記 号 処 理 68－3
（1993. 3. 19）

マウスメソッドの実装
－分散ウィンドウツールキット上の設計－

古坂孝史[1]
情報処理振興事業協会技術センター

広域分散環境のプロセス協調動作に対して、両者の間の通信量を適正なものに調整できる機構としてリモートクロージャ機構を設計した。この設計の概要と性能実験について報告する。

リモートクロージャは、分散環境上のプロセスにより動的にダウンロードされるプログラムを言う。リモートクロージャ機構は、リモートクロージャを規定する言語、リモートクロージャを送信する機構、実行する機構、協調動作を行なう機構からなる。プロセスの協調動作を行なう機構の一つとして、リモートクロージャ内の変数の参照／変更がそれぞれのプロセスで行なえる。

このリモートクロージャ機構をマウスメソッドに適応することで、効率的な処理が期待できる。実験では、リモートクロージャを用いたマウスメソッドを実装してその有効性を確かめた。

A Implement of the Mouse Method
－ A Design of the Mouse Method on the Distributed Environment －

Takashi Kosaka
Information-technology Promotion Agency, Japan (IPA)
Software Technology Center
Shuwashibakoen 3-chome BLDG., 6F
3-1-38 Shibakoen, Minato-ku, Tokyo 105, Japan
kosaka@stc.ipa.go.jp

This paper designs the mechanism of REMOTE CLOSURE which is available for the kosher amount of the communication, in oreder to cooperate processes on the wide area distributed environment. This paper describes about the design and the capacity experiment of the mechanism of REMOTE CLOSURE.

REMOTE CLOSURE is a kind of a program which is down-loaded dynamically by a process on the distributed environment. The mechanism of REMOTE CLOSURE consists with the language specification, the transmission mechanism, the execution mechanism and the cooperation mechanism. In order to cooperate on processes, they can refer/change variables which are in REMOTE CLOSURE.

In order to expect the efficiency processing, the mouse method fits in with the mechanism of REMOTE CLOSURE. In the experiment, this paper implements the mouse method with REMOTE CLOSURE. And this paper show the effectualness of REMOTE CLOSURE.

[1](株)CSK から出向

1 はじめに

マウスやキーボードデバイスから発生するイベントを処理する機構は、ウィンドウシステムやウィンドウツールキットにとって一つの核機構である。特にマウスの操作で発生するイベントに対して起動される処理(本稿では、マウスメソッドと呼ぶ)は、ウィンドウを利用するアプリケーションにとって、重要な位置を占める。マウスメソッドは、表示を伴うアプリケーションの内部操作を行い、利用者との会話性を高めている。例えば、描画された図形を移動する操作やラバーバンド操作のようなマウスメソッドがある。

今日広く用いられているウィンドウシステムであるX Window Systemでは、サーバ/クライアント方式を利用し、今までのライブラリ方式と比べて計算機資源の有効利用を果たしている。更に、転送するデータをバッファリングする機構をライブラリで用意しているため、通信によるオーバヘッドが軽減されている。しかし、マウスの動きに合わせた画面描画内容の変更などの特定の処理では、転送するデータをバッファリングできる範囲が限定される。

Common Lisp用分散ウィンドウツールキット *Yy* Window Tool Kitは、広域ネットワーク上でのGUIの枠組を提供した[1]。この枠組は、機能分担を行なうことにより通信量を減少させた。しかし、現在の機能分担の枠組は静的な環境で行なわれる。そのため、全てのアプリケーションで効果的に通信量が減少しているわけではない。そこで、動的に機能分担できる仕組みを考え、これをマウスメソッドに適応した。

機能分担する仕組みは、ある特定の目的に対して明示的に機能分担する部分をプログラムに記述する。その部分のプログラムを転送し実行することで機能分担が達成できる。本稿は、転送するプログラムをリモートクロージャと呼び、その設計、実装について報告する。

