マルチメディア通信と分散処理 38 － 2
（１９８８．７．１４）

Broadcast機能を有するRemote Procedure Callの実現

山口　英　　下條真司　　宮原秀夫
大阪大学基礎工学部情報工学科

Remote Procedure Call (RPC) は分散アプリケーションを構築する際に非常に有効である。本研究では同時に複数のサーバに対して処理を依頼することのできるRPCの機構を開発した。このRPCでは並列に処理を依頼できるという機能のほか、異機種計算機間でのRPCの実行を実現し、さらに、通信時の障害に対する信頼性を保証している。本稿では、このシステムの特徴・実現について説明する。

Implementation of Broadcast Remote Procedure Call

Suguru Yamaguchi
Shinji Shimojo
Hideo Miyahara
Dept. of Information and Computer Sciences,
Faculty of Engineering Science,
Osaka University.
1-1 Machikaneyama, Toyonaka, Osaka 560, JAPAN

Remote Procedure Call (RPC) provides an effective method for abstracting inter-process communications on distributed applications. We have constructed an RPC system. It has a facility with which processes can call RPC to more than one server at one time. Moreover, this system works on heterogeneous environment and it provides reliability on communication failures. In this paper, we describe its design and implementation.

1. はじめに

近年、高性能ワークステーションを互いに LAN で結合した分散処理システムの構築が盛んである。それにともない、複数のワークステーションを協調して働かせるためのネットワークアプリケーションの必要性が叫ばれている。我々の研究グループではこれまで幾つかのネットワークアプリケーションを開発してきた。[1-3] これらのアプリケーションはクライアント・サーバモデル[4] をベースにして開発されている。つまり、クライアントはサーバに対して処理要求のメッセージを送り、サーバは処理を実行して結果をクライアントに返すという形でサービスを提供している。このようなアプリケーションにおけるプロセス間通信を扱う方法として、 Remote Procedure Call (RPC) による抽象化が有効であり、多くのシステムで実現されている。我々のネットワークアプリケーションの開発の経験から考えると、 RPC には次のような機能が求められる。

(1-1) 信頼性

サーバで提供するサービスにはファイルアクセスを伴うような副作用のある処理がある。したがって、RPCでは副作用のある処理について計算機の障害や通信誤りが発生しても信頼性を保証して正しく実行できるようにしなければならない。

(1-2) 1対N-RPC

クライアント側では複数のサーバに対して同時に処理要求を送ることがある。このため、クライアント側で1回の RPC 呼び出して、複数のサーバに対して同一の処理要求を送ることができるようにしなければならない (これを1対N-RPCと呼ぶ)。

現在我々が利用できるRPCシステムでは上記のような条件を満足するものはない。そのため我々はこれらの要求を満足するようなRPCシステムを構築した。これを我々はSPLICE/RPC†と呼ぶ。 SPLICE/RPC は以上のような要求を満足し、アプリケーションでのプロセス間通信に関する処理の記述を簡便に行なえるようにし、有効な開発環境を提供することを目的として開発された。 SPLICE/RPC では複数の UNIX‡ ワークステーションが Ethernet によって結合されている環境を対象にしている。このような環境ではアーキテクチャの異なる計算機が接続されており、ネットワークアプリケーションも異なる計算機間での通信を考慮しなければならない。したがって、 SPLICE/RPC ではさらに次のことが要求される。

(1-3) 異機種計算機間通信

RPCを実現するプログラム自体が移植性の高いものでなくてはいけない。そのため、kernel に対して直接組み込むといったOSの直接の変更を行なう形での開発は行なわない。さらに、 RPC で扱われるデータはその意味(解釈)が計算機間で異ならないようにする処理が必要である。

開発言語としては、これまで開発されたネットワークアプリケーションも現在の開発においてもC言語を主に使用しているため、SPLICE/RPC でもC言語によるインタフェースを提供している。

