データベースシステム 108－6
（1996. 5. 23）

コンティニュアスメディア・データの表示・演奏を対象とした分散プロセス間同期機構と実現

端山 貴也　　清木 康
筑波大学 工学研究科　慶應義塾大学 環境情報学部

要旨

近年、コンティニュアスメディア・データのデータベース化が急速に進んでいる。コティニュアスメディア・データを対象とした分散処理環境におけるマルチメディア処理システムの実現は、重要な課題の一つとなっている。そのようなシステムにおける重要な機構の一つは、複数のコンティニュアスメディア・データの表示・演奏時の同期である。我々は、このようなコンティニュアスメディア・データの表示・演奏時の同期を支援する分散プロセス間同期システム NAMI の設計および実現を行っている。NAMI システムは、複数の独立したアプリケーションを自由に組み合わせ、一つのアプリケーションとして協調動作することを支援するシステムである。アプリケーション間での協調動作は、アプリケーションに NAMI システムの同期マネージャを組み込むことにより可能となり、コンティニュアスメディア・データの表示・演奏時に同期が実現される。本稿では、NAMI システムにおけるこれらの同期マネージャによるアプリケーション間での同期機構の実現方式について述べる。

A Mechanism for Supporting Distributed Processes Rendering Continuous Media Data and its Implementation

Takanari HAYAMA　　Yasushi KIYOKI
Doctorial Program in Engineering　Faculty of Environmental Information
University of Tsukuba　　Keio University

Abstract

It is important to realize multimedia data processing for continuous media data in distributed computing environment. A sychronization mechanism among several contiunous media during the rendition is required for dynamic integration of those media. We designed and implemented a NAMI system which supports synchronizing processes to render continuous media data in the distributed computing environment. The NAMI system allows several independent applications to be arranged arbitrarily and work as they are integrated in a single application. Synchronization is supported by a Sync Manager of the NAMI system. The Sync Manager is included in each application. In this paper, the implementation method of the synchronization mechanism of the NAMI system is presented.

1 はじめに

コンティニュアス・メディアを扱うマルチメディア処理システムの実現は、重要な課題の一つとなっている [1, 2, 3]。コンティニュアスメディア・データを用いてマルチメディア・ドキュメントを生成する作成者は、自分の目的に応じて最適なデータ形式を選択する。一方、マルチメディアを扱うアプリケーションは、新しい静止画像、動画像、そして、音声などの符合化手法が実用化されるたびに、再実現を行うことが必要となる。例えば、あるマルチメディア・データの表示アプリケーションにおいて、MPEG 動画像を再生する機能を追加しようとする場合、元のプログラムの多くの部分を変更する必要がある。しかし、MPEG 動画像を再生するためだけのアプリケーションであれば、容易に入手することが可能である。実際には、新たなソフトウェアの開発を行う必要がない場合が多い。マルチメディア・データの表示アプリケーションの一部を担うことのできる数多くのアプリケーションが存在する。

一般的に、マルチメディア処理を行うアプリケーションの多くは、互いに独立に開発されているため、他のアプリケーションと協調して動作することを想定していない。しかし、アプリケーションの協調動作を支援する機構が存在し、複数のアプリケーションを組み合わせることが可能となれば、より高い拡張性と柔軟性を持つマルチメディア・アプリケーションを構成することが可能となる [4, 5]。

このような方法でマルチメディアを対象としたアプリケーションを構築するためには、実時間制約のもとでのアプリケーション間の同期機構が必要となる。この機構は、例えば、動画像と音声の再生を行う異なるアプリケーションが存在する場合に、動画像と音楽の再生速度を同調させるために必要である。我々は、このようなアプリケーションの協調動作を支援することを目的として分散プロセス間同期機構とその実現システム NAMI の設計を行った。

本稿では、コンティニュアス・メディアの表示・演奏の形態を分類する。そして、その分類に基づいた同期制御機構とその実現について述べる。

2 NAMI システムの同期制御

NAMI システムの同期制御は、プロセスを単位に行われる。したがって、NAMI システムにおいて、同期制御を行おうとするアプリケーションは、制御対象のプロセスとして扱われる。同期制御は、**協調型処理方式**で行われる。この方式において、各プロセスは、マスタ・スレーブ型処理方式のような主従の関係ではなく、全て同じ権利を持つものとして扱われる。

