マルチメディア通信と分散処理 37 － 2
（１９８８．５．２０）

ＯＳＩ　ＣＣＲの実現

中川路　哲男　　宮内　直人　　勝山　光太郎　　水野　忠則

三菱電機㈱　情報電子研究所

我々は、ＯＳＩ（開放型システム間相互接続）の応用層のプロトコルの一つであるＣＣＲ（コミットメント、同時性制御と回復制御）を実装した。実装にあたっては、オブジェクト指向モデルの導入、セションシミュレータの利用などにより、ソフトウェア構成を汎用的なものにするとともに、開発効率の向上を狙った。本稿では、まず実装の基本方針を概説し、次に実装の方式をソフトウェア構成を中心に述べ、考察を加えている。実装において、オブジェクト指向モデルの有効性を確認した。また、標準のいくつかの不完全な点と試験の困難性なども明らかにした。

Implementation of OSI CCR

Tetsuo Nakakawaji　Naoto Miyauchi　Kotaro Katsuyama　Tadanori Mizuno

Information Systems and Electronics Development Laboratory
MITSUBISHI ELECTRIC CORPORATION
5-1-1 OFUNA KAMAKURA-CITY KANAGAWA 247 JAPAN

We have implemented CCR (Commitment, Concurrency and Recovery) which is one of the protocols in the application layer for OSI (Open Systems Interconnection). In developing this software of protocol, we introduced the object oriented model, and we achieved to make the software architecture flexible and portable. In this paper, we describe the implementation strategies at first. Then, we present the method of the implementation mainly about the software architecture. At last, we consider about our method, and found several imcomplete points in the current standard, difficulty in the testing stage, and so on.

1．はじめに

異種システム間相互接続のために、ＩＳＯにおいてＯＳＩ（開放型システム間相互接続）の検討が進められている。ＯＳＩ基本参照モデルに従って、各層のサービス・プロトコルの規格化も順調に行なわれ、各地で実装や接続実験が試みられる段階を迎えている。

応用層のプロトコルは業務の内容に応じてそれぞれ規定されるため、様々な種類のプロトコルが存在し、その標準化の進行度合もまちまちである。その中でも、世の中の通信形態の８割とも言われる需要の大きさに比して標準化が遅れているのは、ＴＰ（トランザクション処理用の通信プロトコル）[1]である。その原因には、高性能の実現が難しいことや、既に各社独自のトランザクションプロトコルを持っておりその親和性で思惑があることなどもあるが、下位プロトコルとして使用を想定しているＣＣＲ（コミットメント、同期及び回復）[2]の規格化が遅れていることも大きな一因である。我々はＣＣＲの実装方法を検討し、実装を行なうことで規格の完成度や実装の可能性などを探ったので、ここに報告する。

2．ＣＣＲの概要

ＣＣＲは応用層プロトコルの１つであり、分散処理全体の処理の同期をとるためのプロトコルである。その意味で、ＦＴＡＭ（ファイル転送、アクセス及び管理）などのように特定業務に直結したものではなく、ＡＣＳＥ（アソシエーション制御）などのように応用層の中で共通的に使用されるプロトコルとして位置づけられる。ＣＣＲを使用している、或は使用する可能性がある応用層プロトコルとしてはＴＰを始め、ＦＴＡＭ、ＪＴＭ（ジョブ転送操作）、ＲＤＡ（遠隔データベースアクセス）などがあり、分散処理を実現するための基本的なプロトコルということができる。

ＣＣＲでは、同期を保って完遂されるべき処理の単位をアトミックアクションと呼び、これを分散された各システム間で一貫して実行するためのプロトコルを規定している。各システムは、アトミックアクショントリーと呼ばれる木構造で関係付けられ、各枝で上位をスーペリア、下位をサブオーディネイトと呼んでいる。さらに、木構造の最上位はマスタと呼ばれ、アトミックアクションを管理して、その完遂（コミット：資源をトランザクション最終の状態にする）と取り消し（ロールバック：資源をトランザクション開始の状態にする）を判断、指示する。ＣＣＲを使用したときの、アトミックアクショントリーの関係を図１に示す。

