利用者環境を考慮した相互通信のためのミドルウェア

橋本浩二[†] 柴田義孝[†]

ブロードバンドネットワークの普及により，日常的に利用するコンピュータを通信端末とした相互通信サービスも実用的なものとなった．また，利用する端末の性能やネットワークの帯域によっては，非常に高品質な映像と音声による相互通信も可能となりつつある．その一方，端末の性能や利用可能な帯域は，利用者の通信環境によって異なるので，適切なフォーマットによる相互通信を実現するための仕組みが必要となる．RTP で定義されているトランスコーディング機能は，利用者の通信環境を考慮した相互通信を実現するために有効な機能であるが，通信環境の異なる複数の参加者を想定したトランスコーディング機能の動的な利用方法やシステムは確立していない．そこで本論文では，利用者環境に応じてトランスコーディング機能をネットワーク上へ動的に配置し，適切なフォーマットによる相互通信サービスを実現するためのミドルウェアを提案する．複数のマルチキャストセッションを動的に接続して 1 つの通信セッションを構成する仕組みを設計し，プロトタイプシステムによる評価実験を通して，スケーラブルな相互通信サービスの基盤技術となる機能について述べる．

A Middleware for Intercommunication Adapted to User's Communication Environments

Koji Hashimoto[†] and Yoshitaka Shibata[†]

By the wide spread of broad band network services, intercommunication services are realized by even personal computers that are using at ordinary. We can use high quality video communication services if our communication environments have enough resources. However, our computer's capabilities and available network bandwidths are different respectively. Therefore, it is important for the communication system to use suitable formats according to users' communication environments. Although the transcoding function defined by RTP is a useful function to realize intercommunication adapted users' communication environments, dynamic transcoding mechanisms have not established for multi-user environments. In this paper, we propose a middleware system to realize intercommunication services with suitable media format according to users' environments. The middleware system integrates multicast communication sessions by using relocatable transcoding functions. We have designed and implemented the prototype system, and through the results of an experimentation we describe about basic mechanisms for scalable communication services.

1. はじめに

近年，急速に普及を続けるブロードバンドネットワークにより，テレビ電話やビデオ会議などの相互通信サービスは実用的なものとなった．通信に利用する端末の性能や利用可能なネットワークの帯域によっては，非常に高精細な映像や高品質な音声ストリームによる相互通信も可能となりつつある．一方，端末の性能や利用可能なネットワークの帯域は，利用者の環境によって異なるので，たとえばギガビットクラスのネットワーク上で高性能な端末を用いて行われる相互通信セッションに，比較的狭帯域なネットワーク利用者がノートパソコンなどを利用して参加することも十分に想定される．

しかしながら既存の相互通信システムは，H.323 や SIP [1] によるシグナリングプロトコル上で H.263/264, MPEG4 などのビデオフォーマットを用いるものが多く，利用者の通信環境や QoS 要求に応じて複数のビデオフォーマットを用いた相互通信を実現するための十分な仕組みは実現されていない．たとえば既存の相互通信システムを利用して，DV（Digital Video）による通信セッションと MPEG4 による通信セッションを必要に応じて動的に統合し，1 つの相互通信を実現することは困難である．ここで，RTP [2] で定義されているトランスコーディングやミキシングの機能は，利用者の通信環境や QoS 要求に応じた適切なフォーマットによる相互通信を実現するための重要な機能で

† 岩手県立大学
Faculty of Software and Information Science, Iwate Prefectural University

あると考えられる．しかしながら，複数の参加者を想定した相互通信におけるトランスコーディングやミキシング機能の動的な利用に関する方法論やシステムは確立していない．そこで本論文では，利用者環境に応じてトランスコーディング機能をネットワーク上へ動的に配置し，適切なフォーマットによる相互通信サービスを実現するためのミドルウェアを提案する．

既存の関連研究として文献3)では，RTPマルチキャストセッション上のメディアストリームに対し，トランスコーディングやミキシング処理をアプリケーションレベルで行うメディアゲートウェイシステムが提案されている．トランスコーディングやミキシング処理を分散させるためのプロトコルも提案されているが，複数のマルチキャストセッションを統合した通信セッションを実現する方法については述べられていない．

また，ビデオや音声ストリームによるPeer-to-Peer通信を実現するシステムに関する研究[4),5)]も行われており，ライブ型の相互通信においてもスケーラビリティが重要な項目となっている．その1つとして文献6)では，複数のビデオストリームを空間的にタイル状に並べて1本のビデオストリームとし，そのビデオストリームをマルチキャストすることにより，多人数の相互通信を実現している．スケーラビリティを上げるためにビデオストリームのミキシングを行っているが，その動的利用法やフォーマットの適合に関しては言及していない．

一方，相互通信を実現する基盤技術は，アプリケーションとの組合せで様々な応用が考えられ，ミドルウェアとしての側面も重要となる．分散システムのためのミドルウェアとして，リアルタイム処理を必要とする組み込み型分散アプリケーションのためのミドルウェア[7)]や，Peer-to-Peerのオーバレイネットワーク上でpublish/subscribe型のイベント通信を基盤とするミドルウェア[8)]，認証の仕組みを取り入れたセキュアな分散サービスのためのミドルウェア[9)]などが提案されている．また，近年，分散システム構築手段の1つとして，マルチエージェントや移動エージェントを基盤とする研究も行われており，QoSを考慮したエージェントベースのミドルウェア[10)]や，ユビキタスサービスのための軽装なエージェントプラットフォーム[11)]，異機種混合の分散システムにおけるエージェント指向プログラミングを実現するためのミドルウェア[12),13)]などが提案されている．これらのミドルウェアは，各種の分散アプリケーション構築のために様々な機能を提供する．

しかしながら，相互通信機能をアプリケーションに提供することを考慮すると，通信資源の管理を含むセッション管理機能がミドルウェアに含まれるべきであり，既存のミドルウェアの機能では十分とはいえない．本論文で提案するシステムは，音声やビデオによる相互通信を必要とするアプリケーションに対して相互通信機能を提供するミドルウェアである．柔軟な相互通信機能を実現するために移動エージェントベースのトランスコーディング機能[14)]を利用し，複数のマルチキャストセッションを動的に接続する仕組みを実現する．

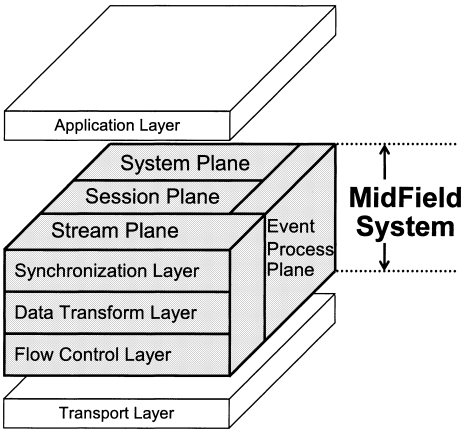

図1 MidFieldシステムアーキテクチャ
Fig. 1 MidField system architecture.

以下，2章では，提案するシステムと動的な相互通信環境の構築について概説する．続く3章では，提案システムにおける相互通信セッションの設計について述べる．そして4章では，提案システムの実装とプロトタイプシステムを用いた相互通信実験の結果を示す．

2. MidFieldシステム概要

利用者の通信環境やQoS要求に応じて適切なフォーマットによる相互通信サービスを実現するためのシステムアーキテクチャを**図1**に示す．図1のMidField☆システム[14)]は，ネットワーク接続されたコンピュータシステムのトランスポート層より上位に位置し，アプリケーション層に対して相互通信機能を提供する．本システムの機能は4つに大別され，Stream Planeは，メディアの同期・変換・フロー制御といったメディア処理を行い，Session Planeは通信セッションの管理を行う．System Planeでは，システム間のメッセージ通信

☆ **Mid**dleware for **F**lexible **i**ntercommunication **e**nvironment by re**l**ocatable **d**ecision

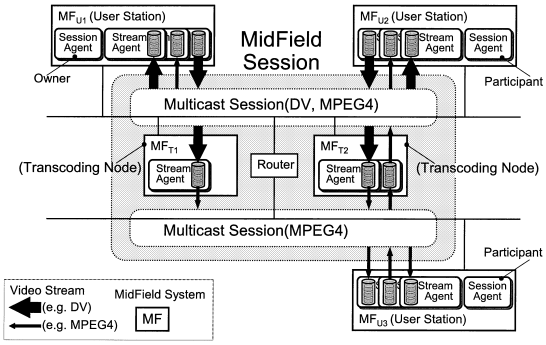

図 2 MidField セッションの動的構成例
Fig. 2 Example of dynamic configuration by MidField session.