NAMI システムにおいて協調動作を支援するアプリケーションは、もともと独立なものとして開発されているため、それぞれ独自の動作方針を持つ。この動作方針は、各アプリケーションの扱うメディア、および、データの特徴を最大限に引き出すようになっているため、制御の際、各アプリケーションの動作方針を尊重する必要がある。協調型処理方式では、容易に各アプリケーションの動作方針を保存することが可能である。

さらに、協調型処理方式では、同期制御を行う際、関係するプロセス間でのみ通信を行えば良い。そのため、複雑な組合せの同期を取る場合に、マスタ・スレーブ方式に比べ有利である。一方、本方式では、全てのプロセス間で通信が行われる。そのため、プロセスが多い場合、マスタ・スレーブ方式に比べ、通信コストが非常に高くなる。しかし、NAMI システムが対象とする応用において、実際に協調動作するプロセスの数は多くとも 5 個程度であることが多い。そのため通信にかかるコストの影響は小さい。また、本方式では、優先度を各プロセスに持たせ、各プロセスが高い優先度を持つプロセスに従うようにすることにより、マスタ・スレーブ方式を包含することが可能である。

2.1 同期制御の記述

NAMI システムにおける同期の記述は、各プロセスの持つ同期点の相対的な関係によって記述される。同期をとるプロセスは、グループごとに互いの所属を識別するための識別子を持つ。

グループ内のそれぞれのプロセスは、自分が何時、誰と同期を取らなくてはならないかを知る必要がある。ここで、あるプロセスのグループが、同期を互いに取りながら開始し終了する一連の流れをセッションと呼ぶ。また、セッションに必要な同期情報と各プロセスに関する情報をセッション情報と呼ぶ。同期情報とは、どのプロセス間が、いつ同期を取るかを示した情報である。セッション Sは、次のように定義する。

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

セッションは、二つのタプルより構成される。GIDはグループ識別子、PS はプロセスの集合、そして、SPは同期点の関係の集合 (同期情報) である。n 個のプロセスから構成される PSは、次のように定義する。

$$PS = \{PS_1, PS_2, PS_3, ..., PS_n\}.$$

それぞれの PS_iは、各プロセスについての必要最小限の情報を持つ。これらは、次の三つのタプルより構成される。

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

最初のタプル $location$ は、プロセスがどこに存在するかを示すものである。次のタプル kは、いくつの同期点をプロセスが持っているかを示している。ここで、各プロセスは、必ず開始点と終了点を持つため、kの値は、この開始点と終了点を除いた値である。以上の二つのタプルは、プロセス PS_i以外のプロセスによって利用される情報である。最後のタプルは、同期点の具体的な値である。これらの同期点は、同期点の関係の集合 SPにおいて、次のように関係付けられる。

$$SP = \{SP_1, SP_2, SP_3, ..., SP_m\}.$$

各 SP_jは、次のように定義される。

$$SP_j = \{P_{finisher} \rightarrow P_{starter}\}.$$

これは、「$P_{finisher}$のプロセスがそれぞれの同期点に到着し次第、$P_{starter}$のプロセスがそれぞれの同期点から動き出す」と読む。各 P_xは、次のように記述される。

$$P_x = \{..., PS_p.T_q, ...\}.$$

$PS_p.T_q$は、プロセス PS_pの持つ同期点 T_qを意味する。図1に示すようなセッションは、図2のように記述可能である。

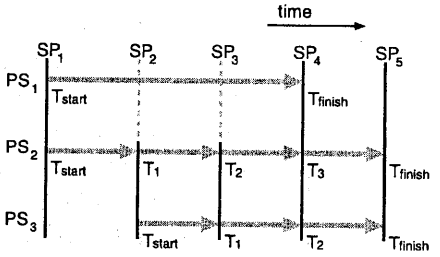

図 1: セッションの例

2.2 プロセスの状態

プロセスがセッション情報をもとに同期を行う際、時間が進むに連れプロセスの状態が変化する。つまり、同期は、プロセスの状態を変化させるきっかけの一つである、と捉えることが可能である。一般的に、マルチメディア処理を行うプロセス (アプリケーション) は、次のいずれかに分類可能である。