ＣＣＲは、アトミックアクションの実行制御に、いわゆる２相コミットプロトコルを採用している。第１相でデータの転送などの処理を行なった後に、マスタは、各システムのコミット可否を問い合わせる。第２相では、その結果に従ってコミット又はロールバックを実行する。問い合わせ及び実行指示は、マスタから順次下位に伝達されていく。問い合わせの結果は逆に下位から上位に上がって行く。その中に１つでも拒否があれば、全体としてロールバックとなる。ＣＣＲのサービスプリミティブとその機能を表１に、２相コミットメントの実行例を図２に示す。

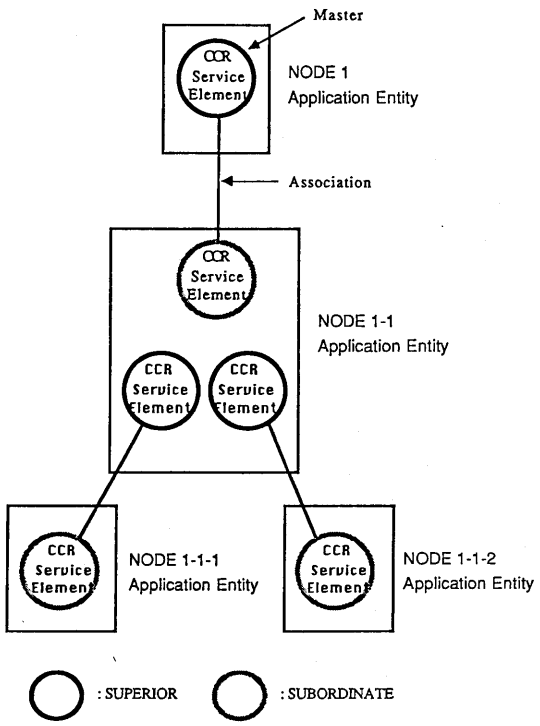
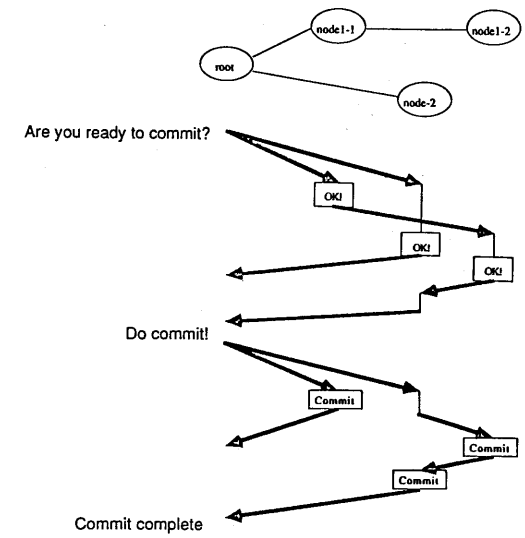


図１　ＣＣＲにおけるアトミックアクショントリー

表1　ＣＣＲサービスプリミティブ

名称	型	方向	内容
C-BEGIN	非確認	SUPERIOR→SUBORDINATE	アトミックアクションの開始
C-PREPARE	非確認	SUPERIOR→SUBORDINATE	コミット可否の問合せ
C-READY	非確認	SUBORDINATE→SUPERIOR	コミット肯定応答
C-REFUSE	非確認	SUBORDINATE→SUPERIOR	コミット否定応答
C-COMMIT	確認	SUPERIOR→SUBORDINATE	コミット完遂要求
C-ROLLBACK	確認	SUPERIOR→SUBORDINATE	コミット取り消し要求
C-RESTART	確認	SUPERIOR→SUBORDINATE SUBORDINATE→SUPERIOR	アトミックアクションの再開

