修 士 論 文

# ビデオネットワークにおける
# 動的な資源発見及び接続機構

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

鈴木 賢治

2002 年 3 月

修 士 論 文

# ビデオネットワークにおける<br>動的な資源発見及び接続機構

指導教官　丹 康雄 助教授

審査委員主査　丹 康雄 助教授

審査委員　篠田 陽一 教授

審査委員　日比野 靖 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

010059　鈴木 賢治

2002 年 2 月 15 日

## 要 旨

ネットワーク内におけるストリーム機器間接続の管理性と信頼性を向上させるための高次の接続概念を提案する。これにより、複数のストリーム機器をつなぎ合わせることで実現可能な遠隔授業、遠隔会議といったアプリケーションを、エンドポイントのユーザ環境に負担を強いることなく、容易に利用できることになる。また、こういったアプリケーションが大規模となり大量の機器を接続する必要に迫られた場合にも、信頼性を保つような機能を設けることが可能となる。

近年、QoS を保証するような品質の高いネットワークや家電を直接接続することができるようなネットワークが登場している。これにより、家電を用いた高品質のビデオ伝送を遠隔地間で行うこと可能となってきているが、複数の機器の複雑な接続をサポートする機能をもったシステムは、従来考えられてこなかった。従って、複数の機器を連携させて利用するためには、結局人間が一つ一つの接続について細かく管理、操作する必要があった。そこで、関係を持つ複数の接続をひとまとめにした接続の単位を新たに導入することにより、複雑な接続管理を容易に行える上に信頼性をも高めることを可能とする手法を提案する。

# 目 次

# 図 目 次

# 表 目 次

# 第 1 章

# はじめに

近年、計算機の性能向上、情報家電機器の登場、ネットワークの性能向上を背景に、計算機及び情報家電を端末としてネットワークを介したビデオデータのやりとりがいっそう盛んに行われるようになってきている。DV, MPEG2, D1 等が代表する高品質な映像、音声のネットワークを介した送受信が可能となりつつあり、遠隔会議、遠隔授業といった遠隔地同士で映像をやりとりするアプリケーションの利用価値、利用効果の顕著な向上が期待されている。また、遠隔医療のような厳しい品質水準を絶対条件とするような用途にも利用することができるようになると考えられる。

しかしながら、現存するシステムはそれぞれの用途に特化している傾向にあり、各々のネットワークシステムを相互運用することは非常に難しい。利用するネットワークの差異を意識せずにビデオ伝送を行うことができれば、新たな用途に利用することが容易になるだけでなく、システムの拡張性、信頼性、利用しやすさの劇的な向上を促すことが可能であると考えられる。

また、従来はネットワーク上にある資源を有効に接続して利用することをあまり考慮されてこなかった。トランスコーダ、マルチプレクサ等の高価な資源をネットワーク上に配備し、中間資源として簡単に接続することができるようになれば、エンドポイントの負担を低く抑えることができる。さらに、中間資源の故障や接続の切り替え等の資源状態の動的推移に対して柔軟に対応可能な機構を設けることが可能となる。特に本研究では、資源状態の動的推移に対処する機能について大きく取り上げることとする。

このように、ネットワーク間相互運用性とネットワーク内資源の有効利用性が向上すれば、従来のアプリケーションは勿論のこと、新たなものに関してもビデオ伝送の利用価値が著しく向上することは間違いない。

さらに、この上で本研究ではこれまでにない接続の管理手法を提案する。従来のビデオ

伝送の接続管理では、ネットワーク上に複数の接続が存在すれば、一つ一つの接続を細かく人間が管理することとなってしまうことになる。これは、接続の管理する単位は、一つ一つの接続であり、他に管理単位がないことに起因している。これでは、大量のストリーム機器の利用が不可欠な大規模なアプリケーションを運営すること困難を極める。そこで、本研究ではセッションという接続単位を提案し、上記の問題点を解決する。

上記の点をふまえ、「計算機」「家電」の機能を、接続されているネットワークの差異を考慮することなく接続し、資源を有効に活用するための管理システムを提案、設計、実装を行った。本稿は以下の構成となっている。

- 第２章

  本研究の背景となるビデオネットワーク技術の動向及び、ターゲットとするビデオネットワークシステムについて述べる。

- 第３章

  本研究のベースとする VIA に関して概要を説明する。

- 第４章

  接続形態について考察する。

- 第５章

  第４章で考察した接続形態を実際のシステムに適応するために必要な概念であるセッションについて提案する。

- 第６章

  資源予約を行うことで信頼性の高いシステムを構築するための考察。

- 第７章

  セッションの VIA への組み込みについて述べる。

- 第８章

  SDL による設計及び、実装に関して説明する。

- 第９章

  本研究により新しく提案された枠組みにおいて、どのような性能を評価すべきなのかを考察する。

- 第１０章

  今後の課題について。複数のリソースマネージャに関する考察。

- 第１１章

  本研究をまとめる。

# 第 2 章

# ビデオネットワーク

ビデオネットワークシステムとは、ビデオを用いるアプリケーションをネットワークを介して行うことができるシステムのことである。従来、様々な伝送形態を用いてビデオをやりとりするコミュニケーションが模索されてきた。ビデオデータを伝送するためには、フォーマットや用途によってネットワークや端末の性能に要求を課すことになる。従って、ビデオネットワークシステムの実現のため、利用可能なネットワークや端末の性能に合わせて、可能なビデオフォーマットやコミュニケーションの形式を選択してきた。

つまり、既存のビデオネットワークシステムは、ネットワークや端末の性能に縛られた上で構築されてきたと考えられる。

ビデオをやりとりするアプリケーションは大きく分けて、以下で述べる三つの形式が考えられる。ここでは、これらについて既存のシステムと絡めながら概観することとする。

## 2.1 リアルタイムコミュニケーション

遠隔会議、遠隔講義といったリアルタイムでのコミュニケーションが重要となるアプリケーションである。このようなリアルタイムコミュニケーションは、特定の地点に人や物が集まることで可能であったものを遠隔地間でも可能とするものであり、今後システムが確立するにつれ様々な利用方法が考えられる。

以下にいくつかのシステム事例をあげる。

### 2.1.1 H.320 系テレコンファレンスシステム

H.320 系テレコンファレンスシステムは、ISDN や IP 網を用いてテレビ会議をするシステムであり、近年かなり普及したシステムである。このシステムでは、映像、音声、ホ

ワイトボードやアプリケーションの共有、ファイル転送等を総合的に扱うことができる。また、MCU(Multipoint Control Unit) を用いることにより、複数の端末同士の接続も可能である。

### 2.1.2 DVTS

DVTS[8] は、DV(Digital Video) データを RTP を用いた IP ネットワークの上で送受信するシステムである。このシステムは、ネットワークに 30Mbps 程度の帯域を要求するため、現在のインターネットで容易に扱うのは難しい。しかし、このシステムは、民生用のカムコーダを利用して、遠隔地間で高い品質の映像、音声のやりとりを可能とするために、遠隔コミュニケーションの利用価値、効果は高い。

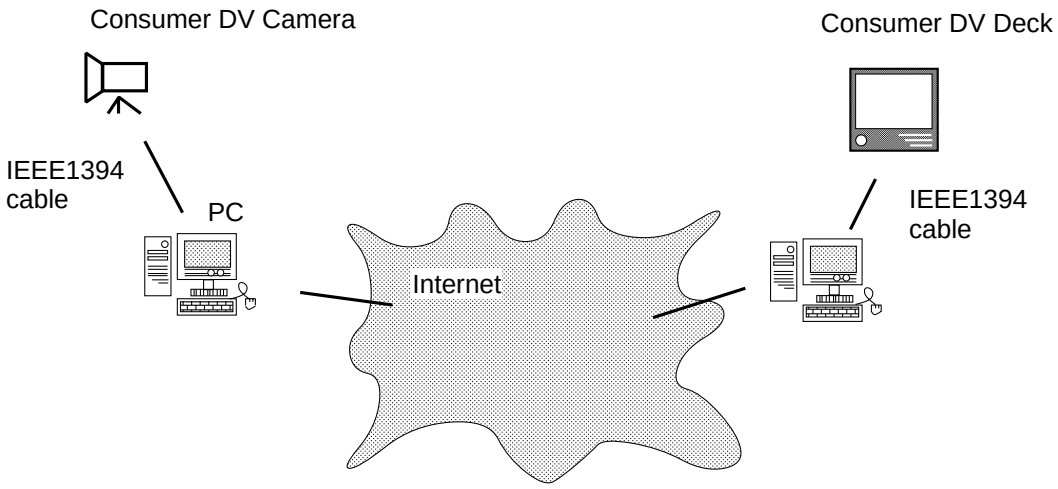



図 2.1: DVTS

近年では、Intserv(Integrated Servece Architecture) や DiffServ といった IP ネットワークの品質を保証しようとするシステムが活発に研究されている。こういった、新しいネットワークの品質保証技術を用いるか、ATM のような QoS を保証できる既存のネットワークを利用することにより、広範ネットワーク上でも、高い品質の映像をやりとりすることができる。

### 2.1.3 VuNet

VuNet[12] は、ATM ネットワークをベースとした DAN(Desk Area Network) の実装である。DAN は、図 2.2ワークステーションやマルチメディア機器が接続された、小規模のローカルネットワークのことをさす。LAN,MAN,WAN と異なり、狭いエリアでのネットワークであるため、ネットワークに参加するホストの数は限られている。こういった、小さなネットワークを考えることにより、ネットワークシステムをシンプルに設計することができるという利点がある。DAN では、ワークステーションがネットワークを介してビデオ機器にアクセスし、接続されている機器同士のデータのやりとりを仲介する。

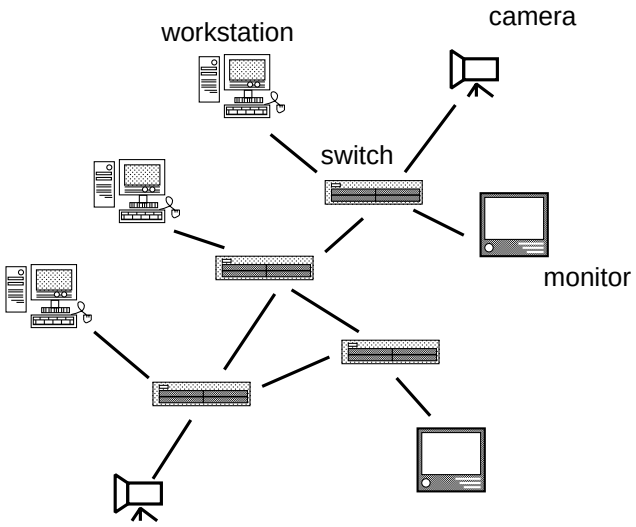



図 2.2: Desk Area Network

## 2.2 VoD

VoD(Video on Demand) は、コンテンツをサーバに蓄積しておきクライアントからの要求に応じて映像を制御しながら配信することができるシステムである。近年の、端末の性能向上や、クライアントアプリケーションのフリーでの配布が相まって、今後急速に広がりを見せると考えられている。ただし、現在は普及の過渡期にあり非常に多くのビデオフォーマットが出回っているために、エンドポイントの環境を何らかの形で整える必要がある。サーバ側では、複数のフォーマットに対して複数の解像度で用意するといった対策を講じる必要がある。もしくは、クライアント側で複数のクライアントアプリケーション

を適宜使い分けるという必要がある。こういった対策では、エンドポイントに大きな負担がかかってしまうのは明らかである。近年では、エンドユーザポイントの端末は必ずしも計算機ではなく、携帯電話や PDA といった、ビデオデータを処理するには性能が足りない機器である場合もあるため、エンドポイントに負担をかけることは映像配信サービスを行う上で致命的な問題となることが考えられる。そこで、最近では、クライアントに応じて元のコンテンツをネットワーク内でトランスコードしてから配信するというサービスを行う方法も見られるようになってきている。[15]

このように、ビデオフォーマットに関しては何らかの方策を講じる必要があることがわかる。

## 2.3 放送

近年、インターネットを介して放送を行うという動きがある。収録から編集、配信にいたるプロセスの中でネットワークを効果的に利用することにより大幅な経費削減や、コンテンツ製作時間の節約等の効果が見込まれる。

## 2.4 まとめ

本章では、遠隔コミュニケーションの形態を元にビデオネットワークを分類しながら考察した。既存のシステムでは以下の問題点をあげることができる。

- システムごとに特化して相互に利用することが難しい
- ビデオフォーマットの際を何らかの方法で吸収する必要がある
- エンドポイントのユーザがコンテンツ供給にはまだ壁がある
- エンドポイントの端末の多様化

本研究では、こういった問題点を解決するためにはエンドポイントの性能をあげるのではなく、ネットワークが問題点を吸収するようなシステムを構築することが有効であると考える。なぜなら、将来エンドポイントの端末がいっそう多様化することが考えられるため、問題の解決を端末側に押し付けるのは難しいと考えるからである。以降の章では、ネットワーク内で上記の問題を解決しようとする VIA について述べ、これを基盤とした新しいネットワークシステムについて提案を行うこととする。

# 第 3 章

# VIA

ビデオネットワーク統合アーキテクチャ Video-network Integration Architecture(VIA)[1][2] は、各種ビデオネットワーク間の相互接続を可能とすることを目的としたシステムである。

VIA では、ネットワークに接続されるストリーム機器を、データの流れる方向を持った入出力口を一つの単位とする資源として抽象化し、この抽象化された資源をネットワーク内で管理する。また、フォーマット変換及び、制御プロトコル変換をシステム内で吸収する。これにより、末端のネットワークが異なっても相互に接続可能となる。

図 3.1は VIA の概念図である。VIA は、本研究の目的に必要な異種ネットワーク間の相互接続及び資源の管理という環境を提供するシステムであるといえる。そこで、本研究では VIA を元にしたシステムを構築することとする。

本章では VIA に関する概要を述べる。

## 3.1 ビデオネットワークの形態

VIA のネットワークは、フロントエンドネットワーク、コアネットワークの二つとそれらを繋ぐゲートウェイから構成される。フロントエンドネットワークは IEEE1394 のような実際にビデオ機器が接続されるネットワークとする。コアネットワークは、ATM ネットワークや RSVP を用いた IP ネットワークといった帯域保証を行えるネットワークとし、異なるフロントエンドネットワーク同士を接続するハブとして見ることができる。

このような構成をとるために、ゲートウェイが個々のフロントエンドネットワークの特性及び、接続されている機器の特性を隠蔽、抽象化する動作をとる。その結果として、コアネットワーク上ではアタッチされているビデオ機器を一元化して扱うことが可能となる。

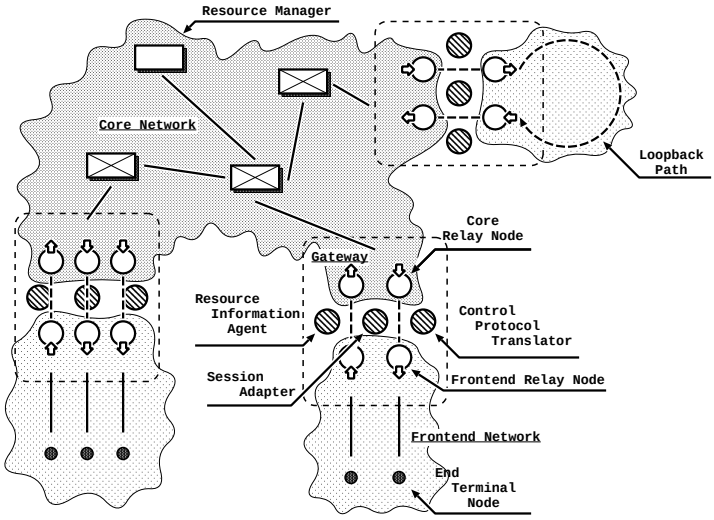



図 3.1: VIA

以降で、システムを構成するそれぞれの要素を順に述べる。

### 3.1.1 フロントエンドネットワーク

フロントエンドネットワークは、ビデオ機器が実際に接続される末端のネットワークである。

フロントエンドネットワークの条件として、ビデオ伝送が可能な通信品質を提供可能であることがあげられる。

### 3.1.2 コアネットワーク

コアネットワークは、複数のフロントエンドネットワークのハブとなるネットワークである。コアネットワークの条件としては、広域ネットワークでのビデオ伝送が可能な通信品質の保証があること及び、コネクション切り替えによるオーバヘッドが小さいことがあげられる。

コアネットワークでは、後述するゲートウェイによってアタッチされた資源を一元的に管理し、接続の確立及び解放管理を行う。コアネットワーク上にはリソースマネージャ (3.2.2) が配置され機能する。

### 3.1.3 ゲートウェイ

ゲートウェイはフロントエンドネットワークとコアネットワークを仲介する。ゲートウェイ内では以下のコンポーネントが動作する。

- リソースインフォメーションエージェント (RIA)

  ゲートウェイの中で核となるコンポーネントである。該当フロントエンドネットワークのコアネットワークへのアタッチ、デタッチを行う。また、フロントエンドネットワークに接続され抽象化されたエンドターミナルノードをコアネットワークで管理可能とするために管理情報を付加してフロントエンドリレーノードに変換し、管理する。詳細は後述する。(3.2.1)

- コントロールプロトコルトランスレイタ (CPT)

  VIA では、フロントエンドネットワークのコントロールプロトコルの差異を吸収するために、コアネットワークでは統一したプロトコルを扱うこととしている。この統一したプロトコルを VNCP(Video Network Control Protocol) と呼ぶ。コントロールプロトコルトランスレイタは、フロントエンドネットワークごとに異なるコントロールプロトコルを VNCP に変換してコアネットワーク上で利用できるようにする機能を提供する。これにより、各々のフロントエンドネットワークで異なるコントロールプロトコルを、相互に運用することが可能となる。詳しくは [1][2] を参照のこと。

- セッションアダプタ (SA)

  実際の機器がビデオネットワークに接続する点である。接続された機器は、エンドターミナルノードの情報として抽象化され、この情報をリソースインフォメーションエージェントへ渡す。機器の挿抜があった時にリソースインフォメーションエージェントに対して通知を行い、資源の状態変化をコアネットワークへ伝播させるきっかけをつくる。シグナリング操作により、接続の確立及び解放を行う。

## 3.2 資源管理

本システムでは接続されている機器の情報を次の 3 種類の段階の形式で資源として保持する。これにより、ビデオネットワークシステムに接続される全ての機器は、データの方向性を持つ口を一つの単位とした資源として保持されることとなる。

1. エンドターミナルノード

   フロントエンドネットワークにビデオ機器が接続されると、セッションアダプタによりエンターミナルノードに抽象化する。データの方向を持ったひとつの資源として認識される。

2. フロントエンドリレーノード

   リソースインフォメーションエージェントが、エンドターミナルノードをシステムで用いる形式に変換し、識別子を付加したもの。

3. コアリレーノード

   コアネットワークで利用できるように、フロントエンドリレーノードにゲートウェイの識別子を付加したもの。リソースマネージャが保持する。

以降で、リソースインフォメーションエージェント及びリソースマネージャについて説明する。

### 3.2.1 フロントエンドネットワーク資源の管理

機器がフロントエンドネットワークに実際に接続されると、リソースインフォメーションエージェントはセッションアダプタからエンドターミナルノードを受け取り、システムで用いる統一的なフォーマットであるフロントエンドリレーノードに変換する。フロントエンドリレーノードには、以下の情報を管理情報として付加することが必要である。

- 資源 ID：資源を識別する ID

  フロントエンドネットワーク内で一意に識別可能とする。この識別子を元にコアネットワークが資源を管理し、接続の確立や解放等を行う。また、ビデオ機器がフロントエンドネットワークに接続される時に生成され、離脱すると抹消される一時的なものである。エンドターミナルノードが持つ、識別子 (IEEE1394 なら NUID) と一意に結び付けられている。

- パーミッション：資源がコアネットワークにアタッチされている条件

  資源はコアネットワークにアタッチされた時に、はじめて資源管理の対象となる。この属性は、アタッチされているか否かの状態を示すものとなる。将来的には、アクセスの条件を設けて資源ごとの利用条件を示すことが期待できる。(例えば、利用できる他のフロントエンドネットワークを指定するといったアクセス制限)

また、フロントエンドネットワーク内で故障等の異常や、機器の抜き差し等を主な原因とする資源状態の動的推移の通知をセッションアダプタから受け取ると、コアネットワークに伝播させる。

### 3.2.2 コアネットワーク資源の管理

リソースマネージャは資源管理、資源発見、資源接続の三つの機能をもつことにより、コアネットワーク内において二つの資源の単純接続を提供する。リソースマネージャは複数のゲートウェイ及び、アタッチされたコアリレーノードを管理する。ゲートウェイ内のリソースインフォメーションエージェントと通信することにより、フロントエンドネットワーク内の資源情報をやりとりすることができる。これにより、ビデオネットワークにアタッチしている全ての資源を管理することが可能となる。

#### 資源管理

リソースマネージャは、アタッチされているゲートウェイ及び資源を管理する。各々の資源は、コアリレーノードインフォメーションとして保持している。リソースインフォメーションエージェントから資源状態の変更通知を受け取ると、コアネットワークレベルで対処する。