を実現するとともに，入出力ビットレートや CPU 利用状況の監視と，メディアストリーム処理に対するアドミッションテストを実行する．また，Event Process Plane では，システム内部で発生する各種のイベントを処理する．

ここで，MidField システムにおける相互通信セッションを MidField セッションと呼ぶ．MidField セッションは，移動エージェントベースのトランスコーディング機能が複数のマルチキャストセッションを動的に接続することで実現される．その動的構成例を**図 2** をもとに概説する．図 2 では，コンピュータネットワーク上に 5 台の MidField システムが接続されており，そのうち 3 台は利用者端末（MF_{U1}〜MF_{U3}），残りの 2 台はトランスコーディングノード（MF_{T1}，MF_{T2}）として稼動している．図中，セッションエージェント（Session Agent）は，図 1 における Session Plane の機能を実装するエンティティであり，ストリームエージェント（Stream Agent）は，Stream Plane の機能を実装するエンティティである．

まず最初に，MF_{U1} のセッションエージェントが MidField セッションを生成する．図 2 における MidField セッションは 2 つのマルチキャストセッションから構成されている．1 つは DV ストリームの送受信を想定したマルチキャストセッションであり，もう 1 つは MPEG4 ビデオストリームの送受信を想定したセッションである．MidField システムでは，MidField セッションを生成したセッションエージェントをセッションオーナと呼び，以後，この MidField セッションの参加者数やストリーム数などの情報を管理する．

次に，MF_{U2} のセッションエージェントが MidField セッションに参加する．ここで，MF_{U1} および MF_{U2} は各々DV ストリームを用いた相互通信を行うために十分な CPU 資源とネットワーク帯域幅を利用できるものとする．そこで，DV ストリームの送受信を想定したマルチキャストセッションを利用し，MF_{U1} と MF_{U2} の各ストリームエージェントは DV ストリームによる相互通信を開始する．

その後，MF_{U3} のセッションエージェントが MidField セッションに参加する．ここで MF_{U3} は，DV ストリームを用いた相互通信を行うための資源を確保できず，MPEG4 ビデオストリームを利用した相互通信を要求する．この場合，MF_{U1} と MF_{U2} の各ストリームエージェントは，各々が送信している DV ストリームを MPEG4 ビデオストリームへ変換するために適切なトランスコーディングノードを選択し，トランスコーディング処理を行うストリームエージェントを送り込む．これをストリームの拡張と呼び，トランスコーディング処理を行うストリームエージェントをトランスコーディングエージェントと呼ぶ．

トランスコーディングノードでトランスコーディングエージェントが処理を開始すると，DV ストリームは MPEG4 ビデオストリームへ変換され，MPEG4 ビデオストリームを想定したマルチキャストセッションへ送信される．これにより，MF_{U3} は MF_{U1} と MF_{U2} のビデオストリームを受信することが可能となる．

一方，MF_{U3} が MPEG4 ビデオストリームを送信する場合も，ストリーム中継用のトランスコーディングノードが選択される．図 2 では MF_{T2} がその中継処理を行うことにより，MF_{U3} が送信した MPEG4 ビデオストリームを MF_{U1} と MF_{U2} も受信することが可能となる．

以上，DV と MPEG4 を想定したマルチキャストセッションをトランスコーディングエージェントが動的に接続することで，2 つのマルチキャストセッションを統合した相互通信セッションを実現する例を示した．続く 3 章では，このような相互通信セッションを実現するためのセッションエージェントとストリームエージェントの機能を述べる．

3. 相互通信セッション

2 章で述べた MidField セッションを実現するために，まず，セッションエージェントが MidField セッションを管理するために利用するセッションプロパティのデータ構造を**図 3** に示す．セッションプロパティは MidField セッションごとに 1 つ存在し，1 つの MidField セッション情報と複数の RTP セッション情報により構成され，セッションオーナが管理する．

MidField セッション情報は，セッションオーナが稼動しているホスト名と MidField セッション識別用

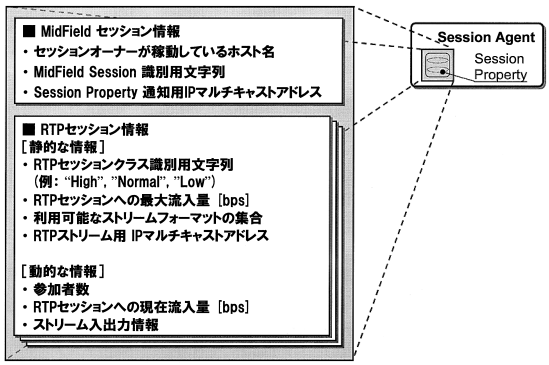

図 3 セッションプロパティのデータ構造
Fig. 3 Data structure of a Session Property.

文字列，そしてセッションプロパティ通知用 IP マルチキャストアドレスで構成される．これらのデータは MidField セッション生成時にセッションオーナが決定し，MidField セッション終了時まで変更されない静的な情報である．

一方，RTP セッション情報は静的な情報と動的な情報により構成される．静的な情報には，RTP セッションクラスを識別する文字列，RTP セッションへのストリームの最大流入量 [bps] と利用可能なストリームフォーマットの集合，そして利用する IP マルチキャストアドレスが含まれる．各 RTP セッションにおけるメディアデータ転送は，この IP マルチキャストアドレスを用いて行われる．参加者は，利用可能なストリームフォーマットと最大流入量を参照することにより，各自の相互通信環境に応じて適切な RTP セッションを選択することが可能となる．動的な情報には，RTP セッションへの参加者数，RTP セッションで用いられているストリームの現在流入量 [bps] とストリームの入出力情報が含まれる．これらの動的な情報は，セッションオーナにより適宜更新され，更新されたセッションプロパティはセッションプロパティ通知用 IP マルチキャストアドレスを用いて参加者すべてに通知される．

このようなデータ構造に基づき，セッションオーナと参加者が次節で述べる状態マシンに従って連係動作することによって，MidField セッションによる相互通信が実現する．

3.1 状態マシン

セッションエージェントは，セッションオーナまたは参加者として，それぞれの有限状態の中で動作する．

表 1 は状態遷移に関連するセッションエージェントの主なアプリケーションインタフェースを示している．セッションエージェントは MidField セッションに関するメソッドをアプリケーションへ提供するとともに，非同期に発生するイベントをアプリケーションへ通知する．

表 1 アプリケーションインタフェース
Table 1 Application interface methods.

外部インターフェースメソッド		
	open() / close()	MidField Session を開く / 閉じる.
	join() / leave()	〃 に参加する / 退出する.
	addStream()	〃 へストリームを追加する.
	removeStream()	〃 からストリームを削除する.
	updateProperty()	最新の Session Property を取得する.
	confirmProperty()	Session Property の確認をとる.
	terminate()	全ての処理を終了する.
アプリケーションへの通知		
	F-SUE	Fire Session Update Event MidField Session の状態を通知する.
	F-SEE	Fire Session Exception Event MidField Session 内で発生した例外を通知する.

表 2 システム内部イベント/メソッド
Table 2 Internal events and methods.

システム内部イベント		
	COMPLETION	拡張を含むストリーム追加処理完了.
	EX	Exception メッセージ配送に関する例外.
システム内部メソッド		
	modifyProperty()	Session Property を更新する.
	admissionTest()	ストリーム追加時の流入量をテストする.
	checkDestination()	ストリームの到達確認を行う.

状態遷移に関連するシステム内部イベントとメソッドは**表 2** に示すとおりである．システム内部イベントとしては，ストリームの拡張を含むストリーム追加処理の完了イベントと，メッセージ受信待ちタイムアウトを含むメッセージ配送に関する例外イベントがある．また，セッションオーナがセッションプロパティを更新するためには modifyProperty() メソッドを使い，参加者が送信ストリームをセッションへ追加する際には，セッションオーナが admissionTest() メソッドを用いて RTP セッションへの流入量オーバを確認する．一方，セッションオーナはセッションプロパティを適宜更新し，参加者に通知する．更新されたセッションプロパティを受信した各参加者は checkDestination() メソッドを呼び出し，各自の送信ストリームがすべての RTP セッションへ到達しているかどうかの確認を行う．

表 3 は，セッションオーナと参加者間で送受信されるメッセージを示している．これらのメッセージにより，MidField セッションへの参加と退出，ストリームの追加と削除，およびセッションプロパティの通知と確認が行われる．

表 3 Session Agent 間メッセージ
Table 3 Inter-session agent messages.