1. 実時間制約のもとで一定作業を繰り返し出力を行う能動的なプロセス (例:音楽の再生など)
2. 実時間制約を持たずに一定作業を繰り返し出力を行う能動的なプロセス (例:画像の表示など)
3. 入力に応じた出力を行う受動的なプロセス (例: Graphical User Interface(GUI) など)

実時間制約のもとで動作するプロセスは、時間に沿って処理を行い、その状態を変化させる。実時間の概念をもたないプロセスは、行った処理量にあわせて、その状態を変化させる。入力

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

図 2: 図 1のセッションの NAMI における記述

に応じた出力を行う能動的なプロセスは、得た入力により、その状態を変化させる。NAMI システムの同期制御によって行われる状態変化の制御は、各プロセスの特徴に応じた状態変化の方法とうまく融合される必要がある。

しかし、NAMI システムにおいて同期は、プロセスによって能動的に行われるものとしている。つまり、各プロセスは、基本的に、自身が望む時に他のプロセスと同期を行う。ただし、他のプロセスが先に同期点に達した場合、あるいは、次の同期を行わなければならない時点に達した場合、NAMI システムが同期を行うように、プロセスに対して指示することができる。これらの指示は、無視することが可能であり、プロセスが望むのであれば、プロセスの状態を強制的に次の同期点へ遷移させることも可能である。このような選択肢の中から、各プロセスは、自身の持つ方針と衝突しないように、他のプロセスとの同期方法を選択する。

NAMI システムでは、前述の通り、同期の指示を強制とすることも可能である。しかし、そのためには、同期を受け付けた時点から、同期点への状態遷移を自動的に行う必要がある。具体的には、プロセスのコンテキストを次の同期点における状態に変更する機構を必要とする。プロセスのコンテキストが、ほぼ同じ状態である受動的なプロセスの場合、事象待ちの状態でコンテキストを保存する。そして、同期処理から復帰し、処理を再開する時は、事象待ちの状態から再開する。一方、同じ処理を繰り返す能動的なプロセスでは、処理の開始時点でコンテキストを保存する。

保存したコンテキストは、同期処理後、元に戻す。その際、必要であれば、プロセス上で動作するプログラム中の変数の値の変更を行う。また、強制的に同期を取る場合に呼び出すユーザ定義の手続きを定義し、同期を取る際には、その手続きを呼び出し、プロセス状態を遷移させることもできる必要がある。

2.3 同期制御機構

NAMI システムの同期制御機構は、プロセス間の同期を支援するため、**情報サーバ** (*info server*) と**同期マネージャ**(*sync manager*) の二つのモジュールからなる (図 3)。情報サーバは、セッション情報を管理する。また、同期マネージャは、情報サーバより得たセッション情報をもとに、各プロセスの状態を管理し、他の同期マネージャと通信することにより、実際の同期を行う。各プロセスは、必ず一つの同期マネー

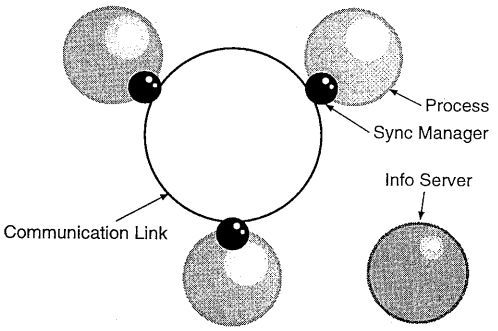

図 3: NAMI システムの同期制御機構

ジャと直接リンクされている。

1. **情報サーバ**

   情報サーバは、同期マネージャの利用する情報の管理を行う。情報サーバの管理する情報は、2.1節において述べたセッション情報である。情報サーバは、同期マネージャが互いを認識し、円滑に同期制御を行えることを支援するためのものである。情報サーバは、伝言板として捉えられるものである。最初に、各同期マネージャは、自身の識別子とグループ識別子をもとにセッション情報を入手する。そして、自身の識別子と所在を伝言板に書き込み、他の同期マネージャの所在を取得する。情報サーバにより、各プロセスは、情報サーバと通信可能である限り、任意の所在で動作することが可能となる。