2 通信によるオーバヘッド

2.1 広域ネットワークの通信によるオーバヘッド

ネットワークの通信特性は、伝送路の通信速度、伝送の遅延、伝送距離から決定される。通信によるオーバヘッドは、これらに依存して決定されることが多い。そのため、通信速度が早く、伝送の遅延が小さく、伝送距離が短ければ通信によるオーバヘッドは少ない。

LANに比べて広域ネットワークは、伝送路の通信速度が遅い傾向にあるため、通信によるオーバヘッドが顕著に現れる。また、何度も通信を行なう場合、LANに比べて広域ネットワークは、一回あたりの伝送の遅延が大きいため、通信によるオーバヘッドが顕著に現れる。

広域ネットワークでの通信によるオーバヘッドを軽減する方法として、下記に示す2つ方法がある。

(1) 通信するデータ量を通信速度に比べて十分小さくする。

(2) 通信の回数を減らす。

(1)は、ネットワーク上の伝送路の通信速度に比べてデータ量が少なければ、通信特性が改善され、通信によるオーバヘッドが小さくなる。(2)は、回数を少なくすることで、一回の通信の伝送の遅延を軽減させ、通信によるオーバヘッドを小さくする。どちらの場合も通信量の減少を行なう。

X Window Systemでは、グラフィック機構を持ったハードウェアデバイスをモデル化し、統一的に取り扱うキューを提供した。また、その最下層での指示をX Protocolとしてまとめている。従って、プロトコルの個々は低機能であり、あるまとまった処理のためには、どうしても通信量が多くなる。そのため、通信速度の遅い広域ネットワーク上では、十分な性能が発揮できない。

*YyonX*では、機能分担を用いることで、通信量の削減を狙い、広域ネットワーク上で十分な性能が発揮できる仕組みを示した。

2.2 現状の *YyonX* での設計

*YyonX*は、1) アプリケーションが稼働するCLOSで記述された *Yy-client*、2) 仮想的な描画

空間を提供し描画処理を行なう *Yy-server* 、3)X-Server の 3 つのプロセスから構成される。X-Server と *Yy-server* との間の通信は X-protocol として設定される。また、*Yy-client* と *Yy-server* 間には、*Yy* -protocol がある。*YyonX* とは、*Yy-client* 、*Yy-server* と両者の間の通信プロトコル *Yy* Protocol を含む。以上の関係を図 1に示す。

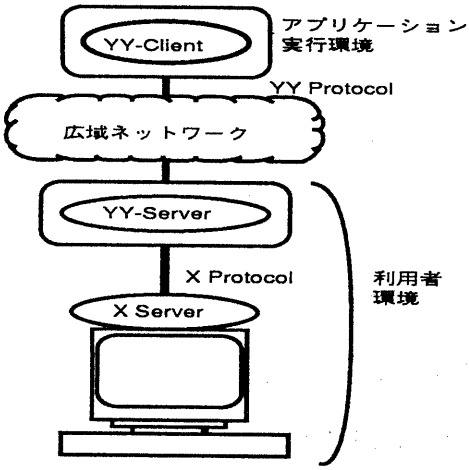

図 1: *YyonX* のプロセスの関係

通信量を減少する設計として下記に示す 3 つの方法を行なった。

(1)「テリトリ」と呼ばれる仮想的な描画空間を設定して、その描画内容を *Yy-server* で保証する機構を設けた [2]。従って、*Yy-client* から再表示に関連するプロトコルを送る必要がない。
(2) 高いレスポンスを必要とする文字入力などは、テリトリにモードを設けて一部機能を *Yy-server* で行なう [3]。従って、*Yy-client* からの細かい命令のプロトコルを送出する必要がない。
(3) 透明テリトリを導入したことにより [4]、そのテリトリに描画された内容だけの移動が行なえる。そのため、描画された内容の移動に対して *Yy-client* からの細かい命令が必要ない。