（コミット完遂の場合）



（コミット取り消しの場合）

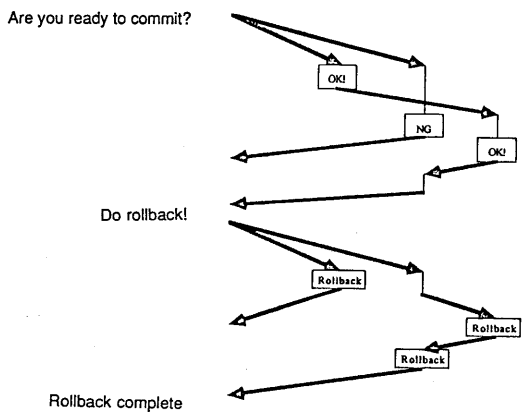


図２　２相コミットメントの実行例

また、ＣＣＲにおける障害時の回復モデルは、全てスーペリアからの指示によって行なわれ、サブオーディネイトは障害後はスーペリアからの指示を待つ。しかし、スーペリア側が回復して、回復指示を発行しない限り、サブオーディネイトはデッドロック状態に陥る。それを回避するために、サブオーディネイト側は障害後一定時間内に回復指示が到着しない場合、ヒューリスティックにコミットかロールバックの判断を行なう権利を持つことが許される。これをヒューリスティックコミットメント機能と呼ぶ。判断内容は、ＣＣＲ関係確立時にスーペリアから指定することもできるし、サブオーディネイトが自律的に決定することもある。

ＣＣＲは分散されたシステム間のプロトコルであるが、基本的にはスーペリアとサブオーディネイトとの１対１の通信として規定されている。即ち、アトミックアクショントリーの途中のシステムに於いては、サブオーディネイトとしてのＣＣＲエンティティと、スーペリアとしてのＣＣＲエンティティが複数存在することになる。その間の複雑な協調動作を規定することは極めて困難であり、この点が通常の１対１のプロトコルと大きく異なる。現規格では、一つのアプリケーションエンティティ内の複数のＣＣＲエンティティで共有するコミットメント変数を定義することによりこれを規定している。

また、いかなる状況に於いても分散システム間でのアトミックアクションの一貫性を保つためには、下位層の通信路切断時にも他のプロトコルのように異常終了することは許されず、回復する手段を提供することが要求される。

以上のようにＣＣＲは他のプロトコルと異なる特性を持つため、規格として正確に完全に規定すること、及びその通りに実装する事が困難であるという特徴を持つ。ＣＣＲの規格化自体は、1984年にＤＰ（規格草案）、翌年にはＤＩＳ（規格案）と順調に進行したが、前述した技術的な困難さや使用する上位のプロトコルがいま一つ具現化しなかったこともあって、２ndＤＩＳ以降進捗が停滞していた。しかし、ＴＰの急速な進行に伴い、最近にわかに注目を浴び、早期規格化が強く望まれている。

3．実装方針

ＣＣＲを実装するに当たり、我々はその実装方法について検討を行ない、以下のような方針で実装を行なった。

(1) 規格が未熟であることを念頭に於いて、現規格の理解と検証を主目的とする。その意味で、記述の厳密でないところ、曖昧なところは、独自の解釈を加えて実装することにした。

(2) ＣＣＲモジュールは複数のアソシエーションに関する処理には関知しないこととする。これは、ＣＣＲの仕様自体があくまで１対１を前提としているためである。例えば、サブオーディネイトからのコミット可否問い合わせ応答を集め、その中に一つでも否定応答があれば否定応答を、全部肯定応答であれば肯定応答をスーペリアに報告するという判断処理は、上位のプロトコルで行なうこととして本ＣＣＲモジュールでは実現しない。

更に、ローカルな資源に対するアクセスはＣＣＲモジュールに含めないこととする。これは前述したように、１つのシステム内に複数のＣＣＲエンティティが存在する可能性があり、上位のプロトコルがこれらの状態を総合的に判断した上で、ローカル資源に対するアクセスが定まるからである。

その意味で、ＣＣＲモジュールには、１本のアソシエーション上の通信プロトコルを実行する単なる順序機械としての機能のみを持たせる。

(3) ＣＣＲエンティティは、スーペリアとして動作する場合とサブオーディネイトとして動作する場合がある。どちらで動作するかは、アトミックアクションコントリーが形成された時点で静的に決定される。また、両者のプロトコル処理には共通のものが少なく、規格の状態遷移表も２つに分けて記述されている。よって、ＣＣＲプロトコルモジュールも、その役割に応じて分けることとした。

(4) コミットメント変数は、一つのアプリケーションエンティティ内の複数のＣＣＲモジュールからアクセスされる変数であり、モジュール間で共有される必要がある。今回はこれをファイルとし、排他制御を行なった。

(5) ヒューリスティックコミットメント機能は、プロトコルマシンへのイベントとして外部から与えられるものとして実現した。即ち、どの様な状況において、どの様な判断をするかといったセマンティクスに関わる部分は、全て上位のアプリケーションプロトコルの担当として、ＣＣＲの処理の範囲外とした。

