協調型同期方式によるメディア間同期とその実現方式

端山貴也† 清木 康††

マルチメディアを対象としたメディア間の時間的関係は，複数のメディア・データの論理的な時間的関係，および，メディア・データを再生するプロセスの物理的な時間的関係に分類できる．論理的に指定された時間的な関係は，再生時に物理的なプロセス間同期に変換される．本論文では，メディア・データの論理的な時間関係を元に再生を行うときに，その曖昧性を除去しプロセス間同期に変換可能な時間関係の記述方式を示す．そして，与えられたメディア・データ間の時間的関係より，メディア・データ再生時のプロセス間の同期機構を実現する方式を示す．本論文で示すプロセス間同期機構の実現方式は，協調型処理方式に基づく協調型同期方式である．本方式において，プロセスは，与えられた時間的関係の情報に基づき，他のプロセスと協調し処理を行う．本方式では，それぞれのプロセスは，独立に動作するため，処理対象のメディアに応じた最適な動作方針を選択可能である．

An Inter-media Synchronization Based on the Coordinated Synchronization Method and Its Implementation

TAKANARI HAYAMA† and YASUSHI KIYOKI††

Temporal relationships among multimedia document components can be classified into logical temporal relationships of multimedia data and physical temporal relationships of processes which render the multimedia data. Logically defined temporal relationships are converted to synchronization among processes to render the multimedia data. However, there often exists a gap between them, which is caused by the vague explanation of logical temporal relationships. In this paper, we present a method to define logical temporal relationships which omit vague explanation of the relationship. Additionally, an implementation method for the inter-process synchronization mechanism which realizes the physical temporal relationships from logical temporal relationships is presented. The synchronization mechanism is based on the synchronization method in the coordinated computational model. Each process refers to the temporal relationships, and plays multimedia data by coordinating with other processes. This method allows each process to select the best policy to process media.

1. はじめに

メディア間の同期問題は，マルチメディア処理における重要な問題の1つである．マルチメディア・ドキュメントは，複数のメディアを時間的，および，空間的に関係付けることにより作成される．複数のメディアを時間的な関係により結び付けるときに，メディア間の同期問題が生じる．

メディア間の同期関係は，論理的なメディア・データ間の時間的関係と，物理的な再生時の時間的関係に分類できる．論理的な関係とは，異なるメディア間の同時再生，連続再生など，メディア・データの内容により決まる時間的関係である．物理的な関係とは，メディアの論理的な関係により定められるメディア再生プロセスの時間的関係である．メディア再生プロセス間の時間的関係は，プロセス間の同期問題として扱われる．論理的なメディア間の関係を記述するために，いくつかの方法が提案されている[1),7)]．それらは，メディアを1つのインターバルととらえ，そのインターバルの関係により，時間的な関係を表すものである．この方法で指定された時間的な関係を物理的なプロセス間の同期として解決しようとしたとき，データとプロセスが時間的な制約を正確に守ることを要求する．そのため，時間的な制約を保証できないシステムでは，論理的な関係と，実際に再現された物理的な関係の間に大きな予測不可能な差異が生じる．

この問題を解決するために，本論文では，複数のメ

† 筑波大学工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 慶應義塾大学環境情報学部
Faculty of Environmental Information, Keio University

ディアそれぞれに任意の同期点を設け，各メディアの同期点の関係により時間的関係を記す方式を示す．本方式により，論理的な関係を物理的な関係として再現する際の差異を予測可能にすることができる．

また，メディアの再生時に，メディア間の時間的関係は，メディアを再生するプロセスの同期関係となる．本論文では，与えられた時間的関係に基づき，メディア再生時のプロセス間同期を，協調型同期方式に基づき実現する方式について述べる．本方式において，各プロセスは，独立して動作する．そのため，処理の対象とするメディアに応じて，プロセスは，最適な動作方針を選択できる．また，本方式は，マルチメディア・システムの柔軟な実現を可能とする．

本論文では，まず，任意の同期点を用いた論理的な時間関係を記述するための方式を示す．そして，その論理的な関係を再現するための協調型処理方式に基づくプロセス間同期機構を示す．さらに，提案方式に基づく同期機構の実現方式を示し，同期機構の満たすべき性能について述べる．最後に，実現した同期機構とその性能評価について述べる．

2. メディアと同期

様々な時間的な論理関係を表すためのモデルが提案されている[1),6),7)]．文献 1) では，任意の 2 つのインターバルの論理関係を *equal*, *before*, *after*, *meets*, *met_by*, *overlaps*, *overlapped_by*, *during*, *contains*, *starts*, *started_by*, *finishes*, *finished_by* の 13 種類の時間的関係に分類している（**図 1**）．13 の時間的関係の多くは，一方のインターバルの終了，あるいは，開始が，もう一方のインターバルに比べ，任意の時間ずれた関係にある．この任意の時間ずれた関係は，空のインターバルを導入することで，容易に記述可能となる[5),7)]．空のインターバルを用いることにより，すべてのインターバルは，必ず，いずれかのインターバルと同時に開始，あるいは，終了する関係になる．文献 7) では，空のインターバルを用いることにより，13 種類の時間的関係を *equals* と *meets* の 2 種類で表す方式が提案されている[7)]．

インターバルという単位で，時間的関係を表す場合，インターバルの開始時と終了時のみが同期点として指定可能である．インターバル単位による記述は，抽象度が高く，マルチメディア・ドキュメントを記述するうえで有用な記述方式である．しかし，抽象度が高いために，物理的な同期関係に変換する際に問題が生じる．同期関係は，インターバル間の相対関係で表されるが，実際に再生されるときは，すべて実時間で動作

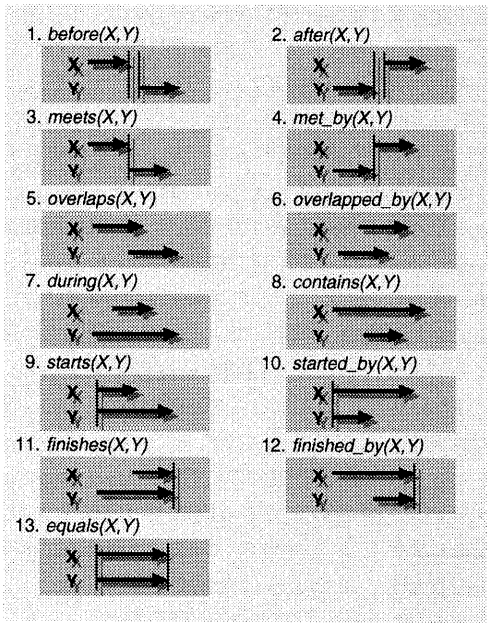


図 1 13 の時間的関係
Fig. 1 13 temporal relationships.