これらを実装するために、*Yy-server* と *Yy-client* では、仮想的な描画空間「テリトリ」をプログラム上で定義した。*Yy-client* では、定義した「テリトリ」を継承して様々なウィンドウの部品などのクラスを定義する。それぞれのクラスのインスタンスを生成するとき、*Yy* Protocol を用いて *Yy-server* 側に「テリトリ」のデータオブジェクト生成を依頼する。従って、「テリトリ」を継承したクラスのインスタンスは、*Yy-server* で「テリトリ」のデータオブジェクトを一つ持つ。

この実装の結果、*Yy-server* と *Yy-client* との間の通信量は、X Window の場合と比べて少なく、伝送の遅延の影響を受にくい機構を示せた。

2.3 現状の *YyonX* のマウスメソッドの通信量

広域ネットワークを利用するアプリケーションでは、そのプログラムのどの部分が伝送の遅延の影響を受けるか、また、どの部分で通信量の削減を行なうか、という問題は、アプリケーションの内容に依存する。そのため、現状の *YyonX* で採り入れた静的な機能分担の方法は、現実には全てのアプリケーションに適応できない。

特にマウスメソッドは、アプリケーションに依存した内容である。しかも、高いレスポンスを必要とする場合が多く、ネットワークの通信特性にその性能が依存する。

YyonX でのマウスメソッドは、*Yy-client* にある「テリトリ」を継承したクラスのインスタンスに関連付けられている。従って、*YyonX* でのマウスメソッドの定義は、CLOS で用いられるメソッドとは異なり、引数のクラスに特有な振舞いをしない。マウスメソッドは、あるクラスのインスタンスに対して特有の振舞いをする。

図 2にマウスメソッドの起動の機構を示した。図で示すように、*Yy-server* では、X Server から送られてくるイベントに対して、「テリトリ」のデータオブジェクトを特定している。

現状の *YyonX* のアプリケーションの一つに、マウスの操作が多い 3 D Image Editor がある。そのアプリケーションは、マウスの移動でワイヤーフレームにより表示されたオブジェクトの視点を変える操作がある。

文献 [5] では、*YyonX* 上で実装した 3 D Image

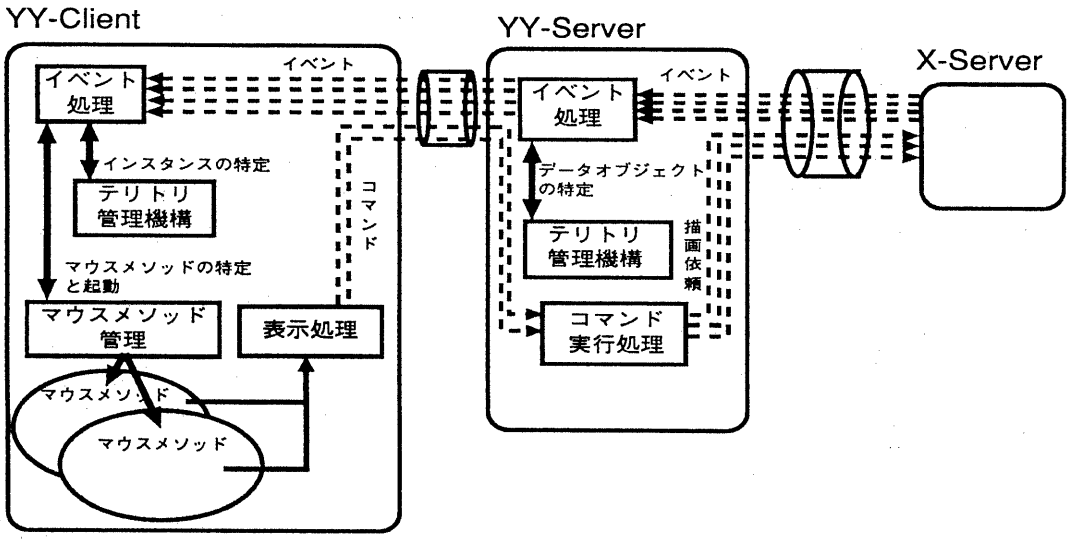

図 2: マウスメソッドの起動機構

Editorでの通信量の実測を行なった。その中で、視点の変更を行なうマウスメソッドでは、X Protocolとほぼ同程度の通信量となることが判った。この結果は、そのマウスメソッドでは、通信量の削減ができていないことを意味する。また、通信量を実測したマウスメソッドは、顕著に通信量が多いが、他のマウスメソッドも同じ枠組を利用しているので、通信量の削減ができないと考えても良い。