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、各プロセスの動作を制御する。各プロセスは、同期マネージャが情報サーバより得た情報をもとに、制御される。同期マネージャの役割は、基本的に、2.2節において述べた状態遷移を支援することである。同期マネージャは、最初に、自身の識別子、および、情報サーバの所在を知っている。その上で、同期マネージャは、最初に次のような手順を踏みセッションの準備を行う。

   (a) 情報サーバに接続し、必要なセッション情報を取得する。

   (b) 自分の情報(プロセス識別子、および、場所)を情報サーバに登録する。

   (c) セッションに関係する他のプロセス(同期マネージャ)の情報を取得する。

   この時点で、同期マネージャは、互いの位置を知り、かつ、何を行えばよいかを把握する。そして、一旦、プロセスに制御を戻す。プロセスは、同期を取りたいと望む時点で、同期マネージャにその制御を渡す。同期マネージャが、制御を強制的に取るのは、プロセスがそれを許した時のみである。また、プロセスは、その実行中に、強制的に制御を奪うことを許可することも、また、許可を取り消すことも可能である。制御が同期マネージャに渡されると、同期マネージャは、次のような手順を踏むことにより、他の同期マネージャと同期を取る。

   (a) 自分よりも優先度の低いプロセスの同期マネージャに対し、緊急メッセージを送り、自分の同期点への到着を知らせる。

   (b) 同期に参加する全プロセスの同期マネージャに対し、自分の同期点への到着を知らせるメッセージを送る。

   (c) 同期に参加する全プロセスの同期マネージャから、同期点への到着を知らせるメッセージを受けとるまで待つ。

   (d) 制御をプロセスに返す。

   同期マネージャは、緊急メッセージを受けとると、プロセスに対し次のいずれかを行う。プロセスが強制的な次同期点への状態遷移を許している場合は、メッセージを受けとると同時に、強制的に次の同期点に状態を遷移させる。プロセスがそれを許していない場合は、単に緊急メッセージの受け取りを知らせるのみである。同期マネージャ

表 1: 同期マネージャのプリミティブ

プリミティブ	機能
NamiInit()	セッションの準備
NamiSync()	同期のために制御の譲渡
NamiClrVarList()	同期時に更新する変数リストの初期化
NamiSetVarList()	同期時に更新する変数リストの設定
NamiUseTimer()	実時間同期制御の許可
NamiCompelledSync()	強制的な同期の許可
NamiSetJmp()	強制的な同期からの復帰点の設定(コンテキストの保存)
NamiSigBlock()	強制的な同期のブロック
NamiSigUnblock()	強制的な同期のブロック解除

表 2: 情報サーバのプリミティブ

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

は、制御をプロセスに返す際、プロセスに対し、次の同期点までの時間、および、残りの同期点の数を毎回知らせる。

2.4 NAMI システムのプリミティブ

NAMI システムの機能群は、プリミティブとして実現されている。NAMI システムのプリミティブは、情報サーバの機能を利用するためのプリミティブと同期マネージャの機能を利用するためのプリミティブの二種類からなる。表 1 は、同期マネージャの機能を利用するための、また、表 2は、情報サーバの機能を利用するためのプリミティブである。後者の情報サーバの機能を利用するためのプリミティブは、同期マネージャも利用する。

NAMI システムのもとで、同期を行うアプリケーションには、これらのプリミティブが組み込まれる。それにより、各アプリケーションは、同期マネージャの制御の元で、他のアプリケーションと協調動作を行えるようになる。

