柔軟な相互通信環境のためのトランスコーディング機能の設計と実装

橋本浩二 柴田義孝
岩手県立大学ソフトウェア情報学部

利用可能な帯域幅や処理能力の異なるコンピュータネットワーク上に相互通信環境を構築する際，メディアデータを適切なフォーマットに変換するトランスコーディング機能を導入することよって，利用者環境に応じた柔軟な相互通信環境の構築が可能となる．筆者らは，移動エージェント技術を基盤としたトランスコーディング機能を，通信パスの適切な中間ノードへ配置し，相互通信環境を動的に構成するミドルウェアシステムの研究開発を進めており，本稿では，柔軟な相互通信環境を実現するためのトランスコーディング機能の設計と実装について述べる．

Design and Implementation of Transcoding Functions for Flexible Inter-Communication Environments

Koji Hashimoto Yoshitaka Shibata
Faculty of Software and Information Science, Iwate Prefectural University

Using distributed multimedia system that can integrate various realtime and non-realtime media data, when the system users communicate with each other by realtime audio video data, the system must guarantee end-to-end QoS (quality of service) according to requirements of the system users and available resources. If the system can dynamically use translator or mixer functions defined by RTP, more flexible peer-to-peer communication is realized. In this paper, we design and implement mobile-agent based transcoding functions.

1．はじめに

コンピュータの処理速度向上と音声や動画像圧縮技術の進歩により，安価なパーソナルコンピュータでも複数のリアルタイムメディアを処理することが可能となった．一方，ADSL や FTTH の普及により一般家庭においても数 Mbps～100Mbps 程度の帯域幅を利用することが可能となり，リアルタイムメディアを利用した相互通信が現実的なものとなりつつある．現在，IP ネットワークを利用して DV(Digital Video)ストリームを転送する技術[1,2]や，プロダクション品質映像(D1)，ハイビジョン映像(HDTV)を配信する技術も研究開発[3,4]されており，利用可能な帯域幅とコンピュータ資源によっては非常に質の高い映像による通信も実現可能となった．

しかしながら，例えば JGN(Japan Gigabit Network)のような超高速ネットワークを利用して約 30Mbps の DV ストリームを用いた相互通信環境を構築しても，自宅や出張先からその相互通信に参加できる仕組みを実現することは困難な状況である．なぜならば，相互通信参加者全てのコンピュータネットワーク環境をあらかじめ把握しておくことは現実的ではなく，例え相互通信に参加できたとしても，コンピュータ資源や帯域幅が十分確保できなければ，リアルタイム通信に支障をきたす可能性があるからである．

構築すべき相互通信環境が静的であり，通信参加者を限定できるなら，例えば DV ストリームを MPEG や M-JPEG，H.263 などのストリームへリアルタイムに変換するトランスコーディング機能を適切な中間ノードにあらかじめ配置しておくことも可能である．しかし，通信参加者数や通信パスが動的に変化することを考慮すると，トランスコーディング機能を動的に配置するための仕組みが必要であると考えられる．このような背景のもと，利用者の QoS(Quality of Service)要求とコンピュータおよびネットワーク資源の利用状況に応じて，適切な中間ノード上でトランスコーディング機能を稼動させるミドルウェアを構築することが本研究の目的である．移動エージェント技術を基盤としたトランスコーディングエージェントを適切な中間ノードへ動的に配置することにより，相互通信参加者の用いる利用者端末の処理速度やネットワーク環境に応じた柔軟な相互通信が可能となる．これまで筆者らは，柔軟な相互通信環境構築のために必要とされる機能をミドルウェアシステムとして実現するためのアーキテクチャを考案し，トランスコーディングエージェントの動的な配置に関して報告[5,6]を行ってきた．本稿では，現在開発を進めているプロトタイプシステムにおけるトランスコーディング機能の設計と実装について述べる．

