複数のネットマウスにより大きな共同作業空間構築を支援するミドルウェアGLIA

西村真一[†,☆] 由井薗隆也[††] 宗森純[†††]

複数のマウスを用いたネットワーク操作を複数ウィンドウに対して行える仕組みをミドルウェアGLIAとして実現し，大きな共同作業空間の構築を支援する．そのために，マウスデバイスごとにマウスモニタとマウスエージェントという組を作成し，マウスエージェントがネットワークを介して移動し，移動先のウィンドウに対して操作を可能とするネットマウスの仕組みを実装した．具体的なアプリケーションとして，GLIAを用いて，分散協調型KJ法支援ソフトウェア，同時描画ができるペイントソフト，70台のPCを用いたスライディングパズルなどを実現した．そして，GLIAのネットマウスを評価するために，複数人による共同作業やマウスエージェントの通信性能を測定した．その結果，(1) 全部で70台の計算機をつなぐことができるとともに，KJ法のような画面の一覧性が重要とされる共同作業における効果が期待できること，(2) マウスエージェントの同時移動が起こるとネットマウスの性能に遅延を感じさせるが，同時移動がなければ10人規模のネットマウスの操作に対する遅延は少ないこと，(3) 既存のJava Swingアプリケーションを GLIAアプリケーションに変更するには，マウスイベントに集中したプログラム作業で行えること，が分かった．

Middleware GLIA for Developing Large Collaboration Space with Multi Networked Mice

Shinichi Nishimura,[†,☆] Takaya Yuizono[††] and Jun Munemori[†††]

Middleware GLIA has been developed for creating large collaboration space. GLIA combines multi windows with multi cursors via a computer network. GLIA has a basic function named as a networked mouse. The mouse consists of a combination of a mouse monitor and a mouse agent, and the mouse agent can move to a window of another computer and access the window. Examples of GLIA applications were developed, such as groupware for the distributed and cooperative KJ method, a paint tool for simultaneous drawing by multi users and a sliding puzzle with 70 PCs, etc. To evaluate the performance of GLIA mice, some trials of building large collaboration space had been performed and the communication performances of the networked mice were measured. Those results showed as follows; (1) GLIA can unite 70 computers. GLIA would improve efficiency of the cooperative work such as the distributed and cooperative KJ method which needs to have a look data. (2) Simultaneous moving of some mouse agents for the networked mice makes feeling of delay, but it is assumed that about ten participants feel less of the delay in the case of few simultaneous moving of mouse agents. (3) The modification of a GLIA application from an existing Java Swing application is mainly related to mouse events.

1. はじめに

近年，我々の身の回りにはデスクトップコンピュータから携帯端末であるPDAや携帯電話，そして，小型センサなど様々なデバイスがネットワーク接続可能な状態で存在している．ユビキタスコンピューティングのための情報環境は着実に，社会に浸透し始めている．このような環境を用いた情報サービスの実現形態としてオフィスワークを支えるシームレスな知的生産活動のためのデジタル環境[1]を課題としてあげることができ，計算機支援協調作業（CSCW）の研究では，専用の共同作業環境や携帯デバイスを利用できる共同作業環境が研究されてきた[2),3)]．このような作業環境では，複数のユーザインタフェースをユーザが有効活用するために，専用のハードウェアやソフトウェア

† 島根大学
Shimane University
†† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology
††† 和歌山大学
Wakayama Universitry
☆ 現在，株式会社ブリッジコーポレーション
Presently with Bridge Corporation Inc.

を開発する必要があり，その負荷を軽減するための開発ツールを作ることが研究を進めるうえでも重要である[4]．

複数のユーザインタフェースを連携させて仮想的な共同作業空間を実現する会議環境を実現する研究として i-Land[3),5)]，iRoom[6)] があり，また，携帯端末である PDA の活用を狙った PebblesDraw[7)] や GDA[8)] がある．i-Land は，ユビキタス環境での共同作業を支援するグループワーク環境の実現を目指しており，大型ディスプレイ 2 台をネットワーク接続した大きなパブリック画面を実現している．小型端末である PDA を使用する PebblesDraw は，1 台の計算機ディスプレイ上に表示された共同作業用ウィンドウ上に複数の PDA を介した操作を行える．また，GDA は，1 つの PDA では表示しきれない情報を扱うために，PDA 上のウィンドウ画面をその場で接続するパブリック画面を構築できる．これらの研究では，いずれも専用の開発環境を実装し，デバイス連携による共同作業支援の新しい可能性を示してきたが，複数のマウスを同時に使用できる大きな共同作業空間を支援してはいない．たとえば，i-Land や GDA は，複数の入力デバイスを同時に用いた共同作業を扱えない．また，PebblesDraw は単一計算機を対象としており，大きな共同作業空間を考慮していない．特に，日本では，衆知を集める発想法として知られた KJ 法[9),☆]を参考にしたソフトウェアやグループウェアの研究が行われてきており，数百個の意見データを一度に画面表示できないことが解決課題とされている．その一覧性の問題を解決するために，従来の KJ 法を参考にしたソフトウェアの研究では，拡大縮小表示機能やパニング機能，テトリス型インタフェースなどの工夫を実装している[10)~12)]．しかしながら，画面の切替えが必要であるために，紙面上の KJ 法が有しているような一覧性は確保されているわけではない．これらより，大きな画面で複数の人が多数のカーソルを使って共同して作業することが望まれており，将来的に計算機による電子化を推進した会議などへのニーズが期待できる．

