システムソフトウエアと
オペレーティング・システム 60－2
マルチメディア通信と分散処理 61－2
（１９９３．７．８）

リアルタイム・オブジェクトむきの IPC について

盛合 敏

NTT 情報通信網研究所

分散コンピューティングシステム上で連続メディアを扱うためのプログラミング・モデルとして，Realtime Projection Model を紹介する．このモデルは，『動的なオブジェクトのイメージを，与えられた再現精度で，別のオブジェクト上に投影する』という考え方に基づいて構成されている．再現精度は，QoS (Quality of Service) とも呼ばれ，リソースの予約をすることで再現精度を保証する．本報告では，このモデルをサポートするための IPC 機構を提案する．そして，Real-Time Mach と ST-II プロトコルに基づく実現について述べる．

An Inter-Process Communication Mechanism for Real-Time Objects

Satoshi Moriai

moriai@nttbss.ntt.jp, NTT Network Information Systems Laboratories,
Yokosuka-shi, Kanagawa, 238–03, Japan.

The real-time projection model, described in this paper, provides a programming model for supporting continuous media on distributed computing systems. This model is based on the general idea such as projecting the dynamic image of an object on another object. Reproduction quality, called quality of service, is guaranteed by reserving resources. The inter-process communication mechanism of this model is proposed. This IPC mechanism is constructed on the Real-Time Mach kernel and the ST-II internet stream protocol.

1 はじめに

パーソナルコンピュータの高性能化やネットワークの高速広帯域化にともない，分散環境において連続メディア (または各種メディアを統合したマルチメディア) を利用するためのハードウェア基盤が整いつつある．しかし，連続メディアを支援するソフトウェアは，ネットワークで多数のコンピュータが接続された環境を有効に利用するには至っておらず，いわゆる分散環境におけるマルチメディアシステムを構築するためのソフトウェア基盤が求められている．

この分野の研究は，リアルタイム・カーネル [7][8]，リアルタイム通信プロトコル [5][6]，マルチメディア・サーバやマルチメディア・ツールキット [2][10] など，精力的に行われている．これらの研究の結果，連続メディアのデータを扱うための個々の “部品” は徐々にそろってきていると言えよう．しかし，全体をひとつのシステムとして構築したときの end-to-end の性能 (遅延時間やスループット) の保証については，解析のためのモデル [1] が提案されているが，実際のアプリケーションに即したモデルへの高度化や，“部品” へのフィードバックが必要である．

本研究では，連続メディアを扱うためのプログラミング・モデルに基づいて end-to-end の性能を保証する枠組みを与え，オペレーティング・システムや通信プロトコルへフィードバックを図ることで，連続メディアのためのソフトウェア基盤を実現することを目的としている．

本報告では，連続メディアのためのリアルタイム・オブジェクト・モデルについて概説し，このモデルをサポートするための IPC 機構について提案する．カーネルとして Real-Time Mach[8] を，通信プロトコルとして ST-II[9] を用いた場合の IPC 機構の実現法や問題点を考察する．そして，プロトタイプ実験について述べる．

2 連続メディアのためのリアルタイム・オブジェクト・モデル

連続メディアのデータを扱う上で最も重要なことは，データの持つ時間的な構造を保ったまま，時間的に変化する情報源を可能な限り正確に再現することにある．しかし，システムの資源が有限であることを考えると，その再現の精度は利用できる資源とのトレード・オフとなる．すなわち，連続メディアのアプリケーションは，時間的に変化する情報源を与えられた精度で再現するものとしてモデル化することができる．このモデルを実時間投影モデル (Realtime Projection Model) と呼ぶ．

Realtime Projection Model (図 1) は，

- スクリーン (Screen) — メモリ・オブジェクトやフレーム・バッファのように，時刻の経過に伴って変化する情報を保持するもの，
- プロジェクタ (Projector) — スクリーンからスクリーンへデータを投影するリアルタイム・オブジェクト，
- 資源 (Resouce) — プロジェクタやスクリーンのための資源