することを要求されるうえに，インターバル内で演奏される曲や動画像の再生時間の調整をあらかじめ正確に行う必要がある．また，リップ・シンクを行う場合，それぞれのメディアは，必ず周期的，かつ，実時間制約のもとで再生されることが要求される．演奏時に，動的にリップ・シンクを制御する場合，インターバルの同期関係からでは，同期をとるべきチェック・ポイントに関する情報を得られない．インターバルの関係のみで論理的な時間関係を表す方式は，再生時に必要な情報を十分に記述できない．また，再生時の環境や状況に応じて，演奏時間は変化する．そのため，あらかじめ用意された論理的な同期関係を満たす保証はなく，再生時に動的に調整を行う必要がある．

以上のような観点から，論理的な同期関係と物理的な時間関係の間の差異が小さくなるような論理的な同期関係の記述方式が必要である．そこで，本論文では，インターバル中の任意の点の関係により時間的関係を表現する方式を示す．インターバル単位ではなく，インターバル中の任意の点の関係により時間的関係を表現することにより，再生時に，柔軟にメディア間の同期処理を行うことが可能となる[3)]．これにより，論理的に同期する必要のある点を実際に同期させることが可能となる．また，インターバル内で再生されるメディアを同期させるとき，間延びさせるのか，あるいは，短縮するのかを自由に記述することが可能になる．

ここで，2 つのインターバル X と Y があるとする．

同期点を任意に設定可能であるとすると，インターバル X と Y の同期点は，次のように定義される．

$$X = \{X_{start}, X_1, X_2, \ldots, X_i, \ldots, X_n, X_{finish}\}$$
$$Y = \{Y_{start}, Y_1, Y_2, \ldots, Y_j, \ldots, Y_m, Y_{finish}\}$$

X_i や Y_j は，同期点であり，X_{start} を起点としたときの相対時間である．X_{start}，および，X_{finish} は，インターバルの開始と終了点であり，従来の同期関係と同様である．X_i は，新たに設ける任意の同期点である．X_{start} と X_{finish} の間の任意の時点に設定される．時間的関係（同期）は，これらの同期点の関係によって表す．ここで，同期点の関係を表すため，新たに *Meet* 関係を導入する．*Meet* 関係は，*meets* 関係とほぼ同等のものである．ただし，*meets* 関係がインターバル全体の関係を表すのに対し，*Meet* 関係は，インターバルの任意の点である同期点の関係を表す．

Meet 関係により従来の *starts*, *finishes*, *meets*, *equals* 関係は，次のように表現できる．

- $starts(X,Y)$ （X と Y の同時開始）
  $Meet(X.X_{start}, Y.Y_{start})$.
- $finishes(X,Y)$ （X と Y の同時終了）
  $Meet(X.X_{finish}, Y.Y_{finish})$.
- $meets(X,Y)$ （X に続いて Y を開始）
  $Meet(X.X_{finish}, Y.Y_{start})$.
- $equals(X,Y)$ （X と Y は同時に開始し終了）
  $Meet(X.X_{start}, Y.Y_{start})$,
  $Meet(X.X_{finish}, Y.Y_{finish})$.

equals 関係のように開始点と終了点を同時に指定する場合，複数の *Meet* 関係を記述する．このように記述することにより，メディア再生時の同期制御と論理的な同期関係の差異を小さくすることが可能となる．また，インターバル単位の時間的関係による記述方式で記述できることは，*Meet* 関係によって記述できる．そのため，あらかじめ抽象度の高いインターバル単位の時間的関係を記述したうえで，*Meet* 関係による記述に置換すること可能である．これにより，抽象度の高い記述方式でマルチメディア・ドキュメント全体を記述してから，実際にどのような形で再生するかを記述できることになり，マルチメディア・ドキュメントを有効に記述可能とする．

3. メディア間同期の制御方式

メディア・データの論理的な同期関係は，その再生時に，物理的な同期関係に変換される必要がある．メディア・データの再生は，メディア再生アプリケーション（プロセス）により行われる．そのため，この物理的な同期関係の問題は，プロセス間の同期の問題となる．ここでは，メディア・データの再生を対象としたプロセス間の同期制御方式を提案する．

3.1 関連研究

マルチメディア・システムの拡張性を高めるために，複数の比較的小さなアプリケーションを組み合わせ，マルチメディア・システムを構築する方式がある．このような方式に基づくシステムには，アプリケーションの同期制御にマスタ・スレーブ方式の機構を用いたMAEstroがある[2]．この方式では，時間や同期情報を管理するマスタが，スレーブである各アプリケーションの実行と終了を管理する．通信量が少ないものの，マスタが実時間制約などを管理する必要があるため，マスタの負担は大きい．

マスタを含め，すべてをアプリケーションとして実装する場合，負荷が大きくなるため，マスタをミドルウェアとして実装する方式もある[9]．また，資源予約と実時間の周期スレッドを用いた同期機構についての研究も多くされている[4),8),11]．いずれの場合も，スレーブは，与えられたCPU時間とメモリ資源内に与えられた処理を行う．スレーブ間の同期タイミングは，スケジューラにより制御される．スレーブは，与えられたCPU時間およびメモリ資源と，行うべき処理の関係を理解する必要がある．

3.2 協調型同期方式

プロセス間の独立性を高めることにより，実行環境に応じて，プロセスは最適な動作方針を選択可能になる．プロセスの動作方針は，その扱うメディアに応じて異なるため，その動作方針も多岐にわたる．本論文では，プロセスの独立性を高める同期処理方式として，協調型処理に基づく**協調型同期方式**を示す[3]．

すべてのプロセスが平等な場合，対象とするメディアの組合せに応じた最適な同期制御を行うことができない．たとえば，動画像と音楽を扱う場合，音楽に動画像を同期させることが多い．これは，人の感覚が，動画像のフレームが飛ばされることよりも，音楽が途切れたり，飛ばされることの方に，人が敏感であるからである．しかし，状況により，音楽より動画像の方が重要な場合もありうる．この場合，動画像が途切れないように再生されなければならない．

この問題を解決するためには，プロセスを**優先度**により区別化する．優先度を同期の強制のためのヒントとして用いる．ここで，プロセスAとBが同期して動作する場合を考える．プロセスAがプロセスBよりも高い優先度を持つ場合，プロセスBは，プロセスAに追従して動作する．プロセスAが，仮に，プロセスBよりも先に同期点に到着した場合，その到

着がプロセスBに知らされる．このとき，プロセスBは，可能な限り次の同期点に進む．しかし，同期点に進むことがその時点で不可能な場合，これを無視，あるいは，保留する．これにより，動画像と音楽において，動画像再生プロセスは，フレームを飛ばすが，音楽再生プロセスは，音符を飛ばさない制御を行うことが可能となる．