一方，科学的可視化における大量情報の表示や電算機演習室・会議室におけるパブリック画面の利用において巨大ディスプレイの活用は期待できる領域であり，マルチディスプレイに関する研究が行われている[13)]．特に，X-Window の 1 つのイメージを複数の計算機にマッピングする Xdmx[15)] は既存のアプリケーションが動くという長所がある．また，複数カーソルを利用できる XCursor[14)] の開発が行われている．これら技術を組み合わせて，複数カーソルを使用できる共同作業空間を部分的に実現できる．しかし，既存アプリケーションを利用するのみでは，共同作業を支援するグループウェア環境として制限が存在する．たとえば，複数のマウスが押された状態を区別して取り扱う仕組みはなく，複数カーソルが同時に描画を行うといった共同作業を行えるわけではない．したがって，同時に 1 人しかアクセスできないような共同作業環境であり，複数ユーザの積極的な同時参加を求める共同作業環境としては必ずしも理想的とはいえない．

そこで，本論文では，ネットワークを介して複数のマウスを複数の計算機上にあるウィンドウで使用可能とすることによって，大きな共同作業空間の構築を可能とするミドルウェア GLIA（Groupware-kit for Linking Interactive Actions）を提案する．まず，2 章で GLIA の設計方針，実装およびアプリケーション開発について述べる．3 章で，GLIA の評価実験について述べ，4 章でその実験結果をもとに考察する．

2. ミドルウェア GLIA

2.1 設計方針

GLIA は，ネットワークを介した複数のマウスの操作を，複数の計算機上にあるウィンドウ上で利用可能とすることにより，広がりのある共同作業空間の構築を支援するミドルウェアである．**図 1** に示すようなシステム構成において，マウスなどの入力デバイスを管理する入力デバイス処理，ネットワークを介したオブジェクト通信を支援する通信処理，そして，入力と通信を組み合わせて得られた情報を出力する GUI 処理が連携することにより，ネットワークを介したマウス操作（ネットマウスと名付ける）を基本機能として実現している．

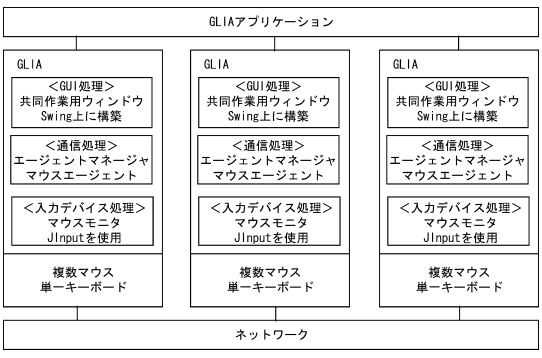


図 1 GLIA のシステム構成
Fig. 1 System configuration of GLIA.

☆「KJ 法」は，株式会社川喜田研究所の登録商標である．

表 1 GLIA の支援機能一覧
Table 1 A list of supported functions of GLIA.

実現機能	内容
ネットワーク対応マウス	ネットワークを介して他計算機にあるウィンドウ上でカーソルを表示して操作.
複数マウスの使用	複数のマウスを接続して複数のカーソルを表示できる．ネットワークを介した利用も可能.
ネットワーク対応キーボード	ネットワークを介したキーボード入力を実現．日本語入力を支援するために 1 計算機 1 入力のみ.
柔軟なウィンドウ結合	複数のウィンドウを接続先を方向指定するだけで接続可能．計算機の起動順に関係なく接続.
Swing に近い開発環境	GUI として Swing を用いた Java アプリケーションを GLIA 上に移植することを支援.
マウスの同時アクセス処理	利用者から見て複数のマウスの選択操作が同時に行われているように表示.
GUI レベルのアクセス制御	マウスイベントに各マウスの情報を含む．その情報を用いて GUI レベルのアクセス制御を記述可能.
オブジェクト転送機能	オブジェクトレベルのデータ通信機能により複数ウィンドウ間での多様なデータ通信を支援.

GLIA の設計方針とその理由を以下に記す．また，**表 1** に実現機能の一覧を示す.

(1) 計算機 10 台接続可能
過去に行われてきた分散協調型 KJ 法の研究では，1 画面（縦 768 画素，幅 1024 画素）に表示される意見数 50 個〜70 個が適当であった．本格的な KJ 法の支援[12] では，500 個の意見の使用が目標とされている．したがって，本研究では 10 画面を扱える 10 台を目標とするため.

(2) 異なる大きさのウィンドウが接続可能
すでにある複数のディスプレイを使用する場合，大きさが異なるモニタがあり，それらを活用したい場面が考えられるため.

(3) マウス 10 個使用可能
グループウェアの研究では，10 人規模の支援が大人数会議とされており，一般的な会議には 10 人程度の支援でよいと考えられるため.

(4) ソケット通信に TCP，UDP のどちらも使用
通信には確実なデータ転送が求められる場合と，確実性は落ちても連続的なデータ転送が行えるとよいため.

(5) マウスの応答速度 1 秒未満
ユーザインタフェースの応答時間は 1 秒未満が望ましいとされるため[16].

(6) マウスに関するアクセス制御のプログラミングを記述可能
会議によっては，1 つのものを 2 人でとりあうと困る場合もあるため.

(7) GUI として Java Swing[17] を使用
異なる OS で動作させるとともに，洗練されたオブジェクト指向の GUI ライブラリを用いた効率の良いシステム開発を行うため.

2.2 開発の概要

GLIA の開発のためにかかった Java のプログラム行数は約 5 千行であり，総クラス数 155（そのうち，内部クラス数 35）であった．そのほか，OS 依存の処理を用いるために JNI（Java Native Interface）[18] と C++を用いた数十行規模のプログラムが存在する.

複数のマウスなどのデバイス装置から直接入力データを取得するために，Java の API である JInput[19] を使用している．また，ネットマウスの実現や複数ウィンドウ間のデータ通信には，Java の Serialization 機能とソケット通信を組み合わせたオブジェクト[18] のデータ通信を実現している．そして，複数マウスカーソルが取り扱える GUI を支援するために Java 標準の GUI 開発環境である Swing[17] を拡張している.