から構成される．スクリーンは独立した存在ではなく，プロジェクタに張り付いている．プロジェクタは，精度 (QoS; Quality of Service) および資源の予約に関して以下の動作をする．

- 指定された精度を保証するために，必要な資源 (一般的には，複数の種類の資源) の予約を行う．
- 必要な資源が予約できない場合や予約した資源が利用できない場合は，利用可能な資源で保証できる精度をアプリケーションに通知する．
- スクリーンを介してパイプライン状に接続した場合，アプリケーションからは終端のプロジェクタに対して精度が指定され，先行するプロジェクタに対して，必要な精度の指定する．

パイプライン状に接続されたプロジェクタの集合をセッションと呼ぶ．システム上には複数のセッションが同時に存在し得る．システム上の資源は有限であることから，資源の利用に関して調停をする必要がある．調停には 2 つのレベルがある．すなわち，

- 同一セッション内の調停
- セッション間の調停

である．調停の結果，各セッションの end-to-end で保証できる精度が決まることになる．

3 プロジェクタ・オブジェクト

プロジェクタは，アクティブなリアルタイム・オブジェクトであり，プロジェクタ自身を制御するための制御スレッド (Control Thread) と，データを投影

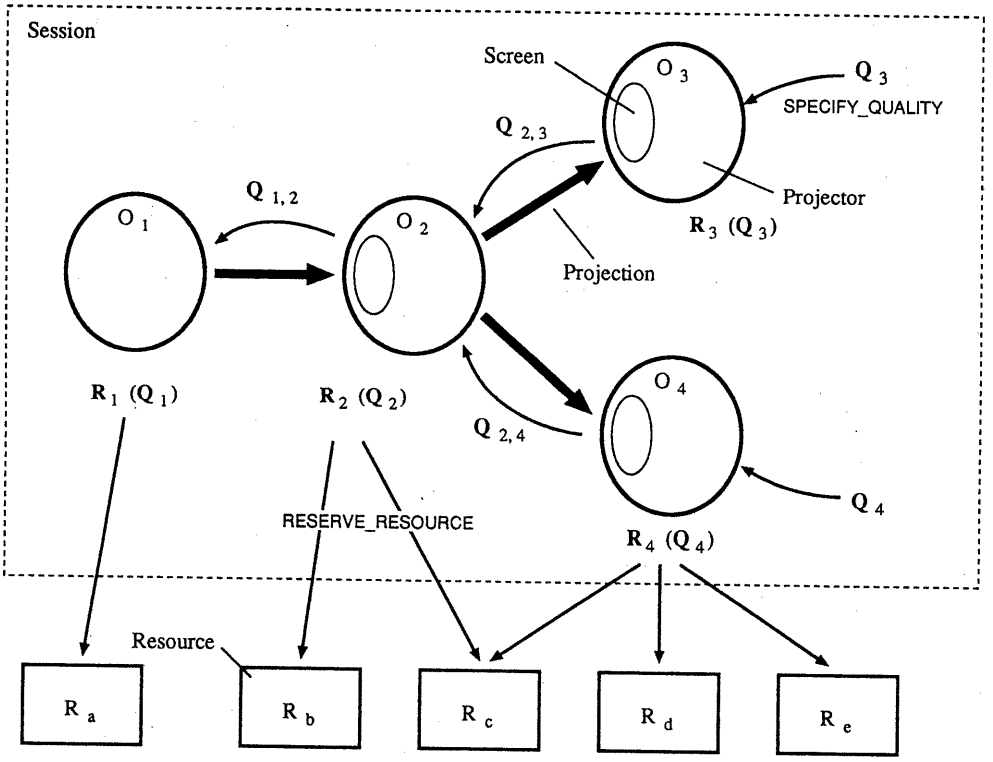


図 1: Realtime Projection Model

するためのデータスレッド (Data Thread) が閉じ込められている．また，プロジェクタはデータの出入口となるスクリーンを持つ．

制御スレッドは非周期的なリアルタイム・スレッドであり，セッション・マネージャや他のプロジェクタからの制御要求を受け付け，その処理を行う．また，必要に応じ，接続された他のプロジェクタに制御要求を行い，自身で処理できない要求の転送を行う．