メッセージ（ R-{MSG} ： 受信メッセージ，S-{MSG} ： 送信メッセージ）	
JSReq / JSRes	Join Session Request / Join Session Response. MidField Session への参加要求 / 応答.
ASReq / ASRes	Add Stream Request / Add Stream Response. MidField Session へのストリーム追加要求 / 応答. ※ASRes [A] : ストリーム追加受理(ACCEPTED) ASRes [R] : 〃 却下(REJECTED)
RSReq / RSRes	Remove Stream Request / Remove Stream Response. MidField Session からのストリーム削除要求 / 応答.
LSReq / LSRes	Leave Session Request / Leave Session Response. MidField Session からの退出要求 / 応答.
SPReq / SPRes	Session Property Request / Session Property Response 最新の Session Property の要求 / 応答.
CPReq / CPRes	Confirm Property Request / Confirm Property Response. Session Property の確認要求 / 応答.
UPRep	Update Property Report 更新された Session Property の通知. ※参加者全員にマルチキャスト.
CSCom	Close Session Command. MidField Session の終了命令. ※参加者全員にマルチキャスト.

表 1 から表 3 において，セッションエージェントが処理するイベントには，(a) 外部インタフェースメソッドの呼び出し，(b) システム内部イベント，(c) 受信メッセージの 3 種類がある．イベント発生時には，(a) アプリケーションへの通知，(b) システム内部メソッドの呼び出し，(c) メッセージの送信といったアクションが実行される．これらのイベントとアクションによりセッションエージェントは役割に応じた状態遷移を行う．

図 4 はセッションオーナの状態マシンを示している．MidField セッションを開始するためには，まずアプリケーションがセッションプロパティを用意する．次に，セッションオーナへセッションプロパティを与えることにより，セッションオーナは初期状態から IDLE 状態へ遷移する．そして，open() メソッド呼び出しにより OPENED 状態へ遷移し，以後 MidField セッションを利用した相互通信が可能となる．

OPENED 状態では，セッションへの参加/退出，ストリームの追加/削除要求を参加者から受信するごとに，セッションオーナはセッションプロパティを更新して，MidField セッションの参加者すべてにその更新内容を通知する．ストリームを追加する場合は，追加する RTP セッションにおける最大流入量を超えないようにアドミッションテストが行われる．また，最新のセッションプロパティ要求を参加者から受信した場合，現在のセッションプロパティを返信する．

一方，アプリケーションが comfirmProperty() メソッドを呼び出すと，CONFIRMING 状態へ遷移し，

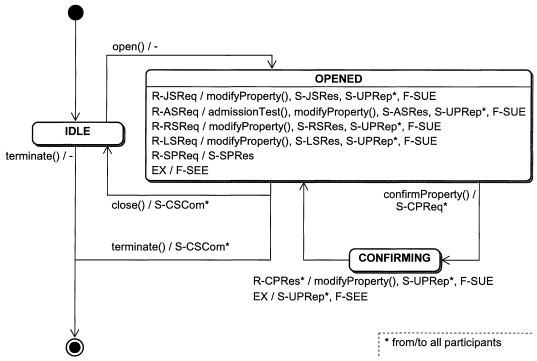

図 4 セッションエージェント（オーナ）の状態マシン
Fig. 4 State machine of Session Owner.

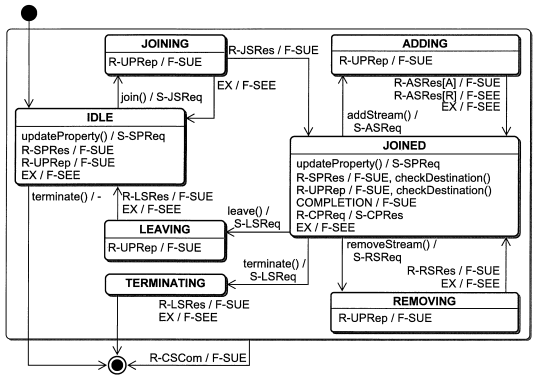

図 5 セッションエージェント（参加者）の状態マシン
Fig. 5 State machine of Session Participant.

参加者全員に対して参加の確認をとる．その際，送信ストリームの入出力情報も集め，セッションプロパティを更新する．この機能は，メッセージ配送におけるエラーなどによりセッションプロパティの整合性をとる必要が生じた場合に利用する．

MidField セッションに参加するセッションエージェントの状態マシンは **図 5** に示すとおりである．まず参加者は，セッションオーナが稼動している MidField システムよりセッションプロパティを取得し，初期状態から IDLE 状態へ遷移する．参加者は，セッションプロパティの RTP セッション情報から適切な RTP セッションを選択し，join() メソッドを呼び出す．join() メソッドは MidField セッションへの参加要求をセッションオーナへ送信し，参加者は JOINING 状態へ遷移する．そして，セッションオーナから参加応答を受けると JOINED 状態へ遷移する．もし，メッセージ配送などのエラーが発生した場合は IDLE 状態へ戻る．また，これらの状態遷移に関する情報やエラー情報は，非同期イベントとしてアプリケーションへ通知

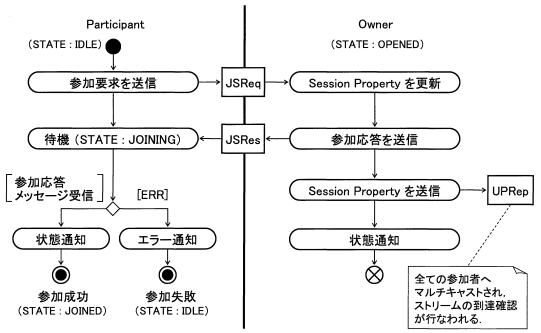

図 6 MidField セッションへの参加
Fig. 6 Joining to a MidField session.

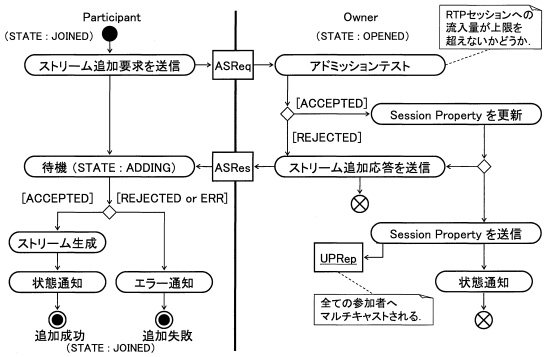

図 7 ストリームの追加
Fig. 7 Adding stream to the joined session.

される．

JOINED 状態に遷移すると，参加者はストリームの追加と削除が可能となる．ストリームの追加には addStream() メソッドを用いて，セッションオーナへストリーム追加要求を送信し，ADDING 状態へ遷移する．セッションオーナ側で実行されるアドミッションテストの結果，要求したストリームを RTP セッションへ追加可能であれば，ストリームを追加して JOINED 状態へ戻る．追加不可能な場合，その理由をアプリケーションへ通知して JOINED 状態へ戻る．また，更新されたセッションプロパティをセッションオーナから受信すると checkDestination() メソッドを呼び出し，各自が送信しているストリームの到達確認が行われる．この到達確認の結果によって，3.3 節で述べるストリームの拡張と縮退が行われる．

次に，MidField セッションへ参加する際のメッセージフローを **図 6** に示す．参加者からの要求に対し，セッションオーナはセッションプロパティを更新し，その参加者へ応答を返す．一方，更新されたセッションプロパティはすべての参加者へマルチキャストで通知される．セッションからの退出とストリームの削除に関するセッションオーナと参加者間のメッセージフローもほぼ同じ流れとなる．

ストリーム追加に関するメッセージフローは **図 7** に示すとおりであり，アドミッションテストの結果ストリームが追加可能である場合，それ以降のフローは図 6 と同様のものとなる．

このように，MidField システムにおける相互通信は，セッションオーナが管理するセッションプロパティをもとに行われる．セッションプロパティが更新されるとセッションオーナはすべての参加者へ通知し，更新されたセッションプロパティを受信した参加者はストリームの到達確認を行い，必要に応じてストリームの拡張と縮退を実行する．

3.2 フォーマットのマッピング

ストリームの拡張により複数の RTP セッションを動的に接続するためには，各 RTP セッションで利用可能なフォーマットのマッピングが可能であり，実装モジュールはマッピングに従ってトランスコーディング処理が可能でなければならない．実装に依存する部分ではあるが，セッション開始前にセッションプロパティの静的な情報を決定する際，フォーマットのマッピングを考慮する必要がある．