現在，GLIA によるすべての機能が動作する OS は MacOSX，WindowsXP である．GLIA の通信処理と GUI 処理の実装はすべて Java で記述されており，OS 依存がなく利用できる．したがって，GLIA では Unix 系 OS（Linux や SUN OS）における入力デバイス処理部分の開発が不完全であっても，Unix 系 OS は，他計算機にある入力デバイスの処理を通信を介して受け付けることができ，共同作業用の GUI 部品として活用できる.

2.3 ネットワークを介したマウス操作

GLIA は起動時に，共同作業空間を関連付けるための設定ファイルを読み込み，各種通信を行うためのネットワーク接続を試みる．その結果，ネットマウスなどの仕組みを利用可能とする．基本機能であるネットマウスを実現するために，GLIA は接続されたマウスごとにマウスモニタとマウスエージェントというペア部品を構築する．マウスモニタはマウスのデバイス状態を取得する．そして，マウスエージェントは，マウスモニタが送信するマウスの情報を受信および保持し，その情報をもとにしたカーソル処理を共同作業用ウィンドウに依頼する．特に，エージェントマネージャの管理下，マウスエージェントがネットワークを介して他計算機に移動できることにより，ネットワーク接続されたすべての計算機に対するマウス操作が実現されている．このネットマウスの仕組みを中心とした実行時のシステム構成例を**図 2** に示す.

2.3.1 共同作業空間の関連付け

ミドルウェア GLIA を用いて複数の計算機上にあるウィンドウを用いた共同作業空間の設定方法について説明する．複数の計算機上にあるウィンドウの位置

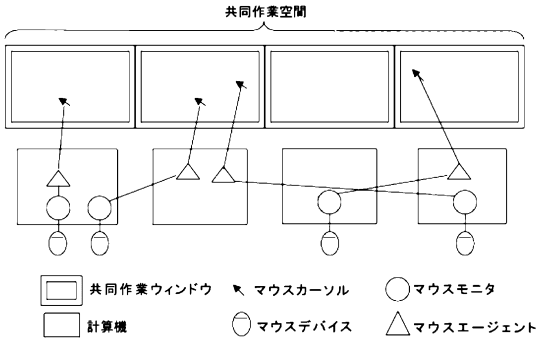


図 2 ネットマウス使用時のシステム構成例
Fig. 2 Example of system composition using networked mice.

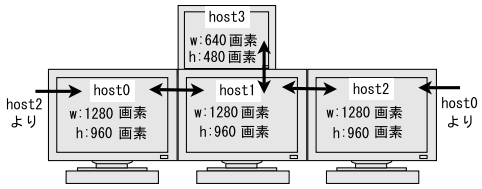
```
<?xml version="1.0" encoding="UTF-8" ?>
<glia>
  < peer name="host0" host="10.210.10.1" port="2000" active="true">
    <left name="host2"/> <right name="host1"/>
    <window x="0" y="0" w="1280" h="960" />
  </peer >

  < peer name="host1" host="10.210.10.2"  port="2000" active="true">
    <top name="host3"/><left name="host0"/><right name="host2"/>
    <window x="0" y="0" w="1280" h="960" />
  </peer >

  < peer name="host2" host="10.210.10.3"  port="2000" active="true">
    <left name="host1"/><right name="host0"/>
    <window x="0" y="0" w="1280" h="960" />
  </peer >

  <peer name="host3" host="10.210.10.4" port="2000" active="true">
    <bottom name="host1"/>
    <window x="0" y="0" w="640" h="480" />
  </peer >
</glia>
```

(a) 協調ウィンドウの設定ファイル例



(b) 協調ウィンドウの物理的設置例

図 3 複数ウィンドウによる共同作業空間の設定
Fig. 3 A setting of a collaboration space with multi windows.

関係などの情報は**図 3** に示す XML ファイルに記述する．

GLIA の通信プログラムはつねに送受信可能な状態であるので，各計算機にある GLIA アプリケーションの起動順番は気にする必要がない．その設定ファイルの最上位要素は< glia >であり，要素< peer >内に各計算機が持つウィンドウの情報を記入する．そのうちで，属性< active >はウィンドウごとに起動する入力デバイスの監視プログラムを動かすかどうかを指定する特殊な設定であり，同一計算機内で複数のウィンドウを関連付けたい場合に使用できる．

ウィンドウの配置方法には，接続方向先のホスト名を指定する接続指定方式とマトリックス状に配置する行列指定方式の 2 方式を用意している．行列指定方式は，すでにウィンドウの接続方式が固定されている場合に使用し，ウィンドウが存在する行と列の値を設定する．一方，接続指定方式は，行列方式と比べて柔軟な方式であり，接続先を指定する要素である< top >，< bottom >，< right >，< left >を用いて，接続先のウィンドウを指定する．

図 3 は接続指定方式を用いた例であり，4 台の計算機上に表示されたウィンドウをトツ状に構成するだけでなく，左端に位置するホスト 0 にあるウィンドウの左側をホスト 2 にあるウィンドウの右側と関連付けている．また，図 3 の例では，ホスト 1 とホスト 3 は大きさの違うウィンドウが関連付けられている．大きさの異なるウィンドウ間でマウスカーソルの移動が生じた場合は，マウスのカーソル位置は自動的に調整される．具体的には，移動先のウィンドウの最大座標値が，移動元のカーソルの座標より小さい場合は，その最大座標より小さい位置にマウスのカーソル位置が収まるように座標変換している．