3 通信によるオーバヘッドの改善

3.1 動的な機能分担の必要性

YyonX で示した静的な機能分担は、通信量の削減に有効である。しかし、すべてのアプリケーションに適応することは、現実的に不可能である。また、マウスメソッドのようにその内容は、アプリケーションに依存しているため、機能分担の共通の枠組を提供することもできない。

しかし、アプリケーションには、その内容に適した機能が実現される。そこで、アプリケーション記述者がプログラム開発中に明示的に機能分担できる枠組を設ければよい。また、アプリケーションが逐次的に動作を行なう場合もあるので、機能分担したプログラムの内容の変更を動的に行なえる機構が必要である。

3.2 機能分担の要求

動的な機能分担する方法の要求を以下のように定めた。

- 特定の処理の全て、またその一部を機能分担できる。
- 特定の処理を定義する時、明示的に機能分担部分を指定できる。
- 特定の処理の一部分を機能分担する場合、残った部分との協調動作ができる。

機能分担を実現する方法の一つに、GMW[6] などで行なっているプログラムなどをダウンロードする方法がある。

特定の処理の一部を機能分担するためには、ダウンロードするプログラムに状態を包含する仕組みが必要である。また、協調動作を行なうためには、そのプログラムの持つ状態に対してアクセスする仕組みも必要である。

Common Lisp では、レキシカルな束縛に対して値を参照したり変更するクロージャがある。レキシカルな束縛は、状態を包含する仕組みやその束

縛にアクセスする仕組みを提供する。このクロージャの性質をネットワーク上で実現すれば、他方にダウンロードしたプログラムの状態を変更できる。そこで、本稿では、ダウンロードするプログラムをリモートクロージャと呼ぶ。

3.3 リモートクロージャ機構の設計

リモートクロージャ機構は、リモートクロージャを記述する言語、リモートクロージャを送信する機構、リモートクロージャを実行する機構、リモートクロージャを管理する機構からなる。

これらを実現するために、以下の機構が必要である。

リモートクロージャを記述する言語：

(1) ある事象に対して行なう処理、(2) 処理に対する状態 (3) 他者との協調を記述する。

送信する機構：

先の記述は、アプリケーション記述者に可読である。即ち、一般的なプログラムとして文字で表現できる。送信機構は、この内容をそのまま転送する。

実行する機構：

ダウンロードされる側にコンパイラやインタプリタを用いたリモートクロージャを実行する仕組みを置く。リモートクロージャは、あるプログラムの一部を切り出したプログラムであるため、ダウンロードする側のプログラムと協調動作を行なう場合がある。そこで、ダウンロードする側にも、基のプログラムを実行できる仕組みを置く。

リモートクロージャは、何らかの外的要因によって、その実行を開始する。そこで、ダウンロードする側される側ともに、それまで動作している処理の中からリモートクロージャを起動する機構を置く。

管理する機構：

リモートクロージャを実行するためには、転送されたリモートクロージャ管理する機構が必要である。そのため、ダウンロードする側とされる側の両者は、リモートクロージャを管理する機構を置く。その機構は、リモートクロージャの登録、削除、変更を行なう。また、その両者上のリモートクロージャを管理する機構は、互いに同じリモートクロージャを関連付ける。

また、複数のプロセスの協調を考えると、幾つかのプロセスから処理がダウンロードされることもある。従って、リモートクロージャが常にユニークな存在になる必要がある。そこで管理する機構は、リモートクロージャをユニークにする仕組みを置く。

3.4 *YyonX* への適応

ダウンロードする側が *Yy-client* であり、受ける側が *Yy-server* である。*Yy-client* は、CLOS で記述されているので、ここにリモートクロージャの記述が可能な拡張を行なう。リモートクロージャの転送は、*Yy* Protocol を拡張して、リモートクロージャの内容を送信する。