#### 資源発見

管理下のネットワーク内に必要な資源が足りない場合、他のリソースマネージャが管理しているネットワーク内に利用可能な資源があれば、これを利用するための機能。

#### 資源接続

各資源に対してシグナリングを促すために、接続する資源を管理しているリソースインフォメーションエージェントに対して接続命令を出し接続情報をデータベースに登録する。

## 3.3 接続管理

VIA では、コアコネクション及びループバックパスの二つの接続管理を行っている。

- コアコネクション

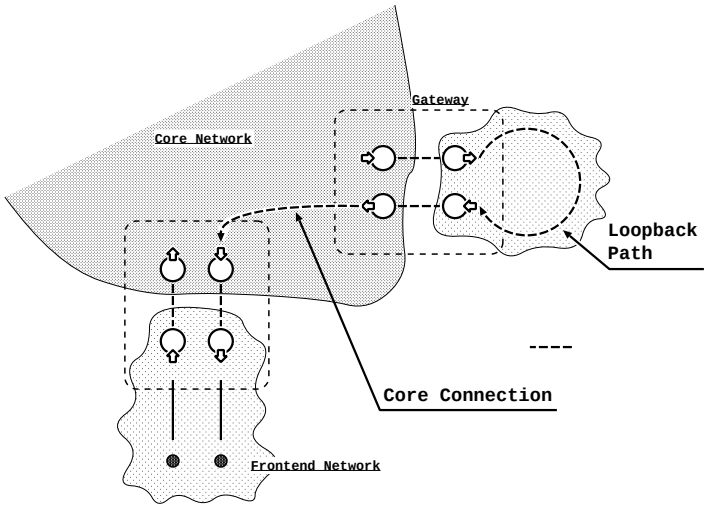



図 3.2: コアコネクションとループバックパス

コアネットワーク内でのコアリレーノード間の接続。

- ループバックパス

  コアネットワークから見ると、あるコアリレーノードからフロントエンドネットワークへ出力されたデータが入力のコアリレーノードに帰ってくるパスのこと。トランスコーダ等の中間資源を通過する場合に用いられる。

## 3.4 全体としての動作

VIA の動作は大まかに、資源及びゲートェイのコアネットワークへの登録動作と資源間の接続動作の二つに分けられる。

本システムでは登録動作を、アタッチ、デタッチと呼ぶ。詳細は [2] を参照されたい。

以下に示すのは、ビデオ機器が実際に接続されて、コアネットワークにアタッチされるまでの流れである。

1. セッションアダプタに機器が接続されると、エンドターミナルノードとして認識される。リソースインフォメーションエージェントに通知する。

2. リソースインフォメーションエージェントは、エンドターミナルノードをフロントエンドリレーノードに変換し保持する。

3. リソースインフォメーションエージェントは、フロントエンドリレーノードをコアネットワークの資源として利用できるように、リソースマネージャに対してアタッチ要求メッセージを送る。

4. リソースマネージャは、フロントエンドリレーノードをコアリレーノードに変換し保持する。

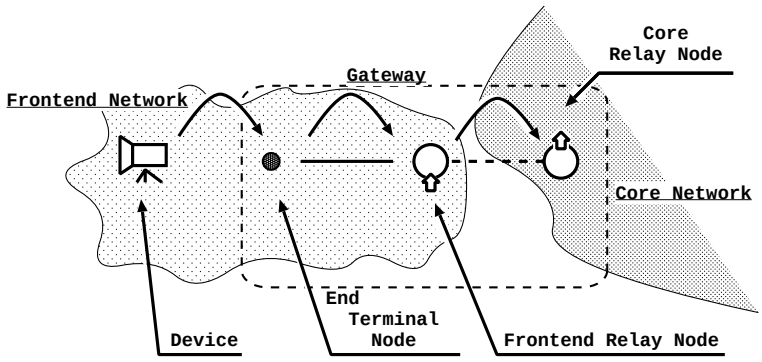



図 3.3: コアリレーノードへの変換

# 第 4 章

# 接続の管理

従来、接続の管理は二つの資源の間の接続を送信側と受信側の識別子及び、付随する管理情報を一つの接続管理情報の単位として管理してきた。前章で紹介した VIA における接続管理の形態も同様である。

しかし、実際に利用するアプリケーションでは複数の資源を複雑に接続する必要が頻繁に発生すると考えられるため、より高水準の接続管理が求められることが予想できる。

本節ではこの問題を解決するために、接続の形態を「基本接続」と「複合接続」の二段階のものとして考えることを提案し、管理の方法を考察する。

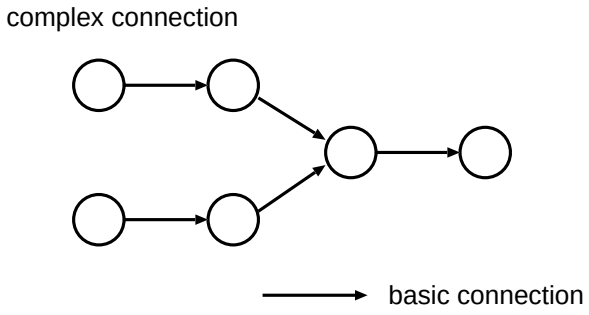



図 4.1: 基本接続と複合接続

## 4.1 基本接続

### 4.1.1 基本接続の定義

本研究では、二つの点とその間を結ぶ一本の線を要素とする接続を基本接続と呼ぶこととする。ビデオネットワークシステムに当てはめると、ネットワーク上の送信側資源と受信側資源をそれぞれ端点とした接続を一つの接続単位とし、これを基本接続と呼ぶことになる。つまり、従来考えられてきた接続管理は基本接続を管理していると考えることができる。

どんな接続形態であっても、必ず基本接続が基本である。

### 4.1.2 VIA における基本接続

VIA における例を示すと、各々のコアコネクションは基本接続であるといえる。また、ループバックパスも基本接続に含めることが可能である。ループバックパスは、コアネットワークからフロントエンドネットワークへの出力口から入力口への接続である。これも、出力口及び入力口を二つの端点とし、ループバックパスを線とする接続と見ることができるからである。

### 4.1.3 基本接続の管理

基本接続は、二つの端点を一つのセットとして管理することが可能である。さらに、接続に識別子を付せば二つの端点の間に複数の接続を確立することも可能となる。このように、基本接続そのものの管理は容易である。なぜなら、他の基本接続との依存関係を考慮する必要がないからである。つまり、一つ一つの基本接続は独立として扱われる。

しかし、大規模なアプリケーションを動作させる場合、複数の基本接続が依存関係を持つと考えたほうが自然である。なぜなら、一般的にビデオを扱う場合、映像や音声の信号は様々な機器を通過させることによって所望の出力を得るからである。

## 4.2 複合接続

複合接続は、複数の基本接続が依存関係を持った集合である。つまり、依存関係のある複数の接続を一つの単位として見るとき、これを複合接続と呼ぶこととする。基本接続では、関係のない線分の集合であったものが、複合接続では、依存関係のある平面的なもの

として扱うことが可能となる。平面的なものとして扱うことで、実際に接続されている接続の形態に近い形で管理することができるようになる。

### 4.2.1 複合接続の管理

複合接続を逐次管理することが可能となれば、時間の経過に伴う資源の状態変化の影響のある範囲を特定して部分的に対処できるようになる。なぜなら、接続関係を平面的に扱うことができるので、依存関係が伝播する幾つかのエリアに分割することが可能となり、資源状態の変化のエリア内での処理や、接続操作の一括処理が可能となるためである。

複合接続の管理をビデオネットワークに導入すると以下のような効果が期待できる。

- 接続構成を柔軟に変更

  接続構成を変更する場合には、関連する基本接続を解放してから、再び必要な基本接続を確立する必要がある。変更の規模が小さければこの対応にかかる負担は小さいが、規模が大きくなり、変更の頻度が増加すると、かなり大きな負担を背負うことになる。複合接続で管理すれば、変更に関して依存関係のある個所をまとめて変更する操作が可能となる。

- フォーマット変換資源を自動的に接続

  大規模なアプリケーションになれば、利用する資源が対応可能なフォーマットの種類は増加すると考えられる。基本接続のみを考慮した接続システムでは、ユーザが適したフォーマット変換資源を中間資源として接続する必要がある。複合接続として管理することができれば、送信側と受信側のフォーマット情報をチェックして、必要であればネットワーク内から適した資源を検索する動作が可能となる。

- 信頼性の向上

  基本接続同士の依存関係を保持しているため、資源に異常が発生した場合、他の資源を検索して回復することを可能とする機構を設けることが可能である。

- ユーザアプリケーションを容易に作成可能

  複合接続を容易に作成できるインターフェースを設けることにより、アプリケーション側の負担を減少させることができると考えられる。アプリケーションは基本接続間の依存関係を、考慮する必要がないからである。

## 4.3 基本接続管理と複合接続管理の関係

本研究では、基本接続管理と複合接続管理を二階層のシステムとして考える。下位の基本接続管理では、システム内の資源の整合性を保ち単純な管理のみ行う。その上で、上位の複合接続管理では、複数の基本接続に関して高度の管理を行い、ユーザアプリケーションに利用しやすいインターフェースを提供する。

システムが基本接続の管理のみを提供する場合、高い信頼性や高水準の接続操作を上位のアプリケーションに押し付ける形となってしまう。その結果、各々のアプリケーションごとに異なった接続管理機構が実現されてしまう可能性が高く、相互に運用するすることが難しくなるであろう。

一方、複合接続を柔軟に管理するための土台がシステムに存在して、相互に利用可能な共有のフレームワークが存在すれば、アプリケーション側で接続に関する管理をする必要がなくなる。その結果として、アプリケーションの差異は接続に関しては負担にはならなくなる。

また、複合接続のみを考慮する場合は、アプリケーション側から見れば問題がないが、実際に接続の確立を行う場合にはどうしても基本接続を積み上げていかねばならないためシステムを設計することが難しい。

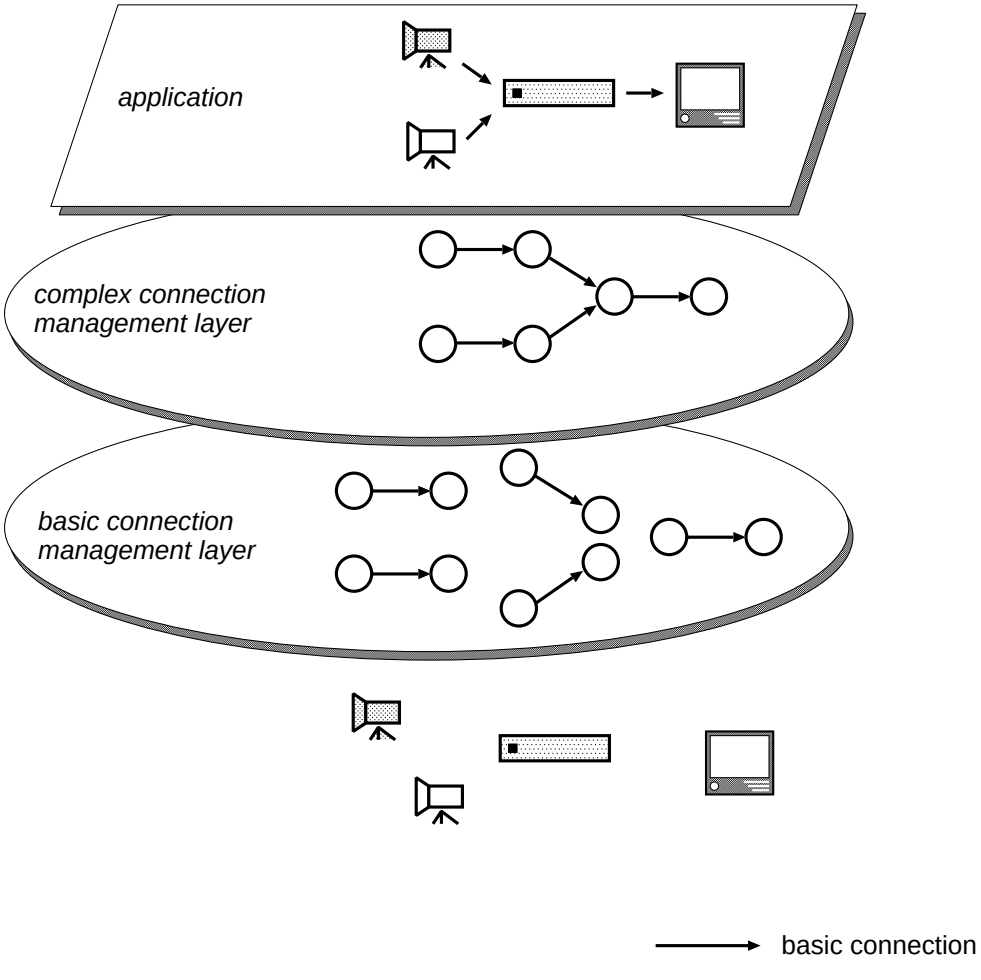



図 4.2: 接続管理の階層構造

# 第 5 章

# セッション

セッションマネージャは複合接続の管理をセッションという単位で扱い、変換資源の自動的な付加接続とセッションに対する操作を行う。セッション単位で管理することにより、複雑な接続であっても、個々の資源の状態変化に柔軟に対応することが可能となる。

セッションマネージャはコアネットワーク内に位置しユーザアプリケーションからのセッション操作要求を受け、リソースマネージャと通信することで動作する。

## 5.1 セッション

従来は、複合接続を管理するをあまり考慮されてこなかった。しかし、今後実際にビデオネットワークを利用する場合、ネットワーク上にある複数の資源を複雑に組み合わせて利用する局面が増加すると考えられる。もし、二つの資源の接続の集合として全ての資源を管理するとなると、あまりに多くの接続を別々の独立したものとして人間が管理しなければならないが、利用するアプリケーションによっては接続する資源が膨大な数に上ってしまうため非常に扱いが難しくなる。また、ビデオネットワークの場合フォーマットタイプの差異を吸収する必要が有るが、これを考慮しながら接続することは非常に困難である。ユーザは、末端の資源及び利用したい中間資源のみの接続を指定するだけで、所望のアプリケーションシステムを設定することを望むであろう。

そこで、本システムでは、複数の資源の接続を一つのまとまりの単位としてセッションと呼び、これを管理することで、上記の問題を解決する。セッションとして複雑な接続をトポロジカルに管理することにより、ユーザにかかる複雑な資源管理及び接続管理の負担を劇的に軽減することが可能である。しかし、このようなトポロジカルな接続を動的に管理することは非常に難しいため、本研究では、セッションを複数のサブセッションの集合

と見なす戦略をとることとする。サブセッションとは、単方向の分岐のない資源接続のことである。

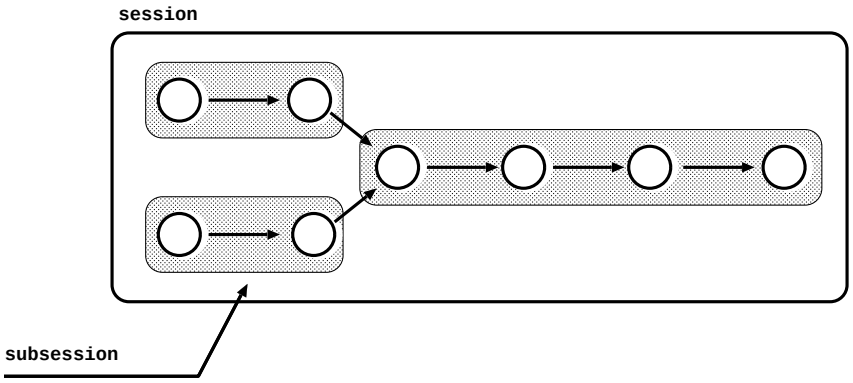



図 5.1: セッション

サブセッションは直線であり分岐しないので、ある資源の状態変化の伝播する範囲はとりあえずは、属しているサブセッション内であるということがわかる。サブセッションのセッションにおける位置づけに依っては、該当サブセッションの状態変化がセッション全体に及ぼすこともあるが、セッションの末端のサブセッションに変化があった場合にはそのサブセッションの外に変化は伝播させる必要はないと考えられる。

セッションをこのようにサブセッションに分けることにより、セッション全域を管理するよりも狭い範囲で接続を管理することができる。

以降でより詳細なセッションの機能、動作等について述べる。

## 5.2 セッション自動生成

セッションマネージャは、ユーザにサービスを提供するために必要な資源をリソースマネージャに問い合わせ、ユーザの要求する資源接続に適応して自動的に変換資源を付加接続する。例えば、DV30Mbps のフォーマットの映像を送信するカメラと MPEG2 で 6Mbps のフォーマットを受信することができるレコーダがあったとしよう。この場合、従来は、フォーマットの差異を吸収するためにトランスコーダをどちらかのエンドポイントに配置し、フォーマットを合わせた上で送受信するか、ネットワーク上にトランスコーダを設置して正しくフォーマットの差異を吸収できるような接続を人間が設定しなければならなかった。図 5.2。

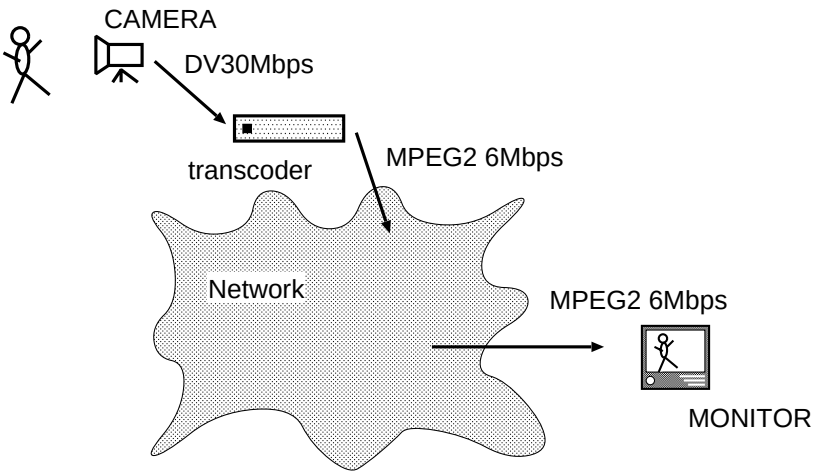



図 5.2: 従来のビデオ接続

規模の小さいアプリケーションであれば、この方法でも可能であり、フォーマットを変換しなければならない局面に遭遇する機会も少ないであろう。しかし、非常に多数の機器を複雑に接続する必要のある場合には、非常に負担の大きな作業となる。また、機器の故障や接続の変更等に対処することは不可能に近い。さらに、エンドポイントにトランスコーダを容易しなければならない場合には、エンドポイントで物理的な配線に悩まされることはおろか、経済的にも大きな負担となる。もし、ビデオネットワーク内にトランスコーダが大量にアタッチされており、それらを管理し利用することができれば、これを簡単に接続することが望ましい。図 5.3。本システムでは、これを実現するための機能を提案、導入する。

コアネットワークで利用する資源にはフォーマットタイプと入力資源の情報が含まれているため、必要な中間資源（例えば、トランスコーダ）をフォーマットタイプと入出力を用いてマッチングを行うことで、検索可能である。セッションを作成する時に、検索により得られる適切な中間資源を自動的に組み込むことにより、自動的に所望のアプリケーションシステムを設定すること実現できる。これは、ユーザの負担を軽減するだけでなく、最適な中間資源の検索や故障時のリカバリ操作等でも威力を発揮する。

また、あらかじめセッションのポリシーを設定しておき、これに従ったセッション作成を行う方法も考えられる。

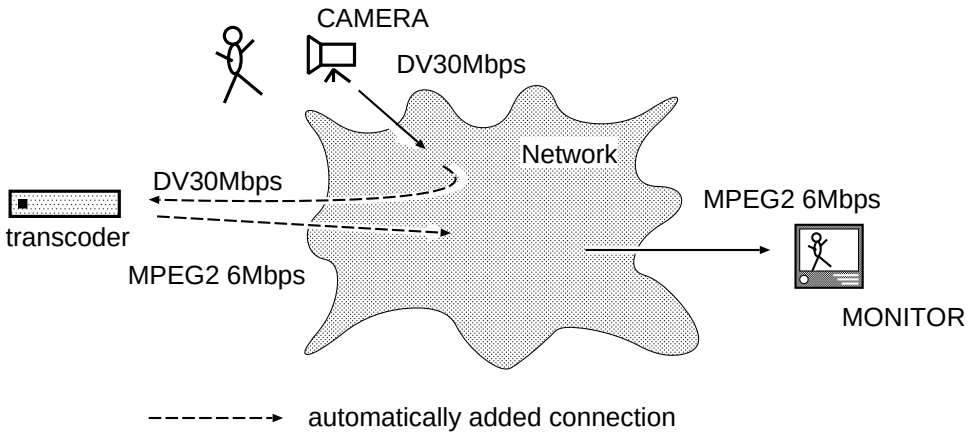



図 5.3: ネットワーク内でフォーマット変換

## 5.3 セッションの動的な変更

ビデオネットワーク上の資源は、故障や、追加、離脱等様々な状態推移が考えられる。つまり、セッションに参加する資源状態は、常に一定の状態を保つわけではなく、動的に推移していくものである。こういった変化は、セッションの規模が拡大するほど多発することが予想され、軽い負担で正常状態に復帰することが求められる。個々の接続を、独立に扱う場合には、セッションを正常状態に復帰させるための負担は大きい。なぜなら、個々の接続に対して全ての接続を人間が把握し、管理しておかなければ、正常状態に直すための手順がわからないからである。本システムでは、セッションとしてビデオネットワーク上の複数の接続を管理しているため自動的に正常状態に復帰させる機構を設けることが可能である。複雑に絡みあう接続であっても、接続と接続の関係を保持しているため、ある資源に問題が発生したことによる悪影響の範囲を絞り機械的に対処することができる。