本稿ではSPLICE/RPCの設計とその実現について述べる。

2. RPCにおける問題とSPLICE/RPCでの解決

RPCシステムを構築する場合、その基本的な機能の面で幾つかの選択が考えられる。これらは開発するRPCシステムの性能に密接に関連する。さらに、これらの選択を行なった後に、幾つかの設計上の問題も解決しなければならない。本章ではこれらの問題を簡単に説明し、さらに SPLICE/RPC での選択について説明する。これらの問題については[5-7] 等においても議論されている。

† SPLICE (Supercomputer, Personal workstations and Local area networks InterConnected Environment) は我々が行なっているスーパコンピュータの効果的な利用環境を構築する研究プロジェクトの名前。

SPLICE/RPCはその一つのプロダクトとして開発された。

‡ UNIXはAT&T Bell研究所の登録商標。以下特に示さない限り4.3BSD UNIXを意味する。

2.1. 通信プロトコル

RPCを実現する場合、トランスポートプロトコルとして Virtual Circuit を使用するか Datagram を使用するかという問題がある。信頼性を考えた場合には通信中にパケットが失われた場合の自動再送機能や、通信をしているプロセスが破壊されていないかをチェックする機能を持っているVirtual Circuit が有利である。しかし、その分 Datagram と比較してオーバヘッドが大きい。また、1対1通信のみが実現されており、同報通信の機能は用意されていない。さらに、UNIXの Virtual Circuit では1つのプロセスが使用することのできるリンクの数は制限されている。一方 Datagramでは Virtual Circuit のような信頼性は保証されていないが、オーバヘッドは小さく、また同報通信の機能もあり、比較的自由な通信形態を構成することができる。**SPLICE/RPC**では同時に複数のプロセスと通信する必要があるので、トランスポートプロトコルとしてはDatagramであるUDP/IPを採用した。

2.2. 制御

関数呼び出しの面から見たRPCではBlocking CallとNon-blocking Callがある。Blocking Callでは通常の手続き型言語と同様にRPCによって処理を依頼したクライアントはサーバでの処理結果が返ってくるまで、次の処理を行なわない。一方、Non-blocking Callではクライアントがサーバに処理を依頼すると、クライアントは直ちに次の処理に移り、サーバでの処理の完了を待たない。 Non-blocking Callの場合、サーバでの処理とは独立にクライアント側で処理が進められるために処理効率の面では有利である。しかしながら、サーバの提供している処理が処理結果をクライアントに返す場合、 Non-blocking Callでは、クライアントの処理の実行を進めながら処理結果を受け取る機構を構築する必要があり実現が難しい。このため、**SPLICE/RPC**では Blocking Callを採用した。

2.3. Call Semantics

RPCにおいては一般的にクライアントとサーバは異なる計算機上に存在する。そのため、クライアント・サーバ間の通信やサービスを実行中にどちらかのプロセスが破壊されたり、どちらかの計算機が故障・停止(クラッシュ)したり、通信中にメッセージが失われたりすることがある。サーバがクラッシュした場合は、クライアントはサーバに対してメッセージの再送を行なうことになる。あるいは、サーバが正しくメッセージを受信したにも関わらず、そのメッセージの ACK が通信中に失われ、クライアント側では送信が失敗したと判断し、メッセージの再送を行なうという状況も考えられる。これらの場合、サーバに対して再送によって発生した同じメッセージが複数到着してしまい、サーバでこの同一のメッセージをどのように扱うかが問題となる。このような場合の RPC の処理の仕方を Call Semantics として Lampson は次のようなクラス分けをした。[7,8]

(1) *Exactly Once Semantics*

クライアントがサーバに対してメッセージを送り処理を依頼した、すなわち RPC を行なった場合には、その処理はサーバにおいて丁度1回だけ行なわれることを意味する。これは手続き型言語における関数と同じ Semantics であり、副作用のある関数でも安全に実行することができる。