3.5 リモートクロージャを利用したマウスメソッドの実装

リモートクロージャを利用したマウスメソッドは、マウスメソッド管理機構とリモートクロージャ管理機構ににより管理する。

リモートクロージャの機構とマウスメソッドの起動機構をまとめた仕組みを図 3に示す。図 3に示すように、*Yy-server* のイベント処理は、「テリトリ」管理機構とリモートクロージャ管理機構によりリモートクロージャ特定する。リモートクロージャがある場合は、そのクロージャを実行する。

一方、*Yy-client* 側のイベント処理は、リモートクロージャの特定をマウスメソッド管理機構とイベント処理機構が行なう。

4 通信量削減ための実験

4.1 リモートクロージャの実装

リモートクロージャによる機能分担での通信量削減を実験するために、リモートクロージャを持つマウスメソッドを実装した。実験のためのリモートクロージャの言語仕様は、C 言語を基本として、リモートクロージャ記述が可能な拡張を行なった。

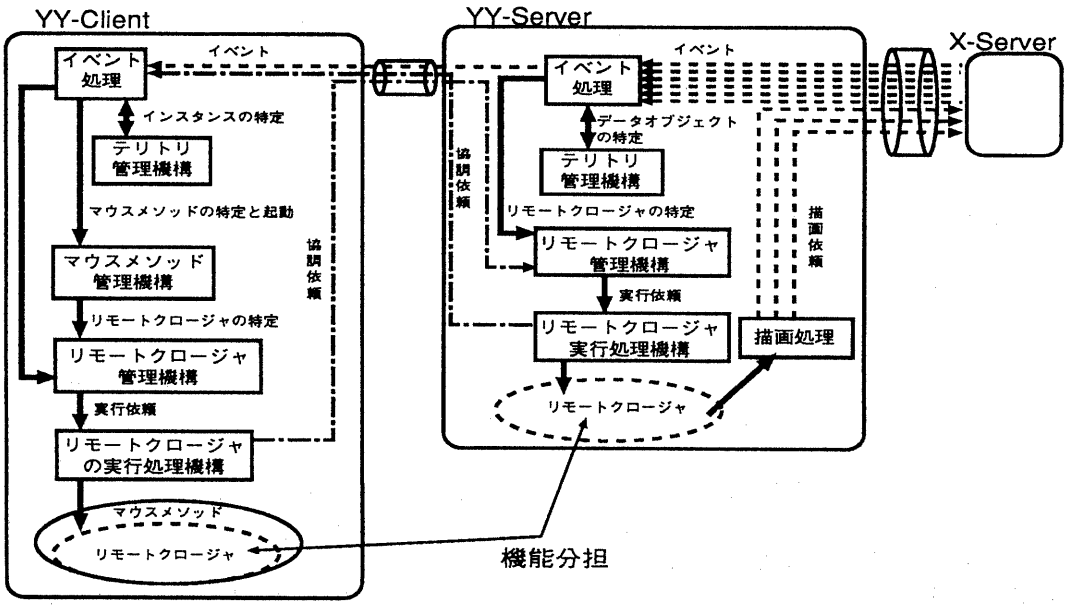

図 3: リモートクロージャを導入したマウスメソッドの起動機構

リモートクロージャ自身は、C 言語の関数になるように定義する。但し、その関数内で Lisp でいうレキシカルな束縛に対して、参照／変更が行なえる。また、その関数内では、サーバ側で定義している関数が利用できる。

利用できるデータ構造は整数値とし、レキシカルな束縛に対する値も整数値とした。

4.2 リモートクロージャ機構の実装

リモートクロージャを用いたプログラムの通信量を調べる実験のために、それを持つマウスメソッド機構を実装した。その実装は、*YyonX* と同じ枠組をもつサーバ／クライアントモデルのウィンドウツールキットである。その構成は、CLOS で記述したクライアント、C 言語で記述し X Server と接続できるサーバ、クライアントとサーバ間と通信する *Yy* Protocol と同じ枠組をもつプロトコルからなる。