2.3.2 ネットマウスの仕組み

ネットワークを介したマウス操作であるネットマウスの処理内容を**図 4** のフローチャートに示す．このネットマウス全体の処理において，マウスの状態を取得する処理，マウスのイベントやマウスエージェントの通信を監視する処理，および，GUI 処理にとりかかる契機となるキューに入れられたマウス情報を取得する処理は別々なスレッドとして並行動作させている．これにより，GUI レベルの描画処理によって CPU 処理が止まり，ある程度の時間間隔で処理されることが期待されるマウスカーソルの処理が悪くなる事態を回避している．また，通信処理においては，マウスエージェントの移動などの確実に伝達される必要がある通信には TCP ソケットを用い，ある程度のデータ落ちは許容され，高速性が要求されるマウスイベントの送信には UDP ソケットを使用している．

ネットマウスの処理は，(1) マウスモニタが JInput を使用して，マウスデバイスの操作情報（マウスの X 軸移動量，Y 軸移動量，ボタン状態）を直接取得することによって開始される．この取得は Swing の Timer クラスを使用し 20 ms 周期で発生する．また，接続されたすべてのマウスの情報を取得する．次に，(2) マウスモニタはエージェントマネージャに取得したマウス情報を送信する．ここで，大きな共同作業画面においてマウスが速く動かせるように，マウスモニタは，

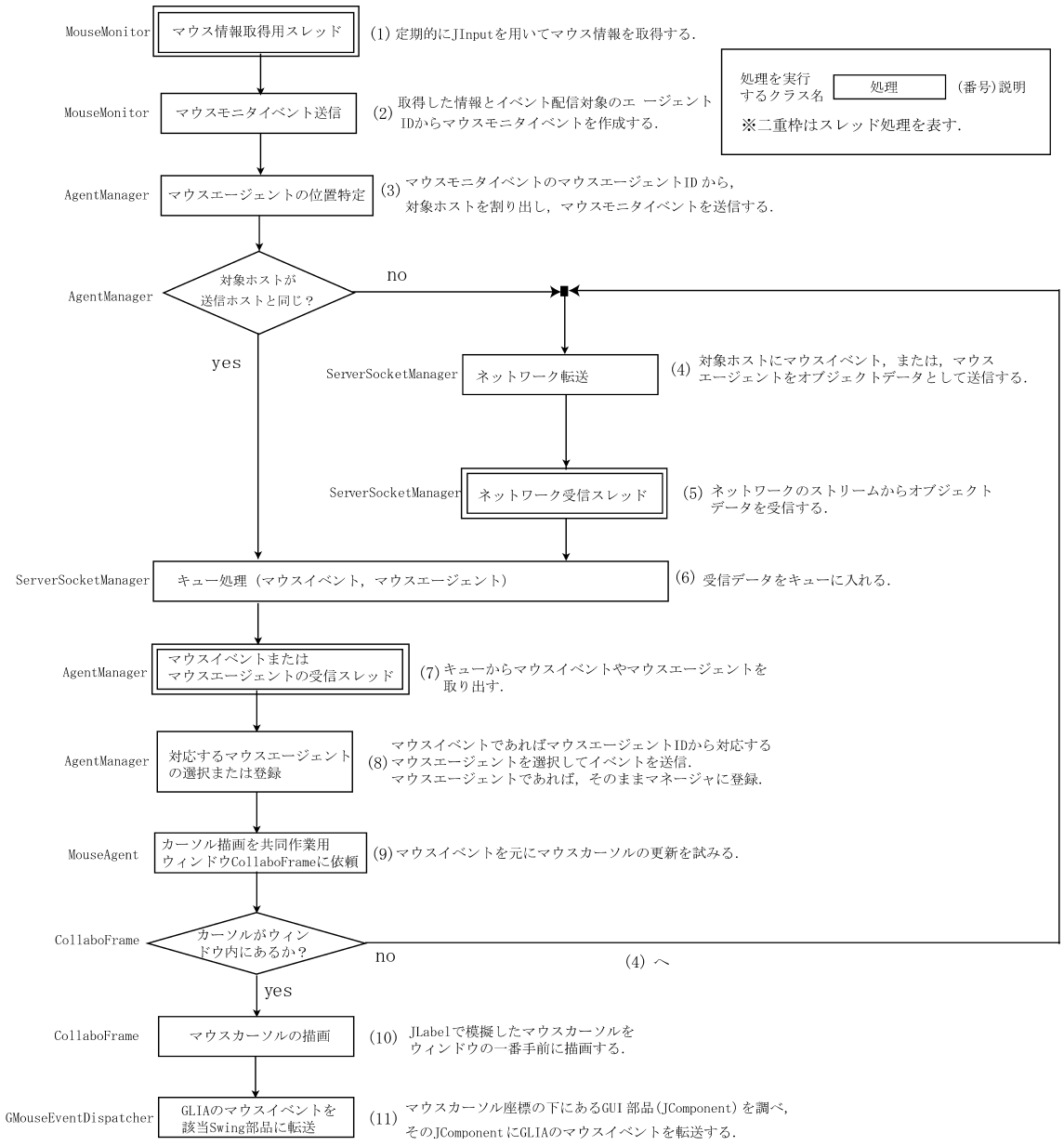


図 4 ネットマウス処理のフローチャート
Fig. 4 A flow chart for handling networked mice.

マウスを速く動かせばマウスの移動距離が大きくなるようなイベントを作り出している.

(3) エージェントマネージャは,マウスモニタから送られてきたイベントが保持しているエージェントIDからマウスエージェントの位置を割り出し,マウスエージェントへマウスイベントを送信する.このイベント送信は,マウスエージェントが同じ計算機にある場合は,(6) キューに入れる.一方,他の計算機に移動している場合は,(4) 対象ホストにイベントが送信され,(5) 相手先のホストのネットワーク受信スレッドを介して (6) 相手先のキューに入れられる.