(6) 下位層としては、プレゼンテーション層のプロトコルモジュールと、セションシミュレータ[3]を用いる。セションシミュレータは、当社で開発した、セション層以下の通信を模擬するツールであり、１台の計算機の上で複数の通信を視覚的に把握することが可能である。ＣＣＲプロトコルでは、大同期などセション層の複雑な機能を利用しているので、その使い方を検証することも容易である。

また、下位プロトコルとしてのＡＣＳＥを扱う部分は、ＣＣＲモジュールでは関知せず、ＣＣＲの利用者であるアプリケーションプロトコルに任せることとする。これは、ＣＣＲではアソシエーションの存在を仮定しており、その回復もそのアプリケーションプロトコルが直接ＡＣＳＥを使用して行なうモデルだからである。但し、コネクションの概念は必要なため、その活性化と非活性化をアプリケーションから指示する必要がある。ここでは、ＣＣＲを応用コンテクストに含むアソシエーションを確立したときに、アプリケーションがＣＣＲモジュールを活性化することにした。

一方、アプリケーションプロトコルのデータは、ＣＣＲモジュールを通過する形で実装する事にした。ＣＣＲでは、本来アプリケーションプロトコルのセマンティクスを解釈しないため、そのデータに対して処理を行なうことはない。しかし、第２相など、アプリケーションがデータを送信してはならない期間があるので、そのチェックを行なう意味も含めて、今回はＣＣＲを通過する方法とした。

以上のことから、今回の実装においては、標準で規定されているサービスプリミティブの他に、ＣＣＲモジュールの活性化と非活性化及びデータ

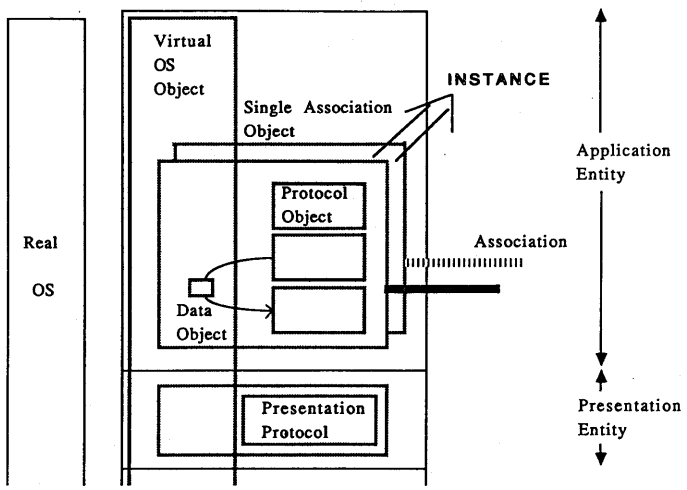


図３　応用層ソフトウェア実装のためのオブジェクトモデル

転送要求・指示というローカルなサービスイベントを定義した。ＣＣＲモジュールの活性化と非活性化及びデータ転送要求は、上位のアプリケーションプロトコルから発行され、データ転送指示は、下位層から報告される。

(7) 応用層におけるソフトウェアの実装方法は、我々が従来から提案している図３のオブジェクトモデルを採用することにする。[4]

このモデルでは、応用層の中に仮想ＯＳオブジェクトと、各アソシエーションに対応する単一アソシエーションオブジェクト及びデータオブジェクトが存在する。更に、単一アソシエーションオブジェクトの中には、ＡＣＳＥ、ＣＣＲなどの各プロトコルを実現するオブジェクトが存在する。各アソシエーション毎のプロトコルモジュールの実体は、それらのインスタンスに相当する。

仮想ＯＳオブジェクトは、応用層モジュールの中で資源を管理する主体である。資源とは、通常のＯＳが管理するファイルなどの資源に加えて、アソシエーションや各プロトコルモジュールも含む。仮想ＯＳオブジェクトは、各プロトコルオブジェクト間の通信、応用層の利用者や下位層との外部的な通信及び実ＯＳの機能を仮想的なインタフェースで提供する。インタフェースを仮想的にすることで、アーキテクチャやＯＳの異なるシステムにも、仮想ＯＳオブジェクトを入れ換えるだけで、プロトコルモジュールはほとんど影響なく移植が可能である。