クライアント側は、サーバとの通信路の設定、初期画面の設定、同期を必要としない描画コマンドの送信、リモートクロージャの設定、同期を必要とするリモートクロージャの送信、イベント発信の要求、イベントの取り込みを行なう。イベント発信の要求は、クライアント側で動的に行なう。

サーバ側は、クライアントとの通信路の設定、X サーバとの通信路の確立とウィンドウの生成、クライアントから送られてくるコマンドの実行、リモートクロージャのコンパイル／リンク、リモートクロージャの実行、クライアント側へのイベントの送信を行なう。リモートクロージャをコンパイル／リンクするので、実行機構は言語処理系に従う。

クライアント側にあるマウスメソッドは、明示的にクライアント側で実行する。リモートクロージャを持つマウスメソッドに対しても、同様に明示的に行なう。そのため、全てのマウスメソッドは、明示的に起動されるのでイベント処理は、自分自身で行なう。また、明示的にマウスメソッドを実行するので、リモートクロージャ管理機構は置かない。

サーバ側は、イベントに対してクライアントにイベントを送るモードと設定されたリモートクロージャを起動する 2 つのモードを置く。これらのモードの切替えは、イベント発信要求により行

なう。従って、リモートクロージャの管理機構はサーバ側に置かず、クライアント側のプログラムで行なう。また、リモートクロージャ実行機構は、そのプログラムがCの関数であるため、その関数を一旦ファイルに書き出し、コンパイル、ダイナミックリンクを利用した。

4.3 実験の環境

実験環境としては、手持ちのSPARC Station 2にクライアントプログラムを実行させ、SPARC IPCにサーバを実行させた。サーバのウィンドウは、SPARC Station 2上に表示した。SPARC Station 2とSPARC IPCとの間は、イーサネットで接続している。図4に実験環境を示した。

4.4 実験の方法

実験は、マウスの移動に追従して3つの円とその円を結ぶ3本の線を描くマウスメソッドを構築して行なった。このマウスメソッドには、*YyonX*と同じ枠組である全てクライアント側で行なう場合(ケース1)、3つの円と3本の線を描画する部分をリモートクロージャした協調動作を行なう場合(ケース2)、殆んど全てをリモートクロージャで行なう場合(ケース3)の3つのパターンを用意した。これら3つのマウスメソッドは、ボタンが押下されるまで、マウスの移動に追従した描画を行なう。

これらのマウスメソッド処理のアルゴリズムを図5に示す。図5の点線で囲った部分は、リモートクロージャにする部分を示す。

マウスの移動に追従するマウスメソッドは、描画した情報を消去するために、その位置などを記

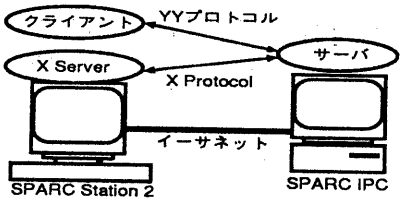

図 4: 実験環境

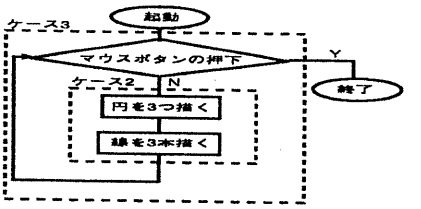

図 5: 実験するマウスメソッド

憶する機構が必要である。ケース1とケース2では、この機構にグローバルな変数を利用した。一方、ケース3では、リモートクロージャの持つレキシカルな束縛の保存の性質を利用した。

この実験では、ある点から別の点までマウスを移動させた時に発生するサーバ/クライアント間の通信量とサーバとX Server間のX Protocolの通信量をtcpdumpを用いて測定した。この測定をそれぞれのケースについて行なった。また、リモートクロージャを転送した後の通信量も調べた。

4.5 結果と評価

表 1: マウスメソッドの通信量の比

実験ケース	*X Protocol*/サーバ-クライアント間
ケース1	2.928
ケース2	8.338 (8.839)
ケース3	48.822 (1017.428)