協調型同期方式は，マスタ・スレーブ方式に比べ，プロセスが互いに直接通信するため，分散環境における通信の複雑さとオーバヘッドが増す．しかし，単一計算機上で実行する場合は，本方式の実現を，IPCにより実現することで，通信量をマスタ・スレーブ方式と同等にすることが可能である．また，協調型同期方式では，プロセスの組合せ方により，マスタ・スレーブ方式と同様の同期制御を行うことが可能である．この場合，スレーブとなるプロセスの優先度を低くし，必ず，マスタとなるプロセスに従うようにする．

4. 協調型同期方式に基づく同期機構の実現方式

同期機構は，様々なメディアを扱うプロセスを対象とする．プロセスは，扱うメディアにより分類できる．ここでは，プロセスの分類に基づいた同期機構を示し，協調型同期方式に基づく同期機構の実現方式を示す．

4.1 プロセスの分類

マルチメディア・アプリケーションは，**図2**のように分類可能である．これらのアプリケーションを複数組み合わせることにより，マルチメディア・アプリケーションが構成される．

マルチメディア・アプリケーションのコンポーネントとなりうるプロセスは，2つのタイプ，**受動的プロセス**と**能動的プロセス**に大別できる．対話を担うGUIを持つ事象駆動型のものは，受動的プロセスにあたる．また，音楽，動画像，静止画像の再生を行うプロセスは，能動的プロセスにあたる．さらに，能動的プロセスは，扱うメディアが時間の概念を必要とするか否かにより，**連続**と**非連続**に分けられる．連続メディアを扱うプロセスには，**周期的**に毎秒30フレームを再生する動画像再生プロセスと，MIDIシーケンサのように**非周期的**に連続メディアを扱うプロセスがある．

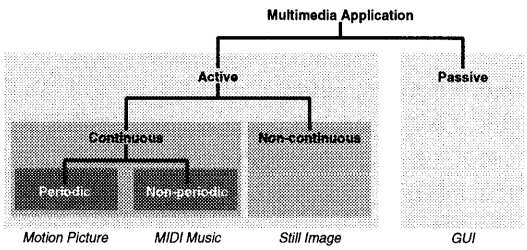


図2 マルチメディアを対象とするアプリケーションの分類
Fig. 2 Classification of multimedia application.

静止画像のような非連続メディアを扱うプロセスは，時間の概念を持たない．このようなプロセスにも表示の開始と終了はあるが，この開始と終了は，連続メディアを扱うプロセスと同時に動作しない限り，時間的制約を受けることがない．これは，扱うメディア自体に，時間の要素がないためである．また，GUIのような受動的なプロセスにおいても，プロセスが受けるイベントの一種としての時間はありうるが，自ら状態遷移をしない．そのため，非連続メディアを扱うプロセス同様，時間の概念を持たない．しかし，画像の表示や，ユーザの入力に対する反応などを指定時間内に行う時間的制約が考えられる．これらの特性は，メディアに応じて一般化することが難しい．このような問題は，実際にプロセスが動作する環境に応じて，環境固有の問題として，解決しなければならず，本方式では，その対応について対象としない．

4.2 同期機構

以上のように分類されたプロセスは，与えられた同期情報に応じて同期する．同期は，プロセスのグループで行われる．グループ内で行われる一連の同期を**セッション**と呼ぶ．

ここで，プロセスがネットワーク上の計算機に分散することを想定する．複数の計算機上において同時に動画像を再生する場合や，特定の計算機にのみ付随する特殊なデバイスを利用する場合がある．これらの場合，プロセスは，ローカルとリモートに分散するグループ内のプロセスと同期する必要がある．加えて，協調型同期方式では，同期関係の情報をグループ内の全プロセスが知る必要がある．各プロセスは，この情報を取得し，解析し，どのプロセスと同期を行うのかを理解する必要がある．

本方式では，このような複雑な処理をプロセスに代わって処理し，プロセスの同期処理を支援するために，**同期マネージャ**を用いる（**図3**）．同期マネージャは，時間的関係を記述した同期情報を元に，各プロセスの同期処理を，他のプロセスと協調し支援する．同期マネージャは，操作プリミティブとして，同期処理支援のための機能を提供する．同期マネージャをプロセスに組み込むことにより，プロセスは，容易に同期に参加可能となる．プロセスは，操作プリミティブを介して同期のために最低限必要な指示を同期マネージャに与えることにより，容易に，同期処理を行う．

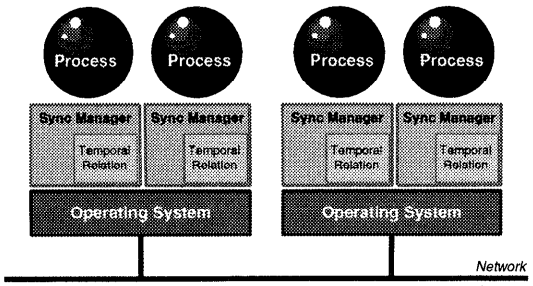


図3 同期機構とプロセス
Fig. 3 Synchronization mechanism and processes.

表1 同期のための操作プリミティブ
Table 1 Primitives for synchronization.

操作プリミティブ	機能
SYNC	同期処理
SAVE_CONTEXT	コンテキストの保存
SET_UPDATE_VAR	強制的な同期時に自動更新する変数
BLOCK_SYNC	強制的な同期のブロック
UNBLOCK_SYNC	強制的な同期のブロック解除

同期マネージャは，同期処理時に，他の同期マネージャとの協調，および，プロセスの状態の管理を行う．同期のインターバルがメディアに依存する単位（フレーム数など）の場合，プロセスの同期点への到着の判断は，プロセス自身が行う．そのため，プロセス自身が同期点への到着を判断し，同期マネージャに対し同期処理を依頼する．同期点への到着をプロセスが判断するため，同期マネージャは，次の同期点までのインターバルと同期点の残数を，同期関係の情報より抽出し，プロセスに知らせる．ただし，同期のインターバルが実時間を単位としている場合，同期マネージャがプロセスの同期点への到着を判断することが可能である．必要に応じて，プロセスは，同期マネージャに同期点への到着の判断を依頼する．

本方式では，同期処理を柔軟に行うため，プロセスの優先度が低い場合，割込みにより強制的に同期処理を行うことを可能にする．この場合，強制的な同期を要求されるたプロセスは，次の同期点に遷移しなければならない．このような強制的な同期を行うために，プロセスのコンテキストの保存と更新を行う．強制的な同期処理後，同期マネージャは，保存済みのコンテキストを用いプロセスの同期処理終了直後の状態に戻す．また，ヒープの状態を変化させる必要がある場合は，ヒープの更新を行う．プロセスは，強制的な同期を行う際に，重要な変数の更新中などのクリティカル・リージョンにある場合がある．そのため，クリティカル・リージョンにプロセスがある間は，強制的な同期を一時的にブロック可能とする．

```
while (not finished processing) {
  if (arrived at synchronization point)
    SYNC;
  /* save context */
  SAVE_CONTEXT;
  if (no more synchronization)
    exit;

 /* media processing comes here */
 BLOCK_SYNC;
 /* critical region */
 UNBLOCK_SYNC;
 /* rest of the media processing comes here */
}
```

図4 操作プリミティブを組み込んだプログラム例
Fig. 4 An example of a program with the primitives.