データスレッドは，周期的なリアルタイム・スレッドであり，スクリーンからのデータの読み取り，データの処理，スクリーンへの書き込み，という一連の動作を繰り返す．

オブジェクト間の通信機構には，スレッドと同様に，制御用の通信機構と，スクリーンを介した連続メディアをリアルタイムに転送する通信機構がある．連続メディアの通信には，

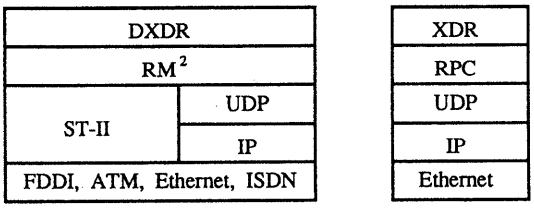
- 定常的であり，通信量が多い，
- メッセージの発信時刻や到着時刻という時間軸上の情報が意味を持つため，時期を逸したデータは無意味となる

という特徴がある．このため，メッセージのコピーやコンテキストの待避や復元を極力避けるとともに，受信されていなくても意味を失った時間的に古いメッセージを次々と捨てる必要がある．すなわち，伝統的なキューによるメッセージバッファや，コピーによるメッセージの転送は，連続メディアの通信には向いていない．また，ふたつのオブジェクトで共有される単純なリングバッファを用いた場合，時間軸上の情報を保ちながらメッセージの転送を行うことはできるが，コピーのオーバーヘッドが生じてしまう．

スクリーンを介した通信では，仮想記憶機構を積極的に利用し，連続メディアの通信に対する要求に応える．すなわち，

- メッセージを保持するページはコピーされることなく次のプロジェクタに渡される，
- メッセージの管理は，受け側のリングバッファを送り側が直接操作することによって行う

という方法をとる．



(a) 本研究のプロトコル体系　　(b) 参照モデル

図 2: プロジェクタ IPC のためのネットワーク通信プロトコル

4 IPC 機構とその実現

4.1 ST-II プロトコル・サーバ

ST-II は，帯域保証型リアルタイム・ストリーム・プロトコルであり，

- 実際のデータの転送に先立って，Flow Spec と呼ばれる帯域幅や遅延時間などの条件，すなわち QoS を指定する機能を持つ，
- データ転送のためのプロトコルを簡素化し，高速なデータ転送を可能としている，
- マルチキャストの機能を持つ

等の特徴がある．ST-II では，データの転送元からの接続要求を契機に，転送元から宛先までの一方向のストリームを確立する[1]．ストリームを確立するために必要な情報については，別の手段によって知られている必要がある．また，フロー制御，データ誤りの検出，および再送処理は他のプロトコル層にまかされている．

そこで，ネットワーク経由のプロジェクタの接続するために，ST-II と UDP/Multicast IP をベースとした図 2(a) に示すプロトコル体系を用いる．ST-II および UDP の上位層には RM^2 がある．これは RPC 層にあたり，遠隔システムへの要求・応答型のメッセージと，連続メディア・データを扱い，レート制御によるフロー制御，再送処理等を行う．DXDR (Dynamic External Data Representaion) は，RM^2 で扱うデータやメッセージのコーディングに用いる．これは，XDR とは異なり，ネゴシエーションによって表現形式を決定する機能を持つ．将来的には，UDP/IP の代わりにリアルタイム・メッセージ・プロトコルを用いる予定である．

[1]オプションによって逆方向のストリームも同時に確立することも可能ではあるが，それぞれは独立したものと考えられる．

以上で述べた通信プロトコルは Real-Time Mach のユーザ・レベルで走行するプロトコル・サーバによって処理される．プロトコル・サーバはプロジェクタ IPC インターフェースを持ち，直接プロジェクタに接続することができる．プロジェクタからの制御要求のうち，Flow Spec 制御などの ST-II で定義されているものについては ST-II を用い，それ以外の制御要求 (例えば，どういったデータをどんな形式で転送するかの要求) は UDP を用いる．この判断は，プロトコル・サーバが行う．