2．MidField システム概要

筆者らが研究開発を進めている MidField (Middleware for Flexible intercommunication environment by relocatable decision) システムの機能モジュール構成を図１に示す．本システムは，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，アプリケーションプログラムに対してマルチメディア通信機能を提供する．Stream Plane はマルチメディアストリーム転送処理を行い，Session Plane は通信セッションの管理を行う．また，System Plane ではネットワークトラフィックや CPU 利用状況の監視を行い，システム利用者が要求する QoS に対するアドミッションテストを実行する．そして 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．

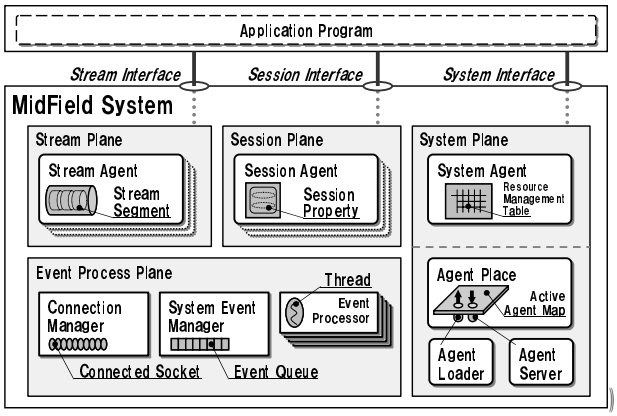

図１：MidField システム機能モジュール構成

通信セッション参加者やメディアストリームの状態管理を行う Session Plane では，利用される相互通信セッションの形態に応じて様々な機能が要求される．これに対し，管理情報と管理機能を移動エージェントとして相互通信セッション参加者のノードへ転送できれば，機能モジュール変更への対応が容易になると考えられる．また，Stream Plane のメディア処理機能は，ローカルのコンピュータシステムに必要な機能モジュールが存在しない場合，他のコンピュータから取得できることが望ましい．これらを考慮し，MidField システムの Session Plane と Stream Plane を，移動エージェントにより実現する．

また，MidField システムの各プレーンの機能は，Straem 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．これらのインターフェースに対応するシステムの処理は，Stream Agent，Session Agent，System Agent が対応する．Stream Agent は，RTP[7]ストリーム処理を実現するために Stream Segment を保持し，RTP ストリームの送受信およびトランスコーディングを行う．Session Agent は，MidField セッション情報である Session Property を保持し，必要に応じて MidField システム間を移動する．また，System Agent はローカルコンピュータシステムの CPU 資源やネットワークトラフィックを監視し，システム利用者が MidField セッションへ参加する際にはアドミッションテストを実行する．一方，AgentPlace はエージェントの生成・起動，移動，終了処理を行い，ローカルシステム内のエージェントを管理する．そして，エージェントの移動を実現するための Agent Loader と Agent Server を利用する．Agent Place も MidField システムにおけるエージェントの１つである．

これらのエージェントはそれぞれ，コンピュータネットワーク上で稼動する MidField システム間でメッセージの送受信が可能である．Event Process Plane の Connection Manager は，MidField システム間を接続したソケットインターフェースを保持し，エージェント間メッセージ通信の排他的制御を行う．また，System Event Manager は，各エージェントが発行するシステム内部イベントを Event Queue に格納し，Event Processor の Thread でイベントを処理する仕組みを実現する．これにより，システム内部のイベント処理を一元管理するとともに，優先順位に基づくイベント処理が可能となる．

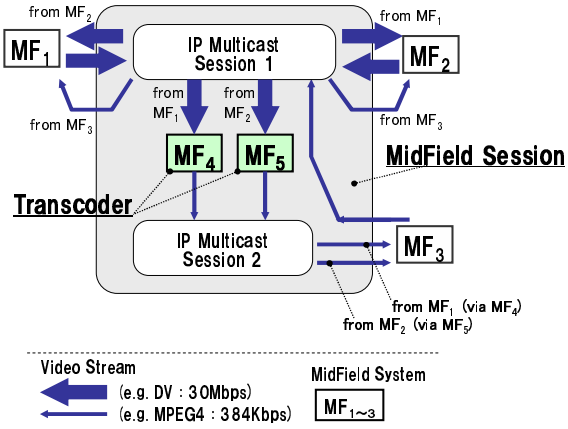

図２：MidField Session 概要

MidField システムは，利用者の通信環境とサービス品質(Quality of Service)要求に応じて複数の IP マルチキャストセッション上に RTP セッションを生成する．そして，生成された RTP セッション間にト