(2) *At Least Once Semantics*

クライアントがサーバに RPC を行なった場合、その処理は少なくとも1回以上サーバにおいて行なわれることを意味する。すなわち、再送によってサーバに複数の同一の処理要求が到着した場合に、同じ処理を複数回実行することを表す。したがって、副作用のある関数では処理結果に問題が生じることがある。

この2つをメッセージの扱いという面で比較すると *Exactly Once Semantics* では、1回の RPC においてクラッシュのためにサーバに複数のメッセージが到着しても、それらの内ただ1つのメッセージだけが確実にサーバによって受け取

られ処理されることを意味する。そのためサーバでは再送によって発生した複数の同一のメッセージを発見し、必要に応じてメッセージを捨てる処理が必要である。一方、 *At Least Once Semantics* では再送によって複数の同一のメッセージが到着しても、それらを同じように処理すればよい。

SPLICE/RPCではその対象としている処理として副作用のあるサービスも含んでいる。このため call semantics として *Exactly Once Semantics* をサポートできるようにプロトコルを構成した。すなわち、**SPLICE/RPC**では再送によって複数の同一のメッセージが送られても受信側のプロセスでは、そのうちの1つだけが確実に受け取られるような処理を行なっている。

Exactly Once Semantics を採用し、副作用のある関数をサービスとして提供するためには、サーバが処理中にクラッシュした場合でも、矛盾が生じないようにしなければならない。このような要求を完全に満たすには、クラッシュから復帰した後サーバはクラッシュが発生した時点での実行状態から処理を再開する機能を持つ必要がある。このような機能は **SPLICE/RPC** では提供しておらず、RPC を利用するアプリケーションにおいて実現する必要がある。

2.4. 通信の形態

UNIXでは計算機間のプロセス間通信は socket を用いて実現する。 socket における通信ではプロセスはポートを確保し、このポートを介してメッセージを送る。ポートには番号が付けられており、メッセージの送り先は送り先の計算機のネットワークアドレスとポート番号を指定して送る。 RPC のためのプロセス間通信の形態として次のようなものが考えらる。

- 特定のポート番号のついた通信用サーバを用意する。クライアントはメッセージをその通信用サーバに送る。通信用サーバはメッセージの内容を見て適当なサービスの処理を行なうサーバに対してメッセージをフォーワードする。(図1-a)
- 通信用サーバを置かず、実際の処理を行なうサーバに対して特定の番号を持ったポートを割り当て、クライアントはそのポートに直接メッセージを送り、処理を依頼する。(図1-b)

SPLICE/RPCではこの2つの方法の中間の方法を取り、各計算機にポート番号の管理用サーバを置き、 RPC サービスを行なうサーバは立ち上がった時に適当なポートを確保し、そのポート番号を管理用サーバに登録しておく。各計算機上の管理用サーバは現在利用可能なサーバとそのポート番号や、サーバの実行状態などの情報を互いにやりとりしている。クライアントは管理用サーバに希望するサービスを実行するサーバのポート番号を問合せ、それによって得られたポート番号を使用して直接サーバにメッセージを送るようにした。

以後、サービスを提供するサーバが動作しているホストのネットワークアドレスとポートアドレスを合わせて、サーバのアドレスと呼ぶ。

2.5. 名前管理

クライアントが各サーバの提供するサービスを受ける場合、そのサーバをどのように指定するかという問題がある。各サーバが提供するサービスには名前が付けられるのが一般的である。したがって、この名前からサーバのアドレスを得ることができればよい。この名前の管理の方法として各種の方法が提案されている。例えば、 SUN RPC[9] では、全てのサービスは番号で参照され、その番号はSun Microsystems社が一括して管理している。アプリケーション開発者からの要求があると SUN がRPCのサービス番号を割り当てていくという方式がとられている。またしばしば行なわれる方法としては特定の計算機に name server を置き、そこで名前の一括管理を行なう方法がある。この name server でサービス名とサーバのアドレスの mapping を行なう。この場合、 name server がある計算機がクラッシュした場合にはRPCを受けることができなくなる可能性がある。