4.2 プロジェクタ IPC インターフェース

プロジェクタ IPC は，以下に示すように Caller Interface と Callee Interface が対称となったインターフェースを持つ．なお，引数については主なものを示している．

Caller Interface

rp_create(screen_name, quality, [entry_point, thread_attribute_hint,] ...)
rp_destroy(screen_name)
rp_control(screen_name, quality, ...)
rp_start(screen_name, counter)
rp_stop(screen_name, counter)
rp_reset(screen_name)
rp_call(screen_name, procedure_name, argument, result)

Callee Interface

rps_create(screen_name, quality, ..., result)
rps_destroy(screen_name)
rps_control(screen_name, quality, ..., result)
rps_start(screen_name, counter)
rps_stop(screen_name, counter)
rps_reset(screen_name)
rps_call(screen_name, procedure_name, argument, result)

Data Manipulation and Synchronization

scr_next(screen_name, data, data_length, [timeout,] ...)

scr_done(screen_name, data, data_length)

rp_create によって，指定精度のスクリーンが生成される．entry_point を定義した場合，スクリーン上で新たなデータが利用できるようになると，そのデータを処理するためのポップ・アップ・スレッドが起動される．rp_destry はスクリーンを削除する．rp_start, rp_stop, rp_reset によって，データの流れを起動，停止，初期化できる．明に精度の変更を行うためには，rp_control を用いる．rp_call は，そのほかの制御を行うためのもので，これはプロジェクタのインプリメントに依存する．rps_* は，接続されたプロジェクタにおいて，rp_* が起動されたときに呼ばれる関数である．プロジェクタが一切関与する必要がない要求を他のプロジェクタに転送する場合は，screen_name を返り値として関数を終了すればよい．scr_next は次に処理すべきデータの位置を返す．timeout を指定することで，特定の時間だけデータが利用できるようになるまで待つことができる．scr_done は処理が終了したことを指示する．

4.3 Real-Time Mach 上でのスクリーンの実現

Real-Time Mach は，リアルタイム機能として，リアルタイム・スレッド，リアルタイム同期機構，リアルタイム・スケジューラを備えた Mach Kernel であり，Mach の仮想記憶の機能が利用可能となっている．

同一のタスク空間内にあるプロジェクタ間のスクリーンを実現するには，全ての仮想アドレス空間が共有されていることを利用できる．スクリーンの読み書きのサイズが等しいならば，仮想ページプールを用意して，リングバッファはポインタの配列として実現できる．スクリーンの読み書きのサイズが異なるならば，読み書きにおいて可能な限り scatter / gather 形の処理を行い，仮想メモリ関係のカーネル呼び出しを最小限にとどめることができる．

一方，異なるタスクにあるプロジェクタ間のスクリーンを実現する場合，スクリーンのための仮想空間をふたつのタスク間で共有してもデータのコピーが避けられないことがある．例えば，プロトコル・サーバではネットワークからの入力をそのデータの内容によって宛先を動的に決めなければならず，共有空間に直接書き込むことはできない．このため，適当な仮想

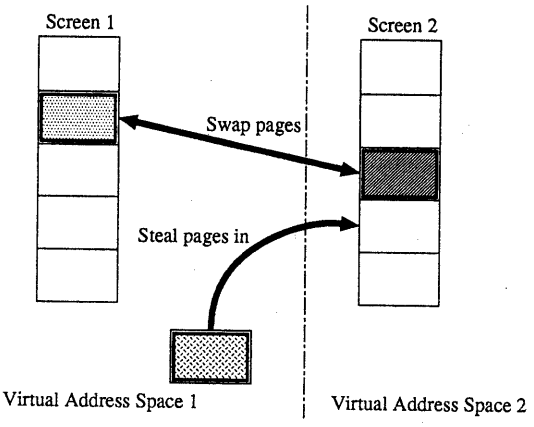


図 3: Mach 上でのスクリーンの実現

ページにデータを書き込んだ後，その仮想ページを共有空間に移動することになるが，共有空間に対する仮想ページの移動は，現在の Mach では正常に動作しない．そこで，異なるタスク間のスクリーンは，仮想空間を共有することなく，受け側で vm_allocate を実行した後，vm_write と vm_deallocate によって仮想ページを移動することになる．