本方式において，これらの機能を実現するための同期マネージャの操作プリミティブを**表1**に示す．連続メディアを周期的に処理するプロセスに対し，操作プリミティブは，**図4**のように組み込まれる．

4.3 同期機構とプロセス

受動的なプロセスに操作プリミティブを組み込む場合には，SAVE_CONTEXT() によりイベント・ハンドラを保存し，同期処理後，イベント処理を継続して行うよう．利用者との対話に応じて，同期を制御する場合には，BLOCK_SYNC() および UNBOLOCK_SYNC() を用いることにより，同期処理を制御する．非連続メディアを扱うプロセスの場合，同期は，開始時と終了時にのみ行われる．そのため，たとえば，静止画像を表示するプロセスでは，表示直前と表示直後に，SYNC() を用いる．連続メディアを扱う場合は，図4のように，メディア処理を開始する前，メディア処理のループ中，そして，メディア処理後に同期を行うようにする．このように，本方式による同期機構は，先に分類したプロセスの処理の中に，操作プリミティブの形で，柔軟に組み込むことができる．

5. 同期処理の時間的制約

連続メディアは，複数のデータ・エレメントを連続的に出力することにより実現される．出力される任意の2つのデータ・エレメント間のインターバルは，メディアにより異なる．同期制御を行う際，このインターバルは，非常に重要である．プロセスは，決められたインターバルの間に，データを読み込み，復号化し，そして，出力する．他のプロセスとの同期処理は，イン

ターバル内でデータ・エレメントを処理した残りの時間で行われる．連続メディア・データの再生にジッターを生じさせないためには，同期処理が，インターバルの残りの時間内に処理を終える必要がある．同期処理が与えられた時間内に終了しない場合，再生時にジッターが生じる．

同期のインターバルが短ければ，メディア間のずれは小さくなるが，細かいジッターが生じる可能性がある．一方，同期のインターバルが長い場合，同期処理のための時間を多く確保できることが期待できるが，メディア間の処理のずれが大きくなる可能性がある．同期処理に必要な時間より，可能な同期のインターバルを求めることができれば，ジッターを生じさせない最適な同期処理を行うことが可能である．また，期待する同期のインターバルより，同期処理に与えられる時間を求められれば，同期の処理時間の目標値とすることが可能である．

ここでは，最適な同期のインターバルや同期の処理時間の目標値を求めるための評価式を示す．この評価式により，最適な同期処理を行うための指針を得ることが可能となる．同期処理のためのCPU時間は，プロセスの扱うメディア・データの処理方式により異なる．そのため，評価式は，メディアごとに求める必要がある．時間的な制約を受ける連続メディアを扱うプロセスの入力と出力の特徴に応じて分類すると，次のようになる．

- 可変長の入力を受け，周期的に出力するプロセス
- 固定長の入力を受け，周期的に出力するプロセス
- 可変長の入力を受け，非周期的に出力するプロセス

ここでは，それぞれの代表的な例として，MPEGデコーダ，PCMプレーヤ，および，MIDIシーケンサの3種類を選択し，同期のインターバルを決定するための評価式について述べる．連続メディア以外を扱うプロセスについては，4.1節において述べたように一般化が困難なため，ここでは，対象としない．しかし，同期処理の時間的制約が最も厳しい連続メディアを対象とするプロセスの制約を満たすことができれば，他のプロセスについても，その制約を満たすことが可能である．

5.1 同期機構に与えられた時間

ここで，同期点の数を m とし，$(j-1)$ 番目と j 番目の同期の間のインターバル（sec）を I_j とする．また，同期処理のために同期機構に与えられた処理時間（sec）を N_j とする．j 番目の同期までに経過した相対時間 P_j は，

$$P_j = \sum_{k=1}^{j} I_k$$

となる．

もし，同期機構が $\min(N_j)$ 内に同期処理を終了することができれば，同期処理によりプロセスの動作が妨げられないことを意味する．つまり，与えられたインターバルより，$\min(N_j)$ を求め，その時間内に同期機構が同期処理を終了できればよい．また，同期機構のCPU時間 N があらかじめ分かっている場合には，$\min(N_j)$ が N 以上になるように，I_j を選択すればよい．複数のメディアの同期処理を行う場合は，すべてのメディアについて，$\min(N_j)$ が N 以上になるような条件の I_j を求めることにより，プロセスの処理に影響を与えないような同期処理を行うことが可能となる．

5.2 可変長の入力を受け周期的に出力するプロセス

パラメータの定義を**表2**に示す．インターバル内で表示されるフレーム数は，正数である必要がある（$I_j \cdot F \in N^*$）．ここで，N_j は，次のように定義できる．

$$N_j = I_j - \sum_{i=P_{j-1}\cdot F}^{P_j \cdot F - 1} (d_i + v_i). \qquad (1)$$

動画像におけるフレームの展開時間は，復号化のための計算量がフレームごとに違うことが多い．特に，MPEGデコーダでは，Iフレーム，Pフレーム，および，Bフレームにより，処理時間が異なる．そのため，復号化，および，表示のためのコストをそれぞれ変数としておく必要がある．

5.3 固定長の入力を受け周期的に出力するプロセス

パラメータの定義を**表3**に示す．PCMプレーヤのようなアプリケーションでは，一度に転送されるデータ量にのみ時間が依存する．そのため，N_j は，以下のように定義される．

$$N_j = I_j \left(1 - f \cdot w \cdot c \left(\frac{1}{W} + \frac{1}{R}\right)\right). \qquad (2)$$

5.4 可変長の入力を受け，非周期的に出力するプロセス

動画像や音声の出力は，基本的に周期的である．しかし，MIDIシーケンサのようなアプリケーションは，非周期的，かつ，実時間でデータを出力する．

表4は，評価式のパラメータの定義である．ここで，すべての同期点 $j\,(j = 1, \ldots, m)$ において，$e_i = P_j$ を満たすような e_i が存在することを仮定する．

表2 評価式のパラメータ（1）
Table 2 Parameters for evaluation (1).