ランスコーディング機能を配置することにより，相互通信環境の動的構成を実現する．ここで，MidField システムにおける相互通信セッションを MidField セッションと呼び，図２はその概要を示している．図２では，システム利用者が３人(MF1,MF2,MF3)，１つの MidField セッションに参加している．ここで，MF1 と MF2 は共に，DV ストリームを送受信可能な通信環境を利用できる．しかし MF3 は，帯域不足または DV 処理に対するコンピュータ資源不足のため，DV ストリームを用いた通信が不可能であり，MPEG4 ストリームによる送受信を要求していると仮定する．このような場合に，MF1,MF2,MF3 が相互通信を行うコンピュータネットワーク上の適切な場所にトランスコーディング機能を配置する．図２では，MF4,MF5 が DV ストリームを MPEG4 へトランスコードすることにより，MF3 も相互通信が可能となる．

これまでに筆者らは，トランスコーディングエージェントの動的な配置に関するアルゴリズムの概要を[5,6]にて報告しており，現在筆者らは StreamAgent の開発を進めている．以下，Stream Agent が保持する Stream Segment の概要を述べ，プロトタイプの実装とサンプルコードを概説する．

３．Stream Segment 概要

図３に示す StreamSegment は，Media Processor を保持し，処理に必要な Plug-In モジュールをつなぎ合わせて実際のメディア処理を実現する．Media Processor は，キャプチャデバイス，ファイル，RTP ストリームからのメディアストリームを入力とし，入力データはフレームまたはパケットの単位で処理される．一方，処理されたメディアデータもフレームまたはパケットの単位で出力される．

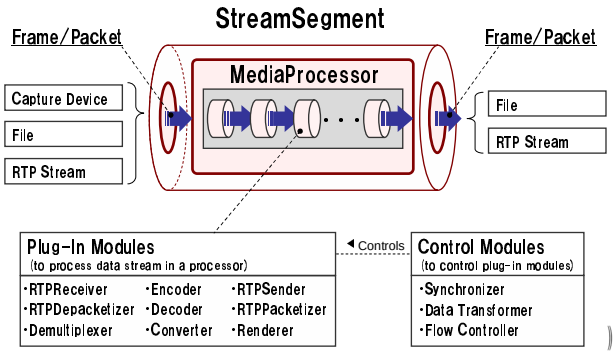

図３：Stream Segment の構成

メディア処理を行う Plug-In モジュール中，RTPSender と RTPReceiver は RTP パケットの送受信を行い，メディアデータフレームとパケットの変換には，RTPPacketizer と RTPDepacketizer が利用される．また，各種の Encoder/Decoder,Demultiplexer,Renderer などを組み合わせて，メディアストリームの処理が可能となる．加えて，これらの Plug-In モジュールはメディア処理を制御するための制御モジュール(Control Modules)を保持しており，Stream Agent がメディア処理を制御することを可能としている．

３．1) RTP パケットの受信

上述した通り，RTP パケットの受信は RTPReceiver により実現される．図４は，RTPReceiver による RTP パケット受信とメディアデータフレームへの変換，Encoder/Decoder への引渡しに関するデータフローと主な処理の流れを示している．