セッション内の資源状態に異変が起こり、これに対処しなければならない場合、その対処の仕方には以下のように大きく二つの方法が考えられる。

- セッション全体をリセットする
- 変化のあったサブセッションで回復する

前者では、セッションの再構成のために複雑な動作をする必要はない。ただし、異常のあった資源が属していたサブセッション以外のサブセッションまで再構成することになり、局所的な異常がセッション全体に波及してしまうこととなる。

後者では、異常のあるサブセッションのみのリセットとなるために、悪影響を最小限にとどめることが可能となる。サブセッションの再構成のための戦略として以下の三つ (図 5.4) が考えられる。[22]

1. local rerouting

   問題の発生した資源のみ迂回する。

2. local-to-end rerouting

   問題の発生した資源の一つ手前からサブセクションの出口となる資源まで迂回する。

3. end-to-end rerouting

   全て迂回する。

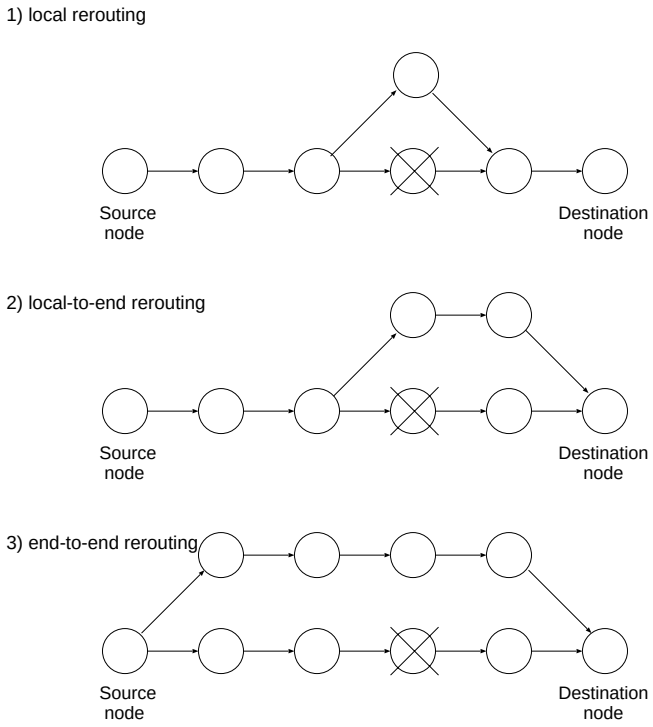



図 5.4: サブセッション再構成戦略

また、迂回する資源を決めるタイミングには二つの戦略が考えられる。

1. 資源に状態変化が発生したとき

2. サブセッションを生成するとき

前者では、回復のためにかかる時間的なオーバヘッドが大きくなる。一方、その時点で利用されていない適した資源がネットワーク上に存在すれば、必ず回復することができるという利点がある。

後者では、サブセッションを生成する時にホットスタンバイもしくはコールドスタンバイする資源を予約しておかなければならない。これにより、回復のためにかかる時間的なオーバヘッドは小さく抑えることが可能である。ホットスタンバイにした場合、実際に利用されていないにも関わらず、資源が予約されているという理由で回復することができないサブセッションが出現する可能性があるため、信頼性及び資源の利用効率は低下する。コールドスタンバイであれば、信頼性及び利用効率を低下させることはないと考えられる。ただし、ビデオネットワーク上の資源状態に変化がある度に予約する資源をチェックしなければならないため、システムの動作上オーバヘッドが大きくなることは避けられない。

さらに、柔軟性の高い信頼性サービスのために予約に優先度を付加することが考えられる。資源に問題が発生した場合、優先度の高いセッションは、優先度の低いセッションが利用している資源を割り込んで利用することができるようになる。信頼性をも考慮したセッションに関する議論は、次章でより詳しく行う。

## 5.4 セッションの実際

本章では、セッションの特徴や機能について説明してきた。以降では、実際にセッションを利用するために必要な考え方を説明する。

### 5.4.1 データ構造

セッションは、平面的な広がりを持った複数の接続の構造をもっている。セッションを計算機で保持するためには、何らかのデータ構造を規定する必要がある。グラフ理論的見地からすれば、セッションは有向の全域木であるため、これを表現できる構造であれば良い。どのような構造を採用するかは、実装上の問題であるが、ここでは以下の二つの構造について考察することとする。以下では、図 5.5のようなセッションを表現することを考える。

セッションのデータ構造を考察する上で、上述したように有向の全域木を表現可能であることが前提条件であるが、その性能を考慮するとサブセッションを柔軟に扱うという観

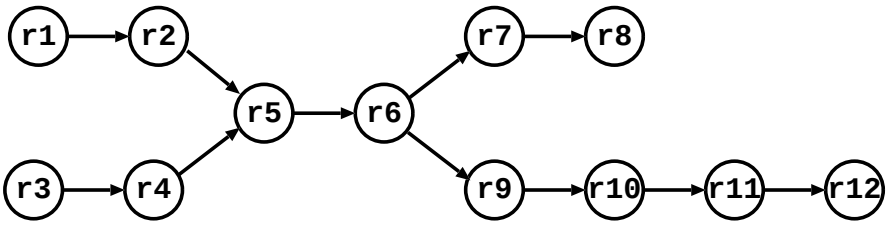



図 5.5: セッションの例

点も検討するべきである。セッションを管理、操作する場合には、セッションを表現すると同時に、その中に存在している複数のサブセッションを認識する必要がある。つまり、サブセッションを柔軟に管理、操作可能かどうかが、本システムの性能の評価を左右する重要なポイントとなる。

こういった議論をふまえた上で、候補となる以下の二つのデータ構造について検討することとする。

- 行列構造
- リスト構造

以降でそれぞれの構造の特徴を述べるが、実際にはうまく組み合わせて考えることが重要である。

### 行列構造

セッションは、有向グラフであるため、隣接行列と似た方法を用いることにより表現することができる。以降、セッションを表現する行列のことをセッション行列と呼ぶこととする。図 5.5を行列表現すると図 5.6となる。

隣接行列では、該当要素に枝の数を記入するが、セッション行列では、その接続の種別を記入することとする。N が記入されている場合には、コアコネクションであることが表現される。つまり、その接続はコアネットワークを介した接続であるということがわかる。L が記入されている場合には、ループバックパスであることが表現される。この場合には、フロントエンドネットワーク内にループバックパスが張られているということが示される。

行列構造を用いることによって、どのような構造の接続であっても簡単に表現することができる。また、行列構造であれば、セッションの構造を示す情報の保存は簡単にでき

Destination resource

| Source resource | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9 | r10 | r11 | r12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| r1 | | **N** | | | | | | | | | | |
| r2 | | | | | **L** | | | | | | | |
| r3 | | | | **N** | | | | | | | | |
| r4 | | | | | **L** | | | | | | | |
| r5 | | | | | | **N** | | | | | | |
| r6 | | | | | | | **L** | | **L** | | | |
| r7 | | | | | | | | **N** | | | | |
| r8 | | | | | | | | | | | | |
| r9 | | | | | | | | | | **N** | | |
| r10 | | | | | | | | | | | **L** | |
| r11 | | | | | | | | | | | | **N** |
| r12 | | | | | | | | | | | | |

**N:Normal Core Connection**
**L:Loopback Path**

図 5.6: セッション行列

る。ただし、セッションの規模が大きくなり参加する資源の数が増加するにつれて、セッション行列の格納に必要な容量は二乗ずつ増加してしまう。さらに、行列内で実際に接続されている部分はそれほど多くならないので、行列が大きくなればなるほど、無駄な容量が増加してしまうという欠点もある。

行列を用いる場合は、セッションそのものの構造を簡単に表現できるという面では優れているが、方向性を持ったサブセッションを抽出するための操作の負担は大きくなってしまう。なぜなら、サブセッションを抽出するためには、行列の中からサブセッションの始点となる点、もしくは終点となる点を探索し、そこから、次の点もしくは前の点をたどる操作が必要となってくるからである。

また、行列を更新するためには、一度行列を全て構築しなおすか、一部分の変更を加えなければならないが、資源の追加や削除、交換等の操作を加えるためには複雑な仕組みを必要とする。このため、セッション規模が大きく資源状態が次々に変化する場合には、危険の伴った動作となってしまう。

さらに、行列を用いる場合には、資源のプロファイルとも言える情報を別の方法で記憶しておかねばならない。例えば、図 5.6の r2 は 2 行目（もしくは 2 列目）であるが、この資源の識別子等の情報は別の場所に、行列中の位置と結び付ける形で保持しなければならない。

このように、行列構造は実現は比較的簡単であるが、性能や効率はあまり高くないと考えられる。行列構造を用いる場合には何らかの工夫を施さねばならないだろう。

### リスト構造

リスト構造は、サブセッションを表現するのに適していると考えられる。サブセッションは、一方向の分岐のない接続であるため、ポインタを一つしか持たない要素をリスト構造にすれば表現できる。図 5.5をリスト構造で表現すると図 5.7の様になる。以降、セッションをリスト表現したものをセッションリストと呼ぶこととする。

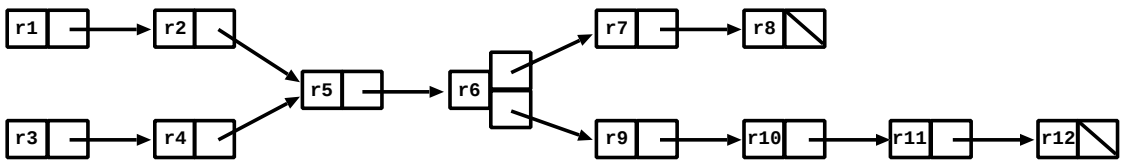



図 5.7: セッションリスト

セッションリストを用いると、サブセッションの抽出にかかる負担が非常に軽くなると

考えられる。無駄な検索をすることなく、サブセッションの始点から終点までたどることが可能だからである。また、接続の存在する部分のみ容量が必要なので、セッション行列と比べると必要な記憶容量も少なく抑えることが可能である。行列構造の時には、行列とは別に資源情報を結び付ける情報構造も設ける必要があったが、リスト構造の場合にはリストのデータ部分に含めることができるというメリットがある。

また、リスト構造であれば、資源の追加、削除、交換、挿入といった操作は、比較的簡単に行うことができるため、大きなセッションになっても十分安全かつ高い性能を保ったままで稼働させることが可能であると考えられる。

しかし、複数の大規模なセッションが同時に稼働するような場合には、セッション情報を何らかの形で保存しておくことが望ましいが、リスト構造の保持、保存は難しい。さらに、図 5.7の r6 のポインタ部分のように、分岐が必要な部分には動的に複数のポインタ部分を追加することが可能でなければならない。

### 5.4.2 セッション生成アルゴリズム

セッションはサブセッションの集合として見ることで、ただ単にセッションという大きな枠で見るよりも、柔軟な動作を行う可能性があるということがわかる。つまり、セッションは管理単位で、サブセッションは動作単位といった要素が強い。ここでは、セッション及び、サブセッションを生成するためのアルゴリズムに関して以下の二つを考察する。本小節でも図 5.5を利用して説明する。

- サブセッション埋め込み法

  先に、必要なサブセッションを生成した後、生成されたサブセッションの集合としてセッションを生成する方法。

- セッション分解法

  セッションを先に生成した後に、セッションを分解することにより、複数のサブセッションを抽出する方法。

これら二つの手法は、セッションとサブセッションのどちらを先に生成させるかの点において相違がある。以下で詳しく説明する。

#### サブセッション埋め込み法

サブセッション埋め込み法では、何も入っていない空の殻であるセッションを作成し、その中にサブセッションを埋め込んでいく方法である。本方法の手順を以下に示す。

1. セッションマネージャは、ユーザからセッション生成の要求を受けて、セッション ID を生成し空のセッションの識別子とする。

2. セッションマネージャは、ユーザアプリケーションから利用したいサブセッションの入力を受け付ける。サブセッションの識別子はユーザアプリケーションが付加し、サブセッション間の関係の情報も渡す。

3. セッションマネージャが、受け付けた複数のサブセッションを検査し、必要なら中間資源を自動的に挿入する。

4. セッションマネージャは、サブセッション間の関係を保存しながらセッションを形成する。

本手法では、サブセッションをリストとして扱い、そのリストの集合としてのセッションを考えるか、セッションそのものもリスト構造として保存することとなる。サブセッションリストの集合としてセッションを保持するのは、それほど難しくはないが、サブセッション間の関係を維持するために何らかの仕掛けが必要となる。一方で、全てリスト構造にすれば、上記の問題は発生しないが、セッションそのものの保存が難しくなる。

図 5.8にサブセッション埋め込み法を示す。図の右側が、初期状態でのユーザアプリケーションの保持しているセッション情報であり、左側はセッションマネージャが保持するセッション情報である。最初、ユーザは右上にあるようなセッションを生成させたいと考える。そして、何らかの方法でユーザアプリケーションに、生成させたいセッションの情報を渡すと、ユーザアプリケーションは要求されたセッションの形態から、サブセッションに分割した上で、サブセッション同士の関係についても保持する。(図 5.8右下)

次に、セッションマネージャは、前もって生成してあった空のセッションに、要求のあったサブセッションを埋め込んで行く。さらにサブセッションを検査して必要であれば、新たに資源を追加する。ここでは、r13、r14、r15、r16 の 4 つの資源が追加されている。

この方法をとると、セッションマネージャのセッション生成動作及び、セッション操作の負担は小さく抑えることができる。しかし、ユーザアプリケーションはサブセッション及びサブセッション間の関係を認識する必要がある。

**セッション分解法**

セッション分解法は、セッションの構成を把握した後に、サブセッションに分解していく方法である。本手法の手順を以下に示す。

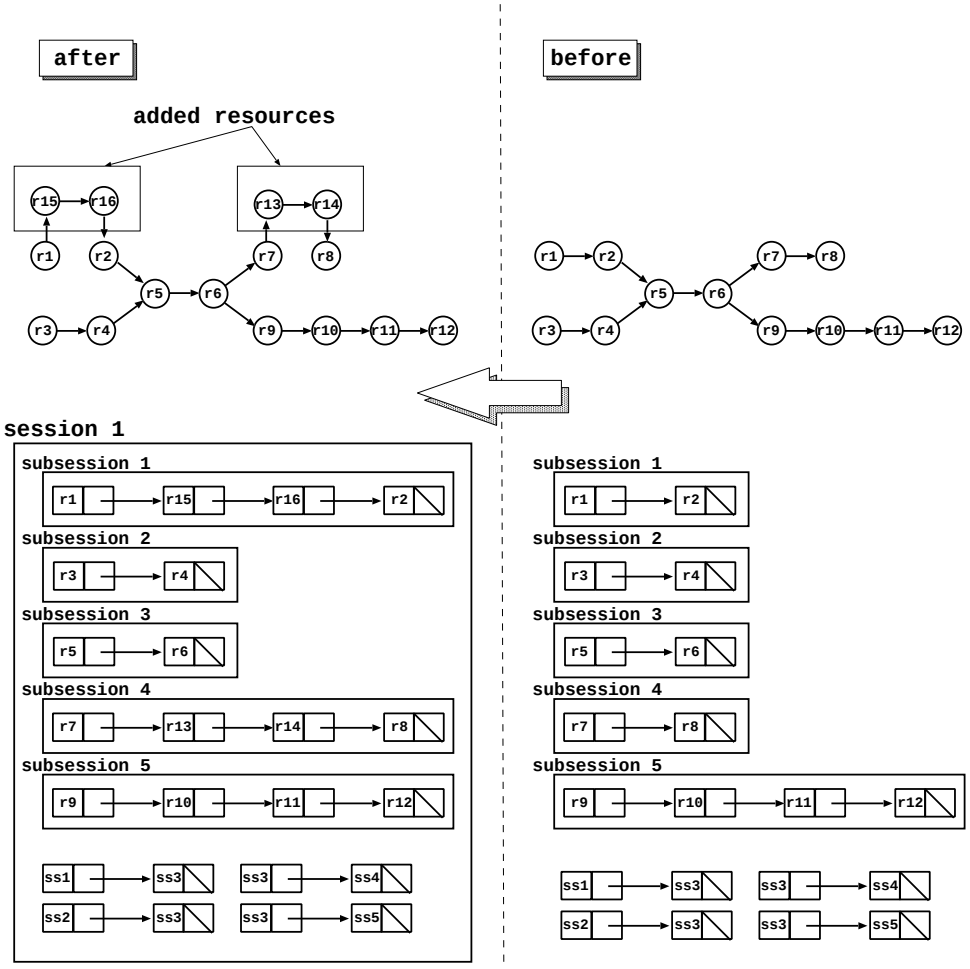



図 5.8: サブセッション埋め込み法

1. ユーザは、必要な資源を指定して複数の基本接続を選択する。
2. ユーザアプリケーションは、ユーザの要求から基本接続の集合として保持する。
3. セッションマネージャが基本接続情報の集合を受け取る。
4. セッションマネージャは、得られた基本接続の集合から、セッション行列を生成すると同時に、セッション ID を付加する。
5. セッション行列からサブセッションを抽出する。
6. 抽出したサブセッションを検査して、必要な中間資源を自動的に追加する。
7. 利用する資源に変化があった場合は、セッション行列を更新する。

本手法では、セッションを行列構造で扱うこととなる。ユーザは、利用したい資源をサブセッションを意識することなく、基本接続として指定していく。ユーザアプリケーションはユーザが指定した基本接続を全て保持して、まとめてセッションマネージャに送る。すると、セッションマネージャはセッション行列及び、セッション ID を生成して、セッションの管理が開始される。セッションマネージャは得られたセッション行列から、サブセッションリストを抽出する。

この方法では、セッション行列から、サブセッションリストを抽出するために複雑な操作を必要とする。また、サブセッションの追加、削除、変更等が発生した場合には、セッション行列を何らかの方法で更新する必要がある。本システムでは資源状態の変化が頻繁に発生することを想定しているので、この動作をできるだけ効率的に行うための工夫が必要となろう。

また、図 5.9にあるように、セッションマネージャの自動的な資源接続の動作により必ずセッション行列は拡大することとなる。本システムの性質上、トランスコーダ等の自動的に接続可能な資源に関しては、ユーザは関与しないため、規模が大きなセッションになると、追加される資源の量はは規模に応じて増加することとなる。その結果として、セッション行列が極端に肥大化してしまう恐れがある。セッション行列の肥大化は、セッション行列の更新や、サブセッションの操作に悪影響を与えるので好ましくない。