各プロトコルオブジェクトは、対応するプロトコル処理を行なう。処理は、仮想ＯＳモジュールからのイベントに対し、受動的に反応することで実行される。プロトコル間のデータの送受は、すべて仮想ＯＳの通信機能を用いるため、プロトコル内の処理はプロセス構成などに依存しない。

データオブジェクトは、利用者や、通信相手からの要求に基づくセマンティクスを運ぶ本体である。これが各プロトコルオブジェクトを通過することによって、プロトコルヘッダの付加や離脱という処理を受ける。通信時に実際に授受されるのは、このデータオブジェクトクラスのインスタンスである。

このように、通信時の動的な主体を、インスタンスとして整理することで、応用層のモデルにしたがった実装が可能となる。

(8) ソフトウェアの記述言語としては、我々が通信ソフトウェア用に開発したオブジェクト指向言語superC[5]を採用して、上記オブジェクトモデルに従った実装を狙い、モジュールの独立性向上を図る。さらに、従来ＦＴＡＭを開発したときのモジュール[4]も最大限流用する。

4．実装結果と考察

前述の実装方針に基づき、実装を行なった。その結果を以下に記す。

(1) ソフトウェア構成

ソフトウェアをオブジェクト指向言語で記述したので、そのソフトウェア構成はオブジェクトのクラス構成に相当する。

ＣＣＲのプロトコルを実現するモジュールは、前述のオブジェクトモデルでのプロトコルオブジェクトに相当する。その中は、アソシエーション管理を行なうクラス、プロトコル処理に対応するクラスから構成した。また、データオブジェクトのためのクラスを、各ＰＤＵ(Protocol Data Unit)に対応したクラスから構成した。さらに、コミットメント変数は、複数のＣＣＲモジュールから参照されるため、これを管理するクラスを独立に設けた。

①アソシエーション管理クラス

前述したように、今回の実装ではＣＣＲモジュールの活性化と非活性化というローカルなサービスイベントを定義している。これは上位のアプリケーションプロトコルが確立したアソシエーション上で、ＣＣＲモジュールを活性化したり、非活性化したりする場合に使用される。本クラスでは、上位のアプリケーションプロトコルからの要求に基づき、アソシエーション対応のＣＣＲモジュールの活性化と非活性化を管理する。具体的には、指定されたアソシエーション番号を基に、ＣＣＲプロトコルマシンの処理を実現しているＣＣＲプロトコル処理クラスのインスタンスの生成と消滅を行なう。活性化時には、スーペリアとして活性化するか、サブオーディネイトとして活性化するかも指定する。

②プロトコル処理クラス

１本のＣＣＲ関係上の、状態遷移表に従った処理を行なうクラスである。状態をインスタンス変数として持ち、入力イベントに対する処理を行ない、対応する動作を行なった後、状態を更新する。スーペリア用のクラスとサブオーディネイト用のクラスがある。

入力イベントとしては、上位のアプリケーションプロトコルからのサービス要求、アソシエーション異常解放などの障害通知、コミットメント変数の変更、下位層からのプロトコルデータ着信通知、タイムアウトなどがある。動作としては、ＰＤＵ（プロトコルデータユニット）の作成と発行、コミットメント変数の更新、タイマの起動、アソシエーションの強制解放などがある。アソシエーションの強制解放は、サブオーディネイトとの障害回復が失敗した場合に、更に自エンティティのスーペリアからの障害回復指示を促すために行なう。

③データクラス

ＰＤＵの各種類に対応して、クラスが存在する。クラス内のメソッドとしては、ＰＤＵの生成と消去、各パラメタの値の設定と獲得、符号化と復号化などがある。

④コミットメント変数管理クラス

コミットメント変数を管理し、その更新や獲得を行なう機能をメソッドとして持つ。

各クラスの関係及びそのステップ数を図４に示す。

(2) 標準の不備

状態遷移表を忠実に実装する事を試みたが、遷移先の状態がない場合も含めて、記述の誤りや曖昧な点が散見された。これらの点については、独自の解釈を行なって、一応実装した。今後標準化活動を通じて、記述の完成度を高めていく必要がある。