SPLICE/RPCではシステムの規模としては非常に小さく、各計算機が密接に結び付いている環境を想定している。そこで、同一のサービス名を持つサーバは同一のサービスを提供すると仮定し、前節で述べたように、サーバのアドレストとサービス名の mapping を行なう分散型の name server を各計算機に配置し、各クライアントはローカルの name server に対して問合せをすることでサーバのアドレスが得られるようにした。

3. SPLICE/RPCの構成

3.1. プロトコル構成

SPLICE/RPC は3つの層から構成される(図2)。 SPLICE/RPC 内で最も下位の階層にある SPLICE/RMTP (Reliable Message Transfer Protocol) では信頼性を保証した1対Nの Multicast 通信の機能を Datagram を用いて提供している。SPLICE/FDTP (Formatted Data Transfer Protocol) では異なる計算機間でも正しくデータがやりとりできるように、各種のデータの標準的な表現方法を定義し、計算機の内部表現をその標準形式に直して通信をする。この階層は下位の通信プロトコルとして SPLICE/RMTP だけでなく他のプロトコルも利用できるような汎用性を持つ。最上位の SPLICE/SRPC (SPLICE Remote Procedure Call protocol) では SPLICE/RMPT と SPLICE/FDTP の機能を利用して、Remote Procedure Call の機能を提供している。この階層では1つのクライアントから同時に複数のサーバに処理を依頼することができるようなRemote Procedure Call も実現している。各計算機に管理用のプロセスを配置して、サーバの名前の管理や、他の計算機の状態のモニタ、さらに利用者のアクセス権の管理を行なうことにより、高度なRPCサービスを提供している。

3.2. SPLICE/RMTP

SPLICE/RPCで提供する1対N-RPCは1つのクライアントから複数のサーバへ同時に処理を依頼できる。このため、SPLICE/RMTPでは1対Nの通信の機能をもつ。 SPLICE/RMTP では1つのメッセージを複数の計算機に対して送信を送る場合は、送信先を明確に指定している場合は1対1通信を送信先の計算機の数だけ繰り返す。一方、送信先を指定せずに、ネットワークに接続されているもの全てに対して送る場合は UDP/IP のBroadcast機能を利用して一回の送信でこれを実行する。

SPLICE/RMTP では処理の依頼のために送られるメッセージの長さに上限を設定しないので、1つのメッセージは複数のパケットに分割して送らなければならない場合がある。サーバが複数のクライアントから同時に処理依頼のメッセージを受信した場合や、クライアントが複数のサーバから処理結果のメッセージを同時に受信した場合には、各メッセージを構成する複数のパケットを同時に受ける。そのため、受信側では複数のパケットからメッセージを再構成する必要がある。 SPLICE/RMPT ではメッセージの reassembling や fragmentation といった処理をしており、ユーザは SPLICE/RMTP を使うことでメッセージを単位として通信を行なうことができる。さらにメッセージが通信中に失われた場合のメッセージの自動的な再送機能などを用意し、信頼性を高めている。

fragmentation では1つのメッセージを複数のパケットに分割する。各パケットにはメッセージが発生した時刻を記録したタイムスタンプと、 1つのメッセージを構成する複数のパケットの内、何番目のパケットかを表す sequence number が付けられている。さらに、各メッセージの最後には trailer と呼ばれるパケットが付けられ、送られたパケットの総数が伝えられる。これによって、メッセージの区切りを発見し、図3のような形でデータがやりとりされる。

reassembling はタイムスタンプと sequence number に基づいて行なわれる。送信中にパケットが失われた場合や、再送によってあるパケットのコピーが複数発生された場合は、sequence number とタイムスタンプによって発見することができ、必要に応じた処理を行なう。複数のクライアント／サーバから同時にメッセ