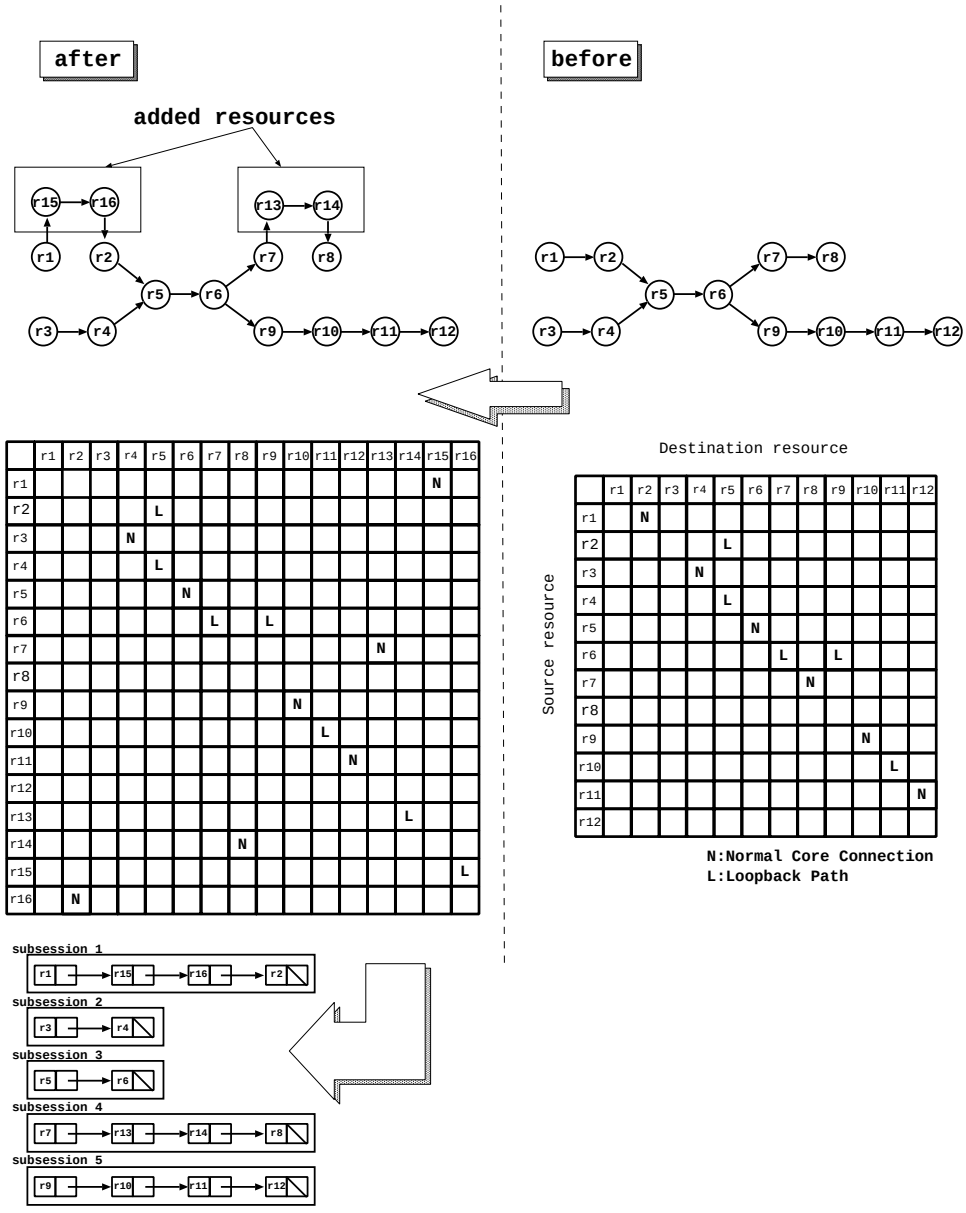



図 5.9: セッション分解法

# 第 6 章

# 高度なセッション管理

前章では、セッションの管理について基本的な内容を説明した。セッションの特徴としては以下の三つがあげられる。

- セッション、サブセッションとしての一括した操作。
- 資源情報を有効に利用することによる中間資源の自動的な追加。
- 資源情報の変化に対して自動的に追随する動作を設置することが可能。

こういった、特徴を生かすことにより、システムの信頼性の向上や個別のアプリケーションで利用したい資源を柔軟にネットワークを介して操作することが可能となる。本章では、信頼性を向上させるために、資源の優先度付き予約に関して検討することとする。これまでの議論では、セッションを生成する時点で利用できる資源を利用してきた。セッションが大規模になれば利用する資源の数は増加するため、中には稼働中に故障等の状態変化が発生するものが現われることが予想される。状態変化が発生した時点で、新たに必要な資源を検索する場合、すでにネットワーク内の資源が利用し尽くされている状況であれば復帰することができない。また、変化が発生してから検索を開始していたのでは、復帰にかかるリカバリタイムが大幅に悪化してしまうことは避けられない。そこで、セッションを生成する時点で予備資源を予約しておき、資源に変化が発生したときには、すぐに切り替えて利用することができるような仕組みを考察する。

## 6.1 セッションの信頼性

今後、途中の資源が故障に見舞われても、予備となる資源に自動的に切り替わってセッションの稼働を可能とすることが求められるような、高い信頼性を必要とするアプリケー

ションがビデオネットワークにも出現すると考えられる。例えば、遠隔医療の中で遠隔手術が行われた場合、ほんの少しのシステムダウンも許すことができないのは想像に難くないだろう。前章までの、セッションの基本的な仕組みを利用することで、自動的に復帰するシステムを構築することが可能であるが、さらに信頼性をあげるための枠組みを構築することが重要である。

そこで、本節では信頼性を向上させるための一つの手段として、資源の予約を検討することとする。本小節では、資源予約の説明の前準備として信頼性に関して考察する。

信頼性には以下のような要素が考えられる。

- リカバリタイム

  ある資源がダウンして、その資源を必要としているサブセッションがダウンしたとき、そのサブセッションが回復するまでの時間。タイムクリティカルなアプリケーションであれば、最も重要視されるべき要素である。

- 信頼度

  セッションやシステム全体としてのダウンのしにくさ。資源を予約して、冗長性のある構造とすることにより、信頼度をあげることができる。

- 稼働率

  故障があっても修理して利用する場合に、システムがアップしていられる率。長い期間でビデオネットワークシステムを見たときに、考えるべき要素である。また、恒久的に可動し続けるセッションに関しても、考慮すべき要素である。

以上の要素を向上させることにより、システム、セッションの信頼性をあげることができる。例えば、リカバリタイムを最小にするためには、予約した資源をホットスタンバイにして瞬時に切り替えることを可能とすることが考えられる。また、信頼度をあげるためには、同じ機能を持つ資源をできるだけ大量にネットワーク上に配備して利用できるようにしておけば良いだろう。

一方、本研究の考えるビデオネットワークシステムでは、一つのアプリケーションに特化したものではなく様々な種類の用途のものが共存するものであるため、ネットワークシステムとしての効率性も向上させる必要がある。

- 資源利用率

  ビデオネットワークシステム上にアタッチされている資源のうち、利用されている資源の割合。あまりに多くの資源がネットワーク上に配備されても、全く利用され

ない資源が現われる可能性がある。そのような資源は、他のビデオネットワークシステム（他のリソースマネージャの管理下）に置かれるべきであろう。

- リカバリオーバーヘッド

  リカバリのために、セッションマネージャの性能としてのリソースを大量に消費してしまうと、正常に稼働しているセッションに対して多大な悪影響を与えてしまいかねない。リカバリオーバーヘッドはなるべく小さく抑えることが求められる。

これらは、トレードオフの関係にあると考えられるが、後述する優先度を調整することにより、アプリケーションにあった信頼性を得ることができるようになると考えられる。

## 6.2 資源の予約

資源の予約を行うことにより、リカバリタイムと信頼度の向上を見込むことができる。ホットスタンバイであってもコールドスタンバイであっても、資源を検索する時間が必要なくなるからである。ホットスタンバイであれば、接続操作に関するオーバーヘッドもほとんどかけずに復帰することが可能となる。ただし、予約した資源を他の資源が利用することができなくなるので、資源利用率は悪化する。

### 6.2.1 資源予約操作のタイミング

資源予約に関する操作には、以下のタイミングを考慮する必要がある。

- セッション生成時

  資源予約そのものは、セッション生成の時に行う。自動的に追加される資源に関しては、資源検索時に適した資源が複数個抽出されると考えられるため、同時に予約も行ってしまう。いくつの資源を予約しておくかに関しても議論が必要であろう。

- 故障発生時

  故障が発生して、予約していた資源に切り替わり復帰を果たした場合には、さらに次の故障に備えて新たな資源予約が必要となる。セッション生成時に複数の資源を予約してある場合には、必ずしも障害発生時に予約を更新する必要はないとも考えられる。

- 新規資源アタッチ時

  セッション生成時には、予約できる資源が存在しなかった場合でも、セッションを稼働している途中で、適した資源がアタッチされれば、これを予約することも考えられる。この場合、他のセッションとの資源の取り合いが発生する可能性が考えられる。複数のセッション間での、協調性を議論する必要がある。

- 予約資源デタッチ時

  予約していた資源がデタッチしてしまった場合には、即時に新たな予約資源を検索し、予約し直さなければならない。

このように、ビデオネットワーク内の資源は、常にその状態を変化させると考えられる。従って、予約を調停する機構をビデオネットワークシステム内に設置する必要がある。本システムでは、セッションマネージャがこの機能を負うこととする。

### 6.2.2 資源予約の形態

資源予約の形態にはいくつかの戦略が考えられる。

- 資源を交換する

  個々の資源に関して、独立に予約を行う。この場合セッション中の他の資源と依存関係なく資源の予約を行うので、一つ一つの資源に対する予約のオプションを自由にとることができる。例えば、一つの資源に対して、いくつの予約をするのかを指定することが可能となる。

- サブセッションを交換する

  サブセッションそのものを交換できるような予約を行う。つまり、故障が発生した場合には、セッションの始点から終点まで全ての資源が切り替わる可能性がある。この場合、資源利用率の極端な低下が考えられるが、リカバリタイムを小さく抑えることが可能となる。

また、予約した資源の管理に関して以下の方法が考えられる。

- 予約資源を他のセッションに利用可能とする

  他のセッションに予約資源を利用させることにより、資源利用率を向上させることができる。ただし、他のセッションに資源を横取りされる場合、新たに予約資源を

検索する必要に迫られるため、オーバヘッド及びセッションとしての信頼性が低下する。

- 予約資源は他のセッションに利用不可能とする

  セッションの信頼性を高く保つことができるが、長期間に及ぶセッションであれば、それだけシステム全体としての信頼性を低下させることとなってしまう。

本小節で示した通り、予約の形態の違いによっても信頼性と効率性の間にはトレードオフが存在することがわかる。

## 6.3 優先度

前小節までで議論したトレードオフは、アプリケーション、セッション、サブセッションごとに、調整できる方が良いと考えられる。そこで、セッションに優先度を設けることにより、信頼性の調整を実現する。

優先度とは、高い優先度を持ったセッションは、低い優先度を持ったセッションよりも有利に資源予約を行うことができるというものである。つまり、ネットワーク内の資源利用の優先制御である。

### 6.3.1 割り込み方法

高い優先度のセッションが低い優先度のセッションの資源を割り込んで利用するが、割り込み方式には以下の二つがある。

- 割り込み優先方式

  ある時点で、資源を利用しているセッションが存在しても、そのセッションを押し退けて、資源を横取りする方式。押し退けられたセッションは、新たに利用できる資源を検索する必要がある。押し退けられたセッションが、さらに低い優先度を持ったセッションから資源を割り込んで利用することも考えられ、ドミノ倒し的に横取りが連鎖する恐れもはらんでいる。

- 非割り込み優先方式

  ある時点で、資源を利用しているセッションが存在した場合、そのセッションが資源の利用を終えるまで、割り込むのを待つ方式。いくら高い優先度を保持していたとしても、待つ必要があるため、リカバリタイムは悪化する。

非割り込み優先方式ではリカバリタイムが非常に悪くなる恐れがあるので、ビデオを利用するアプリケーションには向かない。長い時間映像が跡絶えるのは、ビデオ伝送にとっては致命的な状況であるためである。そこで、割り込み方式としては、割り込み優先方式を選択することが望ましい。

### 6.3.2 優先度の付け方

優先度の付け方には以下の方法が考えられる。

- 目標とする値による優先度付け

  例えば、達成したいリカバリタイム、信頼性等の数値によって優先度を設定する。優先度に上限と下限がない。

- 単純なカテゴリによる優先度付け

  例えば、優先度を 1 から 5 までに分けて、それぞれの優先度にランクをつける。セッションを利用するときに、課金で優先度を設定するといったことが可能となる。

前者の方が、厳密な優先付が可能であるが、どの種類の特性値を利用するかが問題となる。

### 6.3.3 優先度の設定

ここでは、優先度の設定されるタイミングや、優先度の変化について議論する。

優先度が設定されるタイミングは、セッションを生成するタイミングである。ユーザがセッション生成の要求をセッションマネージャに提示する時点で、何らかの優先度が設定される。ただし、セッション稼働中に優先度は変化させる方が適当な局面が現われると考えられる。

## 6.4 予約ポリシ

本章で議論してきたことを、実際にセッションマネージャが管理するには、システムとしての予約ポリシを持ち、これに従う形での予約機構とすることが考えられる。予約ポリシで考慮されるものとして以下のものがあげられる。

- スタンバイ状態

- 予約の個数
- 割り込み設定
- 優先度設定

こういった要素をまとめて予約ポリシとして、管理者が設定できるようなシステムを構築することが望ましい。

# 第 7 章

# VIA の拡張

本研究では、VIA にセッションの概念を取り入れてビデオネットワークシステムを拡張する。大まかに言えば、既存の VIA の上にセッションを管理する機構をおくことにより、接続管理の面から見て全体として二階層のシステムとする。

## 7.1 概観

システムの全体像を図 7.1に示す。上記のように、従来の VIA の枠組みを基本接続の層に当てはめ、新しく複合接続のための層を設ける。コアコネクションとループバックパスを基本接続とし、コアリレーノードを基本接続の端点と見ることができる。既存の VIA でのコネクションは、全て基本接続である。

そして、セッション及びサブセッションはコアコネクション、ループバックパスの集合として表現することが可能である。

## 7.2 セッションマネージャとリソースマネージャの連携

図 7.2のようにセッションマネージャとリソースマネージャの両者がそれぞれ、セッション管理と基本接続管理を行う。

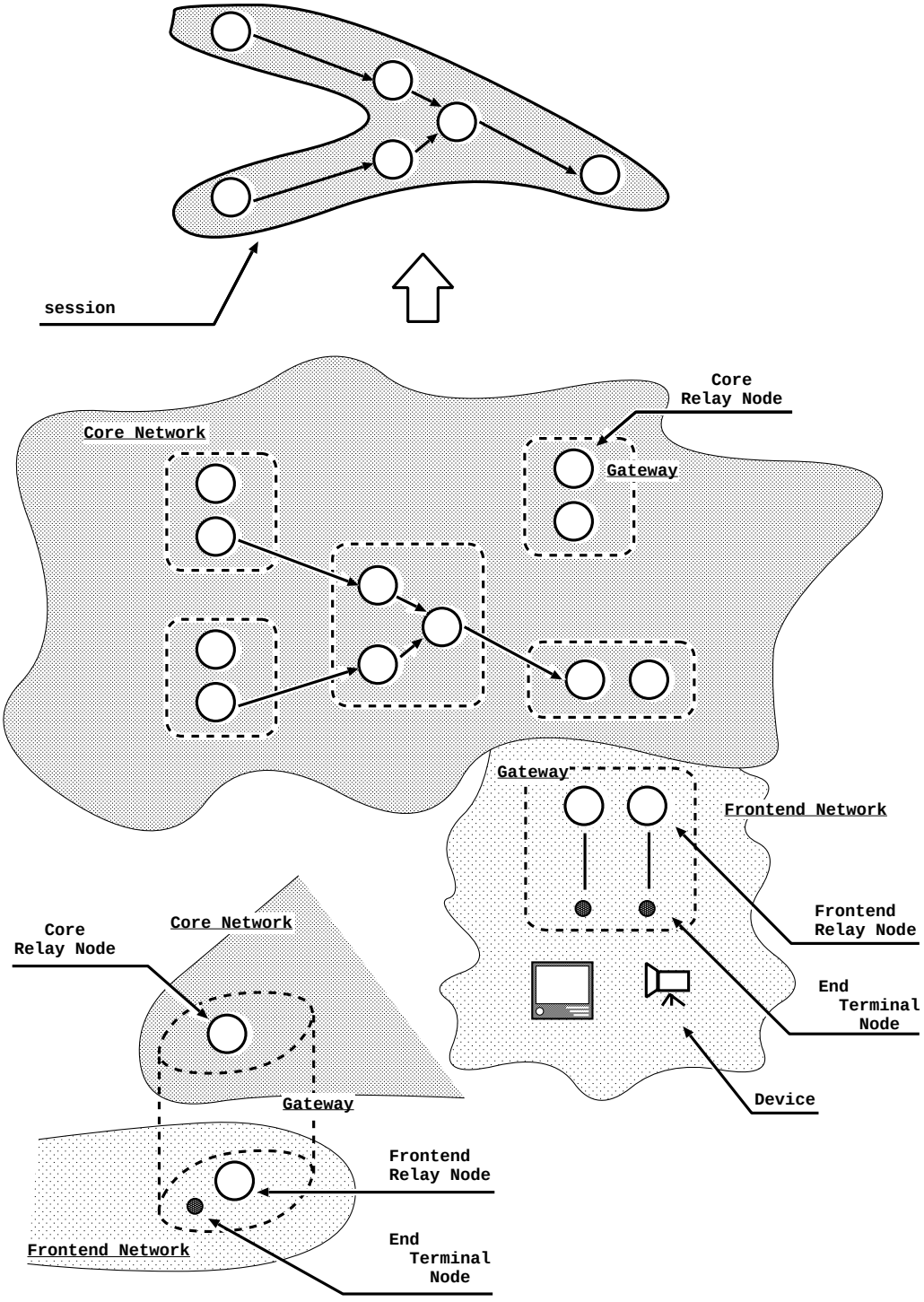



図 7.1: システムの全体像

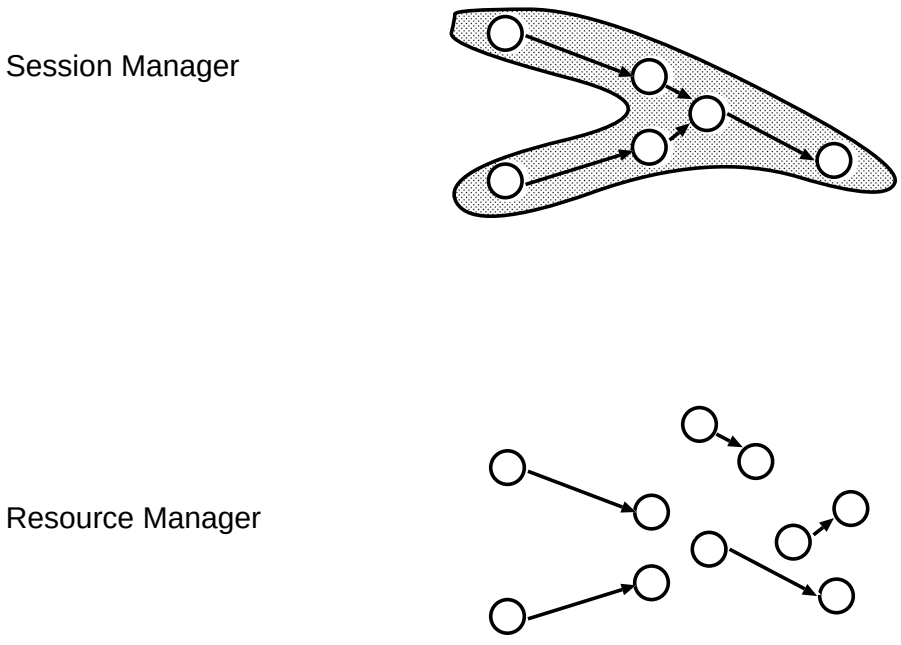



図 7.2: セッションマネージャとリソースマネージャの管理対象

# 第 8 章

# システムの設計及び実装

本研究では、提案するシステムの設計を SDL(Specification and Description Language) を用いて行い、これに準じた形で簡単な実装を行った。本章では、設計したシステムの概観、及び実装したシステムについて述べる。

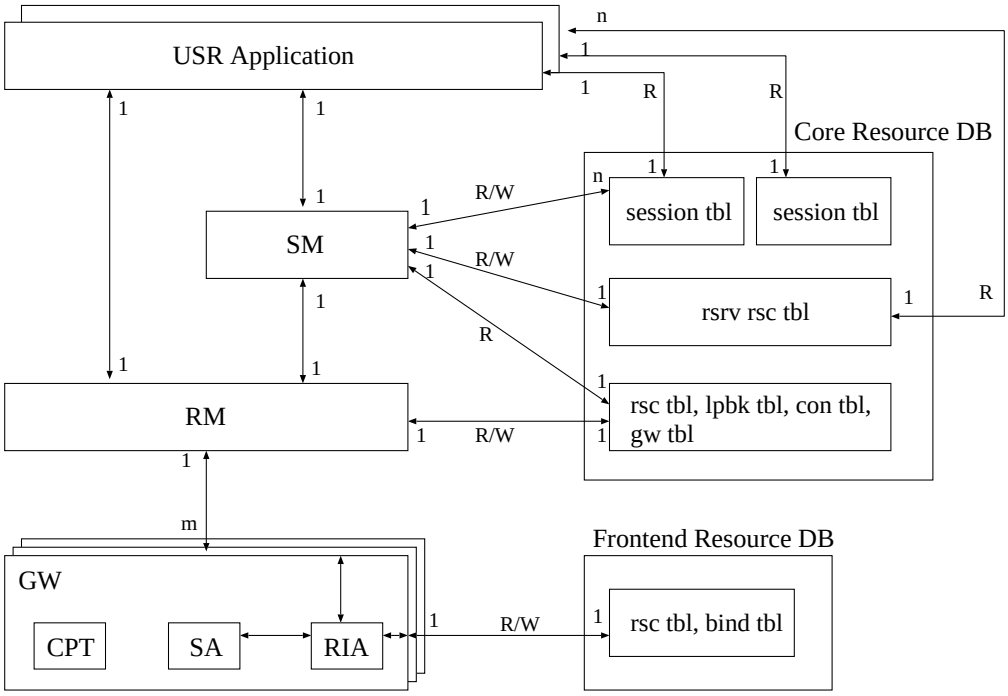
## 8.1 設計したシステムの概観

設計システムの外観を図 8.1に示す。長方形で描かれているのがシステム中の要素である。図の左側の要素は、システムの中で動作するコンポーネントであり、重ねて描かれているものは、システムの中に複数存在することを示している。図の右側の要素は、システムの中に存在するデータベースである。