また、通常のプロトコルでは、予期しないＰＤＵ受信時などはプロトコルエラーとして扱い、上位層と、相手側エンティティに異常報告をして、プロトコルエンティティ自体は異常終了するものが多い。一方ＣＣＲは、障害時にも分散処理システムの一貫性を保つためのプロトコルであり、異常終了することは許されず、プロトコルエラーは各実装任せとしている。しかし、少なくとも上位のアプリケーションプロトコルに異常報告が行なわれるべきであり、相互接続の観点からもプロトコルエラーに関する規定が必要と思われる。

その他、上位のアプリケーションプロトコルに処理が依存する部分があり、ＣＣＲとして独立でない処理に関しても独自の解釈で実装した。ＣＣＲと上位のアプリケーションプロトコル間のサー

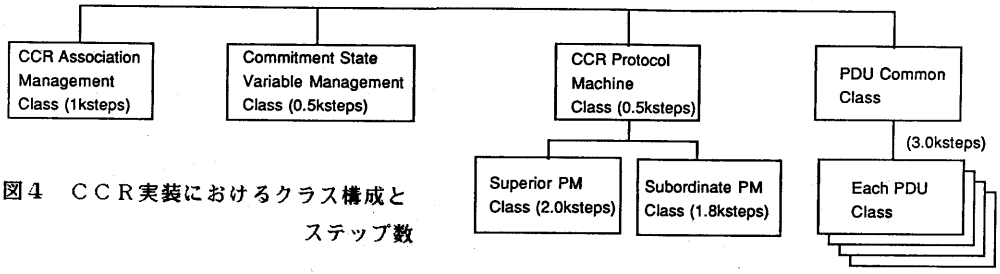


図4　ＣＣＲ実装におけるクラス構成とステップ数

ビス定義をより厳密に規定する必要がある。

(3) ＣＣＲ動作の確認

実装の目的の一つに、ＣＣＲ動作の理解と確認があったため、各イベントに対する処理と状態の変化、コミットメント変数の移り変わりをログとして記録することで、その動作を確認した。また、セションシミュレータを用いることで、下位層の動作も視覚的に確認することが可能であった。セションシミュレータを用いたＣＣＲの動作状況の例を図５に示す。

(4) 下位層マッピング

ＣＣＲプロトコル仕様では、C-BEGIN、C-COMMIT、C-REFUSE、C-RESTARTやC-ROLLBACKなど、ＰＤＵをプレゼンテーション層での確認型サービスプリミティブにマッピングしているものがある。これは処理の同期を取るためには必要であるが、シーケンスによっては冗長と思われるものもあった。

例えば、C-RESTART(ROLLBACK)要求がP-RESYNC要求に、C-RESTART(DONE)応答がP-TYPED-DATAにマッピングされており、P-RESYNC応答は暗黙の内に返信されることとなっている。しかし、C-RESTART(DONE)応答を直接P-RESYNC応答にマッピングしても