Variable	Description
F	Frame rate (fps)
d_i	The time for reading and extracting the i-th frame (sec)
v_i	The time for displaying the i-th frame (sec)

表3 評価式のパラメータ（2）
Table 3 Parameters for evaluation (2).

Variable	Description
f	Sampling frequency (Hz)
w	Sampling quantization (bits)
c	Number of channels
R	Data reading speed (bps)
W	Data writing speed to the audio device (bps)

表4 評価式のパラメータ（3）
Table 4 Parameters for evaluation (3).

Variable	Description
e_i	The absolute time for the i-th MIDI event (sec)
b_i	Number of bits of the i-th MIDI event (bits)
W	Data writing speed to the MIDI device (bps)

$$N_j = e_i - e_{i-1} - \frac{b_{i-1}}{W} \qquad (3)$$

なお，W は，MIDI 規格の機器間のデータ転送速度より，31,250 bps となる．

6. 協調型同期方式に基づく同期機構の実現

本論文で述べた協調型同期方式とその実現方式に基づき，マルチメディア・システムを構成するためのプロセス間同期支援システム NAMI（Network-oriented Administrator for Multimedia Integration）を設計し，実現した．NAMI システムは，協調型同期方式に基づき，プロセス間の同期を図り，メディア間の時間的な同期を実現する（図5）．NAMI システムにより構成されるマルチメディア・システムは，既存のアプリケーションを含めて構成される．各アプリケーションは，それぞれ動画像や音楽などの連続メディアや，静止画像などを扱う．これらを適宜組み合わせることにより，任意のマルチメディア・アプリケーションが構成可能となる．

NAMI システムは，**情報サーバ**と**同期マネージャ**より構成される（図6）．協調型同期方式では，すべての同期マネージャが，同期情報を知る必要がある．

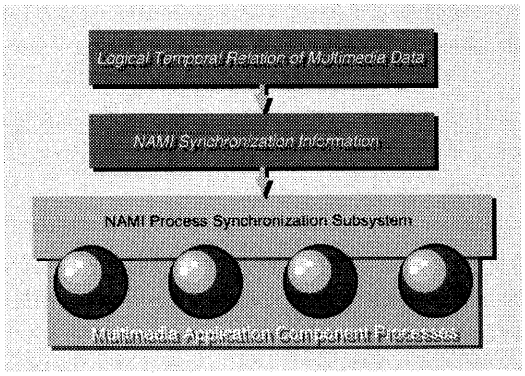


図5 メディア間同期と NAMI システム
Fig. 5 Inter-media synchronization and the NAMI system.

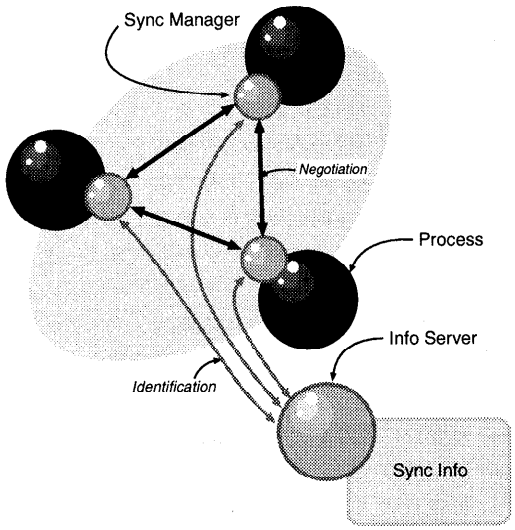


図6 NAMI システムの同期制御機構
Fig. 6 Synchronization mechanism in the NAMI system.

そのため，情報サーバは，同期マネージャが必要とする同期情報を管理し，配布する．同期マネージャは，4.2 節において述べたプロセスの同期処理を支援するモジュールである．

6.1 同期情報

セッションに参加するプロセスは，グループごとに，ユニークなグループ識別子を持ち，同期処理に必要な**セッション情報**を必要とする．セッション情報は，グループ識別子，グループ内のプロセスに関する情報，そして，同期情報より構成される．セッション情報は，情報サーバにより管理される．プロセスは，起動時に，情報サーバの場所とグループ識別子のみを必要とする．情報サーバにより，プロセスは，ネットワーク上の任意の計算機上において起動可能となる．

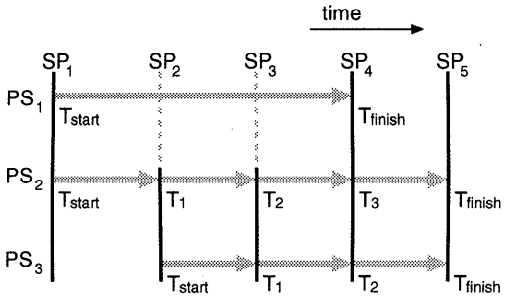


図7 同期の例
Fig. 7 An example of synchronization.

ここで，セッション情報 S は，次のように定義される．

$S_{GID} = \{PS, SP\}$.

GID は，グループ識別子である．PS は，このグループに属するプロセスの集合であり，SP は，同期関係の集合である．n 個のプロセスを含む PS は，次のように定義される．

$PS = \{PS_1, PS_2, \ldots, PS_i, \ldots, PS_n\}$.

PS_i は，以下の3つの要素から構成されるプロセス i に関する情報である．

$PS_i = \{location, k, \{T_{start}, T_{finish}, T_1, T_2, \ldots, T_k\}\}$.

$location$ は，プロセス i の起動した計算機名であり，同期マネージャが同期する際に用いられる．2番目のタプルは，同期点の数である．3番目のタプルは，2章で述べた同期点の集合であり，開始点からの相対時間とプロセスの優先度に関する情報を含む．相対時間は，プロセスがいつの時点で同期処理を行えばよいかを判断するためのものである．そのため，相対時間の単位は，秒などの時間のほか，メディアに応じて決まる単位でもよい．動画像の場合はフレームを単位とすることも可能である．次に，同期関係の集合 SP は，

$SP = \{SP_1, SP_2, \ldots, SP_j, \ldots, SP_m\}$.

と定義される．それぞれの SP_j は，$Meet$ 関係の同期点の集合である．

$SP_j = \{\ldots, PS_p.T_q, \ldots\}$.