MidField システムでは，あるフォーマット（fmt）のメディアストリームを RTP セッション（rs_n）に追加する際，fmt は，rs_n で利用可能なフォーマット fmt' へマッピング可能であるものとし，fmt および fmt' は以下の条件を満たす必要がある．

$$fmt' = rs_n.map(fmt)$$
$$fmt \in \bigcup_{1 \le i \le N} rs_i.fmtSet$$
$$fmt' \in rs_n.fmtSet$$

ここで，rs_n は，MidField セッション内の RTP セッション ($n = 1, \ldots, N$) であり，$rs_n.map()$ は，RTP セッションごとに定義されるフォーマットのマッピング関数である．そして，$rs_n.fmtSet$ は rs_n において利用可能なメディアフォーマットの集合である．また，$fmt \in rs_n.fmtSet$ のとき，fmt' と fmt は同じフォーマットである．

図 8 はフォーマットのマッピング例を示している．図 8 の MidField セッションは 2 つの RTP セッションから構成される．RTP セッション 1（rs_1）で利用可能なフォーマットは DV, MPEG4, PCM であり，RTP セッション 2（rs_2）で利用可能なフォーマットは MPEG4，PCM である．このとき，rs_1 から rs_2 へ DV ストリームを拡張する場合，DV ストリームは MPEG4+PCM へトランスコーディングされる．一

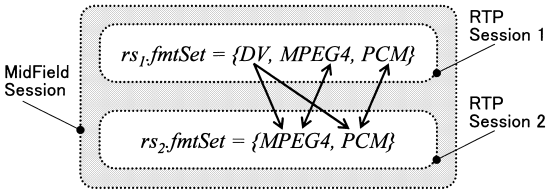

図 8 フォーマットのマッピング例
Fig. 8 Example of mapping format.

方，MPEG4 と PCM ストリームを拡張する場合は中継のみ行われる．

また，ストリームの拡張時には各 RTP セッションの流入量に対するアドミッションテストがセッションオーナによって実行される．アドミッションテストは，セッションプロパティに含まれる RTP セッションごとの最大流入量を上限とした相互通信を実現するために必要な機能である．マッピングされたフォーマットから算出した帯域幅を現在流入量に加えた結果が，最大流入量を超えない場合ストリームの拡張が行われる．

3.3 ストリームの拡張と縮退

3.1 節で述べたように，更新されたセッションプロパティを受信した各参加者はストリームの到達確認をし，必要に応じてストリームの拡張と縮退を行う．その概略を **図 9** に示す．

図 9 (a) に示すとおり，参加者のいる RTP セッションへストリームが到達していない場合，ストリームを拡張する．一方，ストリーム到達確認の結果，参加者のいない RTP セッションへストリームを拡張している場合，図 9 (b) に示すとおりストリームを縮退させる．

ストリームを拡張する際は，各 RTP セッションへ追加すべき出力フォーマットをフォーマットのマッピング関数を利用してセッションエージェント（参加者）が決定し，適切なトランスコーディングノードをストリームエージェントが選択する．そしてストリームエージェントは，自身の出力を入力とするトランスコーディングエージェントを生成し，選択したトランスコーディングノードへトランスコーディングエージェントが移動する．

図 10 は，ストリーム拡張時のストリームエージェントの処理フローを示している．まず初めに，資源利用状況を収集するための要求メッセージを稼働中のトランスコーディングノードへマルチキャストする．これに対して，稼働中のトランスコーディングノードにおける System Plane のエンティティは現在の資源利用状況を返信する．資源利用状況には，各トランスコーディングノードにおいて利用可能な入出力帯域幅の上限値と利用状況 [bps] および CPU 利用率 [%] が含まれる．資源利用状況を受信したストリームエージェントは，各トランスコーディングノードの資源利用状況を要素とする稼動システムリストを生成し，資源利用状況を受信するごとに追加更新する．

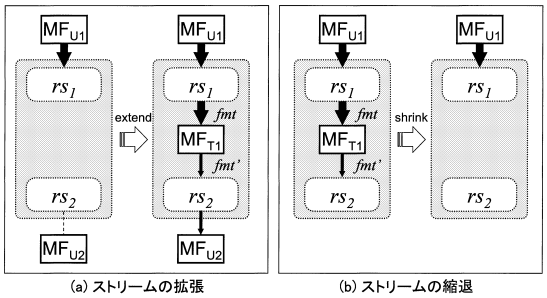

図 9 ストリームの拡張と縮退
Fig. 9 Extending or shrinking a stream.

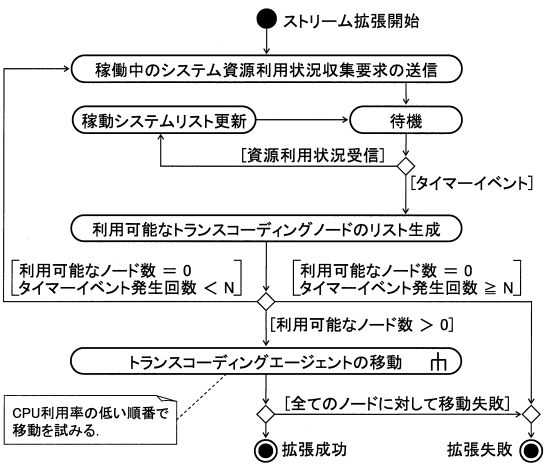

図 10 ストリームの拡張
Fig. 10 Stream extension.

一方，ストリーム拡張開始時にストリームエージェントはタイマを起動し，一定間隔で発生するタイマイベントを受け取る．タイマイベントを受け取ったストリームエージェントは，利用可能なトランスコーディングノードのリストを稼動システムリストをもとに生成する．リストの要素は，必要な送受信帯域幅と十分な CPU 資源を持つノードの資源利用状況である．そして，CPU 利用率の低いトランスコーディングノードから順番に移動を試みる．

もし，タイマイベント発生回数が一定の値（N）未満で，かつ，利用可能なトランスコーディングノードが存在しない場合，ストリームエージェントは資源利

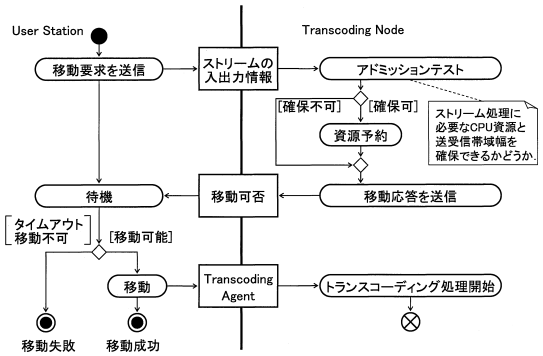

図 11 トランスコーディングエージェントの移動
Fig. 11 Migration of a Transcoding Agent.

用状況を収集するための要求メッセージを再度マルチキャストする．また，利用可能なトランスコーディングノードが存在せず，タイマイベント発生回数が一定の値を超えた場合，必要な資源が確保できないと判断し，ストリームの拡張は失敗となる．次章で述べるプロトタイプシステムでは，タイマイベントの発生間隔を 1 秒とし，発生回数の上限は 3 回としている．

トランスコーディングエージェントがストリーム拡張のために移動する際のフローは **図 11** に示すとおりである．まず，移動先のトランスコーディングノードにおける System Plane のエンティティに対し，ストリームエージェントはトランスコーディング処理のための入出力情報を送信する．次にトランスコーディングノードでは，トランスコーディング処理に必要となる CPU 資源と送受信帯域幅に対してアドミッションテストを実行し，確保可能であれば資源の予約を行う．そして，トランスコーディングエージェントがトランスコーディングノードへ移動し，処理を開始する．

一方，各 RTP セッションで利用可能なストリームフォーマットすべてに対してストリームの拡張が失敗した場合，MidField システムを利用するアプリケーションへ失敗の原因を含むイベントが通知され，ストリーム拡張処理は終了する．

ここで，各 RTP セッションで利用可能なストリームフォーマットがセッションプロパティに複数設定されている場合，あるフォーマットによるストリームの拡張に失敗した際には他のフォーマットでストリームの拡張を試みる．同一フォーマットで品質が異なるストリームを利用することも考慮しており，たとえばストリームの拡張に失敗した場合，ストリームの品質を徐々に下げながら拡張を行うといった制御が可能となっている．

ただし，各 RTP セッションごとにあらかじめ設定しておくフォーマットは，3.2 節で述べたマッピングの条件を満たしている必要がある．つまり，ある RTP セッションで利用可能な任意のフォーマットは，他の RTP セッションで利用可能なフォーマットへのトランスコーディング処理が実装レベルで可能でなければならない．