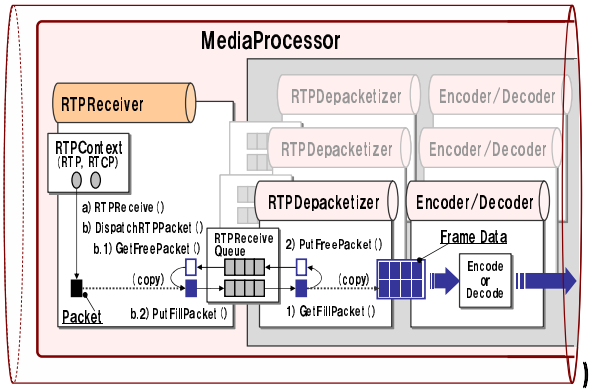

図４：RTPReceiver による RTP パケットの受信

RTPReceiver は，RTP/RTCP パケットの送受信を行うために，２つのソケットインターフェースを利用する．また，受信した RTP パケットを RTPDepacketizer へ引き渡すために，RTPReceiveQueue を保持する．RTPReceiveQueue には，受信データで満たされたパケット用のキューと空パケット用のキューがあり，あらかじめ確保されたパケットデータ用メモリ領域へのポインタが，２つのキューへの入出力を繰り返す．

まず，図４中 a) RTPReceive() メソッドにより，RTPReceiver が RTP パケットを受信すると，次に b) DispatchRTPPacket() メソッドによって RTP パケットを対応する RTPReceiveQueue に振り分ける．本システムでは，RTP パケットヘッダ内の SSRC フィールド値に対応した CNAME 毎に RTPReceiveQueue を管理している．これにより，RTP パケットの SSRC フィールド値に対応した RTPReceiveQueue への振り分けが

可能となる．対応する RTPReceiveQueue が存在する場合，b.1) GetFreePakcet() メソッドにより，空のパケットデータ領域を取得し，受信したパケットデータをコピーした後，b.2) PutFillPacket() メソッドを使って，受信パケットデータのコピーを RTPReceiveQueue へ追加する．一方，RTPDepacketizer は，1) GetFillPacket() メソッドにより受信パケットデータのコピーを取得し，Frame Data 用のメモリ領域へコピーし，2) PutFreePakcet() メソッドにより RTPReceiveQueue へ戻す．RTPDepacketizer では，１フレーム分のデータコピーが完了した時点で，Frame Data を Encoder/Decoder へ引き渡す．

３．2) RTP パケットの送信

RTP パケットの送信は RTPSender により実現される．図５は，そのデータフローと主な処理の流れを示している．

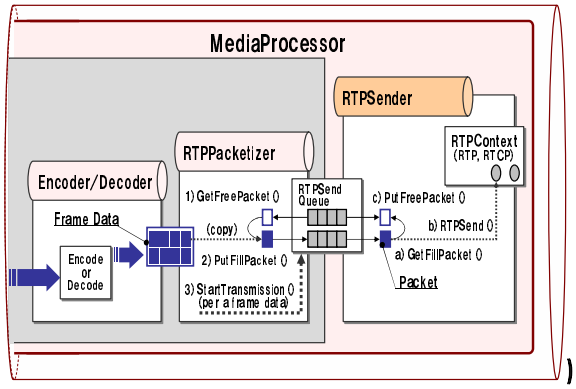

図５：RTPSender による RTP パケットの送信

図５中，RTPPacketizer は Encoder/Decoder より Frame Data を取得し，パケット化したデータを RTPSendQueue に追加する．１フレーム分のデータをパケット化したら，StartTransmission() メソッドにより，送信開始の合図を RTPSender へ送る．RTPSender では，１フレーム送信開始の合図を受けると，a) GetFillPacket(), b) RTPSend(), c) PutFreePacket() を繰り返し，１フレーム分の RTP パケットを送信する．

本システムでは，RTP で定義されている SSRC とそれに対応する CNAME 毎に RTPSendQueue と RTPReceiveQueue を管理することにより，単一 RTP セッション内の複数の送信元を識別している．また，処理すべきメディアデータ毎に RTPPacketizer/RTPDepacketizer を実装し，適切な Encoder/Decoder モジュールに接続することにより，トランスコーディング機能を実現する．