エージェントマネージャは (7) キューに対してマウスに関するオブジェクトが入ることを受信待ちするスレッドを持ち,キューに何らかのデータが送られてきた場合,そのデータを受信順に取り出し,(8) 対応するマウスエージェントを選択して,データを送信する.

(9) その情報を受け取ったマウスエージェントは,その情報に基づくカーソルの描画を共同作業用ウィン

ドウである CollaboFrame に依頼する．ここで，CollaboFrame は Swing の標準ウィンドウである JFrame を拡張した部品である．その依頼時，CollaboFrame はカーソルの移動がそのウィンドウ内に収まるか判定する．

もしウィンドウ内に収まれば，(10) カーソルをウィンドウ上に表示する処理を行う．その共同作業用ウィンドウでは，複数のカーソルを表示するために，Swing の JLabel クラスにカーソル画像を貼り付けたものを必要な数だけ，そのウィンドウの最上位に描画している．このカーソル画像は自由に指定できる．ただし，JFrame が持つ描画領域と別な描画領域を生成して利用する JDialog などの Swing 部品では，カーソル表示が消えるという問題が起こる．この問題を解決する方法として，既存の JDialog クラスを，CollaboFrame のように GLIA 専用クラスとして開発することが考えられる．

逆に，ウィンドウ外にカーソルが移動する場合は，CollaboFrame は，(4) その移動方向をマウスエージェントを介してエージェントマネージャに通知する．その結果，(5)〜(9) エージェントマネージャは，該当するマウスエージェントをカーソルの移動方向に転送し，(10) 転送先で再びマウスカーソルを表示する処理を別な計算機上に表示された共同作業用ウィンドウに依頼することになる．

(11) 最後の処理は，マウスの状態（たとえば，第 1 ボタンが押されているかどうか）を対応する GUI 部品へ転送する処理である．共同作業用ウィンドウ上にカーソルを描画した後，カーソル座標下に存在する Swing 部品を調べる．そして，該当する Swing 部品に，マウスの状態を持つイベントを転送する．このイベントは Swing のマウスイベント処理に用いられる MouseEvent クラスを継承している．よって，GLIA のマウスイベントは，従来の Swing 部品内のマウスイベントと同じようにイベント配信される．また，そのイベントには送信元であるマウスの識別情報も含んでおり，GUI レベルでのアクセス制御のプログラミング開発に応用できる（付録 A1 参照）．

2.4 ネットワークを介したキーボード入力

GLIA のネットワークを介したキーボード入力は計算機あたり 1 つのキー入力のみ支援している．その仕組みでは，GLIA 起動時にキーボードを 1 つのマウスエージェントへ対応付け，このマウスエージェントを介してキーボード入力を他計算機上に送信している．特に，日本語入力のためのカナ漢字変換機能を用いたテキスト入力のために，キーボードがある計算機上にテキスト入力が行える GUI 部品を用意している．また，接続された複数のマウスのいずれもキーボードと切り替えて関連付けることができ，そのための切替えメニューを実装している．

ここで，各計算機に 1 台のキーボードという形態で，ネットワークを介したキーボード入力を実装している理由について説明する．半角の英数字入力に限るのであれば，JInput を用いて複数のキーボードごとに入力状態を取得することが可能であり，複数キーボードに対応したネットワーク入力が実現できていた．一方，日本語入力を利用するためには，カナ漢字変換機能と呼ばれる OS 常駐型のプログラムが使用される．そのカナ漢字変換機能は，一般的に，並行処理などを用いて同時利用する仕組みが支援されていないと推測されるからである．たとえば，MacOSX と Windows において，各 OS に標準装備されたカナ漢字変換機能インタフェースをあるアプリケーションで用いている場合，別なアプリケーションにマウス操作を移した場合，その変換が勝手に確定される現象が起こる．したがって，新たにカナ漢字変換機能を並行処理できる実装または仕組みを検討しなければ，日本語入力を可能とする複数キーボードに対応したネットワーク入力は実現できない可能性が高い．

2.5 GLIA による JAVA アプリケーション開発

GLIA による Java アプリケーション開発は，Swing を用いた GUI アプリケーション開発とほぼ同等な方法で行うことができる．簡単なサンプルプログラム（ウィンドウ上にボタンを表示し，ボタンがマウスで押されたことを知らせるもの）の GLIA 版と Swing 版を**図 5** に対比させる．GLIA のクラスライブラリの読み込み指定を除いた場合の違いは次の 3 点である．(1) SwingSample では JFrame を使用し，GLIASample では CollaboFrame を使用する．(2) GLIASample では適当なマウス番号を CollaboFrame から取得する．(3) GLIASample ではマウスがクリックされたときに呼び出される mouseClicked メソッド内で，マウスイベントからマウス番号を取得して，theMouseID と比較している．GLIA 版のプログラム例を見ると，JButton のような Swing 部品の再利用ができることや，Swing と同じ方法でイベントリスナを GUI 部品へ登録できることが分かる．したがって，既存の Swing アプリケーションを再利用した GLIA アプリケーションの開発が期待できる．

また，GLIA では，マウスのイベント情報に，イベントを発生したマウスの名前（番号）を含んでいるので，その名前情報をもとに GUI レベルでのアクセス

```
import glia.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class GliaSample extends MouseAdapter{
   long theMouseID;
   public GliaSample(){
      JButton b = new JButton("click");
      b.addMouseListener(this);
      CollaboFrame frame = new CollaboFrame();
      frame.getContentPane().add(b);
      frame.pack();
      frame.setVisible(true);
      Mouse[] list = frame.getMouses();
      theMouseID = list[0].getID();
   }
   public void mouseClicked(MouseEvent e){
      GliaMouseEvent ge = (GliaMouseEvent)e;
      if (ge.getMouseID() == theMouseID){
         System.out.println("clicked!");
      }
   }
   public static void main(String[] args){
      new GliaSample();
   }
}
```