ところが，vm_write 等のシステムコールは，copy-on-write の機構によって実現されているため，比較的高価なものとなっている．また，データを書き込む度に複数回のカーネルとのやりとりが必要となり，コピーの回避による効果が薄れてしまう [3]．効率化のためには，

vm_steal(target_task, destination, source, size) — 仮想ページを宛先に移動する，

vm_swap(target_task, destination, source, size) — 仮想ページのマッピング情報を送元と宛先で交換する

といった，copy-on-write に基づかないシステムコールが必要となる．このシステムコールを利用した場合のスクリーンの実現を図 3に示す．また，連続しない複数のページを一括して処理できれば，さらにカーネルとのやりとりを減らすことができる．

4.4 セッション・マネージャ

セッション・マネージャは，ひとつながりとなったプロジェクタをひとつのセッションとして管理し，プロジェクタとアプリケーションおよびリソース・マネージャとの橋渡しを行う．セッション・マネージャ

は以下のようなインターフェースを持つ．

Projector Interface

sm_probe(resource_id, requirements, result)
sm_reserve(projector_id, resource_id, requirements, reservation_id, result)
sm_free(projector_id, resource_id, reservation_id, result)

Application Interface

sm_create(session_id, precedence)
sm_destroy(session_id)
sm_wait(session_id)
sm_change(session_id, precedence)
sm_register(session_id, object_name, parameter, object_id)
sm_unregister(session_id, object_id)
sm_project(session_id, object_id1, object_id2, screen_name, quality)
sm_erace(session_id, object_id1, object_id2, screen_name)
sm_control(session_id, quality, ...)
sm_start(session_id, counter)
sm_stop(session_id, counter)
sm_reset(session_id)
sm_call(session_id, procedure_name, argument, result)

プロジェクタは，sm_reserve によって resource_id で指示される資源の予約や予約の変更を行い，sm_free によって資源の予約を取り消す．予約できるだけの資源があるかどうかは sm_probe によって調べることができる．requirements は，資源を “いつ”，“どれだけ” 必要か，その “希望値” と “受認限度” を表す．資源予約パラメータの詳細は資源の種類によって異なり，ネットワーク資源については帯域幅，パケットサイズ，パケットレートのそれぞれの最小値と希望値が用いられる．実際に予約された値が sm_reserve の返り値となる．資源の予約は，その資源を管理しているリソース・マネージャによって行われ，セッション・マネージャがセッション間の調停を行う．

4.5 予備実験

プロジェクタ IPC とセッション・マネージャのプロトタイプを，UNIX 上に作成したユーザ・レベル・スレッド・パッケージ上に実現した．なお，ST-II の部分は実現されていない．予備実験はプロジェクタの構成，セッションの管理，およびネットワーク資源の管理についての検討を目的としている．

プロトタイプでは，セッション・マネージャにネットワーク資源を管理するリソース・マネージャを組み込んでおり，それ以外の資源の管理は行っていない．ネットワーク資源は帯域幅の最小値と希望値によって予約する．総資源量として帯域幅 c が与えられているとする．プロジェクタ P_i は，他のプロジェクタとのデータ転送のために最低限確保すべき帯域幅 r_i と望ましい帯域幅 d_i を指定してセッション・マネージャへ予約の要求をする．セッション・マネージャは，予約状況に基づいて，それぞれのセッション i が現在利用してよい帯域幅 u_i を求め，応答を返す．また，既に予約されている予約値を変更した場合は，それぞれのプロジェクタへ通知する．u_i は

$$u_i = min\left\{\frac{r_i}{\sum_n r_n}c,\ d_i\right\}$$

によって求められる．

送り側プロジェクタは周期スレッドによってデータを送り出し，受け側プロジェクタはポップ・アップ・スレッドによってデータを受け取る．RM^2 プロトコルはプロジェクタと同一のタスク空間で走行するプロトコル処理スレッドによって処理される．RM^2 は基本的にはブロック型のプロトコルであるため，複数のプロトコル処理スレッドを生成しておくことで，間断なく処理が行えるようになっている．なお，送り側プロジェクタのデータスレッドの周期 g は，最後に送りだしたデータのパケット長を l とすると，$g = l/u_i$ によって与えられる．この値はセッション・マネージャからの通知に基づいて随時変更される．