3 NAMI システムの評価

NAMI システムの記述性、および、同期に要する時間を確認するために、実際のアプリケーション・プログラムに NAMI システムのプリミティブを組み込んだ。現在、NAMI システムは、SunOS4.1.4、HP-UX9.05、IRIX5.3 上で動作している。アプリケーションを NAMI システム上で動作させるためには、ライブラリとして提供されている NAMI システムの同期マネージャをリンクする。そして、ライブラリに組み込まれている NAMI プリミティブを呼び出すようにプログラムを書き換える必要がある。ここでは、2.2節で述べた 3 種類のアプリケーションを対象に、実験を行った。本実験で選択した 3 種類のアプリケーションは、MIDI プレーヤ (実時間アプリケーション)、MPEG デコーダ (非実時間アプリケーション)、そして、Tcl/Tk(受動的アプリケーション) である。

3.1 実時間アプリケーションへの組み込み

MIDI プレーヤのプログラムとして、自作のものを使用した。このアプリケーションは、Standard MIDI File(SMF) を読み込み、解釈し、実時間で MIDI 楽器の制御を行うものである。音楽は、情報が欠落すると問題になるので、このアプリケーションは、外部からの割り込みを無視する。そのため、音楽が途中で飛ぶことはないが、場合によっては、演奏がぎくしゃくする場合もある。

基本的に、他のアプリケーションと同期するタイミングは、MIDI 楽器の制御と同期する必要がある。実際、このような同期を用いるのは、動画像の特定のシーンと音楽の特定のフレーズをあわせるためである。そのため、MIDI 楽器

の制御と他のアプリケーションとの同期するタイミングは、時間的に同時に発生する。

また、このような実時間を扱うアプリケーションは、実時間制約を持たないアプリケーションを実時間で動かすための外部タイマとして用いることも可能である。

3.2 非実時間アプリケーションへの組み込み

MPEG デコーダは、UNIX 上で動作するフリー・ソフトウェアを用いた [7]。このアプリケーションには、NAMI システム、および、次の同期点にすぐに移らなくてはならない時に描画できなかったフレームをスキップする機能を組み込んだ。加えた変更および追加は、38 行であった。

このアプリケーション単体では、実時間処理を行えない。しかし、実時間アプリケーションと同期をとらせることにより、20 フレーム毎秒の動画像再生を行わせることも可能である。この時、表示できなかったフレームは、組み込んだ変更によりスキップされる。また、このような時に用いる実時間アプリケーションは、単に時間をカウントするもので良く、NAMI システム上では、非常に簡単に実現することが可能である。

3.3 受動的アプリケーション

受動的なアプリケーションは、何らかのイベントが発生するのを待ち続ける。例えば、GUI アプリケーションなどでは、ユーザからの入力を待つ。このような受動的なアプリケーションの例として Tcl/Tk を選択した [6]。Tcl/Tk の演算子として、NAMI プリミティブを定義し、ユーザからの入力と同様に他のアプリケーションとの同期も扱う。同期マネージャを組み込んだ Tcl/Tk では、他の GUI 部品にコールバック・ルーチンを指定するように、同期処理のコールバック・ルーチンが指定される。

3.4 NAMI システムの同期

NAMI システムを組み込んだ 3 種類のアプリケーションを用いて、マルチメディア・アプリケーションを構成した。Tcl/Tk による GUI アプリケーション、および、MPEG デコーダは、MIDI プレーヤの優先度を最大にすることにより、MIDI プレーヤの管理する時間にあわせるようにした。GUI アプリケーションは、MIDI プレーヤの同期点への到着の知らせを無視することにより、アプリケーション全体の動作を一時的に停止させることを可能とする。

実際にアプリケーションを 2 台の計算機上で動作させ、同期に要する時間を計測した。MPEG デコーダと GUI アプリケーションの 2 つのビジュアル・アプリケーションを SparcStation5 上で動作させ、MIDI プレーヤを HP9000/715 上で動作させた。MPEG デコーダは 20 フレーム毎、MIDI プレーヤは 1 秒毎に同期を取るように設定した。この時、MPEG デコーダは、処理に間に合わないフレームをスキップする。