４．プロトタイプシステムの設計と実装

MidField システムのプロトタイプシステムを実装するにあたり，筆者らは開発言語として Java (Ver.1.4)，C, C++ の各言語を利用している．これまで，RTP 送受信を含むメディア処理モジュールの実装には JMF (Java Media Framework) [8] を利用してきたが，DV 送受信を考慮すると JMF による実装では十分なパフォーマンスが得られず，現在は Windows XP の環境で DirectShow (DirextX 9.0) [9] を利用している．そして，上述した Encoder/Decoder などの機能の実現には DirectShow の既存のフィルタを用いている．また，各メディアタイプ毎に RTPPacketizer と RTPDepacketizer を DirectShow のフィルタとして実装し，RTPSender と RTPReceiver は Lucent Technologies の RTPlib 1.02b [10] を利用して実装している．以下，UML を用いて MidField システムの API 関連クラス概要を述べ，続いて内部クラス構成と Stream Agent のクラス構成について述べる．

４．1) MidField システム API 関連クラス概要

MidField システムはアプリケーションプログラム対して柔軟な相互通信機能を提供する．図６はその API 関連クラスの概要を示している．

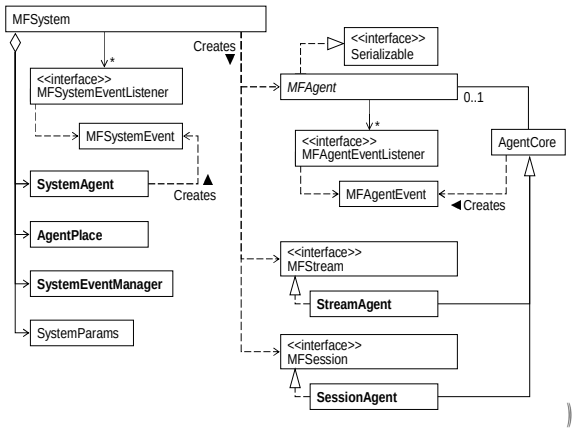

図６：MidField システム API 関連クラス構成

アプリケーションプログラムは，図６における MFSystem クラスを用いて，本システムの起動，終了，各種の操作を行うことが可能となる．MFSystem クラスは，図２における System Interface を実現する．また，MFSystem クラスは，図２における Stream Plane の API となる MFStream インターフェースや

Session Plane の API となる MFSession インターフェースをアプリケーションプログラムに提供する．MFStream および MFSession インターフェースは，それぞれ StreamAgent と SessionAgent により実現される．

一方，MFAgent クラスを拡張したクラスを定義することにより，アプリケーションレベルでの移動エージェントを作成することも可能である．

これらのエージェントは，エージェント間通信と移動の機能を保持する AgentCore を拡張することによって実現される．

4．2) MidField システム内部クラス構成

図7は，MidField システムの内部クラス構成を示している．AgentCore を拡張して，SystemAgent，AgentLoder，AgentServer，AgentPlace の各エージェントを実現する．本システムの中核となるのは AgentPlace クラスとなる．

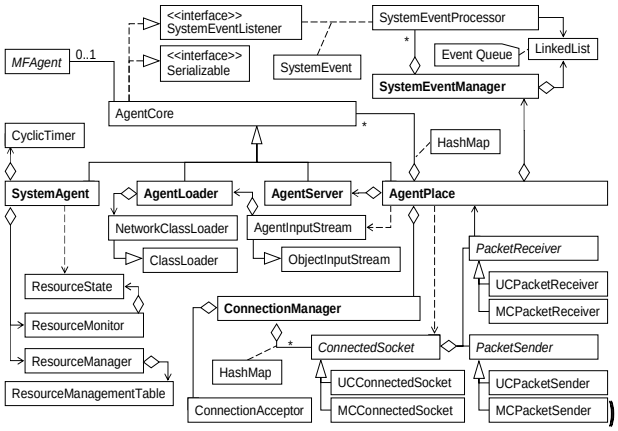

図7：内部クラス構成

AgentPlace は HashMap を用いてローカルシステムの AgentCore インスタンスを管理し，SystemEventManager を利用してシステム内部イベント処理を複数の SystemEventProcessor へ委譲する．また，AgentPlace は ConnectionManager を利用してエージェント間通信を実現する．ConnectionManager は，コネクション接続要求を受け入れる ConnectionAcceptor と，複数の ConnectedSocket を管理し，また，ユニキャストとマルチキャストに対応したエージェント間メッセージ通信を可能としている．