たとえば，**図7** で現される同期は，**図8** のように記述される．

図8にあるセッション情報は，**図9** の形で情報サーバに管理される．情報サーバへのセッション情報の登録は，セッションを開始しようとするユーザによって行われる．情報サーバは，ユニークなグループ識別子を用いて，各セッションの情報を管理する．

6.2 同期マネージャ

同期マネージャは，他の同期マネージャと協調し，プロセスに対して，とるべき動作の示唆をする．プロセスは，同期マネージャからの要求に応じて，同期処理を同期マネージャに依頼する．同期マネージャは，セッション開始前に，次のようにセッション情報を入手する．

(1) 情報サーバに，プロセス識別子をもとに，自分の所在を登録する．
(2) すべての同期マネージャが情報を登録した時点で，情報サーバより，セッション情報を取得する．
(3) 他の同期マネージャと通信をし，互いを認識する．

この時点で，同期マネージャは，すべてのプロセスの所在，および，同期情報を知る．プロセスは，他のプロセスと同期をとるために，同期マネージャに同期処理を依頼する．プロセスが，プリエンプティブに同期処理を行うことを許した場合のみ，強制的な同期が行

$$
\begin{aligned}
S &= \{PS, SP\} \\
PS &= \{PS_1, PS_2, PS_3\} \\
SP &= \{SP_1, SP_2, SP_3, SP_4, SP_5\} \\
PS_1 &= \{host_1, 0, \{T_{start}, T_{finish}\}\} \\
PS_2 &= \{host_2, 3, \\
&\qquad \{T_{start}, T_{finish}, T_1, T_2, T_3\}\} \\
PS_3 &= \{host_3, 2, \\
&\qquad \{T_{start}, T_{finish}, T_1, T_2\}\} \\
SP_1 &= \{PS_1.T_{start}, PS_2.T_{start}\} \\
SP_2 &= \{PS_2.T_1, PS_3.T_{start}\} \\
SP_3 &= \{PS_2.T_2, PS_3.T_1\} \\
SP_4 &= \{PS_1.T_{finish}, PS_2.T_3, PS_3.T_2\} \\
SP_5 &= \{PS_2.T_{finish}, PS_3.T_{finish}\}
\end{aligned}
$$

図8 図7の同期の記述例
Fig. 8 An example of description of synchronization in Fig.7.

```
# The Sample Session Information
#
NPR:3:1234:InfoServerHost;
#
DEF:1:2:s:2:;
DEF:2:1:s:4:1,2,3;
DEF:3:1:s:3:1,2;
#
SYN:1::1-s,2-s;
SYN:2::2-1,3-s;
SYN:3::2-2,3-1;
SYN:4::1-f,2-3,3-2;
SYN:5::2-f,3-f;
```

図9 セッション情報の例
Fig. 9 An example of session information.

われる．同期処理を行うとき，同期マネージャは，先に述べた協調型同期方式に基づき，以下の処理を行う．

(1) 自身の優先度よりも低い他の同期マネージャに対して，即座に同期処理を行うよう要求する．
(2) 他のすべての同期マネージャに，自身の同期点への到着を告知する．
(3) 他の同期マネージャが同期点に到着するまで待つ．
(4) 次の同期点までの時間と同期の残数を計算し，制御をプロセスに戻す．ただし，強制的な同期を行った場合，要求に応じてプロセスのヒープを更新し，保存してあるコンテキストにプロセスの状態を戻す．

同期マネージャは，高い優先度を持つ同期マネージャからの同期の要求を受けた場合，その旨をプロセスに知らせる．プロセスが強制的に同期をとることを許している場合，同期マネージャは，強制的に同期処理を行う．

6.3 同期のための操作プリミティブ

NAMI システムの機能は，操作プリミティブとして提供される[3]．操作プリミティブは，情報サーバへのセッション情報やプロセス情報の登録要求，および，検索要求を発行するための操作プリミティブ，および，同期マネージャに対して同期処理の要求などを行うための操作プリミティブから構成される．同期マネージャの機能を利用するために，本質的に必要な操作プリミティブは，4.2 節において示したとおりである．NAMI システムにおいては，この操作プリミティブを**表 5** に示すように実現した．表 1 に示した操作プリミティブに加え，セッションの開始準備や同期マネージャに動作方針を伝えるための操作プリミティブを加えた．また，情報サーバの機能を利用するためには，**表 6** に示す操作プリミティブを用いる．実際に，同期マネージャの操作プリミティブを組み込んだ例を**図 10** に示す．行 16 における`NamiInit()`では，同期マネージャの初期化を行う．セッション開始は，行 32 の`NamiStartSession`において行われ，情報サーバよりセッション情報を取得し，他の同期マネージャと同期の準備を行う．行 33 の`NamiSync()`では，実際の同期処理を行う．

表 5 同期マネージャの操作プリミティブ
Table 5 Primitve operations for synchronization manager.

プリミティブ	機能
`NamiInit()`	セッションの準備
`NamiStartSession()`	セッションの開始
`NamiSync()`	同期処理
`NamiClrUpdateVar()`	同期時に更新する変数リストの初期化
`NamiSetUpdateVar()`	同期時に更新する変数リストの設定
`NamiSaveContext()`	強制的な同期からの復帰点の設定（コンテキストの保存）
`NamiBlockSync()`	強制的な同期のブロック
`NamiUnblockSync()`	強制的な同期のブロック解除
`NamiUseTimer()`	実時間同期制御の許可
`NamiCompelledSync()`	強制的な同期の許可

表 6 情報サーバの操作プリミティブ
Table 6 Primitve operations for information server.

プリミティブ	機能
`namiIsInit()`	情報サーバとの接続
`namiIsRegisterGrp()`	同期グループの登録
`namiIsUnregisterGrp()`	同期グループの抹消
`namiIsRegisterProc()`	プロセスの登録
`namiIsGetInfo()`	任意のプロセスの情報の取得
`namiIsGetSyncinfo()`	同期情報の取得

7. NAMI システムの性能

5 章においては，同期機構が満たさなくてはならない時間的制約について述べた．本論文では，カリフォルニア大バークレイ校において開発された Mpegplay2.0 をもとに同期処理の性能を評価する[10]．評価には，UNIX 上で実現した NAMI システムを用いた．評価は，NAMI システムの同期処理に要する処理時間と Mpegplay2.0 の処理時間より，5.2 節で述べた式に基づき，NAMI システムで実際に可能な同期のインターバルを求めることにより行う．