ージを受信した場合には各メッセージを構成するパケットが入り混じって到着することになるが、trailer を見るまで、バッファに貯められ、trailer を受信したところで、reassembling の処理を行なう。通信時に何らかのエラーが発生した場合には受信側では再送要求を送信側のクライアント／サーバに送る。一方、送信側でも timeout を設定しておき、その時間が来るまでACKが返らない場合には再送を実行する。再送要求に反応が無い場合や再送してもACKが送られてこない場合にはエラーとする。

3.3. SPLICE/FDTP

SPLICE/FDTPは異機種計算機間でのデータのやりとりにおいて、基本的な型のデータから複雑な構造を持つデータまでを、その意味を変えること無くやりとりするための機能を提供する。SPLICE/FDTPにおいては整数型のデータや配列、構造体などを計算機依存性の無い決められたフォーマットに変換して送信を行ない、受信したデータについてはその計算機での内部表現に変換を行なうといった、フィルタとしての役割を果たす。これによって、byte orderやword boudaryの異なる計算機間でのデータのやりとりを正しく行なうことができる(図4参照)。

3.4. SPLICE/SRPC

SPLICE/SRPC はサービスを提供するサーバに関する情報（例えばポート番号等）の管理とクライアントからの問合せの処理(後述)を行なう。SPLICE/SRPC は name server である SPLICE server と、クライアントがRPCを行なう時に利用するインタフェース関数のライブラリから構成される。SPLICE server は各計算機に 1 つ存在する daemon として構成されており、現在利用可能なサーバの情報や、各計算機にある複数のサーバの実行状態の監視、クライアントによるサーバに関する情報の問合せの処理を行なう。SPLICE server では全ての情報を broadcast によってサーバ間でやりとりしながら分散管理をしており、特定のサーバによる集中的な情報の管理を行なっていない。

クライアントではRPCを実行する場合に {hostname,servicename} という組を指定することでサービスを受けるサーバを指定する。この情報を SPLICE server に送りデータベースを検索することで、{hostname, servicename, portID, vna, nrv, accessmode} という組にmappingする。portID はサーバのポート番号、vnaはRPCでの引数の数、nrvは処理結果の個数、accessmodeはサーバのアクセスモード(後述)の状態を表す。

UNIX においては各ユーザは userID と groupID を持っている。UNIX における各種のツールに対するアクセスはこの userID と groupID に基づいてに制限することができる†。これと同じことを SPLICE/RPC でも採り入る。SPLICE/RPCでは、サーバに設定されるアクセスモードによってサーバが提供するサービスを利用することのできるユーザを制限することができる。このアクセスモードは RPC を実行するユーザの UNIX における userID と groupID をもとにサーバにおいて処理の依頼を受けつけるかどうかを判断する。これがうまく機能するためには、SPLICE/RPC で対象とする計算機上では、全ての人の userID と groupID が同じように設定されていることが必要とされるが、SPLICE/RPC が対象としているシステムの規模は小さいためにこの実現は十分実際的であると考えられる。

ユーザプロセスに対して提供される関数群としては、SPLICE server に対して情報の照会を行なう srpc_query() という関数と、RPC を実際に行なう srpc_call(), srpc_multi_call() という関数が用意されている。srpc_query() ではサービス名からそのサーバが存在する計算機の名前、ポート番号などを得ることができる。srpc_call() は 1対1-RPC を行なう関数であり、srpc_multi_call() は 1対N-RPC を行なう関数である。