矢印は、要素と要素の間で直接関係を持ち、メッセージのやりとりを行うことを示している。矢印上に示されているアルファベットはアクセス制限である。R が読み込み許可、W が書き込み許可、R/W が読み書き許可を示している。矢印上の数字は関係が何対何なのかを示している。例えば、リソースインフォメーションエージェントは、フロントエンドデータベース中のリソースインフォメーションテーブルと１対１に読み書き可能な関係にある。

フロントエンドネットワーク側から説明する。システム内にはフロントエンドネットワークの個数分だけゲートウェイが存在する。ゲートウェイの内部には、コントロールプロトコルトランスレイタ、セッションアダプタ、リソースインフォメーションエージェントが存在している。このうち、リソースインフォメーションエージェントはフロントエンドデータベースを利用して、フロントエンドネットワークの資源を管理する。セッションアダプタは、実際にストリーム機器が接続される点であるため、リソースインフォメー

system overview




SM: Session Manager
RM: Resource Manager
GW: GateWay
CPT: Control Protocol Translator
SA : Session Adapter
RIA : Resource Information Agent

rsrv tbl : reservation information table
session tbl: session table

con tbl : connection information table
rsc tbl : resource information table
lpbk tbl : loopback information table
gw tbl : gateway information table

bind tbl : bind information table

R : Read only
R/W : Read/Write

図 8.1: システム設計の概観

ションエージェントと資源情報及び、接続メッセージのやりとり等を行い、フロントエンドの資源情報を逐一更新していくことになる。

コアネットワーク側では、リソースマネージャとセッションマネージャが一つずつ存在している。それぞれは、コアリソースデータベースを利用して、コアネットワーク上で扱える資源を管理、操作することとなる。コアリソースデータベース内の個々のテーブルは、リソースマネージャとセッションマネージャの間に、アクセス権の違いがある。リソースマネージャは、資源情報の整合性を保つ役割を持ため、他のコンポーネントには資源情報の書き込み権限を与えない。セッションマネージャは、リソースマネージャにより提供されるコアリレーノードの情報を用いてセッション情報を保持するが、セッション情報の操作は行うものの、資源そのものの管理は行わないので、資源情報に対する書き込み権限は必要ない。一方、セッションに関するテーブルは、セッションマネージャのみ読み書き権限を持ち、ユーザアプリケーションには読み込み権限のみ与える。セッションに関する整合性はセッションマネージャが担当しているため、ユーザアプリケーションは、セッションの情報を直接データベースで検索することはできるが、直接操作することはできない。

## 8.2 SDL による設計

本システムは SDL により設計した。システム図、ブロック図、プロセス図の三つの図を用いることにより、システムの仕様を定めることができる。本システムの SDL による設計については、詳細を付録 A として添付した。

## 8.3 実装システムの概観

図8.2に概観を示す。コアネットワークにATM、フロントエンドネットワークにIEEE1394のネットワークをおく。コアネットワークの条件として、広域ネットワークにおいて高速かつQoSを保証すること等があげられるが、これらの条件を満すものとしてATMを用いる。また、本研究では家電をネットワークに直接接続することを積極的に取り入れるため、IEEE1394ネットワークをフロントエンドネットワークに用いる。また、コアネットワークとフロントエンドネットワークをブリッジするターミナルシステム(TS)をセッションアダプタとして用いる。ターミナルシステムは、リソースインフォメーションエージェントからシグナリングメッセージを受け取ると、ATM上でシグナリングを行う。また、IEEE1394のポートにIEEE1394機器が接続されると、リソースインフォメーションエージェントに対して接続された機器に関する情報を通知する。リソースインフォメーションエージェントとターミナルシステムとの間では、CLIP(Classical IP over ATM)技術を用いている。

実際に実装を行ったのはゲートウェイ上のリソースインフォメーションエージェント、コアネットワーク内に存在するリソースマネージャ及びセッションマネージャである。資源管理のためのDBMSとしてPostgreSQLを用いた。各々のコンポーネントはC言語でコーディングし、libpqを用いてDBMSにアクセスする。また、PHPを組み込んだウェブサーバを用いて簡単なユーザインターフェースを作成した。PHPを用いることで、各コンポーネントとのメッセージのやりとり及び、DBMSへのアクセスをウェブブラウザ上でグラフィカルに行うことが可能である。以下、この三つのコンポーネントについて詳細に述べる。(図8.3)

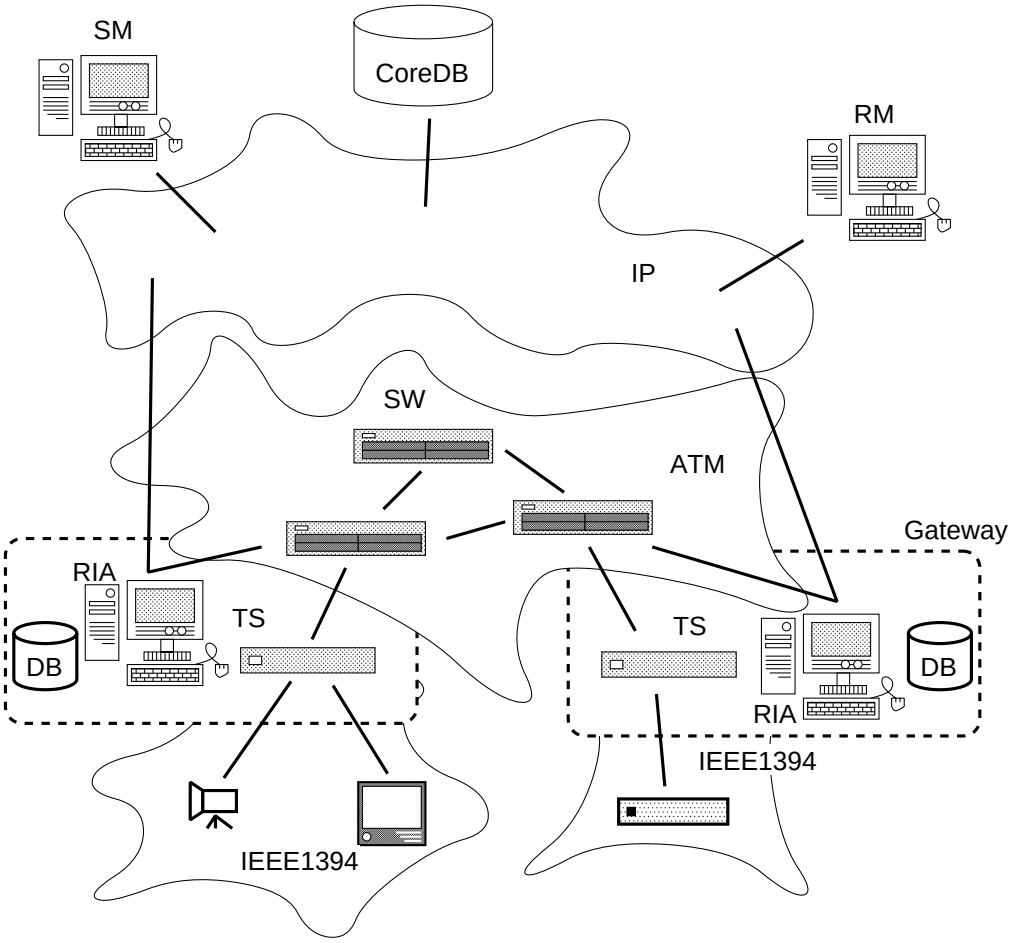

SM
CoreDB
RM
IP
SW
ATM
Gateway
RIA
TS
DB
TS
DB
RIA
IEEE1394
IEEE1394

図 8.2: 実装の概観

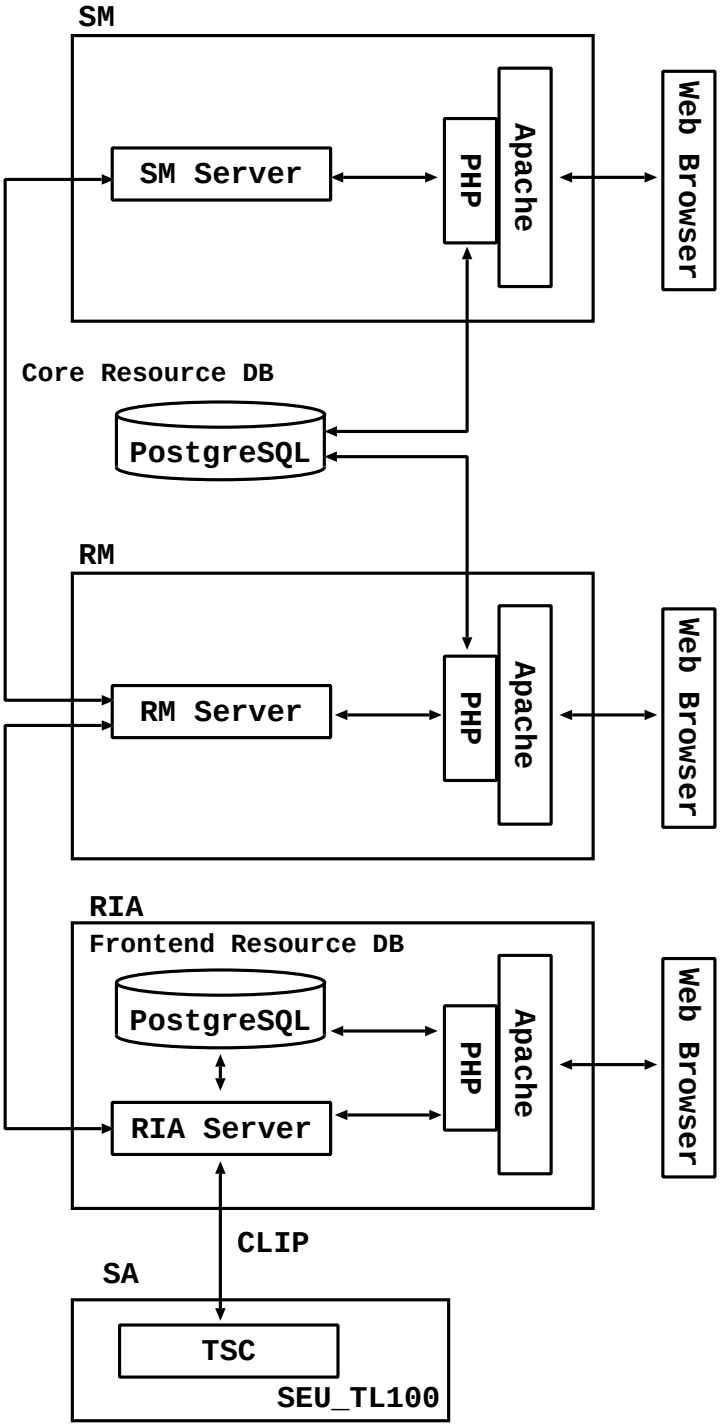



図 8.3: 実装したコンポーネント

## 8.4 リソースインフォメーションエージェント

リソースインフォメーションエージェントの内部は、ターミナルシステム、リソースマネージャ及び他のリソースインフォメーションエージェントとメッセージをやりとりするリソースインフォメーションエージェントサーバと、フロントエンドネットワークの資源情報及びフロントネットワークそのものに関する情報を保持するデータベース、ユーザインターフェースを提供するためのウェブサーバから構成される。

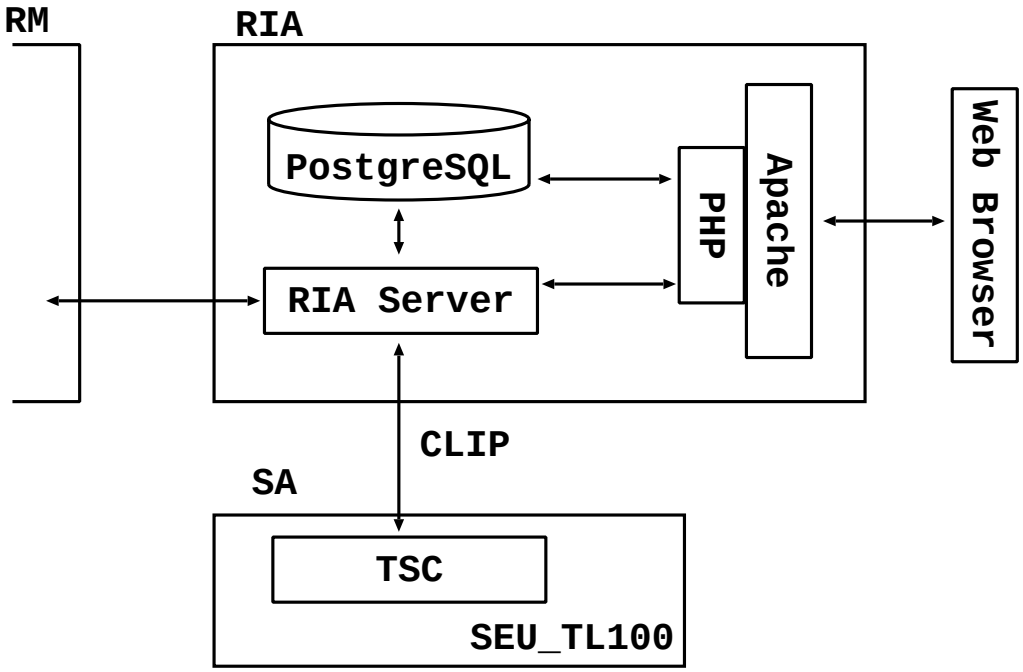



図 8.4: リソースインフォメーションエージェントの実装

リソースインフォメーションエージェントサーバと他のコンポーネントとのメッセージのやりとりのための API は付録 B に詳細が示されている。

### 8.4.1 リソースインフォメーションエージェントの動作

リソースインフォメーションエージェントには大きく分けて三つの動作がある。フロントエンドネットワークに接続される機器の認識及びフロントエンドリレーノードへの変換を経た後、コアネットワークから利用できるように準備する。また、実際のシグナリングに関する動作も行う。以下にこれらについて詳細を述べる。

### フロントエンドリレーノードの登録

ターミナルシステムに機器が物理的に挿抜されると、REGISTERBRIDGE メッセージがリソースインフォメーションエージェントに送られる。

機器が接続された時には、このメッセージ中にエンドターミナルノードの情報及び、ターミナルシステムそのものの情報が含まれており、これをデータベースに格納することで新しい資源及び、フロントエンドネットワークに関する情報をリソースインフォメーションエージェントが保持することができる。

同時に、フロントエンドリレーノード ID を生成し、フロントエンドリレーノードの識別子とする。本実装では、DBMS が自動的にユニークな ID を生成する。

逆に機器が離脱した時には、ターミナルシステムそのものの情報のみが含まれているために離脱した資源を確定することができ必要であればその資源をコアネットワークからデタッチする。

### フロントエンドリレーノードの設定

ここでは、登録されたフロントエンドリレーノードに必要な情報を付加する。付加する情報は以下の三つである。

- 資源の名前

  人間が見てその資源の場所がわかるような名前がふさわしい。例えば、「大講義室正面カメラ」などとすれば、直感的な情報をユーザインターフェースに渡すことが可能となるであろう。オプショナルな設定である。

- 資源の扱えるフォーマット

  エンドターミナルノードの情報から扱えるフォーマットを抽出することが望ましいが、現段階ではビデオ機器が接続される度に設定を行う。必須の設定である。

- ループバック

  ビデオ機器の操作からループバックの情報を得られる方が望ましいが、現段階では、必要に応じてループバックを設定する。ループバックを行う機器にとっては、必須の設定である。

#### コアネットワークへのアタッチ、デタッチ

フロントエンドリレーノードを登録しただけでは、コアネットワーク上からは資源を見ることができない。以下に示すように、まず、ゲートウェイそのものをコアネットワークに登録し、その後、そのゲートェイが管理しているフロントエンドネットワークのリレーノードとして、フロントエンドリレーノードを登録する。登録することをアタッチ、離脱することをデタッチと呼ぶこととする。

- ゲートウェイのアタッチ、デタッチ

  リソースマネージャに対して、ゲートウェイそのものをアタッチ及びデタッチするよう要求する。これにより、該当ゲートウェイのリソースインフォメーションエージェントが管理するフロントエンドネットワークをコアネットワークにアタッチ及び、デタッチすることが可能となる。アタッチの時にはリソースマネージャによりゲートェイ ID が与えられる。

- フロントエンドリレーノードのアタッチ、デタッチ

  ゲートェイがアタッチされている状態下で、フロントエンドリレーノードをアタッチし、コアネットワーク内で当資源を利用することを可能とする。

#### ターミナルシステムへのシグナリング命令

リソースマネージャから CONNECT 及び DISCONNECT メッセージを受け取ると、該当するターミナルシステムに対してシグナリング動作をするよう命令する。これにより、実際に接続を確立、解放することが可能となる。

### 8.4.2 リソースインフォメーションエージェントのデータベース

リソースインフォメーションエージェントはフロントエンドネットワークの資源情報を保持している。以降で説明する情報は、ゲートウェイをコアネットワークのアタッチやデタッチの際にやりとりされる。また、エンドターミナルノードがフロントエンドネットワークに登録されると自動的にデータベースの内容が更新されて、フロントエンドリレーノードに変換され、データベース内で保持される。

#### フロントエンドリソースインフォメーションテーブル

ターミナルシステムからのエンドターミナルノードの接続の通知を受けるとリレーノード ID を生成し、エンドターミナルノードの情報を元にデータの流れる方向を保持する。

コアネットワークに入力される資源を「IN」、フロントエンドネットワークへ出力する資源を「OUT」とする。資源の詳細情報及びアイコンイメージの URL は後述するリソーススペシフィックインフォメーションテーブルを参照することにより、生成する。また、上記の設定により、フォーマット、資源の名前が保持される。さらに、コアネットワークにアタッチしている状態の時は利用許可を「PUBLIC」とし、デタッチしている状態の時は「PRIVATE」とする。利用許可のエントリは将来的には、より細かい制御を可能とすると考えられる。

| rnid | direction | format | prmssn | rsc_name |
|---|---|---|---|---|
| リレーノード ID | データの流れる方向 | フォーマット | 利用許可 | 資源の名前 |

| rsc_spec_info | icon_image |
|---|---|
| 資源の詳細情報 | アイコンイメージのある URL |

表 8.1: フロントエンドリソースインフォメーションテーブル

### ループバックインフォメーション

上記のように設定されたループバック情報を保持する。データの流れる方向性から考えると、必ず出力資源から入力資源へのループバックとなる。

| outrnid | inrnid |
|---|---|
| 出力資源のリレーノード ID | 入力資源のリレーノード ID |

表 8.2: ループバックインフォメーションテーブル

### フロントエンドバスインフォメーション

このインフォメーションテーブルはフロントエンドネットワークの種類によって異なる。本実装では IEEE1394 ネットワーク及びターミナルシステムとして LINKUNIT を用いているため、以下のようなテーブルとなる。

| bus_name | bus_atm | bridge_name | bridge_ip | cid |
|---|---|---|---|---|
| バスの名前 | バスの ATM アドレス | ブリッジの名前 | ブリッジの IP アドレス | コネクション ID |

表 8.3: フロントエンドバスインフォメーションテーブル

### フロントエンドバインドインフォメーション

フロントエンドネットワークの種類により、フロントエンドバスインフォメーションのような各々のネットワークに特有の情報を保持する必要が生じるが、コアネットワークには、このような差異を隠蔽する。そのために、フロントエンドネットワーク特有の情報をコアネットワークで利用できる情報に変換する必要がある。本インフォーメーションテーブルは、コアネットワークとフロントエンドネットワークの情報を結び付ける上で重要である。

| rnid | nuid | bridge_name | bus_name |
|---|---|---|---|
| リレーノード ID | ノードユニーク ID | ブリッジの名前 | バスの名前 |

表 8.4: フロンエンドバインドインフォメーションテーブル

### リソーススペシフィックインフォメーション

本実装では、IEEE1394 を用いているため接続される機器には、機器特有の NUID (ノードユニーク ID) が与えられている。NUID は上位 48 ビットがベンダーの ID として登録されており、下位に関しては規定がなくベンダーによりまちまちである。本研究では、全て同一ベンダーの機器を利用するため、製品の特定が可能である。この情報はフロントエンドリソースインフォメーションで用いられる。

| nuid_spec | vend_name | prod_name | comment | icom_image |
|---|---|---|---|---|
| nuid の上位 48 ビット | ベンダーの名前 | 製品の名前 | コメント | アイコンイメージのある URL |

表 8.5: リソーススペシフィックインフォメーションテーブル

## 8.5　リソースマネージャ

リソースマネージャはコアネットワーク内に位置し、アタッチしているゲートェイ及び資源を管理する。また、セッションマネージャからのメッセージにより必要な接続メッセージをリソースインフォメーションエージェントへ発行する。

リソースマネージャの内部は、リソースインフォメーションエージェント及びセッションマネージャとメッセージをやりとりするリソースマネージャサーバと、コアネットワークの情報を保持するデータベース、ユーザインターフェースを提供するためのウェブサーバから構成される。