7.1 同期処理に要する時間

NAMI システムの同期処理の性能を評価するため，2 つのプロセスを，イーサネットで接続された 2 台の計算機において動作させた．異なる優先度を 2 つのプロセスに与えることにより，低い優先度を持つプロセスが高い優先度を持つプロセスに追従する形とした．なお，ここでいう優先度とは，UNIX のプロセス・スケジューリングにおける優先度ではなく，NAMI システムのプロセス同期機構における優先度を指す．本実験では，優先度の高いプロセスの同期処理に要した時間（`NamiSync()`に要する時間）を計測した．計測したのは，優先度の高いプロセスが同期処理を開始し，優先度の低いプロセスに対して割込みをかけ，そのプロセスからのメッセージを受信して同期したことを確認し，同期処理から復帰するまでの時間である．時間の計測は，同期処理の開始時点と終了時点における時間を UNIX の`gettimeofday()`関数により取得し，そ

```
 1:#include <sys/time.h>
 2:#include "NamiProc.h"
 3:
 4:myskip()
 5:{
 6:  printf("done\n");
 7:}
 8:
 9:main(argc, argv, envp)
10:     int argc;
11:     char *argv[], *envp[];
12:{
13:  int i, j, v, time;
14:
15:  /* Initializing NAMI system. */
16:  NamiInit(&argc, argv, envp, NULL,
17:           myskip, NAMI_NO);
18:
19:  if (nami_cunit != U_MSEC)
20:    exit(-1);
21:
22:  /* allow compelling sync and timer */
23:  NamiCompellingSync(NAMI_YES);
24:  NamiUseTimer(NAMI_YES);
25:
26:  /* update the variable automatically */
27:  v = i = 1;
28:  NamiSetUpdateVar(NAMI_INT, NAMI_ADD_VAR,
29:                   &i, &v);
30:
31:  /* start session now */
32:  NamiStartSession(NULL);
33:  while(NamiSync()) {
34:    if (!NamiSaveContext())
35:      break;
36:    printf("%d: wait (%d sync left)\n",
37:           i, nami_counter);
38:    while(1);
39:  }
40:
41:  exit(1);
42:}
```

図10 NAMIプリミティブを用いたプログラム例
Fig. 10 A sample program with the NAMI primitives.

の差を求めることによって行った．ここで計測した時間は，同期に参加するプロセスが積極的に同期を行った場合の最も条件の悪い場合における時間となる．

表7 と**表8** は，NAMIシステムにおける同期処理の性能示す．同期は151回行い，そのときの平均値，最大値，および，最小値を求めた．高い優先度を持つプロセスは，3種類の異なる計算機において動作させた．低い優先度を持つプロセスは，SunSS5 70 MHz上に固定した．同期処理に要した時間の最大値は，平均値の7倍近くなる．UNIXのスケジューラを用いているため，処理時間にばらつきが見られる．しかし，いずれの計算機でも，92%以上のケースで3,500 μsecの以内に同期処理を終了している（表8）．これは，同期処理時間を3,500 μsecと想定して同期のインターバルを決定したとき，ジッターが生じる可能性は，8%未満であることを意味する．

表7 2プロセスの同期処理に要した時間（μsec）
Table 7 Time consumed to synchronize between two processes (μsec).

Machine	Mean	Max.	Min.
SGI Indy 200 MHz	2,657	17,203	316
HP9000/715 50 MHz	2,998	20,433	627
Sun SS5 85 MHz	3,352	24,523	913

表8 2プロセスの同期処理に要した時間の分布
Table 8 Distribution of time to synchronize between two processes.

Machine	<3,000 μsec	<3,500 μsec
SGI Indy 200 MHz	90.73%	93.37%
Sun SS5 85 MHz	82.30%	92.72%
HP9000/715 50 MHz	82.12%	92.05%

7.2 MPEGデコーダの場合

MPEGデコーダは，フレームの復号部とフレームの表示部より構成される．MPEG I動画像のサンプル・データを用いて，実際にフレームの復号部と表示部において消費されたCPU時間を計測した．使用したサンプル・データに関する情報は，**表9** に示す．**図11** と**図12** の計測結果は，SGI Indy（200 MHz）上で計測されたものである．これらの図は，あるフレームを展開し，表示するのに要した時間を示す．各図中のDecodeは，MPEGデータの展開に要した時間であり，Displayは，表示に要した時間である．そして，Totalは，展開に要した時間と表示に要した時間の合計であり，実際に，あるフレームを表示するのに要した時間である．この実測値と5章において示した評価式(1)を用い，$\min(N_j)$ を求めた．ここで，F を30 fpsとし，I_j を500 msecに固定した．変数 d_i と v_i は，図11より抽出した．これらの値を用いて N_j を求めた結果が**表10** である．この表より，$\min(N_j)$ は，174,761 μsecであることが分かる．この値は，図7にある同期処理に要する時間の平均値値未満であることから，同期処理を15フレームごとに行った場合でも，MPEGデコーダの動作を妨げないことが分かる．

次に，I_j を1000 msecに固定し，F を変化させた場合の $\min(N_j)$ を求めた．結果は，**図13** に示すとおりである．

表9 MPEG Iのサンプル・データの仕様
Table 9 A specification of the sample MPEG I video stream.

Frame Size	352×240
Frame Order	IBBPBBPBBPBBPBB
# of Frames	148
# of I-Frames	10
# of P-Frames	40
# of B-Frames	98

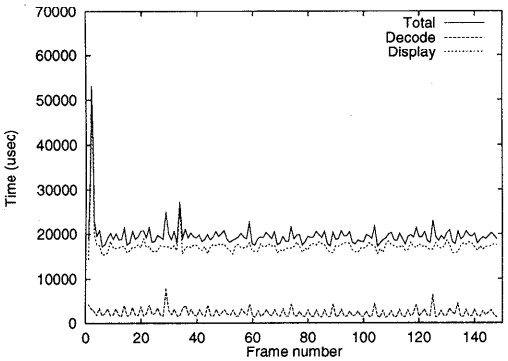


図11 サンプル・データの再生に要した時間（8 bit color）
Fig. 11 Time consumed to browse the MPEG video stream (8 bit color).

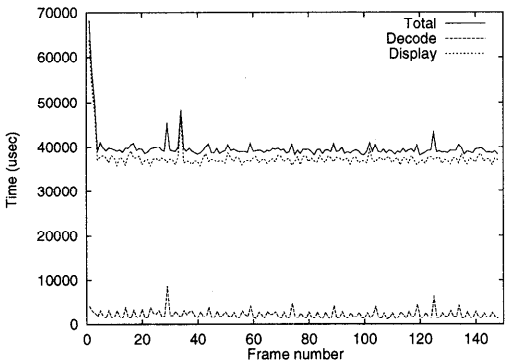


図12 サンプル・データの再生に要した時間（24 bit color）
Fig. 12 Time consumed to browse the MPEG video stream (24 bit color).