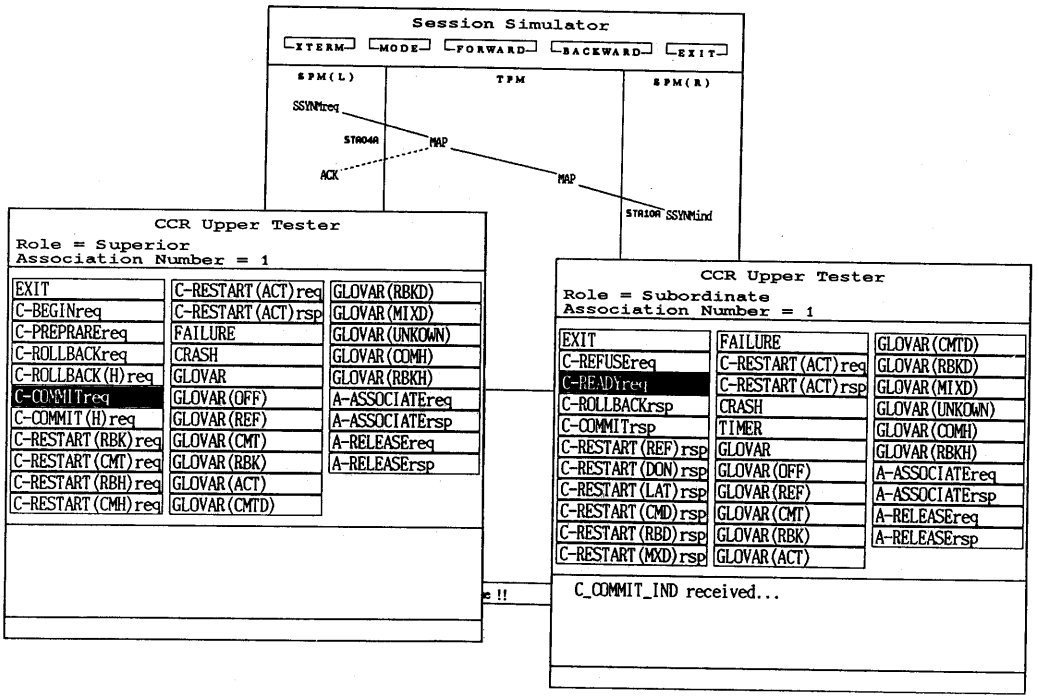


図５　セションシミュレータを用いたＣＣＲの動作確認

特に問題はないと思われる。

時と場合に応じて、転送量が最適になるようなマッピングを考慮する必要がある。

(5) 試験

今回の実装で最も困難であったのは、試験であり、いまなお試験中の状態である。

その原因の一つには、網羅性の問題が挙げられる。ＣＣＲの特徴の一つに、シーケンスが一様でないことがあり、特に回復時には様々なシーケンスが想定され、これらを一様に網羅するシーケンスを作成すること自体が困難である。そのために現在我々は、状態遷移表から試験シーケンスを自動生成するツール[6]を作成中であり、今後それを用いて更に詳細な試験を行なう予定である。また、プログラム自体を異常終了させて、次に２次記憶上の情報からプロトコル処理を再開始する試験も行なう必要があると考えている。

また別の原因としては、複数のアソシエーションに渡る処理の試験の複雑さが挙げられる。一つのアプリケーションエンティティ内の複数のＣＣＲエンティティに対すて、同時に発生するイベントを試験する環境を、実現することが困難だからである。更にその場合の試験シーケンスを導出する事は更に複雑である。これはプロトコルの仕様記述自体が、分散処理において厳密に規定しにくいことにも起因しているのであろう。今後、分散処理システムにおけるプロトコルの形式的な記述方法、試験方法などを検討していく必要がある。当面のＣＣＲの試験方法としては、ＣＣＲを調整・管理するトランザクションなどの上位アプリケーションプロトコルが規定され、その使用方法に応じて試験することで行えるものと考えている。

５．おわりに

本稿では、我々の実装したＣＣＲについて報告した。実装においては、オブジェクトモデルを採用し、オブジェクト指向言語superCを記述言語としたことで、モジュール性の高いソフトウェア構成で実装する事ができた。更に、セションシミュレータのようなツールを用いることで、その動作を視覚的に把握することができ、理解と試験に役立った。

一方、現在の規格に不明な点が多少有り、いくつかの仮定や解釈を設定しなければならなかった。また、特に試験を網羅的に行なうことの困難さから、十分な試験を終えていない状態にある。今後、分散処理システム全体を形式的に記述する方法を見いだしていくことが必要である。

国際的な標準化の動きとしては、１相コミットメント機能の追加要求、複雑なヒューリスティックコミットメント機能の分離、性能を重視した"Presumed Abort"機能の実現などが課題となっている。特に、"Presumed Abort"機能は、ＴＰを検討しているグループからの要求機能であり、有効な機能と思われるので、今後更に検討していく必要がある。

＜参考文献＞

[1] ISO:"Information Processing Systems - Distributed Transaction Processing", ISO/IEC JTC1/SC21 N2606-2608 (1988).

[2] ISO:"Information Processing Systems - Commitment, Concurrency and Recovery", DIS 9804-9805/2 (1987).

[3] 勝山、中川路、宮内、水野：「セションシミュレータによる通信ソフトウェアの開発」，第37回マルチメディアと分散処理研究会 (1988).

[4] 中川路、勝山、水野：「ＯＳＩ　ＦＴＡＭの実現」，第35回マルチメディアと分散処理研究会 (1987).

[5] 勝山、佐藤、中川路、水野：「オブジェクト指向型言語spiceCによる通信ソフトウェアの開発」，第33回マルチメディアと分散処理研究会 (1987).

[6] 佐藤、宗森、水野：「通信プロトコル試験シーケンス生成ツールの設計」，昭和62年前期電子情報通信学会情報・システム部門全国大会 (1987).