3.4 CPU 利用率の計測

ストリーム拡張時におけるトランスコーディングノードの選択には，稼働中のシステムにおける利用可能な入出力帯域幅と CPU 利用率を用いる．ストリームの入出力情報から必要な帯域幅を見積もることは可能であるが，必要となる CPU 資源はトランスコーディングノードの性能によって異なる．そこで，MidField システムにおけるトランスコーディングノードは，トランスコーディング処理に必要となる CPU 資源をあらかじめ測定しておき，テーブルとして利用する．MidField システムでは，CPU 利用率の移動平均の標準偏差を用いて，トランスコーディング処理に必要となる CPU 資源を特定する簡単な方法を導入した．

CPU 利用率は，ノードの負荷を測る 1 つの値となる一方，一定の負荷状態においてもその値にばらつきが見られるので，平滑化した値を用いる必要がある．特定の処理のみを連続的に行う場合，一定時間の移動平均をとることによって CPU 利用率の値が収束することを前提とし，まず，ある時刻 t における N 秒間の移動平均を $cpu\_ma(t)$ により求める．ここで，$cpu\_raw(t)$ は，時刻 t における CPU の利用率である．

$$cpu\_ma(t) = \frac{1}{N} \sum_{i=t-N+1}^{t} cpu\_raw(i)$$

次に，ある時刻 t における移動平均の T 秒間の標準偏差を，$cpu\_sd(t)$ により求める．ここで，$\overline{MA}$ は，ある時刻 t における移動平均の T 秒間の平均である．

$$cpu\_sd(t) = \sqrt{\frac{1}{T} \sum_{j=t-T+1}^{t} (cpu\_ma(j) - \overline{MA})^2}$$

MidField システムでは，あるメディア処理を行っている際，$cpu\_sd(t)$ の値が十分小さくなった時刻 t における移動平均値を，そのメディア処理に必要な CPU 資源としている．次章で述べるプロトタイプシステムを用いた機能評価実験では，$N = T = 60$ 秒 とし，$cpu\_sd(t) < 1$ を満たす時刻 t における $cpu\_ma(t)$ を，そのメディア処理に必要な CPU 利用率とした．

4. プロトタイプシステム

本論文で提案している MidField システムのプロト

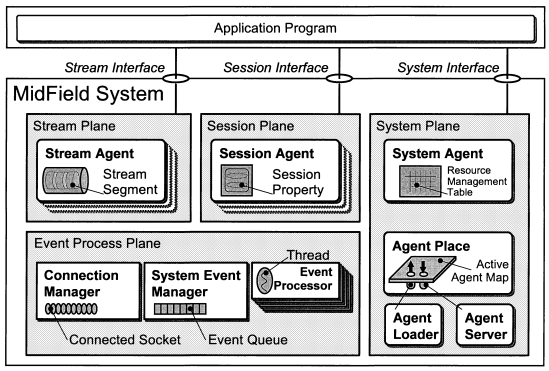

図 12 機能モジュール構成
Fig. 12 Functional modules' configuration.

タイプと簡単なビデオ会議アプリケーションを，Windows XP 上に Java，C，C++言語を用いて実装した．プロトタイプシステムの実装に利用した PC は，Dell WORKSTATION PWS360，Intel(R) Pentium(R) 4 3.20 Ghz，2.00 GB RAM のマシンで，ネットワークアダプタは Intel(R) PRO/1000 MT Network Connection を使用している．

以下，図 1 で示した MidField システムの機能モジュール構成と実装技術について述べ，簡単な機能評価として 2 つの RTP セッションで構成される MidField セッションによる相互通信を行った結果を示し，その考察を述べる．

4.1 機能モジュール構成

MidField システムの機能モジュール構成を**図 12** に示す．MidField システムの各プレーンの機能は，Stream Interface，Session Interface，System Interface によりアプリケーションプログラムへ提供される．そして，これらのインタフェースに対して，Stream Agent，Session Agent，System Agent がアプリケーションからの要求を処理する．

Stream Agent は Stream Segment 用いて RTP ストリームの送受信およびトランスコーディング処理を実現する．プロトタイプシステムの実装には主に Java 言語を用いているが，多くの CPU 資源と大量のメモリコピーを必要とする Stream Segment の実装には C++および C 言語を用いた．Stream Segment における各種のメディア処理には DirectX 9.0 DirectShow [15] を，RTP 通信部には RTPlib 1.02b [16] を利用している．RTP のペイロードとなるフォーマットとしては，DV，MJPEG，MPEG4，PCM に対応しており，DV から MJPEG+PCM へのトランスコーディングと DV から MPEG4+PCM へのトランスコーディングが可能となっている．一方，RTP セッションを動的に接続するために Stream Agent は適切なトランスコーディングノードへ移動するが，移動の際は JVM 上の実行イメージと，関連するクラスデータが必要に応じて転送される．したがって，トランスコーディングエージェントが利用する Stream Segment の実装モジュールは，各トランスコーディングノードにあらかじめインストールされているものが利用される．

Session Agent は Session Property を所有し，複数の RTP セッションを統合して MidField セッションを実現するための管理機能を実現する．Session Property のデータ構造は図 3 に示したとおりである．MidField セッションでは複数の IP マルチキャストアドレスを用いるが，プロトタイプシステムではあらかじめ設定された特定のアドレス空間から，利用するマルチキャストアドレスを決定する簡単な仕組みを実装している．

System Agent は CPU 利用率と RTP ストリームの入出力ビットレートを監視し，トランスコーディングエージェントが移動する際には，アドミッションテストを実行する．また，アドミッションテストを行うために，メディア処理に必要となる CPU 資源をあらかじめ計測した結果を Resource Management Table 内に保持している．CPU 利用率の取得には，PDH（Windows Performance Data Helper DLL）ライブラリを利用した．

Agent Place は各エージェントの生成，起動，移動そして終了処理を担当する．プロトタイプシステムでは，Java のオブジェクトシリアライゼーション機能を利用して簡単な移動エージェントの仕組みを実現している．また，移動したエージェントのインスタンスが必要とするクラスデータをオンデマンドで移動元からロードするための Agent Loader と Agent Server を実装した．Agent Place も MidField システムにおけるエージェントの 1 つである．

MidField システムにおける各エージェントのスーパクラスはエージェント間通信用のプロトコルハンドラを実装している．これにより，MidField システム内の各エージェントはそれぞれ任意のエージェントとのメッセージ送受信が可能となっている．現在のプロトタイプシステムにおけるエージェント間通信プロトコルはテキストベースの独自のものであるが，スーパクラスのプロトコルハンドラを追加実装することで，既存のエージェント間通信プロトコルへの対応も考慮した実装となっている．また，各エージェントの名前は，MidField システムが稼動している端末の IP アドレスと，その MidField システムで生成されたエージェントのシーケンス番号を含む文字列で構成される．

MidField システムでは，エージェントの名前解決機構をシステム外部に想定しているが，現在のプロトタイプシステムには名前解決機構との連携部分は含まれていない．

一方，Event Process Plane の Connection Manager は，MidField システム間を接続したソケットを管理し，システム間のパケット通信を実現する．MidField システム内のエージェントがメッセージを交換する際，Connection Manager は必要に応じてソケットを生成する．一度生成されたソケットのインスタンスは，宛先 IP アドレスをキーとしてハッシュマップに格納され，以後，同じ宛先 IP アドレスへのパケット送信時に利用される．

System Event Manager は，各エージェントが生成するシステム内部イベントを Event Queue に格納し，Event Processor の Thread で処理する仕組みを実現している．システム内部イベントとしては，エージェントの生成・到着およびエージェント間メッセージ用のパケット受信イベントがある．

プロトタイプシステムのエージェントの機能は，FIPA [17] 準拠のエージェントシステムなどに比べると簡略化されたものであるが，MidField セッションを実現するために必要となるエージェント間メッセージ通信機能を含め，任意のトランスコーディングノードでストリームエージェントを動作させるために必要な機能は十分に備えている．

4.2 トランスコーディング処理と CPU 利用率

System Agent はトランスコーディング処理に必要となる CPU 資源をあらかじめ計測し，その結果を保持する．トランスコーディング処理で必要となる CPU 利用率を決定する仕組みは 3.4 節で述べたとおりであり，**表 4** はその計測結果の一部を示している．

ここで，ビデオのフレームサイズは DV（720 × 480 [pixel]）を基準とし，MJPEG と MPEG4 のフレームサイズは DV の 1/4（360 × 240 [pixel]）としている．PCM のフォーマットは，量子化ビット数 16 [bit]，サンプリング周波数 32 [kHz]，チャンネル数を 2（ステレオ）とした．

表 4 より，プロトタイプシステムの実装で用いた PC では，DV ストリームを MJPEG+PCM ストリームへトランスコーディングするために約 38.1%の CPU 資源を必要とし，DV ストリームを MPEG4+PCM ストリームにトランスコーディングするためには約 20.5%の CPU 資源を必要とする．これより，CPU 資源のみを考慮するなら，DV ストリームを MJPEG+PCM ストリームにトランスコーディングする場合，1 台の PC で 2 本分，MPEG4+PCM ストリームにトランスコーディングするのであれば，1 台の PC でも 4 本分の処理を受け持つことが可能であると期待できる．

表 4 トランスコーディング処理と CPU 利用率
Table 4 CPU utilization for transcoding processings.