実験結果を表1に示す。UNIX OSやネットワーク、操作に関連して多少の測定誤差が考えられるので、絶対値ではなく個々の値の比率で統一する。表1に示す比の値は、大きければ大きいほどサーバ-クライアント間の通信量がX Protocolに比べて少ないことを示す。また、表中の括弧の中の値は、リモートクロージャの転送後の通信量の比である。従って、ケース1では、サーバ-クライアント間の通信量は、X Protocolの1/3程度である。リモートクロージャの転送を含めた通信量は、ケース2では、1/8.3程度、ケース3では、1/49程度である。また、リモートクロージャの転送後

の通信量は、ケース2では、1/8.8程度、ケース3では、1/1017程度である。

ケース1とケース2では、その差が約3倍である。3つの円と3本の線を描く関数をそれぞれプロトコルで送るケース1とそれらを一つにまとめたケース2では、それぞれのプロトコルのバイト数が132バイト、36バイトである。これらのバイト数の差は、およそ3倍違う。また、リモートクロージャの転送を含めた通信量とそうでない場合の通信量の差は、殆んどない。従って、リモートクロージャのプログラムを転送する通信量は、この場合影響していない。

また、ケース3では、リモートクロージャの転送を含めたサーバ-クライアント間の通信は、Xと比べて1/49になっている。リモートクロージャの転送を含めないと、1/1017となっている。この結果は、全ての処理をリモートクロージャで行なっているため、そのプログラムが大きくなる。結果として、リモートクロージャの転送の影響が現れた。

ケース2とケース3の結果から、リモートクロージャの部分が固定で、その制御が頻繁に変更するような場合は、一部機能を切り出すリモートクロージャが良い。また、リモートクロージャを定義し、その内容の変更が頻繁にない場合、ケース3の全ての処理をリモートクロージャにする方法が良い。

5 おわりに

動的な機能分担による通信量削減ための方法を考えた。動的な機能分担の枠組提供するために、リモートクロージャを定義し、マウスメソッドに適応させた。リモートクロージャの設計とリモートクロージャをもつマウスメソッドの機構を実装／実験を行なった。

結果として、マウスメソッド等への有効性が示せた。また、処理の内容や分散協調方式は、アプリケーション記述者が自由にできる。従って、アプリケーションに応じた最適化の可能性を持つ。

今後の課題として、リモートクロージャを利用したマウスメソッドの *YyonX* の組み込み、リモートクロージャ自身の言語仕様の詳細設計、リモートクロージャを転送する時の通信量の削減、マウスメソッド以外の他の枠組への適応などがある。

謝辞：
本研究は、IPA技術センタプロジェクト「分散環境用プロセス間通信プロトコルの研究」プロジェクトの一貫として行なった。本研究を進めるのに当たり、ご意見を頂いた本プロジェクトCG、WG委員、*Yy*プロジェクトメンバに感謝します。また、リモートクロージャの設計では、有益な意見を頂いた共同研究者の田中啓介氏に感謝します。

参考文献

[1] Masayuki Ida, et.al. : An Overview of *Yy* and *YyonX* - a clos based window tool kit and its implementation-, Proc. of Europal'90, pages 245-252, March 1990

[2] 田中啓介、他： *YyonX* における *Yy-server* の設計, 情報処理学会第40回全国大会, mar. 1991

[3] 田中啓介、他： *Yy-server* における入力機構について、情報処理学会研究報告書,91-SYM-59, Mar. 1991

[4] 太田幸雄、他： *Yy* へのカラーグラフィックス機能の組み込みについて、情報処理学会研究報告書,91-SYM-58, Mar. 1991

[5] 田中啓介： *YyonX* における通信プロトコルの性能測定と *Yy* プロトコルに対する一考察、Technical reprot, *Yy* Project, Sep. 1992

[6] bit別冊石田晴久編集 :Xウィンドウの仲間たち, pp77-92, 共立出版, May. 1992