割当てを指定した QoS シナリオを，システムの動作や性質とは別に記述できる，という特徴を持つ．試作したコンパイラを用いた実験により，シナリオに指定した通りの QoS 制御を実現し，かつ高速に動作する目的コードを得られることを確認した．これにより，QoS シナリオを変更するだけで様々な QoS 制御方針の効果を確認，評価するといった，ラピッドプロタイピングが可能になった．

ビデオ会議などの複数ノードへの同時配信を含むアプリケーションを LOTOS^{T+} で記述し，その実装を行うことが今後の課題である．

参考文献

[1] 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: 49 – 54 (1997).

[2] Campbell, A., Aurrecoechea, C. and Hauw, L.: “A Review of Qiority of Service Architectures”, *Proc. 4th IFIP Int. Workshop on Quarity of Service (IWQOS'96)* (1996).

[3] 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*: 31 – 45 (1995).

[4] Fischer, S. : “Implementation of multimedia systems based on a realtime extension of Estelle”, *Formal Description Techniques IX*: 310 – 326 (1996).

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

[6] ISO: “Final Committee Draft 15437 on Enhancements to LOTOS”, *ISO/IEC JTC1/SC21/WG7* (1998).

[7] Leonard, L. and Leduc, G. : “An Introduction to ET-LOTOS for the description of time sensitive systems”, *Computer Networks and ISDN systems*, 29(3): 271-292 (1997).

[8] 辰本比呂記，安本慶一，安倍広多，東野輝夫，松浦敏雄，谷口健一：“リアルタイムスレッドを用いた実時間 LOTOS コンパイラの設計と実装”，マルチメディア，分散，協調とモーバイル (DICOMO'98) シンポジウム論文集 (1998).

[9] Vissers, C. A., Scollo, G. and Sinderen, M. v.: “Architecture and Specification Style in Formal Descriptions of Distributed Systems”, *Protocol Specification, Testing, and Verification VIII*: 189 – 204 (1988).