3 つのアプリケーションを同時に動作させ、音楽に生じる遅れに注目した。結果として、同期制御によって生じる遅れは、MIDI プレーヤにほとんど影響を及ぼさなかった。また、同期処理に要する時間を計測すると、3 プロセス間での同期の場合、平均して 10msec 未満という結果が得られた。MIDI プレーヤは、実時間で電子楽器を制御する。つまり、MIDI の制御では、音を鳴らした後、音の長さだけ待ち、音を止める。テンポが 120(4 分音符が一分間に 120) の楽曲において 32 分音符は 62.5msec である。これは、音を鳴らしてから音を止めるまで、62.5msec の余裕があることを意味する。ある音と次の音の境目では、間隔が非常に狭い場合もある。しかし、特殊な効果を求める場合以外で、10msec より短いことはほとんどない。

一方で、使用した MPEG デコーダは、HP9000/715 上で、平均 10 フレーム程度の再生能力を有する。従って、1 フレーム辺りの処理速度は、平均 100msec である。実際には、I フレーム、P フレームや B フレームなどの種類により、展開に要する時間が異なる。そのため、1 フレームあたりの処理時間には、かなりのば

らつきがある。しかし、10フレーム単位で同期をとる限り、同期処理は、動画像の再生に大きな影響を及ぼさない。

より高度な同期処理を行うには、実時間の保証や、同期処理にかかる時間の短縮が必要となる。しかし、現実に使用できるアプリケーションに組み込んだ場合でも、本システムは、そのアプリケーションの性能にほとんど影響を与えずに、同期を支援できる。

4 おわりに

本稿では、コンティニュアス・メディアを扱うアプリケーションの協調動作を支援することを目的とした分散システムであるNAMIシステムについて述べた。NAMIシステムは、アプリケーション間の協調動作を協調型処理方式に基づく同期制御により支援する。NAMIシステムにおけるアプリケーション間の同期制御の特徴は、各アプリケーションの方針を尊重することと、自由にアプリケーションを組み合わせることができることである。NAMIシステムの機能は、プリミティブとして実現されている。本稿では、プリミティブを実際のアプリケーションに組み込み、その記述性、および、性能を確認した。

今後は、アプリケーションの行う表現の協調について検討を行っていく。アプリケーションは、独自のユーザ・インタフェースを持つことが多い。そこで、その独自のユーザ・インタフェースの上に、統一的なユーザ・インタフェースを重ね合わせることにより、ユーザ・インタフェースにおいても、各アプリケーションが協調できるような機構を実現する予定である。また、ストリーム指向型並列処理システムSMASHによるコンティニュアスメディア・データ処理系と結合することにより、より柔軟にデータを利用できるようにする[8, 9]。

参考文献

[1] L.Hardman, D.C.A.Bulterman, G.van Rossum, "Structured Multimedia Authoring," *Proc. of 1st International Conf. on Multimedia.* pp.283-289, Aug. 1993.

[2] L.Hardman, D.C.A.Bulterman, G. van Rossum, "The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model," *Commun. ACM*, Vol.37, No.2, pp.50–62, Feb. 1994.

[3] Y.Kiyoki, T.Hayama, "The Design and Implementation of a Distributed System Architecture for Multimedia Databases," *Proceedings of FID '94*, Oct. 1994.

[4] T.Hayama, Y.Kyoki, "A Distributed Process Coordinator for Rendering Multimedia Resources in a Multimedia System," *Proceedings of Multimedia Japan '96*, Mar. 1996.

[5] George D.Drapeau, Howard Greenfield. "MAEstro—A Distributed Multimedia Authoring Environment," *1991 Summer USENIX Conference in Nashville, Tennessee*, 1991.

[6] J.K.Ousterhout, "Tcl and the Tk Toolkit, " Addison-Wesley, 1994.

[7] L.A.Rowe, B.C.Smith, "A Continuous Media Player," *in Network and Operating System Support for Digital Audio and Video*, Springer-Verlag Lecture Notes in Computer Science #712, pp.376–386, 1993.

[8] 佐藤, 廣木, 清木, "ストリーム指向型並列処理によるマルチメディア・データの統合方式", 並列処理シンポジウム JSPP'95(論文集), pp.137–144, May 1995.

[9] A. Sato, M. Hiroki and Y. Kiyoki, "A stream-oriented parallel processing system for continuous media integration," to appear in Proceedings of the IASTED (International Association of Science and Technology for Development) Conference on Applied Informatics, Feb. 1996.