4．3) Stream Agent クラス構成

Stream Segment を含む Stream Agent のクラス構成を図8に示す．既に述べた通り，本プロトタイプシステムでは，メディア処理に DirectShow を利用している．従って，エージェントシステム部分は Java 言語を用いた実装を行っているが，メディア処理と RTP 送受信部分は C++/C 言語を利用して，その間を JNI(Java Native Interface)により接続している．StreamSegment は Java 側と C++側の実装があり，C++側の StreamSegment によってメディア処理と RTP 送受信の機能が実現される．また，StreamSegment は，MediaProcessor として MediaSinker，MediaPlayer，MediaSender のいずれかのインスタンスを保持する．これらのインスタンスは SegmentManager によって生成され，IMediaEventEx や IMediaControl などの DirectShow が提供する COM インターフェースを通して，各種のイベント取得やメディア処理の制御が可能となる．

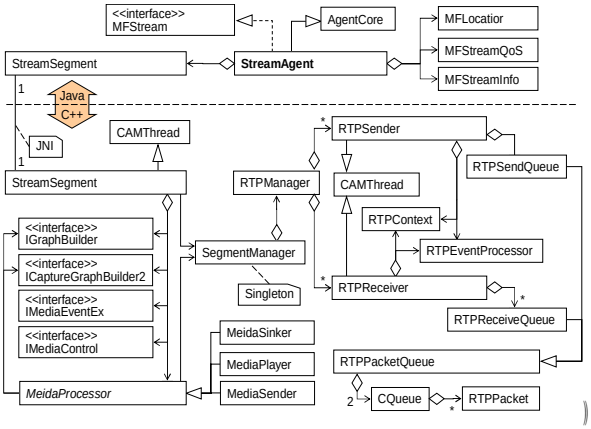

図8：StreamAgent のクラス構成

また，SegmentManager は RTPManager を保持しており，要求された RTP セッションへの RTP/RTCP パケット送受信を行うために RTPSender と RTPReceiver を RTPManager が生成し，管理する．そして，RTPSendQueue と RTPReceiveQueue を DirectShow が提供する各種のフィルタへ接続することにより，図4，図5に示した RTP パケット送受信とトランスコーディングが可能となる．

4．4) サンプルコード

現在，本稿でこれまで述べたクラス構成によるプロトタイプシステムを実装し，簡単なテストが行える段階となった．RTP ストリームとして，まずは DV，M-JPEG，PCM を想定し，各々の RTPPacketizer と RTPDepacketizer を実装した．図9は，本システムを利用して RTP ストリームの送信とトランスコーデ