(a) GLIA版のGUIプログラム例

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingSample extends MouseAdapter{

   public SwingSample(){
      JButton b = new JButton("click");
      b.addMouseListener(this);
      JFrame frame = new JFrame();
      frame.getContentPane().add(b);
      frame.pack();
      frame.setVisible(true);
   }


   public void mouseClicked(MouseEvent e){
      System.out.println("clicked!");
   }



   public static void main(String[] args){
      new SwingSample();
   }
}
```

(b) Swing版のGUIプログラム例

図 5 GLIA による JAVA アプリケーション開発
Fig. 5 A Java application development by GLIA.

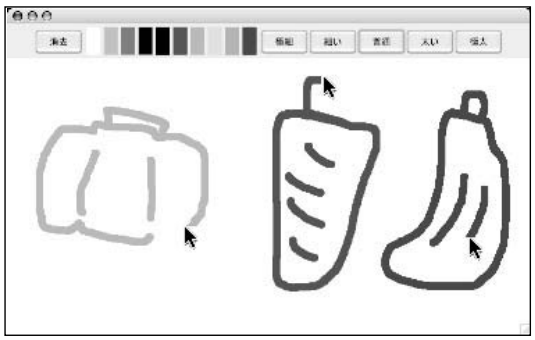


図 6 複数マウスの同時描画機能を持つペイントソフト
Fig. 6 Paint software with a simultaneous drawing function by multi mice.

制御を付録 A1 のように記述できる．そして，**図 6** に示すような複数の利用者がマウスで同時に描画操作を行えるペイントソフトも開発できる．この図 6 では，3 人の利用者がそれぞれカーソルを同時に動かして絵を描いている様子を示している．各々が，線の色，太さをそれぞれ決めることができる．このペイントソフトは，複数のウィンドウ利用時でも使用でき，他のウィンドウ上にカーソルが移動しても，同じ色と太さで描画を続けることを実現できた．そのためには，マウスエージェントに線の色や太さの情報をオブジェクトとして追加・取得のメソッドを使用している．さらに，GLIA では，テニスゲームのテニスボールのように頻繁にウィンドウ間で移動するボールを付録 A2 のようにオブジェクト通信用メソッドを用いて記述できる．

3. 評価実験

3.1 大画面共同作業環境の構築実験

GLIA を用いて大画面の共同作業環境の構築の可能性を調べるために，分散協調型 KJ 法[10] を支援する KUSANAGI の GLIA 化とスライディングパズルを行えるアプリケーションを開発し，試用する．ここで，GLIA 化とは既存の Java で開発されたアプリケーションを GLIA 上で動かせるアプリケーションに修正することを指す．

KUSANAGI は，日本の代表的な発想法の 1 つである KJ 法を参考にした分散協調型 KJ 法を支援するグループウェア GUNGEN[10] を Java によって実装したものである．分散協調型 KJ 法では，ブレインストーミングの精神に則って意見を出す意見入力の作業後，出されたすべての意見を吟味してグループ編成する島作成を行う．KUSANAGI は，従来の GUNGEN と比べて仮想画面機能により複数の画面を取り扱える機能を備えており，画面の一覧性を支援していた．たと

えば，3 台の計算機を横に並べた場合，各計算機に別々の領域を表示でき，3 画面規模の作業空間を同時に閲覧可能であった．しかしながら，ネットワークを介したマウス操作が行うことができないため，意見の移動が別計算機の画面領域に及ぶとき，いちいち別計算機にあるマウスに手を切り替える必要があり実用的ではなかった．もちろん，同時に複数の人がマウスを使用することもできなかった．そこで，開発した GLIA を使用して KUSANAGI を改良することによって分散協調型 KJ 法を大きな画面を使用し，複数人での作業を試みた．具体的には，島根大学の大学院生 2 人，学部生 1 人の計 3 人で意見ラベル 211 個を用いた場合の実験を行った．この共同作業は島根大学の構内で行われ，計 5 台の eMac（Apple Computer 製，CPU は PowerPC G4 であり，1.4 GHz が 1 台，1.25 GHz が 3 台，700 MHz が 1 台）を使用した．すべての eMac は OS が MacOSX10.4，ディスプレイの大きさは 17 インチ，解像度は横 1152 画素×縦 864 画素であった．

スライディングパズルは，“15 パズル” の名で知られる左上から右下へ順番に数字を並べるパズルゲームで，各計算機上にあるウィンドウを 1 マスとして利用できるものである．行列指定方式で長方形状に関連付けられたウィンドウ設定であれば，計算機環境が許す限り何台でも動作する．このパズルゲームは計算機演習室のような環境での使用を想定しており，マウスカーソルの画像を大きな円（横 480 画素×縦 480 画素）に変更したものを用いている．このパズルが島根大学総合理工学部数理・情報システム学科の AV 実習室にある計算機 70 台で動くかどうかを確認した．この計算機はすべて同一機種であり CPU は Intel Celeron 2.66 GHz，OS は Windows XP，ディスプレイの大きさは 17 インチ，解像度は横 1280 画素×縦 1024 画素であった．また，使用したネットワークケーブルは 100base-T であった．

3.2 ネットマウスの通信性能の測定実験

GLIA の基本機能であるネットマウスの通信性能を調査するために，マウスのイベント通信およびマウスエージェントの移動にかかる時間を計算機 50 台を使用して測定した．マウスのイベント通信の性能を測定するためには，同一ウィンドウにあるマウスエージェントを別な計算機から操作した場合の通信時間を測定し，そのマウスエージェント数の影響を調べた．これは遠隔にあるカーソルを動かすためのマウスイベント通信時間を測定したことになる．そして，マウスエージェントの数は同時に利用しているネットマウスの数に対応し，共同作業者の数と対応付けることができる．また，マウスエージェントの移動時間については，同時に移動するマウスエージェント数の影響に加えて，マウスエージェントが移動するホスト数の影響を調べた．ここで，移動するホスト数は別な共同作業用ウィンドウに移動した数であり，ネットマウスで利用した共同作業空間の大きさと関係がある．