SPLICE/SRPC では図5に示されるような形態で RPC が実行される。

† 例えばgroupIDが10のユーザのグループに対してのみ実行を許すとか、userIDが6001のユーザに対してのみ実行を許すといったことが可能である。

4. 評価

これまでに筆者らは **SPLICE/RPC** を4.3BSD UNIXが稼働しているIBMRT/PC上に実現した。この実現において現状でどの部分の負荷が一番大きいかを調べるために UNIX に用意されている profile という性能解析ツールを用いて調べたところ、信頼性のある通信の機能を提供している SPLICE/RMTP での処理が **SPLICE/RPC** の処理の内、大部分を占めていることが分かった。これは信頼性を保証するためにメッセージ毎の ACK を送っていることや、 fragmentation と reassembling の処理を実行しているためであると考えられる。このため、現在 SPLICE/RMTP と同等の信頼性や機能を持つ新たなプロトコルを開発しており、これを用いて性能の改善をはかることを予定している。[10]

5. おわりに

SPLICE/RPC で扱うことのできるデータの種類としては、現在のところメモリ上に確保されているもののみである。しかしながらファイルに対する入出力をするために、ファイル記述子などを渡して、処理をすることが多い。これを RPC で実現するためにファイルをサーバ側にコピーし、サーバではそのコピーに対して処理を行ない、処理終了後クライアントにコピーしなおすというような方法も考えられるる。しかし、データベースのデータファイルといった非常に大きなファイルをサーバがコピーすることは不可能な場合が多く、また、コピーの実行によって処理効率が落ちてしまう。このような場合、クライアントはサーバに対してファイルの記述子を送り、サーバでは必要に応じてクライアントに対してデータの参照や変更を行なってもらうようなプロトコルを構築することで実現することができる。また、クライアントに対して割り込みが発生した場合、現状のシステムでは RPC によって処理を行なっているサーバに対しては、クライアントが割り込まれた情報までは送られることはない。しかしながら、このような情報もサーバに送られることが望ましい。**SPLICE/RPC** ではこれらの機能を実現するために、現在プロトコルの再構成を検討している。

さらに、現在の **SPLICE/RPC** ではメッセージに対して暗号化等の security に対する措置をしていない。今後このシステムを多くの計算機で使用すると考えられるので、このような機能も付加していくことも重要と考えられる。

本稿ではSPLICE/RPCについて説明をした。今後は IBM RT/PC 以外の機種への移植を進めるとともに、機能の拡充をしていく予定である。

References

1. 下條真司,東 浩,宮原秀夫, "オブジェクト指向を用いた分散型ネットワーク環境," 情報ネットワーク研究会, 電子通信情報学会, 1987年3月13日.
2. 東 浩,下條真司,宮原秀夫, "オブジェクト指向を用いた分散型ネットワークＯＳ、O^2NE," 情報処理学会第35回全国大会, 情報処理学会, 1987.
3. 松本範久,田中 悟,下條真司,宮原秀夫, "LAN上のワークステーションにおける教育支援システムにおける通信の諸問題について," 情報ネットワーク・交換合同ワークショップ, 電子情報通信学会, 1988.
4. S. J. Leffler, R. S. Fabry, W. N. Joy, P. Lapsley, S. Miller, and C. Torek, "An Advanced 4.3BSD Interprocess Communication Tutorial," in *UNIX Programmer's Supplementary Documents*, vol. 1, Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Berkeley, CA, April, 1986.
5. Andrew S. Tanenbaum and Robbert Van Renesse, "Distributed Operating Systems," *ACM Computing Surveys*, vol. Vol.17, No.4, pp. 419-470, ACM, December 1985.
6. R. J. Souze and S. P. Miller, "UNIX AND REMOTE PROCEDURE CALLS: A PEACEFUL COEXISTENCE?," *Proc. of the 6th International Conference on Distributed Computing Systems*, pp. 268-277, Cambridge, MA, May 19-23, 1986.
7. S. K. Shrivastava and F. Pansieri, "The Design of a Reliable Remote Procedure Call Mechanism," *IEEE Transactions on Computers*, vol. C-31,7, pp. 692-696, July 1987.
8. B. J. Lampson, "Remote procedure calls," in *Lecture Notes in Computer Science*, vol. 105, pp. 365-370, Springer-Verlag, New York, 1981.
9. Sun Microsystems, *Remote Procedure Call Programming Guide*, Sun Microsystems, Mountain View, CA, February 1986.
10. 東 浩,山口英,下條真司,宮原秀夫, "ＬＡＮにおけるグループ同報通信の実現," 情報処理学会第37回