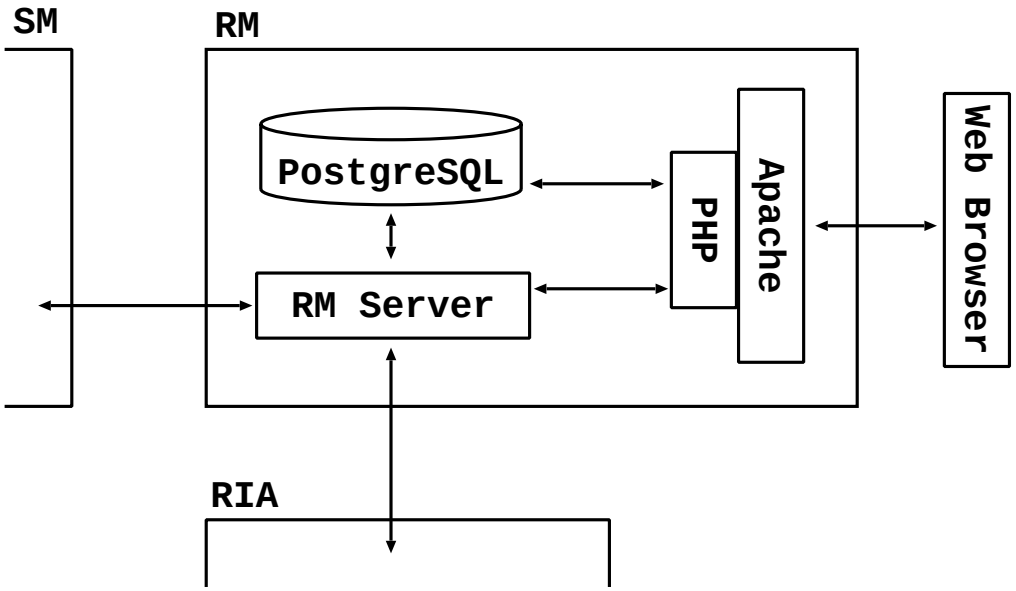



図 8.5: リソースマネージャの実装

リソースマネージャサーバと他のコンポーネントとのメッセージのやりとりのための API は付録 B に詳細が示されている。

### 8.5.1　リソースマネージャの動作

#### コアネットワークの受付

リソースインフォメーションエージェントからのアタッチ及びデタッチメッセージを受け付けることにより、ゲートェイ及び資源をデータベースに保持する。重複するゲートェイ及び資源が無いように管理し、コアネットワーク上での資源の一意性を保証する。

#### 接続管理

コアネットワークを介する接続を管理する。セッションマネージャからの接続メッセージを受けると該当するリソースインフォメーションエージェントに対して資源の接続を促すメッセージを送信する。確立した接続を管理することにより、突然の資源のデタッチがあった場合、セッションマネージャに資源のデタッチを通知する。

### 8.5.2 リソースマネージャのデータベース

リソースマネージャはコアネットワークのゲートェイ情報及び資源情報を保持している。以降で説明する情報は、リソースインフォメーションエージェントからアタッチメッセージを受けると保持され、デタッチメッセージを受けると抹消される。

#### コアリソースインフォメーションテーブル

リソースインフォメーションエージェントから資源をアタッチするメッセージを受信するとコアネットワークで利用できる資源としてデータベースに保持する。

| gid | rnid | direction | format | rsc_name |
|---|---|---|---|---|
| ゲートウェイ ID | リレーノード ID | データの流れる方向 | フォーマット | 資源の名前 |

| rsc_spec_info | rsc_lock | icon_image |
|---|---|---|
| 資源の詳細情報 | 資源がロックされているかどうか | アイコンイメージのある URL |

表 8.6: コアリソースインフォメーションテーブルテーブル

#### コアコネクションインフォメーションテーブル

接続メッセージを受信し接続の確立をリソースインフォメーションエージェントから確認すると、該当する接続情報を保持する。また、フロントエンドネットワーク内のループバックに関しても一つの接続と見なし管理する。

#### ゲートウェイインフォメーションテーブル

ゲートウェイのアタッチ要求を受けるとデータベースの機能を用いて ID を生成し該当ゲートェイの情報を保持する。

| srcgid | srcrnid | srcdir | src format |
|---|---|---|---|
| 送信側ゲートウェイ ID | 送信側リレーノード ID | 送信側データ方向 | 送信側フォーマット |

| dstgid | dstrnid | dstdir | dstformat |
|---|---|---|---|
| 受信側ゲートウェイ ID | 受信側リレーノード ID | 受信側データ方向 | 受信側フォーマット |

表 8.7: コアコネクションインフォメーションテーブル

| gid | name | ip | port |
|---|---|---|---|
| ゲートウェイ ID | ゲートウェイの名前 | ゲートウェイの IP アドレス | RIA のポート番号 |

表 8.8: ゲートウェイインフォメーションテーブル

## 8.6 セッションマネージャ

セッションマネージャはコアネットワーク内に位置し、ユーザからの要求に対してセッションの生成、セッション構成の変更を行う。また、リソースマネージャから資源状態の変化により資源が利用できなくなったことを通知されると、他の適した資源を検索しサブセッションの再構成によりセッションを正常状態へ回復し、管理しているセッション構成の変更を自動的に行う。現段階では、セッション構成を変更する機能は実装できていない。

セッションマネージャの内部は、ユーザアプリケーション及びリソースマネージャとのメッセージをやりとりするセッションマネージャサーバと、セッション情報を保持するデータベース、ユーザインターフェースを提供するためのウェブサーバから構成されている。

### 8.6.1 セッションマネージャの動作

#### セッション生成

ユーザアプリケーションは生成するセッションに組み込みたい全ての接続を、セッションマネージャに通知する。セッションマネージャは、受信した全ての複数の接続からセッション行列を生成する。セッション行列を生成することにより、複雑なセッションでもその構成をトポロジカルに管理することが可能である。セッション行列生成後、セッション

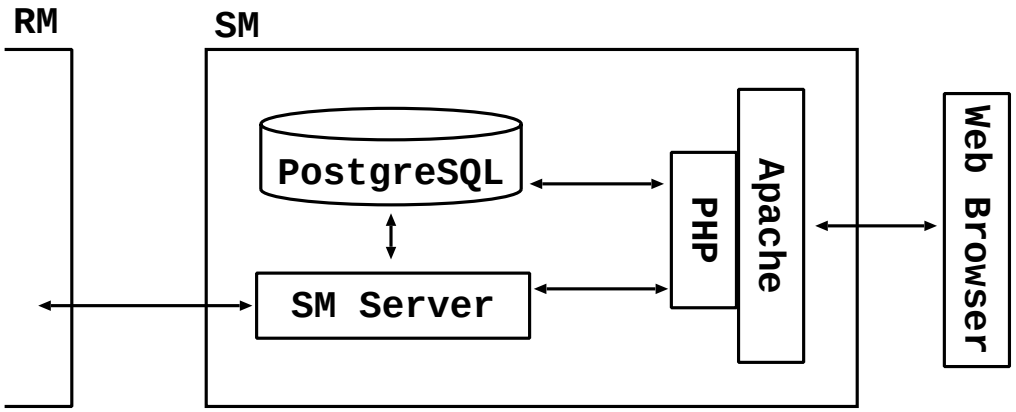



図 8.6: セッションマネージャの実装

からサブセッションを抽出する。得られたサブセッションは、分岐のない一方向の接続であるリストとして管理される。次に、サブセッション中の全ての接続に関してフォーマット変換の必要があるかどうかのチェックを行う。これは、コアリレーノードのフォーマットのエントリを比較することにより実現する。もし、フォーマット変換が必要であれば、コアリソースから適したフォーマット変換を行う資源を検索し、伝送データがフォーマット資源を通過するような接続をサブセッションに挿入する。すべてのサブセッションで自動的に必要なフォーマット変換資源を挿入することができたら、更新されたサブセッションの通りにリソースマネージャに対して接続メッセージを発行する。リソースマネージャから接続の確立の確認通知を受けると、データベースにサブセッション情報を保持する。

### 8.6.2 セッションマネージャのデータベース

#### セッションインフォメーションテーブル

セッションは表 8.9の様に管理される。セッション ID はデータベースが生成する。

#### サブセッションインフォメーションテーブル

サブセッションは表??の様に管理される。サブセッション ID はデータベースが生成する。

| sid | sname | usrname |
|---|---|---|
| セッション ID | セッションの名前 | ユーザの名前（セッションの管理者 ) |

| level | matrix |
|---|---|
| 予約レベル (1〜5) | セッション行列の表示 |

表 8.9: セッションインフォメーションテーブル

| sid | ssid | subsession |
|---|---|---|
| セッション ID | サブセッション ID | サブセッションの表示 |

表 8.10: サブセッションインフォメーションテーブル

## 8.7 動作検証

今回は、リソースマネージャとリソースインフォメーションエージェント及び、限定された形でセッションマネージャを実装した。セッションマネージャは、セッションを生成するときに、自動的にフォーマット変換資源を検索し、接続する機能を持っている。ただし、資源の状態変化による資源の交換機能や、資源を予約する機能等は未実装である。

検証を行った形態を図 8.7に示す。

検証には、二台の DV カメラ及び二台のメディアコンバータを利用した。二台のカメラのうち一台はモニタ用に用いた。今回はフォーマット変換資源としては、メディアコンバータを二台繋げて、DV30Mbps を受信して DV20Mbps として送信するトランスコーダと見立てて検証を行った。また、図のように、カメラを DV30Mbps、モニタを DV20Mbps 用とした。

本検証では、カメラからモニタへ直接接続するようなセッションを生成するように、ユーザアプリケーションに対して要求を出した。

実際には、フォーマットを検査した結果として、トランスコーダが必要なことを自動的に検知してネットワーク内から適した資源を検索し、トランスコーダと見立てた資源を発見して自動的にセッションに組み込まれる。結果的に、ユーザはカメラからモニタへの接続を命令しただけで、フォーマットの差異を気にすることなく接続を行うことが可能であった。

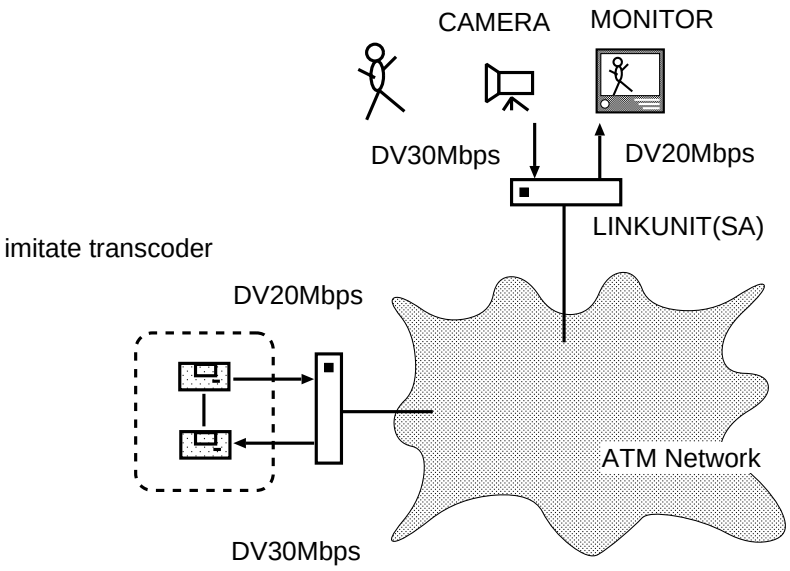

CAMERA
MONITOR
DV30Mbps
DV20Mbps
LINKUNIT(SA)
imitate transcoder
DV20Mbps
ATM Network
DV30Mbps

図 8.7: 動作検証

# 第 9 章

# 性能評価に関する議論

本章では、本研究で提案しているシステムに関する評価について議論する。本提案システムでの、ポイントをおさらいすると以下の項目があげられる。

- セッション管理のしやすさ
- セッションの信頼性
- システムの効率性
- システムの拡張性

これらについて以下で述べ、今後の課題としてするべきシステムの性能評価の指針を示すこととする。

## 9.1 セッション管理のしやすさ

本システムでは、セッションという概念を取り入れているために、複数の資源を複雑に接続することを必要とするアプリケーションを利用するユーザに対する負担を、低く抑えることを考慮している。

### 9.1.1 フォーマット変換

システムの実装の結果、フォーマット変換を考慮する必要なくセッションを構築できるという機能は強力であることがわかった。従来必要であった、必要な資源を検索するという作業及び検索した資源を適当に接続するという作業が不要になるためである。

今、末端の受信資源（モニタ）がネットワーク上に $N$ 個存在し、全ての資源が何らかのフォーマット変換を必要と仮定する。従来は、全ての資源を基本接続として手動で接続する必要があったので、一つの資源を接続するために、

- 必要な資源の検索
- 二つの基本接続の設定

を行う必要があった。この例では、$N$ 回の資源検索及び $2N$ 回の基本接続設定が必要であった。

しかし、本システムが稼働し、十分な数の資源がネットワーク内に存在すれば、上記の作業は一切必要ない。

### 9.1.2　セッションの一括操作

セッション単位で、資源を管理しているため、従来は基本接続ごとに必要であったような操作を、セッションに対する一括操作によって行うことができるようになる。

ここでは例として、確立したセッションを完全に解放することを考える。小規模のセッションであれば、全ての基本接続を一つずつ解放していく負担はそれほど大きくないが、大規模になればなるほどセッションの解放は非常に大きな負担となることが考えられる。

## 9.2　セッションの信頼性

本システムの資源予約、資源状態の変化による資源の交換の機能により、セッションの信頼性を高く保つことが可能となる。

ここでは、全平行修理と単一修理の場合に分けて、稼働率を求めた。まず、共通する前提を表 9.1に示す。

システム内に、$N$ 個の同一機能を持った資源が存在し、$N_s$ 個のサブセッションがこの機能を持った資源を必要としていると仮定する。この仮定のもとで、定常状態になった時にシステム中の全てのサブセッションが稼働できる確率を求める。故障は故障率 $\lambda$ で発生し、修理は修復率 $\mu$ で行われる。故障はポアソン到着、修理は指数分布に従うと仮定する。

| 変数 | 説明 |
|---|---|
| $N$ | システム内に存在する同一機能の資源数 |
| $N_s$ | サブセッションの個数 |
| $\lambda$ | 故障率 |
| $\mu$ | 修復率 |
| $\rho$ | $\lambda/\mu$ |
| $p_i$ | 状態 $i$ の確率 |

表 9.1: 変数

## 全平行修理

全平行修理では、故障が発生した全ての資源に対して平行して修理が行われる。この場合ケンドールの記法では、$M/M/N/N$ となる。

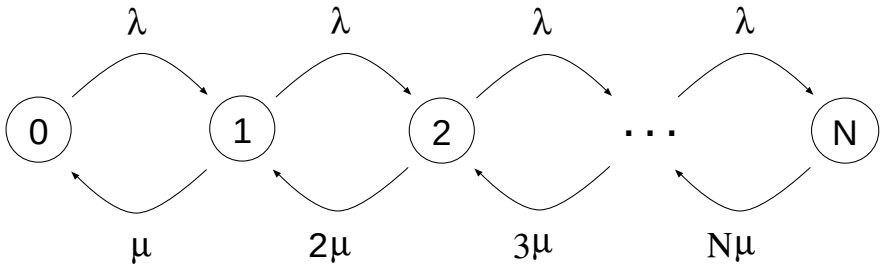



図 9.1: 全平行修理

システム中の全てのサブセッションが、稼働するためには同時に故障する資源が $N-N_s$ 以下でなければならない。従って、状態 $0$ から状態 $N-N_s$ までの間は、全てのサブセッションが稼働することが可能である。この議論より稼働率 $A$ は、

$$\rho = \frac{\lambda}{\mu} \tag{9.1}$$

$$p_0 = \left[1+\rho+\frac{\rho^2}{2}+\ldots+\frac{\rho^N}{N!}\right]^{-1} \tag{9.2}$$

$$= \left[\sum_{i=0}^{N}\frac{\rho^i}{i!}\right]^{-1} (0 \le n \le N) \tag{9.3}$$

$$p_n = p_0 \rho^n \frac{1}{n!} \tag{9.4}$$

$$A(\infty) = \sum_{i=0}^{N-N_s} P_i \tag{9.5}$$

となる。

## 単一修理

単一修理では、故障の発生に対して一つずつ修理が行われる。この場合ケンドールの記法では、$M/M/1/S$ となる。

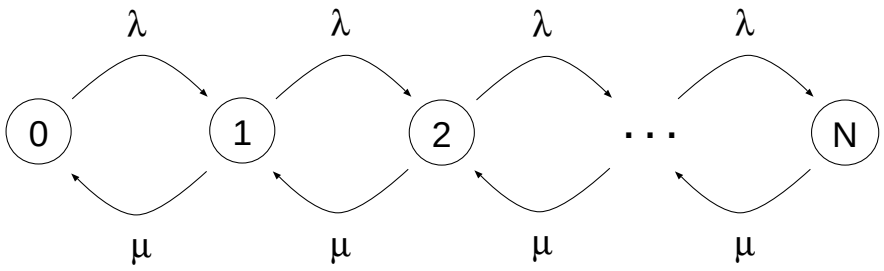



図 9.2: 単一修理

全平行修理と同様の議論により稼働率 $A$ は、

$$\rho = \frac{\lambda}{\mu} \tag{9.6}$$

$$p_0 = \frac{1-\rho}{1-\rho^{N+1}} \tag{9.7}$$

$$p_n = \rho^n p_0 \tag{9.8}$$

$$A(\infty) = \sum_{i=0}^{N-N_s} p_i \tag{9.9}$$

$$= p_0 \frac{1-\rho^{(N-N_s+1)}}{1-\rho} \tag{9.10}$$

$$= \frac{1-\rho^{(N-N_s+1)}}{1-\rho^{N+1}} \tag{9.11}$$

となる。

## 9.3　システムの効率性

本ビデオネットワークシステム上では、様々なアプリケーションが同時に複数のセッションを稼働させることが考えられる。よって、システムが健全に稼働し続けるためには、全体として高い効率が求められる。

システムの効率性としては、システム内の資源利用率を考えることができる。これは、システムが稼働している期間にどれくらいの割合で資源が実際に利用されているかという尺度である。

本システムでは、ネットワーク上にある資源が中間資源として自動的に検索されて利用される仕組みが設けられている。このため、実際にはネットワーク内に所望の資源が存在しても、これを検索することができずに利用を断念するという事態は発生しない。よって、適した資源がネットワーク内に一つでも存在すれば、必ず利用されるので資源利用率は高く保つことができると考えられる。

今後、こういった資源の利用率に関して定量的に評価を行うべきである。

## 9.4　システムの拡張性

本システムでは、異なるフロントエンドネットワークが現われても、新たにコアネットワークへアタッチするためのゲートウェイを作成することにより、ビデオネットワークに参加することができる。これは、コアネットワークがハブの様な役割を果たしているためである。従って、拡張性は高いと考えられる。ただし、アタッチする資源の数が増加すると、コアネットワークでの資源管理に必要な性能にはより高いものが求められることとなってしまう。コアネットワーク上の資源を管理するデータベースも肥大化してしまうため、なんらかの仕掛けを設けない限り、システムの規模には限界があるものと考えられる。

つまり、資源を管理しなければいけないことにより、量的な拡張性には限界があるものの、多様性の拡張に追随する能力は高いと考えられる。今後、なんらかの定量的な評価を行うべきである。

# 第 10 章

# 今後の課題

## 10.1　複数のリソースマネージャ

現段階では、ビデオネットワークシステムを一つの閉じた空間内で扱ってきた。リソースマネージャがネットワーク上に出現すると、そのリソースマネージャにフロントエンドネットワークが、アタッチすることによりビデオネットワークシステムが構成される。つまり、コアネットワークはフロントエンドネットワークにアタッチされることではじめて、その管理機能を発揮できる受動的なものであると言える。管理しているコアネットワークに、十分なだけの資源がアタッチされている場合は問題はないが、アプリケーションが大規模になれば、必要な資源の数も増加すると考えられる。

そこで、ネットワーク上に存在する複数のリソースマネージャがメッセージを交換することにより、他のリソースマネージャが管理する資源を利用できるようなモデルを考えることができる。この結果として、利用できる資源の数が飛躍的に増加し、これまで閉じた世界で運用してきたビデオネットワークをインタービデオネットワークとして利用することができると考えられる。

複数のリソースマネージャが連携して資源情報の交換及び資源の貸借を行うための形態に、いくつかのモデルが考えられる。

- 集中管理型

  リソースマネージャを集中的に管理するコンポーネントをネットワーク上に高々一つだけ配置し、他のリソースマネージャと連携をとりたいリソースマネージャはこの集中管理コンポーネントにアタッチするという構成をとる。

  全ての資源を集中的に管理できるため、各リソースマネージャと集中管理コンポー

ネントの持つ資源情報にだけ整合性が保たれることを保証できれば、ネットワーク内の資源の整合性は比較的簡単に保つことができる。また、資源利用の最適化機能も設けやすくなると考えられる。