Ethernet で接続された Sun 4/65 の上でプロトタイプを動作させたときのデータの処理サイズと平均スループットを図 4に示す．ここでは，帯域幅の制御はしていない．プロトコル処理スレッドが 4 つ程度で Ethernet の帯域幅をほぼ使いきっている．

データの処理サイズを 3kB 一定として，セッション・マネージャによって帯域幅制御を行ったときの例

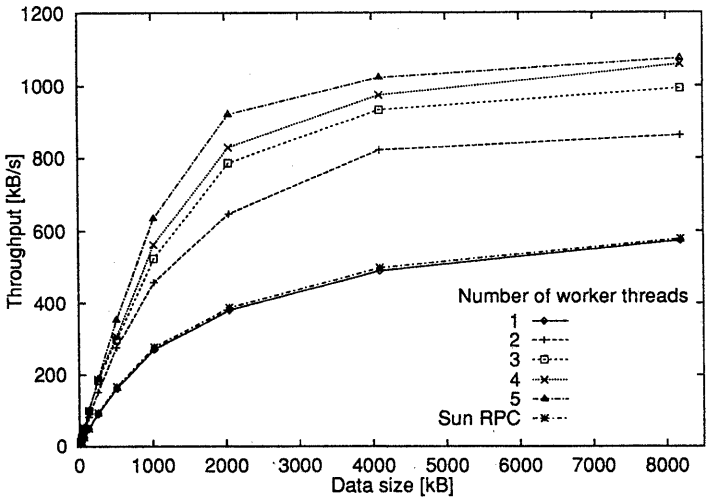


図 4: プロトタイプシステムの Ethernet 上の IPC のスループット

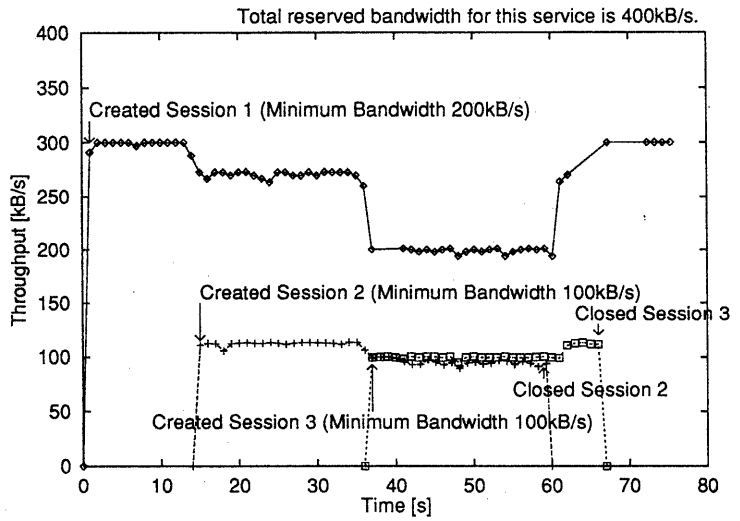


図 5: プロトタイプシステムの帯域幅制御の例

を図5に示す．送り側プロジェクタと受け側プロジェクタを同時に動作させたマシンを3台用意し，プロジェクタは互いに他のマシン上のプロジェクタに接続されている．$c = 400$ とし，$r = 300$, $d = 200$ の資源要求をするセッション1と，$r = 200$, $d = 100$ の資源要求をするセッション2，3があるときの様子を示した (単位は全て kB/s)．はじめはセッションが1つしかないため，セッション1は希望どおりの 300kB/s の帯域幅を得て動作できるが，他のセッションが生成されるたびに，使用できる帯域幅が制限される．しかし，3つのセッションが存在しても，それぞれのセッションが最低限必要とした帯域幅は確保されている．この状況では，新たなセッションの生成はセッション・マネージャに拒絶される．