一方，ネットワークを介したマウスの移動によって生じる通信遅延がユーザの操作感に及ぼす影響を測定した．ユーザは 2 つの計算機上に表示されたウィンドウ間でカーソル移動した場合の違和感を 4 段階で評価した．その際，ネットマウスの移動にかかる通信処理に加えて人工的な遅延を加えた．その遅延は，0 ms から 400 ms の範囲で，20 ms 刻みで変化させた．実験の参加者は，島根大学の大学院生 4 人，学部生 4 人の計 8 人である．実験参加者の半分は，0 ms から遅延を 20 ms 刻みで増やすことによって，その他半分は，400 ms から遅延を 20 ms 刻みで減らすことによって測定した．

3.3 Java Swing アプリケーションの GLIA 化実験

すでにある Java の Swing アプリケーションを GLIA アプリケーションに移植した開発について検討する．そこで，そのような Java アプリケーションである KJ 法支援ソフト KUSANAGI，ペイントソフト，オセロゲーム，○×ゲームを GLIA 化する作業を行った．

4. 適用結果と考察

4.1 大画面共同作業環境の構築

GLIA を用いて大きな画面で共同作業を可能とした KUSANAGI を利用した分散協調型 KJ 法を実行した様子を**図 7** に示す．このシステムでは，参加者それぞれが 1 つのカーソルを用いて同時に複数のラベルを移動させることができた．ただし，複数の人が同じ意見を移動することがないように，同時に 1 人しか意見ラベルを動かせないアクセス制御を実現している（アクセス制御の実現方法は付録 A1 を参照）．この図 7 では，1 画面では収まらない 211 個の意見ラベルを取り扱い，3 人の参加者が 5 つの PC を使用した共同作業を行っている．白い長方形で表示されているものが意見ラベルである．この結果では，島作成にかかった時間は 64 分であった．

この島作成時間を，1 人で行った場合，3 人で行った過去の GUNGEN-DXII などの実験結果と比べると作業時間が短縮される結果となった．具体的には，島根大学の教員 A が 1 人で同一の意見データ 211 個で同じ環境で KUSANAGI を用いた分散協調型 KJ 法

図 7 大きな共同作業空間を持つ分散協調型 KJ 法支援グループウェア KUSANAGI の画面例
Fig. 7 An example screen of KUSANAGI: a groupware for the distributed and cooperative KJ method with a large collaboration space.

図 8 計算機 70 台を用いたスライディングパズル
Fig. 8 A sliding puzzle with 70 computers.

の作業を行った場合の島作成時間は 175 分である．別テーマであり，意見データ 287 個を用いて 3 人による分散協調型 KJ 法を行った場合，GUNGEN-DXII によるテトリス型インタフェースを用いた場合 235.7 分，通常の GUNGEN（これは共同作業画面に同時に 1 人しかアクセスできない制御を行っていた）を用いた場合，355.3 分であった[12]．したがって，大きな画面を用いた分散協調型 KJ 法を複数人で同時作業を可能とした KUSANAGI の GLIA 化は時間効率面で有効に作用したと考えられる．

GLIA を用いて開発されたスライディングパズルを 70 台の計算機上で動作させた様子を**図 8** に示す．このパズルを 2 つのネットマウスを使って動作を確認できた．図 8 の “16” と表示されているディスプレイの右横のディスプレイに表示された黒丸が大きな画像を用いたカーソルである．計算機 70 台を横 10 列，縦 7 台の組合せで動作させており，横 12800 画素×縦 7168 画素の大画面を GLIA で取り扱えることが分かった．

4.2 ネットマウスの通信性能

ネットマウスの通信性能としてマウス操作によって生じたカーソル移動を他計算機のカーソルに反映されるイベントの通信時間およびマウスエージェントが別な計算機上のウィンドウに移動する時間を測定した結果が**表 2** である．また，通信遅延時間が被験者の感じる遅さに及ぼす影響を**表 3** に示す．その通信遅延時間については，マウスエージェントが 2 台の計算機を移動するときの通信時間が 50 ms であるので，その半分の 25 ms を片方向の通信時間として近似した値をベース時間とし，その時間に人工的な遅延値を加えたものを遅延時間とした．

表 3 における違和感の値を四捨五入した値とし，遅延時間との関係を見ると，遅延時間が 65 ms までは，被験者はマウスの反応に違和感を感じず，遅延時間が 125 ms までは少し違和感を感じるという結果となった．次に，この結果を表 2 にあてはめて検討する．遠隔からのマウス操作の際，マウスエージェントの移動がなければ（マウスイベントの通信のみの場合），10 個のマウスエージェントの場合でも，違和感を感じない 22 ms の通信性能がでており，10 人がマウス操作を行っていても違和感を感じないと推測される．また，マウスエージェント 1 個が 2 台の計算機を移動する

表 2 ネットマウスの通信時間
Table 2 Communication time of networked mice.

		マウスエージェント数（カーソルの同時操作または同時移動に対応）				
		1 個	2 個	3 個	5 個	10 個
イベント通信のみ		21 ms**	21 ms**	21 ms**	22 ms**	23 ms**
マウスエージェントの移動ホスト数	2 台	50 ms**	83 ms*	97 ms*	157 ms	302 ms
	3 台	66 ms*	92 ms*	119 ms*	182 ms	351 ms
	5 台	86 ms*	120 ms*	161 ms	249 ms	480 ms
	10 台	152 ms	201 ms	266 ms	404 ms	769 ms
	20 台	281 ms	371 ms	465 ms	713 ms	1630 ms
	30 台	377 ms	501 ms	686 ms	1246 ms	2311 ms
	40 台	463 ms	653 ms	923 ms	1591 ms	2910 ms
	50 台	557 ms	867 ms	1092 ms	1943 ms	3806 ms

**違和感なし：65 ms 以下，*少し違和感：125 ms 以下

表 3 マウスエージェント移動の遅延時間の印象