入力フォーマット	出力フォーマット	CPU利用率	必要な帯域幅
中継処理			
DV/RTP	DV/RTP	17.4 %	入出力： 約28.8 Mbps
MJPEG/RTP	MJPEG/RTP	10.5 %	入出力： 約20.0 Mbps
MPEG4/RTP	MPEG4/RTP	1.0 %	入出力： 約0.5 Mbps
PCM/RTP	PCM/RTP	0.9 %	入出力： 1.024 Mbps
トランスコーディング処理			
DV/RTP	MJPEG/RTP + PCM/RTP	38.1 %	入力： 約28.8 Mbps 出力： 約20.0 Mbps
DV/RTP	MPEG4/RTP + PCM/RTP	20.5 %	入力： 約28.8 Mbps 出力： 約1.5 Mbps

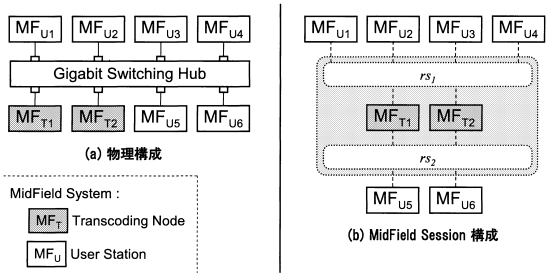

図 13 機能評価実験環境
Fig. 13 Functional evaluation environment.

そこで，本論文で提案している MidField セッションの動的構成機能を評価するために，DV ストリームと MPEG4+PCM ストリームを用いた相互通信実験を行った．

4.3 相互通信実験

MidField セッションの動的構成機能を評価するための実験環境を**図 13** (a) に示す．1 台のギガビットスイッチングハブ（PLANEX COMMUNICATIONS SF-0408G）に 6 台の利用者端末（MF_{U1}〜MF_{U6}）と 2 台のトランスコーディングノード（MF_{T1}，MF_{T2}）を接続した環境で，図 13 (b) に示す MidField セッションを構成した．また，この MidField セッションを構成する 2 つの RTP セッションで利用可能なフォーマットは以下のとおりとした．

$$rs_1.fmtSet = \{DV, MPEG4, PCM\}$$
$$rs_2.fmtSet = \{MPEG4, PCM\}$$

MF_{U1}〜MF_{U4} は RTP セッション rs_1 を利用して相互通信を行い，MF_{U5} と MF_{U6} は RTP セッション rs_2 を利用して相互通信を行う．必要に応じてトランスコーディングエージェントがトランスコーディングノード（MF_{T1}，MF_{T2}）へ移動し，rs_1 と rs_2 を動的に接続することにより，DV ストリームと

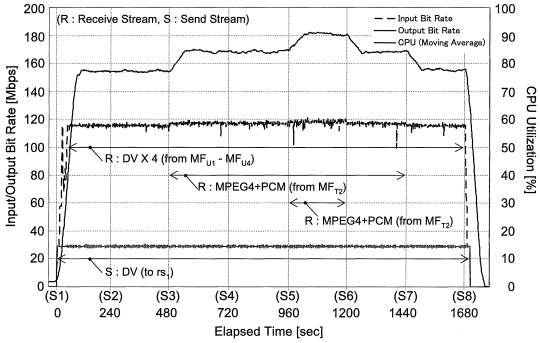

図 14 CPU 利用率と入出力ビットレート（MF_{U1}）
Fig. 14 CPU utilization and I/O bit rate (MF_{U1}).

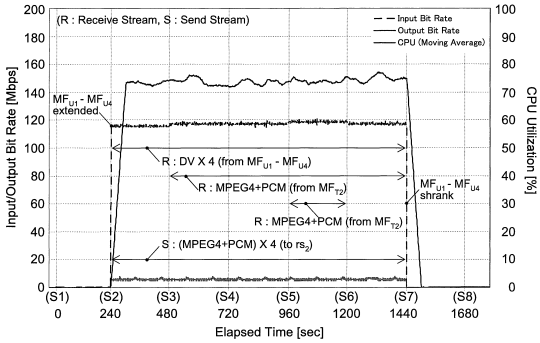

図 15 CPU 利用率と入出力ビットレート（MF_{T1}）
Fig. 15 CPU utilization and I/O bit rate (MF_{T1}).

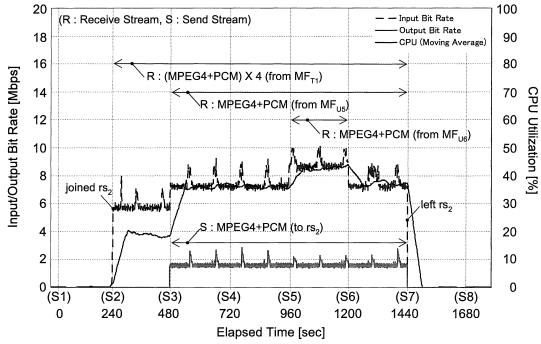

図 16 CPU 利用率と入出力ビットレート（MF_{U5}）
Fig. 16 CPU utilization and I/O bit rate (MF_{U5}).

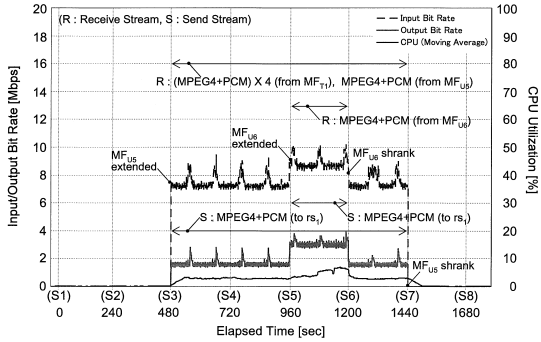

図 17 CPU 利用率と入出力ビットレート（MF_{T2}）
Fig. 17 CPU utilization and I/O bit rate (MF_{T2}).

MPEG4+PCM ストリームによる 6 者間通信を行う．

下記 (S1)〜(S8) に実験のシナリオを示す．

(S1) MF_{U1}〜MF_{U4} が rs_1 で相互通信を開始する．
(S2) MF_{U5} が rs_2 へ参加する．
(S3) MF_{U5} がストリームを追加し送信を開始する．
(S4) MF_{U6} が rs_2 へ参加する．
(S5) MF_{U6} がストリームを追加し送信を開始する．
(S6) MF_{U6} が rs_2 から退出する．
(S7) MF_{U5} が rs_2 から退出する．
(S8) MF_{U1}〜MF_{U4} が rs_1 から退出する．

この実験を通して，下記 (a)〜(c) に示すストリームの拡張と縮退機能の動作確認を行った．

(a) 参加にともなうストリームの拡張
(b) ストリームの追加にともなうストリームの拡張
(c) 退出にともなうストリームの縮退

実験結果として，MF_{U1} と MF_{T1} および MF_{U5} と MF_{T2} における CPU 利用率と入出力ビットレートを，**図 14**，**図 15**，**図 16**，**図 17** に示す．

以下，シナリオに沿った実験結果を述べる．

(S1) まず最初に MF_{U1}〜MF_{U4} が rs_1 へ参加する．そして MF_{U1}〜MF_{U4} 各々が DV ストリームの送信を開始し，各自が送信しているストリームを含めて 4 本分の DV ストリームを受信し始める．図 14 からは，(S1) から (S2) にかけて MF_{U1} が DV ストリームの送信を開始し，4 本分の DV ストリームを受信し始めたことが確認できる．