しかし、ネットワーク内の全ての利用可能な資源情報が集中することから、集中管理コンポーネントにはかなり高い性能及び信頼性が要求される。

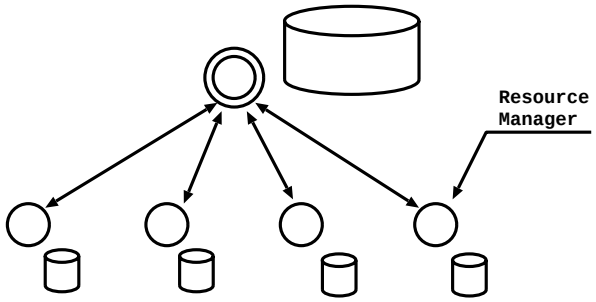



図 10.1: 集中管理型

- 完全分散型

  複数のリソースマネージャがピア・ツー・ピアで直接通信することで、必要な資源を発見する。これにより、ネットワーク上に特別な性能を持つコンポーネントを配置する必要がなくなる。局所的なリソースマネージャのダウンが全体に波及することはないので、ある程度の信頼性を得ることができる。

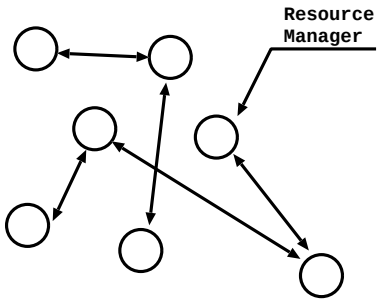



図 10.2: 完全分散型

- 従属接続型

リソースマネージャが、ゲートウェイに成り済ます方法である。つまり、上位のリソースマネージャからはフロントエンドネットワークがアタッチされているように見えるが、実際は、コアネットワークがアタッチされるということになる。

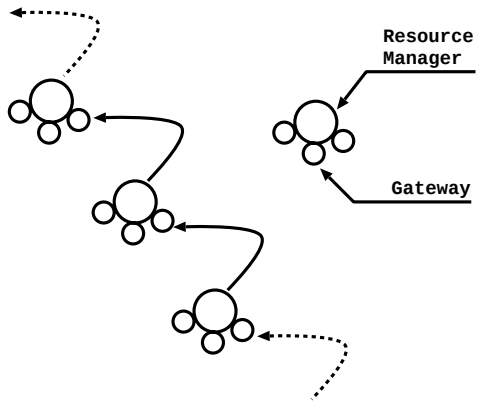



図 10.3: 従属接続型

## 10.2 資源の記述

近年、XML[5] を用いることにより様々なデータを記述しようという議論が活発になってきている。例えば、文書のために Dublin Core[6] という資源の記述のための定義が決定されいる。また、MPEG7[7] は動画像の記述のための定義のセットである。このような、データのためのデータはメタデータと呼ばれ、画像や文書といった資源の検索等に利用されることが見込まれている。

さて、本研究ではストリーム機器を資源として捉えているが、ビデオネットワークシステムが大規模になり、よりインテリジェントな機能を付加させるという局面に立った場合、本提案システムの資源記述は貧弱であると言える。

例えば、UPnP[4] は、TCP/IP ベースのホームネットワーク向けのプロトコル仕様であるが、この中で、デバイスは XML で記述されており、各デバイスはこの XML データを HTTP 経由で交換し、通信を行う。

一方、資源の機能を重視してそのサービスを扱うものとして、Jini[3] がある。

また、本研究ではリレーショナルデータベースにより資源を保持したが、より柔軟な機能を追加するためには、オブジェクト指向データベースを導入することも検討するべきであろう。資源の性質は非常に多様であるため、柔軟性の高い情報格納機構を利用すること

が望ましい。

こういった、資源情報の管理を行うことにより、ネットワークを介した資源の制御をも可能とすることができると考えられる。今後、ストリーム機器の情報管理に適した記述システムを検討することが重要である。

# 第 11 章

# おわりに

ビデオネットワークを利用するためのシステムの枠組みを提案した。本研究では、まず既存のビデオネットワークを眺めた上で、その問題点を明らかにした。

近年、ネットワーク、計算機の性能が飛躍的に向上したことで、高品質のビデオフォーマットを用いた様々な形態の遠隔コミュニケーションシステムを構築することが可能となってきている。マルチメディアデータを伝送するためのネットワークとして、電話系、計算機系、家電系等のそれぞれ異なった背景を元に発達してきたものが扱われている。従来、こういったネットワーク上に、用途や経済性を重視して適した遠隔コミュニケーションのためのシステムを構築してきた。このため、現存するシステムはそれぞれの用途に特化したものとなっている傾向が強い。これではビデオネットワークとしては、限定された利用しか見込むことができない。

また、近年はビデオフォーマットが多種に及んでいるため、フォーマットの差異を吸収するなんらかのシステムが必要である。さらに、ネットワークを介してビデオデータをやりとりすることができる端末は、今後多種類に及ぶと見込まれているため、ビデオネットワークシステムはいっそうシステム固有のものとなってしまう恐れがある。

上記の問題点をふまえた上で、利用しやすいビデオネットワークシステムを構築するために必要な要素について、異種ネットワーク間相互運用性及び資源管理の面から検討した。そこで、こういった問題を解決するためのベースとなるシステムとして、本研究室で提案している VIA を紹介した。

その上で、ビデオネットワークシステムの利用をさらに有意義なものとするため、セッションという概念を提案した。セッションは、複雑な接続の意味のあるまとまりとしての単位である。これを管理することにより、遠隔コミュニケーションのためのシステムを容易に利用することが可能となるだけでなく、システムの信頼性を高めることもできるよう

になる。

本稿では、セッションのさらなる可能性を探るため、さらに高度な機能として、資源の予約機構、資源の交換機構に関して考察を加えた。また、セッションを実際に実現するためにいくつかの手法を検討した。

本研究では、VIA の上位にセッションを管理する層を配置し、システム全体を二階層とすることにより、システムを実現するための設計が可能とした。本研究では、SDL を用いてシステム全体の設計を行い、大部分をプロトタイプとして実装した。また、動作検証を行いシステムの有用性を確認した。さらに、こういったシステムの性能の評価に関して今後、考えるべき事項について考察した。

# 参考文献


[1] 中田潤也, 丹康雄. 異種ビデオネットワーク間接続に関する研究. 情報処理学会, 第 61 回 (平成 12 年後期) 全国大会, pp. 1J-02, 2000.

[2] 中田潤也. 異種ビデオネットワーク間接続に関する研究. 修士論文 September 2000.

[3] Sun Microsystems. JINI NETWORK TECHNOLOGY. http://www.sun.com/jini/

[4] Microsoft. Universal Plug and Play. http://www.upnp.org/

[5] W3C. Extensible Markup Language(XML). http://www.w3.org/XML

[6] Dublin Core Metadata Initiative. http://dublincore.org/index.shtml

[7] MPEG7. http://www.darmstadt.gmd.de/mobile/MPEG7/

[8] WIDE project. DV Stream on IEEE1394 Encapsulated into IP. http://www.sfc.wide.ad.jp/DVTS/

[9] Akihiro Nakao, Larry Peterson, and Andy Bavier. Constructing End-to-End Paths for Playing Media Objects. The fourth IEEE Conference on Open Architectures and Network Programming, April 2001.

[10] Ninja Project, Computer Science Division, University of California, Berkeley. Ninja Project. http://ninja.cs.berkeley.edu/

[11] S. Chandrasekaran, S. Madden, and M. Ionescu. Ninja Paths: An Architecture for Composing Services Over Wide Area Networks. http://ninja.cs.berkeley.edu/pubs.html.

[12] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, David L. Tennenhouse. The VuNet Desk Area Network: Architecture, Implementation, and Experience. MIT Laboratory for Computer Science Cambridge, MA 02139

[13] Stuart Wray, Tim Glauert & Andy Hopper. The Medusa Applications Environment.

[14] Andy Oram. PEER-TO-PEER. O'REILLY.

[15] generic media. http://www.genericmedia.com/

[16] Yasuo Tan. Scaling up IEEE1394 DV network to an enterprise video LAN with ATM technology. In Digest of technical papers of IEEE International Conference on Consumer Electronics 1999, 1999.

[17] Yasuo Tan. Scalable digital video network with IEEE1394 and ATM. In International Distributed Conference 1999, 1999.

[18] Yasuo Tan, Takashi Nomura, Hirofumi Tamori and Kouji Koshiba. Plug and Play Campus Digital Video Network with IEEE1394 and ATM. In Proceedings of the International Conference on Computer Communication 1999, 1999.

[19] SONY. ATM-IEEE1394 Link Unit.
http://www.sony.co.jp/sd/products/Professional/LINKUNIT/index.html

[20] 倉岡貴志, 丹康雄. マルチメディアネットワークサービスの家電機器による利用法に関する一手法. 情報処理学会 第 59 回（平成 11 年後期）全国大会, pp.4C-02, 1999.

[21] Ferenc Velina, Dieter Hogrefe, Amardeo Sarma 共著, 若原 恭, 長谷川 晴朗 共訳. コンピュータ通信シリーズ 5, プロトコル仕様作成の最新形式記述技法, 仕様記述言語 SDL. カットシステム, 1996.

[22] Kang G, Shin and Seungjae Han, The University of Michigan. Fast Low-Cost Failure Recovery for Reliable Real-Time Multimedia Communication. IEEE Network, November/December 1998, p56-63.

[23] 岡田博美. 電子・情報工学講座 16 情報ネットワーク. 培風館.

# 謝辞

本研究をまとめるにあたり、終始熱心なご指導を賜りました丹康雄助教授に心から深く感謝致します。また、常日頃から多くのご協力を頂いた丹研究室の皆様には心から御礼申し上げます。

また、北陸での生活をより有意義なものとしてくださった友や関係した全ての人々に御礼を申し上げます。最後に、様々な面で私を支えてくださった家族に感謝します。

# 付録 A
# SDL による設計

## SDL のマクロ

```
/*************************************************************

 Macro DataTypeDef
 System JaistVideoLAN
 System JaistVideoLAN の SDL 図の中で用いられる新しい型の宣言

*************************************************************/


/* ID */
syntype ID = Integer constants 0:10000; endsyntype;

/* Relay Node ID (24bit)*/
syntype RNid = Integer constants 1:8388608; endsyntype;

/* Gateway ID (8bit)*/
syntype Gid = Integer constants 1:256; endsyntype;

/* IPV4 Address */
syntype IpAddr = Charstring; endsyntype;
```

```
/* Port number (16bit)*/
syntype Port = Integer constants 0:65535; endsyntype;

/* ATM Address */
syntype AtmAddr = Charstring; endsyntype;

/* Session ID */
syntype Sid = ID; endsyntype;

/* SubSession ID */
syntype Ssid = ID; endsyntype;


/***********************************************
 RM が提供する情報
 要するに、RM の DB テーブルの仕様。
*/


/* Crid
 コアリソース識別
 gid  : gateway ID
 rnid : relaynode ID
*/
newtype Crid struct
    gid     Gid;
    rnid    RNid;
endnewtype Crid;


/* Dir
 データの流れる方向
```

```
*/
newtype Dir
    literal in, out;
    operator value: Dir -> Integer;
endnewtype Lock;

/* Lock
 ロックされているか否か
*/
newtype Lock
    literal sm, user, free;
    operator value: Lock -> Integer;
endnewtype Lock;

/* C_Rsc  Core Resource
 Core Resource Information table
 コアネットワークに見える資源情報
*/

newtype C_Rsc struct
    rsc_id           Crid;
    direction        Dir;
    format           Integer;
    rsc_name         Charstring;
    rsc_spec_info    Charstring;
    rsc_lock         Lock;
    icon_image       Octetstring;
endnewtype C_Rsc;

/* Loopback
  Loopback Information table
  存在するループバックコネクション
*/
```

```
newtype Loopback struct
     gid         Gid;
     in_rnid     RNid;
     out_rnid    RNid;
endnewtype

/* C_Con  Core Connection
 Core Connection Information table
 コアネットワーク上での接続情報
*/
newtype C_Con struct
    src_rsc        Crid;
    dst_rsc        Crid;
    format         Integer;
endnewtype C_Con;

/* Gateway_id
 Gateway Information Table
 ゲートウェイ情報
*/
newtype Gateway struct
    gid         Gid;
    name        Charstring;
    ip_addr     IpV4addr;
    port        Port;
endnewtype Gateway;


/*********************************
 GW が提供する情報
 要するに、RIA の DB テーブルの仕様
*/
```

```
/* Permission
 フロントエンド資源の利用許可
 private : 不許可
 public  : 許可
*/
newtype Permission
    literal private, public;
    operator value: Permission -> Boolean;
endnewtype Permission;

/* F_Rsc Frontend Resource
 Frontend Resource table
 フロントエンド資源情報
*/
newtype F_Rsc struct
    rnid          RNid;
    direction     Dir
    format        Integer;
    prmssn        Permission;
    rsc_name      Charstring;
    rsc_spec_info Charstring;
    icon_image    Octetstring;
endnewtype F_Rsc;

/* F_Bind
  Binding Frontend rnid to spec_nid
  エンドターミナルノード ID をフロントエンドリレーノードに変換する
*/
newtype F_Bind struct
    rnid          RNid;
    spec_nid      Integer;  /* NUID 等 */
    /*
```

```
    フロントエンドネットワークに固有の情報を追加する必要がある

     ex) for LinkUnit( ATM-IEEE1394 bridge )

      bus_name      Charstring;
      bridge_name   Charstring;
    */
endnewtype F_Bind;


/* Fnet_Spec_Info
  Frontend network Specific Information
  フロントエンドネットワーク固有の情報

  ex) IEEE1394 Network を Frontend Network として使うとき
  newtype F_Bus
       bus_name      Charstring;
       bus_atm       AtmAddr;
       bridge_name   Charstring;
       bridge_ip     IpAddr;
       cid           Integer;
  endnewtype F_Bus;

  このような情報が必要

*/

/* Rsc_Spec_Info
  Resource Specific Information
  リソース固有の情報

  ex) IEEE1394 Network を Frontend Network として使うとき
```

```
  newtype Rsc_Spec_Info
        vend_nuid    Charstring;
        vend_name    Charstring;
        prod_name    Charstring;
        comment      Charstring;
        icon_image   Octetstring;
  endnewtype;

  この情報が、F_Rsc!rsc_spec_info に記述される

*/


/*********************************
 SM が使うデータ構造
 DB に入れるものと入れないものがある。
*/


/* Rsrvlevel */
syntype Rsrvlevel = Integer constants 0:5; endsyntype;

/* Reserve
 Resource Reservation Information table
 予約資源
*/
newtype Reserve struct
    session_id      Sid;
    subsession_id   Ssid;
    connection_id   Cid;
    replace_target  Cid;
    level           Rsrvlevel;
endnewtype Reserve;
```

```
/* Cidset  Connection ID set
 コネクション ID の集合
*/
newtype Cidset
    Powerset(Cid)
endnewtype Cidset;

/* Subsession
 Subsession Information table
 サブセッション構造
 サブセッションはコネクション ID の集合を持っている
*/
newtype Subsession
    subsession_id  Ssid;
    connection_ids Cidset;
endnewtype Subsession;

/* Cridset
 コアネットワーク資源の ID の集合
*/
newtype Cridset
    Powerset(Crid)
endnewtype Cridset;

/* Matrix
 マトリックスのジェネレータ
*/

/* Matrix( 行数, 列数, 要素のソート )*/
generator Matrix(type Integer, type Integer, type element)
operators
    /* ある要素のデータを取り出す */
```

```
    value  : Matrix, Integer, Integer -> element;
    /* ある要素にあるソートのデータを格納する */
    str    : Matrix, Integer, Integer, element -> Matrix;
    /* 一行を追加する */
    reduch : Matrix, Integer -> Matrix;
    /* 一列を追加する */
    reducv : Matrix, Integer -> Matrix;
    /* 一行を追加する */
    enlarh : Matrix, Integer -> Matrix;
    /* 一列を追加する */
    enlarv : Matrix, Integer -> Matrix;
endgenerator;

/* Contype */
newtype Contype
literals  Rsrvl, Unconnect;
operators
     value : Contype -> Integer;
endnewtype Contype;

/* Eletype */
newtype Eletype
literals  Start, End, Null;
operators
     value : Eletype -> Integer;
endnewtype  Eletype;

/* Selement
 セッションマトリックスの要素
*/
newtype Selement struct
    contype   Contype;
    eletype   Eletype;
```

```
endnewtype Selement;

/* Smatrix
 セッションマトリックス
 正方行列
 Matrix ( 行数, 列数, 要素 )
*/

newtype Smatrix
    Matrix( Integer, Integer,  Selement )
endnewtype Smatrix;

/* Session
 Session information table
 セッションの情報
*/
newtype Session struct
    session_id   Sid;
    conset       Cidset;
    matrix       Smatrix;
endnewtype Session;
```

# システム仕様

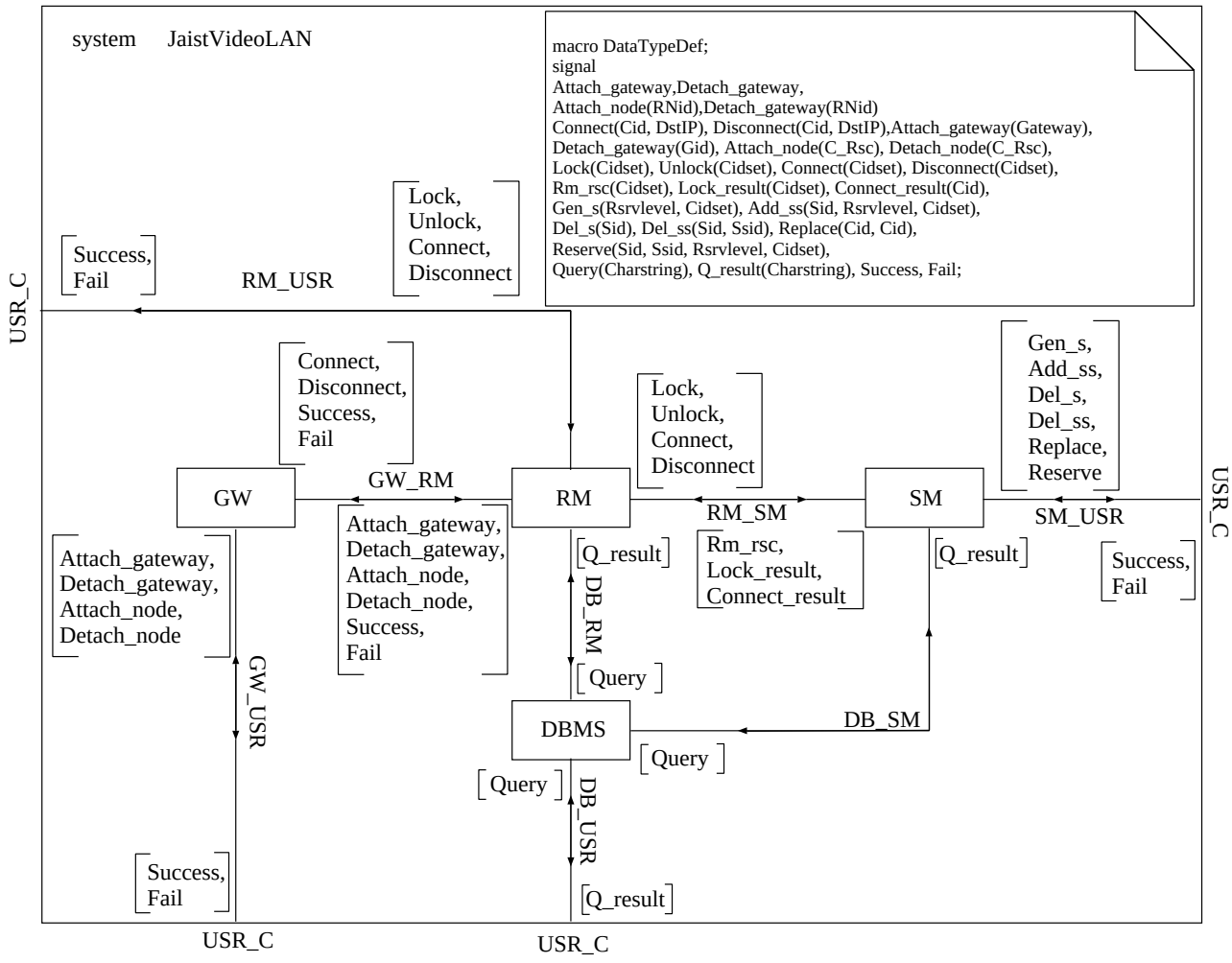



図 11.1: システム図

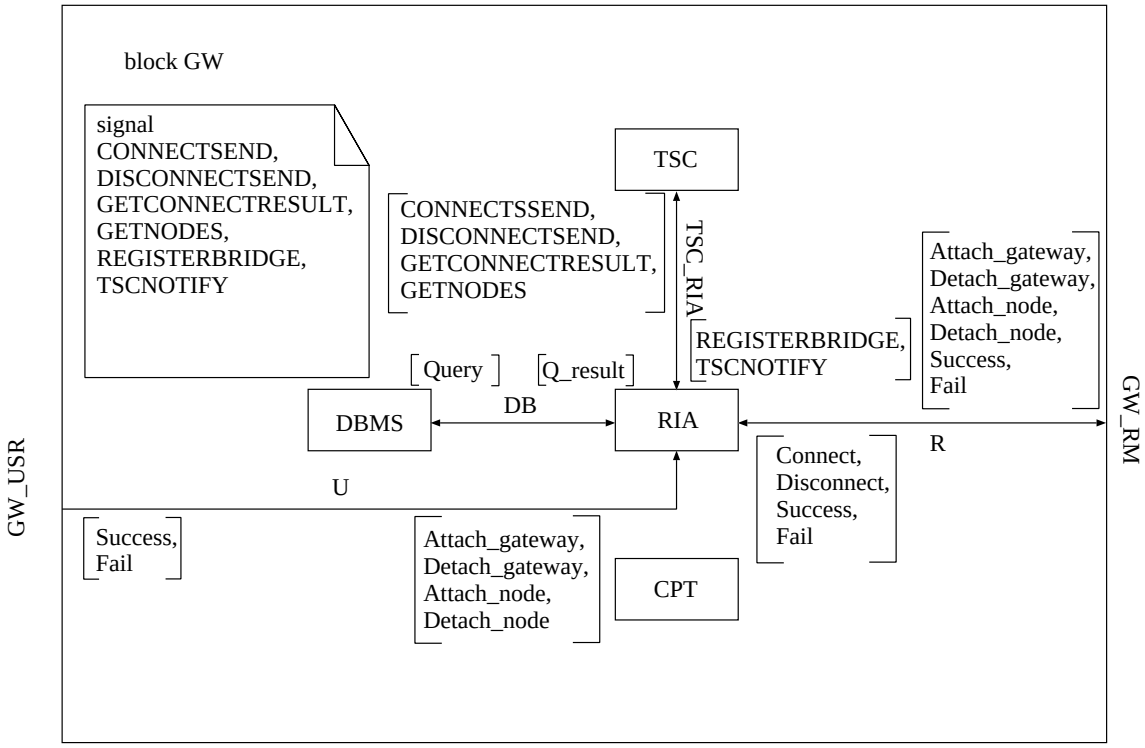

block GW
signal
CONNECTSEND,
DISCONNECTSEND,
GETCONNECTRESULT,
GETNODES,
REGISTERBRIDGE,
TSCNOTIFY
TSC
CONNECTSSEND,
DISCONNECTSEND,
GETCONNECTRESULT,
GETNODES
TSC_RIA
REGISTERBRIDGE,
TSCNOTIFY
Attach_gateway,
Detach_gateway,
Attach_node,
Detach_node,
Success,
Fail
Query
Q_result
DBMS
DB
RIA
R
GW_RM
GW_USR
U
Connect,
Disconnect,
Success,
Fail
Success,
Fail
Attach_gateway,
Detach_gateway,
Attach_node,
Detach_node
CPT