5 関連研究

性能の保証という観点から，本研究に関連した主な研究について述べる．

CM-resource model および Meta-scheduling[2] は，システム要素間の調停を行い，資源を予約することで end-to-end の性能をアプリケーションに保証するモデルである．これは，アプリケーションを複数のセッションに分解し，コスト関数に基づいて各セッションの遅延時間の和が最小となるように各セッションに遅延時間を割り当てる．この概念は本研究においても遅延時間の保証に適用できる．しかし，各セッションでのメッセージサイズやメッセージの到着率が異なる場面での解法が必要である．CM-resource model では，システムに十分に資源があることを仮定しており，その仮定が成り立たない場合のアプリケーション間の調停については答えていない．本研究では，アプリケーションの precedence を考え，それに基づいてアプリケーションの調停を行おうと考えている．

CBSRP[4] は，FDDI ネットワークにおいて，その利用状況に応じて動的な QoS 制御を行う．QoS として，非連続的な “QoS クラス” を導入しており，ユーザからの QoS の指定を容易にしている．FDDI 以外の資源の扱いや，圧縮符号化されたビデオ等での QoS 制御が課題となっている．本研究においても，ユーザからの QoS の指定を容易にする機能は取り入れるべき機能であると考える．

6 おわりに

連続メディアを扱うためのプログラミング・モデルとして Realtime Projection Model を提案した．そしてこのモデルをサポートするための IPC 機構とその実現方法や問題点について述べた．

現在，ST-II プロトコル・サーバ，IPC 機構のインプリメントを行っており，その結果について，機会をあらためて報告したい．また，

1. アプリケーションの要求する精度を保証できるかどうかを検定するアルゴリズム，
2. 要求精度から必要な資源量を推定するアルゴリズム，
3. アプリケーション間の調停アルゴリズム，
4. スケジューラへのセッションの概念の導入，
5. CPU，仮想記憶，デバイス等において，“いつ”，“どのくらい” という資源予約へ対応できるリソース・マネージャ

について検討する予定である．

参考文献

[1] Anderson, D. P.: Meta-Scheduling for Continuous Media, Technical Report UCB/CSD 90/599, UC Berkeley (1990).

[2] Anderson, D. P. and Homsy, G.: A Continuous Media I/O Server and Its Synchronization Mechanism, *Computer*, **24**, 8, 51–57 (1991).

[3] Barrera III, J. S.: A Fast Mach Network IPC Implementation, in *Proceedings of the USENIX 1991 Mach Symposium* (1991).

[4] Chou, S. T.-C. and Tokuda, H.: System Support for Dynamic QOS Control of Continuous Media Communication, in *Third International Workshop on Network and Operating System Support for Digital Audio and Video*, 322–327 (1992).

[5] David R. Cheriton, : VMTP: Versatile Message Transaction Protocol Specification, *RFC 1045* (1988).

[6] Protocol Engines Inc., : *Xpress Transfer Protocol Definition*, Revision 3.6 (1992).

[7] Stankovic, J. A. and Ramamritham, K.: The Spring Kernel: A New Paradigm for Real-Time Operating Systems, *ACM Operating Systems Review*, **23**, 3 (1989).

[8] Tokuda, H., Nakajima, T., and Rao, P.: Real-Time Mach: Towards Predictable Real-Time Systems, in *Proceedings of the USENIX 1990 Mach Workshop* (1990).

[9] Topolcic, C.: Experimental Internet Stream Protocol, Version 2 (ST-II), *RFC 1190* (1990).

[10] 米田健, 林正薫, 松下温: 分散環境に適したマルチメディア制御モデル, 情処研報 (DPS), **93**, 36, 59–66 (1993).