(S2) 次に，MF_{U5} が rs_2 へ参加する．この結果，MF_{U1}〜MF_{U4} は rs_2 へストリームを拡張する．MF_{U5} が rs_2 へ参加したことにより，MF_{U1}〜MF_{U4} はストリームの到達確認を行った．その結果，MF_{U1}〜MF_{U4} はストリームの拡張処理を開始し，トランスコーディングノードとして MF_{T1} が選択された．そして，MF_{U1}〜MF_{U4} の各ストリームエージェントはトランスコーディングエージェントを生成し，そのトランスコーディングエージェントが MF_{T1} へ移動し，トランスコーディング処理を開始した．図 15 の (S2) を見ると，MF_{T1} が rs_1 から 4 本分の DV ストリームを受信し，MPEG4+PCM ストリームを rs_2 へ送信し始めたことが確認できる．一方，MF_{T1} がトランスコーディング処理を始めたことにより，RTP セッション rs_2 へ MPEG4+PCM ストリームが 4 本流れ始める．これは，図 16 (S2) で MF_{U5} が rs_2 へ参加した後，入力ビットレートが約 6 Mbps で推移していることから確認できる．プロトタイプシステムで

用いている MPEG4+PCM ストリームのビットレートは最大約 1.5 Mbps 程度なので，MPEG4+PCM ストリームを 4 本受信すると約 6 Mbps となる．

(S3) シナリオ (S3) では，MF_{U5} が MPEG4+PCM ストリームを rs_2 へ追加して送信を開始する．その結果，MF_{U5} は rs_1 へストリームの拡張を行った．このストリームの拡張ではトランスコーディング処理を必要とせず中継のみが行われた．図 17 の (S3) より，その中継処理を MF_{T2} が担当していることが確認できる．ここで，MF_{T2} は MPEG4+PCM ストリーム 1 本を rs_2 から rs_1 へ中継するが，図 17 のとおり MF_{T2} は 5 本分の MPEG4+PCM ストリームを受信し，1 本分の MPEG4+PCM ストリームを送信している．これは，MF_{T2} が中継処理を行うために rs_2 で利用されている IP マルチキャストセッションに参加したため，rs_2 へ流入するすべてのストリームを受信しているからである．一方，MF_{T2} の中継した MPEG4+PCM ストリームを MF_{U1} が受信していることは，図 14 の (S3) から確認できる．図 14 と図 15 に対し，図 16 と図 17 の入出力ビットレートのスケールが異なることに注意し，図 14 の (S3) 以降の入力ビットレートを見てみると，若干（約 1.5 Mbps）増加している．また，受信した MPEG4+PCM ストリームを再生表示するために CPU 利用率も増加している．

(S4) 次にシナリオ (S4) では MF_{U6} が rs_2 へ参加する．各利用者端末ではストリームの到達確認が行われるが，すでに MF_{U5} が rs_2 へ参加しており，rs_2 に対してストリームの拡張が行われているので，拡張処理は必要ではない．実際，各図の (S4) ではストリームの拡張に関する変化が見られない．

(S5) 続くシナリオ (S5) では，MF_{U6} が MPEG4+PCM ストリームを rs_2 へ追加し，送信を開始した．その結果 rs_1 へのストリーム拡張が行われた．中継ノードには MF_{T2} が選択されている．その結果は各図の (S5) より確認できる．

(S6) シナリオ (S6) で MF_{U6} は rs_2 から退出する．これにより，MF_{U6} が rs_2 へ送信していた MPEG4+PCM ストリームが削除される．また，MF_{U6} のストリームを中継していた MF_{T2} からも MPEG4+PCM ストリームが削除されたことが図 17 の (S6) から確認できる．

(S7) そして，シナリオ (S7) では MF_{U5} が退出する．ここで rs_2 の参加者数は 0 となり，MF_{U1}〜MF_{U4} はそれぞれ rs_2 へ拡張していたストリームを縮退させた．ストリームが縮退した結果，MF_{T1} におけるトランスコーディング処理は終了する．これは各図の (S7) より確認できる．

(S8) 最後に，シナリオ (S8) では MF_{U1}〜MF_{U4} が rs_1 から退出して，相互通信セッションも終了する．

4.4 考　　察

相互通信実験を通して，シナリオ (S2) では参加にともなうストリームの拡張を確認し，(S3) と (S5) ではストリームの追加にともなうストリームの拡張を確認した．また，シナリオ (S7) では参加者の退出によるストリームの縮退も確認した．実験に用いた MidField セッションは RTP セッション 2 つから構成される簡単なものであり，ネットワーク帯域幅は十分確保可能な環境を利用した．しかしながら，トランスコーディングエージェントをトランスコーディングノードへ必要に応じて配置することにより，2 つの RTP セッションから構成される 1 つの相互通信セッションを実現することが可能となった．以下，実験結果をふまえて提案システムを考察し，検討課題を述べる．

（考察 1）不必要なデータの受信破棄

トランスコーディングノードにおいて，トランスコーディング処理の対象となるストリームデータを受信するためには，IP マルチキャストセッションへ参加する必要がある．しかしながら，処理の対象ではない不必要なデータも受信してしまう．今回の実験で MF_{T1} はトランスコーディング処理を行うために rs_1 へ参加し，MF_{U1}〜MF_{U4} が送信している 4 本の DV ストリームを受信している．一方で (S5)〜(S6) の区間では，MF_{U5} と MF_{U6} が rs_2 へ送信している 2 本の MPEG4+PCM ストリームを MF_{T2} が rs_1 へ中継している．したがって，rs_1 に参加している MF_{T1} は，トランスコーディング処理の対象ではない 2 本の MPEG4+PCM ストリームを受信し，これを破棄しなければならない．同じ理由で，(S5)〜(S6) の区間において MF_{T2} は，rs_2 から rs_1 へ中継すべき MPEG4+PCM ストリーム 2 本以外に，MF_{T1} が rs_2 へ送信している MPEG4+PCM ストリームを 4 本分受信し，破棄する必要がある．

今回の実験では，トランスコーディング処理に対して受信破棄の処理はほとんど影響していない．しかしながら，処理の対象ではないストリームデータの受信と破棄に対しても，トランスコーディングノードではネットワーク資源と CPU 資源を消費する．実際，プロトタイプシステムを実装した PC で 1 本の DV ストリームを受信破棄すると，約 6%の CPU 資源を消費する．したがって，トランスコーディングノードが参加する RTP セッションを制限したり，トランスコーディ

ングノードへのストリーム転送にはユニキャストを利用したりするなど，不必要なデータの受信と破棄を避ける仕組みを導入することにより，トランスコーディングノードにおけるネットワーク資源と CPU 資源の無駄な消費を減らすことが可能であると考えられる．

（考察 2）受信ソケット数と CPU 利用率

プロトタイプシステムを実装した PC において，DV ストリームを MPEG4+ PCM ストリームへトランスコーディングするために必要となる CPU 資源は表 4 のとおり約 20.5%であり，4 本分の処理を行う場合，約 82%の CPU 資源が必要となる．実際，4 本分のトランスコーディング処理を行うために受信ソケットを 4 つ使用した場合，CPU 資源は約 82%であることを予備実験で確認している．しかしながらシナリオ (S2) におけるストリーム拡張の結果，MF_{T1} が 4 本分のトランスコーディング処理を行うために消費した CPU 資源は図 15 より約 73%程度であることが確認できる．

プロトタイプシステムでは 1 つの IP マルチキャストセッションへ流れるすべてのストリームを 1 つのソケットで受信し，各ストリームの識別には RTP パケット内の SSRC（送信元識別子）を用いている．つまり，4 本分のトランスコーディング処理に対して受信ソケットを 4 つ使用した場合に比べて，1 つのソケットですべての受信処理を行う場合，CPU 資源の消費が少なくなると考えられる．したがって，受信ソケット数を考慮した受信データ量に対する CPU 利用率の関係を明確にすることで，トランスコーディングノードにおけるストリーム拡張時のアドミッションテストの精度を上げることが可能であると考えられる．

（考察 3）利用可能なフォーマット

3.3 節で述べたとおり，MidField セッションの仕組みとしては多種フォーマットの利用を考慮している．しかしながら，利用可能なフォーマットの数を増やすためには，コーデックとトランスコーディング処理モジュールの追加実装が必要となる．

現在のプロトタイプシステムでは，DV から MJPEG+PCM および MPEG4+PCM へのトランスコーディング処理モジュールを実装している．コーデックは OS 付属のものを利用しており，受信した DV フレームのビデオデータを RGB または YUV 形式に展開した後，MJPEG または MPEG4 へ圧縮している．

利用可能なフォーマットを増やすことによって相互通信の適用範囲は広がると考えられるが，効率の良いトランスコーディング処理モジュールを追加実装するためには，利用するコーデックに依存する各種のパラメータを調整する必要があり，コーデックごとの個別対応が必要となる．しかしながら，フォーマット追加実装のためのフレームワークを整備することは本システムの適用範囲を広げると考えられ，今後の検討課題とする．