図 11.2: ブロック GW

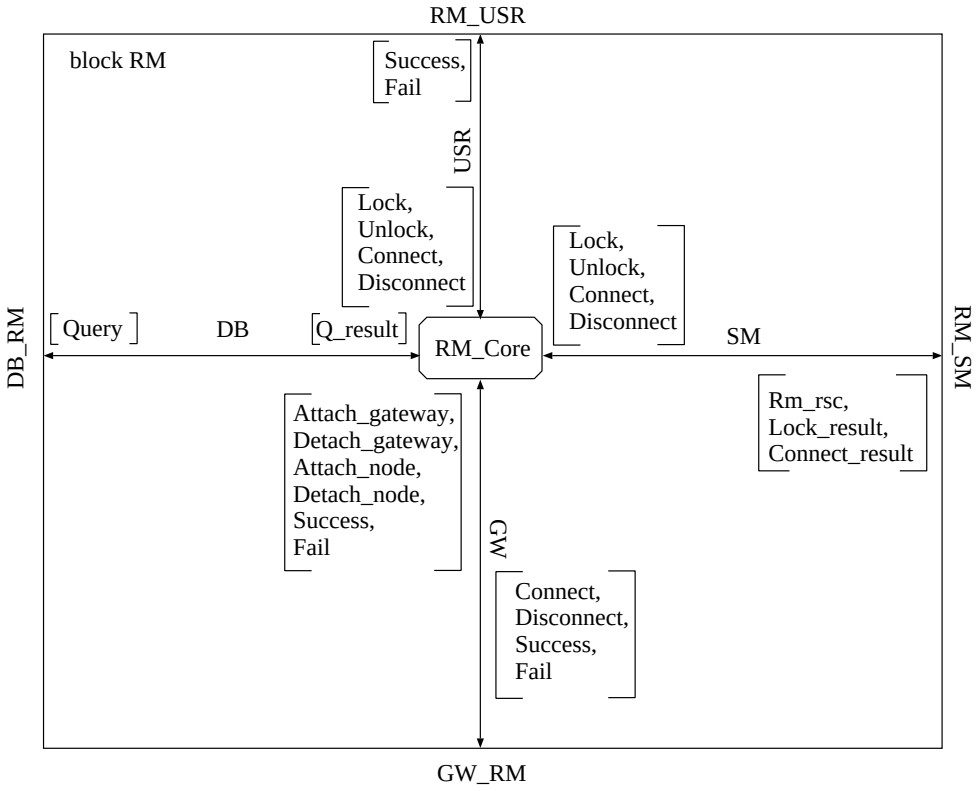

RM_USR
block RM
Success,
Fail
USR
Lock,
Unlock,
Connect,
Disconnect
Lock,
Unlock,
Connect,
Disconnect
DB_RM
Query
DB
Q_result
RM_Core
SM
RM_SM
Rm_rsc,
Lock_result,
Connect_result
Attach_gateway,
Detach_gateway,
Attach_node,
Detach_node,
Success,
Fail
GW
Connect,
Disconnect,
Success,
Fail
GW_RM

図 11.3: ブロック RM

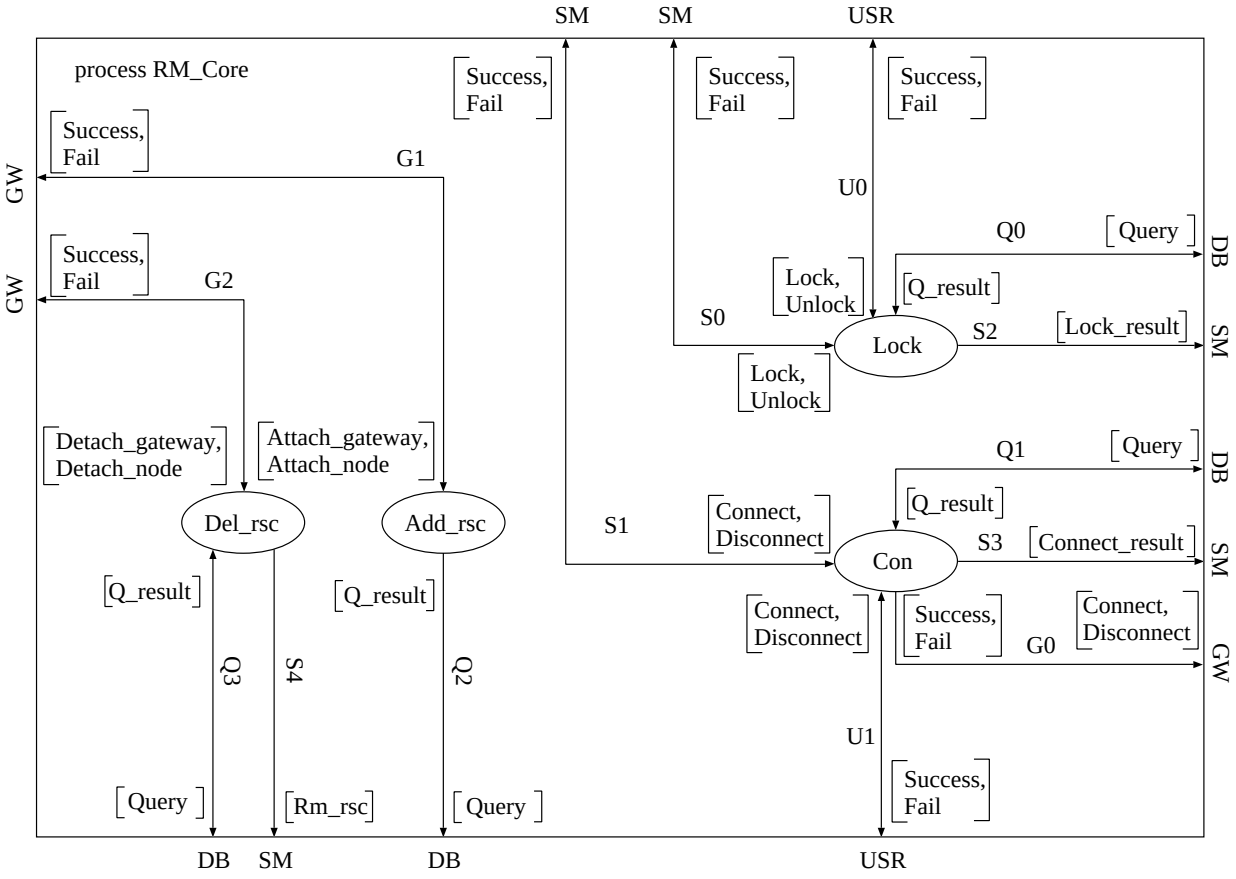



図 11.4: プロセス RM_Core

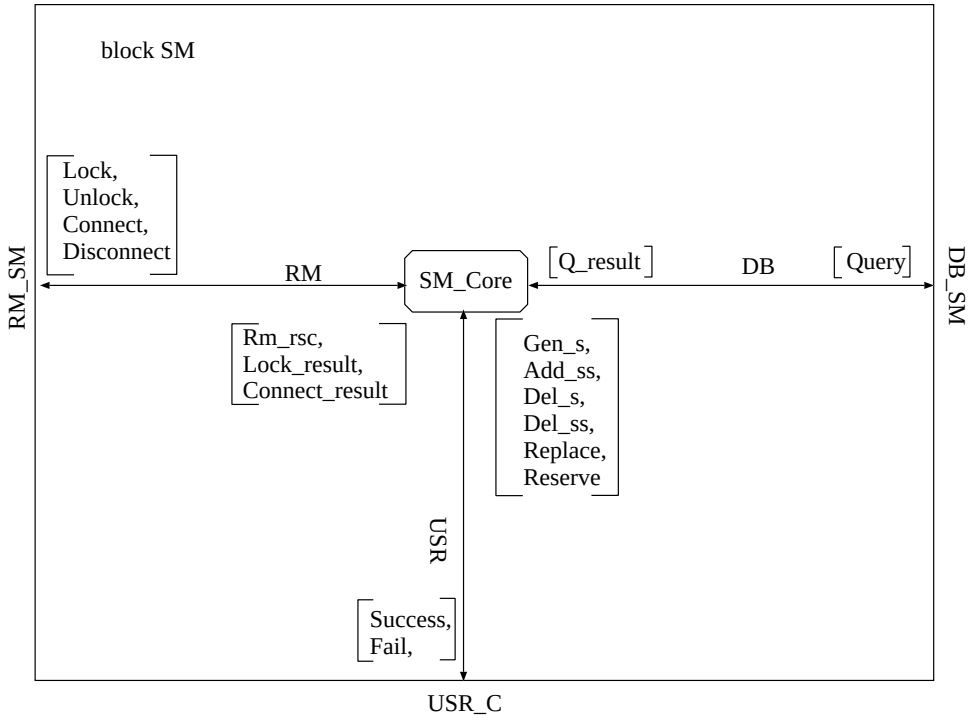

block SM
Lock,
Unlock,
Connect,
Disconnect
RM_SM
RM
SM_Core
Q_result
DB
Query
DB_SM
Rm_rsc,
Lock_result,
Connect_result
Gen_s,
Add_ss,
Del_s,
Del_ss,
Replace,
Reserve
USR
Success,
Fail,
USR_C

図 11.5: ブロック SM

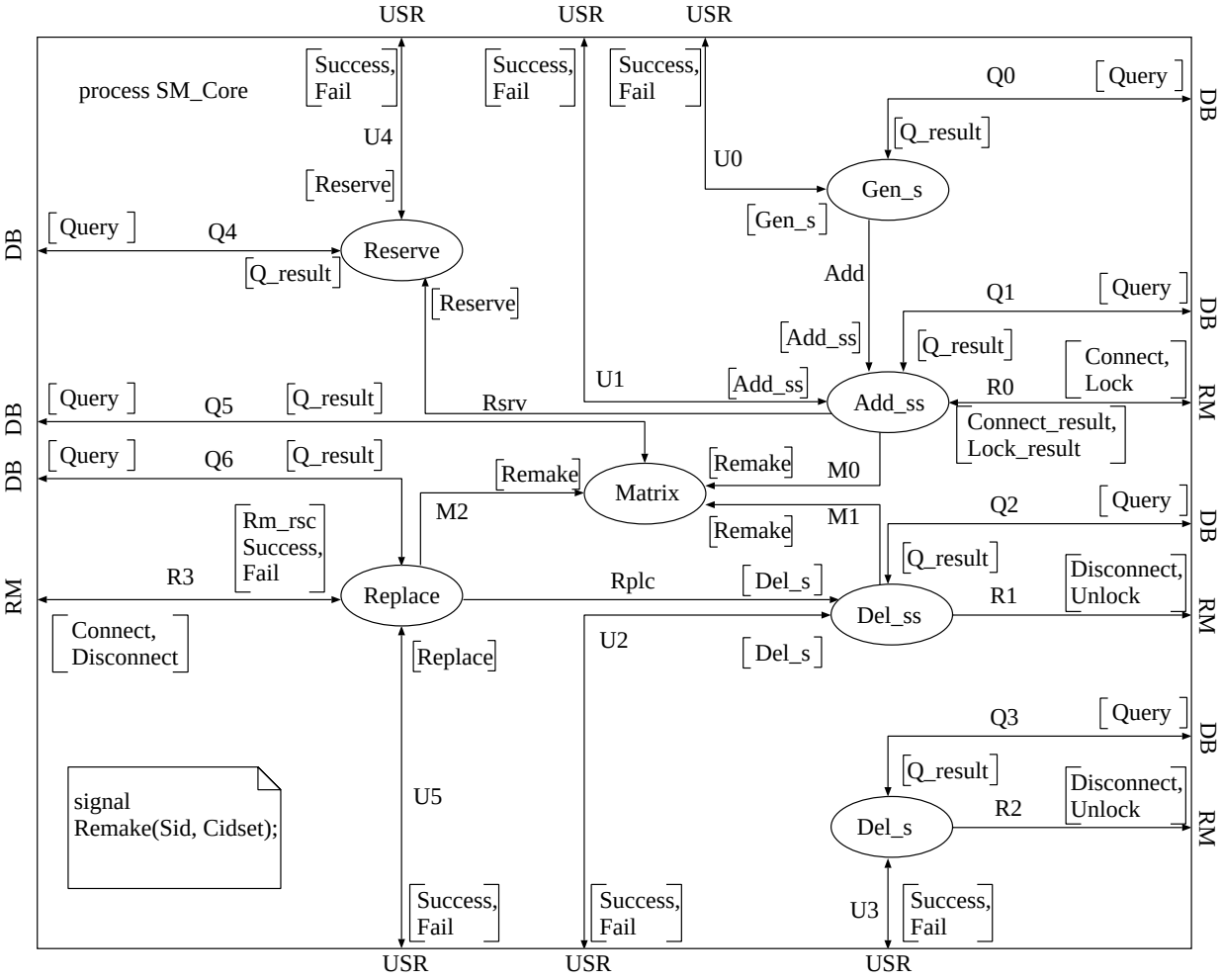



図 11.6: プロセス SM_Core

# 付録 B
# システムの API

```
/**********************************
 *   RIA の RM のための API
 */
```

Connect

資源の確立

書式

```
Connect < Cid > < DstIP >
```

詳細

受け取った Cid から実際のアドレスに変換して、SA にシグナリング命令を送る。

返り値

```
SUCCESS or  FAIL
```

Disconnect

接続の解放

書式

Disconnect < Cid > < DstIP >

詳細

受け取った Cid から実際のアドレスに変換して、SA に接続の解放命令を送る。

返り値

SUCCESS or FAIL

```
/**********************************
 *    RM の RIA のための API
 */
```

Attach_gateway

ゲートウェイのアタッチ

書式

Attach_gateway < Gateway >

詳細

ゲートウェイ情報を引数として受け取り、データベースへの登録を行う。

返り値

GatewayID or FAIL

Detach_gateway

ゲートウェイのデタッチ

書式

Detach_gateway < Gid >

詳細

デタッチするゲートウェイを表す Gid を引数として受け取り、データベースからの抹消を行う。
また、Rm_rsc を発行して、SM に通知する。

返り値

SUCCESS or FAIL

Attach_node

コアリレーノードのアタッチ

書式

Attach_node < C_Rsc >

詳細

コアリレーノード情報を引数として受け取り、データベースへの登録を行う。

返り値

SUCCESS or FAIL

Detach_node

コアリレーノードのデタッチ

書式

Detach_node < Crid >

詳細

デタッチする資源 ID を表す Crid を引数として受け取り、データベースからの抹消を行う。
また、Rm_rsc を発行して、SM に通知する。

返り値

SUCCESS or FAIL

```
/**********************************
 *      RM の SM のための API
 */
```

Lock

資源のロック

書式

Lock < Cidset >

詳細

ロックするコネクション資源 ID を表す Cid の集合である Cidset を引数として受け取り、データベースのリソーステーブル上の lock フィールドを更新する。
ロックに失敗した場合、その Cid を返す

返り値

SUCCESS or CID

Unlock

資源のアンロック

書式

Unlock < Cidset >

詳細

ロックを解除するコネクション資源 ID を表す Cid の集合である Cidset を引数として受け取り、データベースのリソーステーブル上の lock フィールドを更新する。
アンロックに失敗した場合、その Cid を返す

返り値

SUCCESS or Cid

Connect

接続を確立

書式

Connect < Cidset >

詳細

接続するコネクション資源 ID を表す Cid の集合である Cidset を引数として受け取り RIA に対して、Connect 命令を発行する。
成功したら、Core Connection Information table に登録する。
接続に失敗した場合、その Cid を返す。

返り値

SUCCESS or Cid

Disconnect

接続の解放

書式

Disconnect < Cidset >

詳細

接続を解放するコネクション資源 ID を表す Cid の集合である Cidset を引数として受け取り受取り RIA に対して、Disconnect 命令を発行する。成功したら、Core Connection Information table を更新する。
接続の解放に失敗した場合、その Cid を返す。

返り値

SUCCESS or Cid

```
/**********************************
 *     SM の USR のための API
 */
```

Gen_s

セッションの生成

書式

Gen_s  < Rsrv_level > < Cidset >

詳細

予約レベルとコネクション資源 ID を表す Cid の集合である Cidset を引数として受け取り、セッションマトリックスを作成、サブセッションの抽出を行い、Add_ss を呼び出すことで、セッションを作成する。
成功すると、データベースにセッションを登録する。(セッション
マトリックスを登録する)
セッション ID を返す。

返り値

Sid or FAIL

Add_ss

サブセッションの追加

書式

Add_ss  < Sid > < Rsrv_level > < Cidset >

詳細

セッション ID と予約レベルと Cidset を引数として受け取り、サブセッションを作成する。この際、必要な中間資源を検索し、自動的にサブセッションの中に取り込む。この時発見した資源は Reserve を呼び出すことにより予約する。
利用する資源は RM に対して、Lock を発行することによりロックする。
最終的に、RM に対して、Connect 命令を発行することでサブセッションを実現する。
受け取ったセッションに変更を加える場合は、セッションマトリックスを変更する。
サブセッション ID を返す。

返り値

Ssid or Fail

Del_s

セッションの削除

書式

Del_s  < Sid >

詳細

セッション ID を引数として受け取り、セッションを削除する。実際には Disconnect, 及び Unlock 命令を発行する。
また、予約資源を解放する。(予約テーブルを更新する。)
セッションマトリックスを削除する。
サブセッションテーブルを削除する。

返り値

SUCCESS or FAIL

Del_ss

サブセッションの削除

書式

Del_ss  < Sid > < Ssid >

詳細

セッション ID、サブセッション ID を引数として受け取り、セッションを削除する。実際には Disconnect, 及び Unlock 命令を発行する。
また、予約資源を解放する。(予約テーブルを更新する)
セッションマトリックス及び、サブセッションテーブルを更新する。

返り値

SUCCESS or FAIL

Reserve

資源の予約

書式

```
Reserve  < Sid > < Ssid > < Rsrv level > < Cidset >
```

詳細

セッション ID、サブセッション ID、予約レベル、Cidset を引数として受け取り、資源を予約する。
予約テーブルを更新する。

返り値

```
SUCCESS or FAIL
```

Replace

資源の交換

書式

```
Replace  < Cid > [ < Cid > ]
```

詳細

コネクション資源 ID を引数として受け取り、指定があれば（二つ目の引数があれば）その資源に交換する。
指定がなければ、予約テーブルから、利用可能な資源に交換する。
可能な資源がなければ、Del_ss して、サブセッションを削除する。

セッションマトリックス及び、サブセッションテーブルを更新する。

返り値

SUCCESS or FAIL