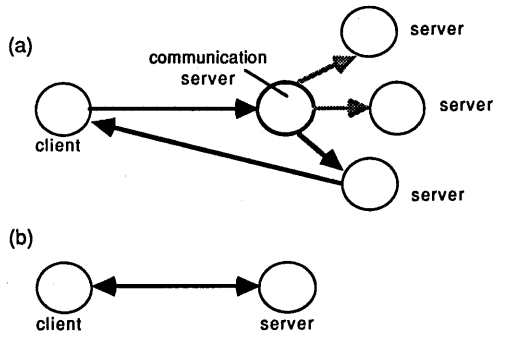

Fig.1 Communication between client and server

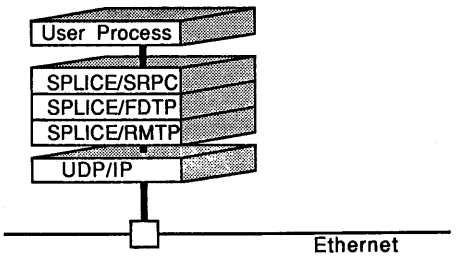

Fig. 2 Protocol Layers for SPLICE/RPC

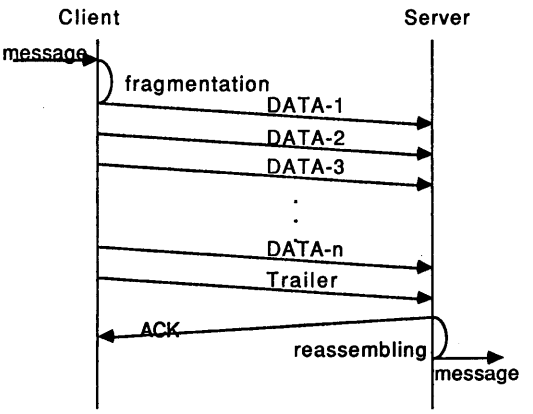

Fig. 3 Message fragmentation/reassembling on SPRICE/RMTP

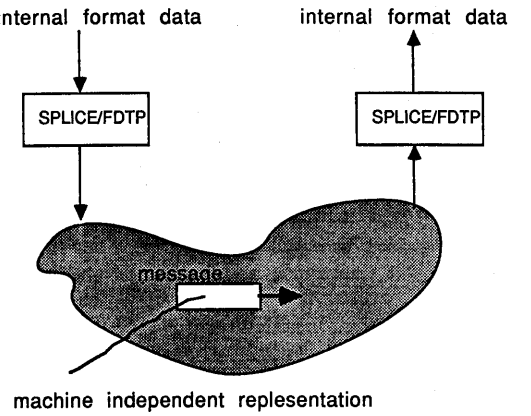

Fig.4 SPLICE/FDTP

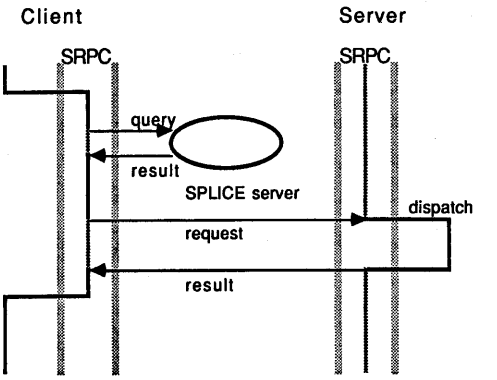

Fig.5 RPC execution