（考察 4）フォーマットと相互通信拠点数

既存のテレビ会議システムで多く用いられている H.263/264 や MPEG4 などのフォーマットに対し映像品質を考慮する場合，DV の利用は 1 つの選択肢であると考えられる．MidField システムを利用すれば DV ストリームを利用した相互通信が可能となるが，図 14 の (S5)〜(S6) より，DV ストリームを 1 本送信しながら DV ストリーム 4 本と MPEG4+PCM ストリーム 2 本を MF_{U1} が受信するためには，約 90%の CPU 資源が必要となった．したがって，たとえ帯域幅が十分確保できる環境であっても，実験で用いた PC を利用する限り，DV ビデオストリームのみを利用した相互通信拠点数は，せいぜい 5 地点が上限となる．

一方，図 16 の (S5)〜(S6) より，MPEG4+PCM ストリームを 1 本送信し，6 本受信している MF_{U5} の CPU 利用率は約 43%であることが確認できる．通常のテレビ会議で話者の映像を伝達する程度の用途であれば，今回利用した MPEG4 の映像でも十分であると考えられるが，DV と比較すると映像の品質は明らかに低い．しかしながら相互通信拠点数を考慮した場合，実験で用いた PC で MPEG4+PCM ストリームのみを利用するなら 12 地点程度の相互通信に対応できると考えられる．

利用するフォーマットの品質と同時接続可能な相互通信拠点数にはトレードオフが生じるため，相互通信の内容を考慮して通信資源を調整する仕組みが検討されるべきである．たとえば，数拠点の相互通信において品質の良い映像と音声ストリームが要求される場合や，映像と音声のフォーマットは会話可能な程度で多くの相互通信拠点を接続したい場合がある．また，特定の参加者のみ高品質な映像の送信を行い，それ以外の参加者は比較的低品質なフォーマットを利用するといったことも想定される．

MidField システムは 1 つの MidField セッションにおいて複数フォーマットの利用が可能であり，相互通信の内容を考慮した通信資源調整を実現するための 1 つのプラットフォームになると考えられる．今後，(考察 1)〜(考察 3) を考慮してスケーラビリティを向上させるとともに，フォーマットと相互通信拠点数のトレードオフにも柔軟に対応するための仕組みを検討する．

5. まとめ

本論文では，利用者環境に応じて適切なフォーマットによる相互通信サービスを実現するためのミドルウェアを提案した．提案システムは，音声やビデオストリームを用いる相互通信アプリケーションに対して，通信機能と通信資源の管理を含む相互通信セッションを提供するミドルウェアである．

柔軟な相互通信セッションを実現するためには移動エージェントベースのトランスコーディング機能を利用し，IP マルチキャストセッション上の RTP セッションをトランスコーディングエージェントが動的に接続する仕組みを設計した．そしてプロトタイプシステムを実装し，DV ストリームと MPEG4+PCM ストリームを用いた 6 者間相互通信実験を通し，2 つの RTP セッションからなる 1 つの相互通信セッションが動的に構成可能であることを確認した．提案システムをミドルウェアとして利用することにより，相互通信の形態や利用者の QoS 要求に応じて，様々な用途に利用することが可能であると考えられる．

今後，4.4 節の考察で述べた検討項目以外に，より多くの相互通信拠点を接続するための仕組みを検討する．相互通信拠点数のスケーラビリティを上げるため，複数のストリームを 1 本のストリームへ統合するミキサの機能を導入し，動的に配置する仕組みを考案する．一方，提案システムは IP マルチキャストの利用を前提としているが，ミキサの機能とあわせて，マルチキャストとユニキャストの変換機能の実現も今後の課題とする．

謝辞 本研究は，文部科学省科学研究費（課題番号：15700066）および，総務省戦略的情報通信研究開発推進制度（整理番号：032102002）の支援によるものである．ここに記して謝意を表す．

参考文献

1) Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and Schooler, E.: SIP: Session Initiation Protocol, RFC3261 (2002).
2) Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications, RFC3550 (2003).
3) Ooi, W.T. and Renesse, V.R.: Distributing Media Transformation Over Multiple Media Gateways, *Proc. 9th ACM International Multimedia Conference* (2001).
4) Hama, T., Asatani, K., Nakazato, H. and Tominaga, H.: P2P Live Streaming System with Low Signal Interruption, *Proc. 18th International Conference on Advanced Information Networking and Applications* (*AINA*), pp.605–610 (2004).
5) Guo, L., Chen, S., Ren, S., Chen, X. and Jiang, S.: PROP: A Scalable and Reliable P2P Assisted Proxy Streaming System, *Proc. 24th International Conference on Distributed Computing Systems* (*ICDCS*), pp.778–786 (2004).
6) Gharai, L., Perkins, C., Riley, R. and Mankin, A.: Large Scale Video Conferencing: A Digital Amphitheater, *Proc. 8th International Conference on Distributed Multimedia Systems* (2002).
7) Krishna, A.S., Schmidt, D.C. and Klefstad, R.: Enhancing Real-time CORBA via Real-time Java features, *Proc. 24th International Conference on Distributed Computing Systems* (*ICDCS*), pp.66–73 (2004).
8) Pairot, C., Garcia, P. and Skarmeta, A.F.G.: DERMI: A Decentralized Peer-to-Peer Event-Based Object Middleware, *Proc. 24th International Conference on Distributed Computing Systems* (*ICDCS*), pp.236–243 (2004).
9) Freudenthal, E. and Karamcheti, V.: DisCo: Middleware for Securely Deploying Decomposable Services in Partly Trusted Environments, *Proc. 24th International Conference on Distributed Computing Systems* (*ICDCS*), pp.494–503 (2004).
10) Guedes, L.A., Oliveira, P.C., Faina, L.F. and Cardozo, E.: QoS Agency: An Agent-based Architecture for Supporting Quality of Service in Distributed Multimedia Systems, *Proc. IEEE Conference on Protocols for Multimedia Systems — Multimedia Networking* (*MmNet*), pp.204–212 (1997).
11) Nishiyama, S., Hattori, G., Ono, C. and Horiuchi, H.: Lightweight FIPA Compliant Agent Platform on Java-enable Mobile Phone for Ubiquitous Services, *IPAJ Journal*, Vol.45, No.2, pp.575–585 (2004).
12) Jaroodi, J.A., Mohamed, N., Jiang, H. and Swanson, D.: Middleware Infrastructure for Parallel and Distributed Programming Models in Heterogeous Systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.14, No.11, pp.1100–1111 (2003).
13) Ferreira, P., Veiga, L. and Ribeiro, C.: OBIWAN: Design and Implementation of a Middleware Platform, *IEEE Trans. Parallel and Distributed Systems*, Vol.14, No.11, pp.1086–1099 (2003).
14) Hashimoto, K. and Shibata, Y.: Dynamic

Transcoding Functions by Extended Media Stream, *Proc. 18th International Conference on Advanced Information Networking and Applications*, Vol.1, pp.334–339 (2004).
15) Microsoft DirectX. http://www.microsoft.com/windows/directx/
16) RTPlib. http://www-out.bell-labs.com/project/RTPlib/
17) Foundation for Intelligent Physical Agents (FIPA). http://www.fipa.org/

(平成 16 年 5 月 12 日受付)
(平成 16 年 11 月 1 日採録)

橋本 浩二（正会員）

1996 年東洋大学大学院工学研究科電気工学科専攻博士前期課程修了．同年（株）CSK 総合研究所入社．1998 年岩手県立大学ソフトウェア情報学部助手．2001 年東北大学大学院情報科学研究科博士後期課程修了．博士（情報科学）．2002 年岩手県立大学ソフトウェア情報学部講師．分散マルチメディアシステムの構築とエンド間 QoS 保証の研究に従事．情報処理学会平成 15 年度山下記念研究賞受賞．IEEE，電子情報通信学会各会員．

柴田 義孝（正会員）

1950 年生．1985 年 UCLA コンピュータサイエンス学科修了．Ph.D. in Computer Science．1985〜1988 年まで Bellcore（旧 AT&T ベル研究所）にて専任研究員としてマルチメディア情報ネットワークの研究に従事．1989 年より東洋大学工学部情報工学科助教授．1997 年同大学教授．1998 年より岩手県立大学ソフトウェア情報学部教授．高速パケットビデオ，マルチメディアプロトコル，ハイパーメディアシステム，感性情報処理等の研究に従事．IEEE，ACM，電子情報通信学会各会員．