図13中の *8 bit Color(real)* と *24 bit Color(real)* は，式(1)より求めた $\min(N_j)$ である．パラメータ d_i と v_i は，図11と図12より求めた．*8 bit Color(approx.)* と *24 bit Color(approx.)* は，d_i+v_i の平均値に基づく近似値である．復号と表示に要する合計時間は，8 bitカラーの場合，19,632 μsecであり，24 bitカラーの場合，39,770 μsecである．

図13中の *NAMI Sync* は，1000 msec内に指定された回数の同期を行う場合の同期処理に必要なCPU時間である．*NAMI Sync* の場合，横軸は，1000 msec内に行う同期の回数であり，縦軸が必要なCPU時間となる．この図より，24 bitカラーで表示する場合は，25 fpsまでフレーム・レートをあげることが可能である．しかし，この場合，NAMIシステムが同期処理に使用できるCPU時間がない．フレーム・レートを24 fpsに設定した場合は，1000 msec間に15回の同期処理を行うことが可能となる．最も良い同期の質とフレーム・レートを両立するためには，*NAMI Sync* と必要とする質の交点に近い値を選択すればよい．たとえば，MPEG動画像を24 bitカラーで表示する場合は，*NAMI Sync* と *24 bit Color(real)* の交点近傍を選択すればよい．この場合，交点は，23.5 fps付近であることから，23 fpsにフレーム・レートを固定すれば，各フレームごと同期をとることが可能であることが分かる．同様に，8 bitカラーで表示する場合には，44 fpsに設定すればよいことが分かる．

表10 N_j の値（8 bit color，フレーム・レート30 fps，同期インターバル500 msecごと）
Table 10 The value of N_j (8 bit color, frame rate is 30 fps, synchronization in terval is 500 msec).

	N_j (μsec)	# of Frames
N_1	174,761	15
N_2	202,018	15
N_3	202,388	15
N_4	208,007	15
N_5	214,032	15
N_6	212,862	15
N_7	212,889	15
N_8	211,415	15
N_9	208,171	15
N_10	247,882	13

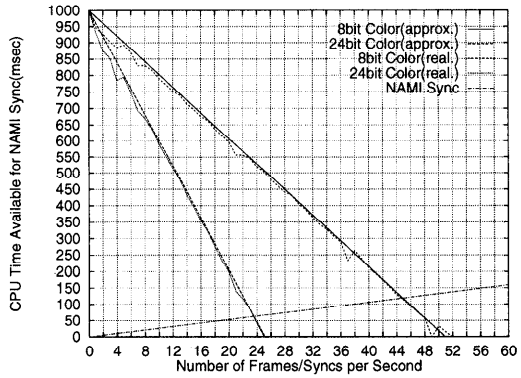


図13 MPEGplay2.0で必要とするCPU時間とNAMIシステムに与えられたCPU時間
Fig. 13 The CPU Time consumed by the MPEGplay2.0 and the CPU time available for the NAMI system.

8. おわりに

本論文では，メディア間の時間的論理関係の記述方式とそれを対象としたメディア間同期機構の実現方式を提案した．さらに，提案方式に基づく同期システムを実現し，その性能についての評価を行った．メディア間の同期処理を行うことを想定してメディア間の時間的論理関係を指定する場合，各メディアの時間軸上に任意の同期点を設け，その同期点間を *Meet* 関係で表す方式が有効であることを示した．

時間的論理関係を実際にメディア間の同期関係として具現化するするための協調型同期方式は，メディアを再生する各プロセスが互いに協調して同期する．この方式により，プロセスは，それぞれ扱うメディアと動作方針に従い，最適な動作を行うことが可能となる．本論文では，さらに，協調型同期方式に基づく同期機構とその操作プリミティブ群を示した．提案した同期機構と操作プリミティブ群を，動作形態により分類したプロセスに，自然な形で組み込めることを示した．

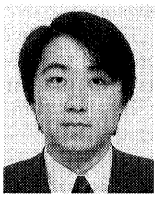
協調型同期方式に基づく同期機構の実現例 NAMI システムは，相対的な同期点の関係（*Meet* 関係）に基づき，プロセス間の同期を実現する．NAMI システムの同期機構は，あるインターバルで同期処理を行う際に，同期機構が満たすべき一般的な時間的制約より，メディア処理を行うプロセスの動作を妨げずに同期処理を行うことを可能とする．

今後は，対話などにより，同期点が動的に変化する場合の同期を扱えるように時間的論理関係を拡張し，その実現を行っていく予定である．

参考文献

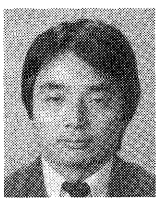
1) Allen, J.: Maintaining Knowledge about Temporal Intervals, *Comm. ACM*, Vol.26, No.11, pp.832–843 (1983).
2) Drapeau, G. and Greenfield, H.: MAEstro—A Distributed Multimedia Authoring Environment, *1991 Summer USENIX Conference*, Nashville, Tennessee (1991).
3) Hayama, T. and Kiyoki, Y.: A Distributed Process Coordinator for Rendering Multimedia Resources in a Multimedia System, *Proc. Multimedia Japan '96*, pp.168–175 (1996).
4) Kawachiya, K. and Tokuda, H.: Resource-Management Mechanism for Dynamic QOS Control of Multimedia, *Proc. Multimedia Japan '96*, pp.14–21 (1996).
5) Kiyoki, Y. and Hayama, T.: The Design and Implementation of a Distributed System Architecture for Multimedia Databases, *Proc. FID '94*, pp.374–379 (1994).
6) Little, T. and Ghafoor, A.: Interval-Based Conceptual Models for Time-Dependent Multimedia Data, *IEEE Trans. Knowledge and Data Engineering*, Vol.5, No.4, pp.551–563 (1993).
7) Masunaga, Y.: An Object-Oriented Approach to Temporal Multimedia Data Modeling, *IEICE Trans. Inf. & Syst.*, Vol.E78-D, No.11, pp.1477–1487 (1995).
8) Nieh, J. and Larm, M.: Integrated Processor Scheduling for Multimedia, *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp.215–218 (1995).
9) Nishio, N. and Tokuda, H.: A Middle-Ware for Continuous Media Processing in the Keio-MMP Project, *Proc. Multimedia Japan '96*, pp.278–284 (1996).
10) Rowe, L. and Smith, B.: A Continuous Media Player, *Proc. 3rd Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp.376–386 (1993).
11) Yavatkar, R. and Lakshman, K.: A CPU Scheduling Algorithm for Continuous Media Applications, *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp.223–226 (1995).

(平成 9 年 5 月 14 日受付)
(平成 10 年 1 月 16 日採録)



端山 貴也（学生会員）

1993 年筑波大学情報学類卒業．1995 年同大学院理工学研究科修士課程修了．現在筑波大学工学研究科電子・情報工学専攻．マルチメディアシステム，データベースシステムの研究に従事．



清木 康（正会員）

1978 年 慶應義塾大学工学部電気工学科卒業．1983 年同大学院工学研究科博士課程修了．工学博士．同年，日本電信電話公社武蔵野電気通信研究所入所．1984～1995 年筑波大学電子・情報工学系講師，助教授を経て，1996 年より慶應義塾大学環境情報学部助教授，現在に至る．データベースシステム，知識ベースシステム，マルチメディアシステムの研究に従事．ACM，IEEE 各会員．