ィングを行うための簡単なサンプルコードである．

図９では，まず MidField システムを起動し，RTP ストリームを送信するためのストリームエージェントを生成している(02～06 行目)．次に，ストリームエージェントの入出力を設定し(07,08 行目)，送信を開始している(10 行目)．この例では，入力として DV カメラを設定し，マルチキャスト IP アドレスとポート番号, TTL の値と共に，DV/RTP での出力を設定している．また，14 行目～16 行目ではトランスコーディング用のストリームエージェントを生成し，その入力として先の DV/RTP を設定し，入力と異なる IP マルチキャストセッションに対して M-JPEG/RTP と PCM/RTP の出力を設定している．さらに 17 行目では，設定した入出力に基づくトランスコーディング機能を実行するホストへ移動するための呼び出しを行っている．

```
01: // MidField System の起動
02: MFSystem mfs = MFSystem.invokeSystem();
03:
04: // 送信ストリームの生成・送信開始
05: // 入力：ビデオキャプチャデバイス(DVカメラ)，出力：RTPストリーム(DV/RTP)
06: MFStream sender = mfs.newStream();
07: sender.setInput(
        "capture://DVCamera&Default");
08: sender.setOutput(
        "stream://StreamAgent00/224.100.100.100/20000/4&DV_RTP");
09: sender.open();
10: sender.start();
11:
12: // トランスコーディングストリームの生成・移動，移動先でトランスコーディング開始
13: // 入力：RTPストリーム(DV/RTP)，出力：RTPストリーム(M-JPEG/RTP, PCM/RTP)
14: MFStream xcoder = mfs.newStream();
15: xcoder.setInput(
        "stream://StreamAgent00/224.100.100.100/20000/4&DV_RTP");
16: xcoder.setOutput(
        "stream://StreamAgent01/224.100.100.200/20000/4&MJPEG180X120_RTP",
        "stream://StreamAgent02/224.100.100.200/20000/4&PCM_RTP");
17: xcoder.migrate("destination_hostname");
```

図９：サンプルコード

５．まとめ

メディアデータを適切なフォーマットに変換するトランスコーディング機能を導入することよって，利用可能な帯域幅や処理能力の異なるコンピュータネットワーク上に相互通信環境を構築する際，利用者環境に応じた柔軟な相互通信環境の構築が可能となる．現在筆者らは，移動エージェント技術を基盤としたトランスコーディング機能を，通信パスの適切な中間ノードへ配置し，相互通信環境を動的に構成するミドルウェアシステムの研究開発を進めており，本稿では，現在開発を進めているプロトタイプシステムにおけるトランスコーディング機能の設計と実装について述べた．

現在，DV/RTP，M-JPEG/RTP および PCM/RTP に対応したトランスコーディング機能を実装し，簡単なテストが行えるようになった．しかしながら，相互通信セッションの動的構成方法に関する詳細はまだ検討不足であり，今後，セッション情報告知プロトコルやセッション識別子交換モジュールの詳細設計を行い，CPU 占有率と利用可能な帯域幅によるトランスコーディングノード決定アルゴリズムを考案し，これらを含む移動エージェントの再配置プロトコルを実装する予定である．また，評価用アプリケーションの開発を行い，簡単なテストベッドを構築し，動的なトランスコーディングノードの決定と，リアルタイムトランスコーディング機能の評価を行う予定である．

参考文献

[1] 杉浦，小川，中村，村井，"民生用 DV を用いたインターネットビデオ会議システム"，情報処理，Vol.40，No.7，pp.698-702，Jul.，1999．

[2] Akimichi Ogawa. DVTS(Digital Video Transport System). http://www.sfc.wide.ad.jp/DVTS/.

[3] 勝本道哲，原田雅博，中川晋一，"D1 over IP による高品位動画像転送・蓄積システムの設計"，情報処理学会マルチメディア通信と分散処理研報，No. 95，pp.85-90，1999．

[4] 勝本道哲，木俵豊，櫻田武嗣，北口善明，杉浦一徳，中川晋一，"超高画質画像配信システムにおけるコンテンツ管理の提案"，情報処理学会シンポジウムシリーズ Vol.2001，No.13，pp.91-96，2001．

[5] 橋本浩二，柴田義孝，"トランスコーディング機能による柔軟な相互通信環境の実現"，マルチメディア通信と分散処理ワークショップ，IPSJ Symposium Series Vol.2002，No.15，pp.189-194，Oct.，2002．

[6] Koji Hashimoto and Yoshitaka Shibata, "Design of a Middleware System for Flexible Intercommunication Environment", Proc. of the 17th International Conference on Advanced Information Networking and Applications (AINA2003), pp.59-64, Mar., 2003.

[7] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson., "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[8] http://java.sun.com/products/java-media/jmf/

[9] http://www.microsoft.com/windows/directx/

[10] http://www-out.bell-labs.com/project/RTPlib/