Table 3 User's impression of delay by the movement of a mouse-agent.

遅延時間 (ms)	遅延印象
25	1.0
45	1.1
65	1.1
85	1.5
105	2.0
125	2.3
145	2.6
165	2.9
185	2.9
205	3.0
225	3.0

1：違和感なし，2：少し違和感
3：違和感あり，4：大変違和感

場合（移動ホスト数 2 台の場合）も違和感を感じない 50 ms で通信が行えていることが分かる．次に，やや違和感を感じるケースについて見ると，同時に 2 個のマウスエージェントが 5 台の計算機を移動するとき（移動ホスト数が 5 台の場合），120 ms の時間がかかっており，最も多くの通信処理を行っているケースである．

以上より，マウスエージェントの移動が重なる可能性が低く，かつ，5 台を超えるマウスエージェントの移動が起こることが少なければ，10 人規模の利用者がいても，通信遅延に違和感を感じない GLIA のネットマウス利用が期待できる．

4.3 Java Swing アプリケーションの GLIA 化

既存の Java Swing アプリケーションを GLIA アプリケーションに移植した結果を**表 4** に示す．表 4 には，元の Swing アプリケーションのプログラム行数と GLIA 化した後のプログラム行数を示すとともに，その変更行数とマウスイベント処理数を示している．

表 4 Java Swing アプリケーションの GLIA 化にかかったプログラム行数
Table 4 The number of code lines of programming for developing a GLIA application by modifying a Java Swing application.

アプリケーション名	Swing	GLIA 対応後	マウスイベント処理数	変更行数
KUSANAGI	8002	8482	12	480(6%)
ペイントソフト	181	227	5	60(30%)
オセロ	723	733	1	14(2%)
○×ゲーム	133	137	1	8(6%)

GLIA のネットマウスを利用するにはマウスイベント処理の書き換えが必要であり，基本的に Swing アプリケーションを GLIA へ移植するための変更はマウスイベント処理関連に集中した作業となる．したがって，ペイントソフトはマウスイベント処理数が多く，その内部でマウスごとの複雑な処理を行っているために変更行数が多い．同様に，KUSANAGI もマウスイベント処理数が多いことに加えて，マウスイベント処理に関連するドラッグ継続機能を実装しているために変更行数が多くなっている．ここでドラッグ継続機能とは画面間で GUI 部品をドラックして移動できる機能である．

今後は，すでにある Java アプリケーションの GLIA 化を支援するために，コンパイラ技術の応用が期待される．具体的には，マウスのイベント処理部分を解析し，そのイベント処理部分を GLIA のマウスイベント処理に変換するなどのパッチ処理を行うことが検討課題としてあげられる．

5. おわりに

ネットワークを介した複数のマウス操作を複数の計算機上にあるウィンドウ上での操作を可能とするミドルウェア GLIA を実現することにより，大きな画面で

の共同作業を実現可能とした．GLIA では，Java の Swing を拡張することにより，単にマウスカーソルを表示するだけでなく，マウスの同時操作や GUI レベルのアクセス制御を支援している．GLIA を用いて，複数の共同作業アプリケーション（分散協調型 KJ 法支援ソフト，同時描画が可能なペイントソフト，70 台規模の 15 ゲームなど）の構築を評価するとともに，ネットマウスの通信性能を測定した．これらの結果より，次のようなことが分かった．

(1) 全部で 70 台つなぐことができるとともに，KJ 法のような画面の一覧性が重要とされる共同作業における効果が期待できる．
(2) ネットマウスの利用者が 10 人でも，同時に越えるマウスエージェントの移動が少なければ，カーソルの連続移動が 5 台まではマウスの遅延を許容できる．
(3) Java アプリケーションを GLIA 化するには，Swing の GUI イベントを主に追加プログラミングすればよい．

今後の予定としては，GLIA をより有効活用できる共同作業アプリケーションの事例を増やすこと，開発支援環境の充実，および，遠隔地間での共同作業を支援するために画面共有機能を設けることを検討している．

謝辞 本研究の一部は日本学術振興会科学研究費補助金（基盤研究（B），研究課題番号 18300043,「センサーと絵文字によるチャットコミュニケーションが相互の理解度向上に及ぼす影響」）による．また，本研究を行う場を提供していただいた島根大学加藤裕一教授，高橋正和准教授，および，評価実験に参加していただいた加藤研究室の皆様に深く感謝いたします．

参 考 文 献

1) Stefik, M. and Brown, J.S.: *Toward Portable Ideas, in Technological Support for Work Group Collaboration*, Olson, M.H. (Ed.), pp.147–165, Lawrence Erlbaum Associates (1989).
2) Russel, D., Streitz, N. and Winograd, T.: Building Disappearing Computers, *Comm. ACM*, Vol.48, No.3, pp.42–48 (2005).
3) Streitz, N., et al.: i-LAND: An interactive Landscape for Creativity and Innovation, *Proc. CHI'99*, pp.120–127 (1999).
4) Greenberg, S.: Enhancing Creativity with Groupware Toolkits, *Proc. CRIWG 2003*, LNCS, Vol.2806, pp.1–9, Springer-Verlag (2003).
5) Tandler, P.: Architecture of BEACH: The software infrastructure for roomware environments, *CSCW 2000, Workshop on Shared Environments to Support Face-to-Face Collaboration* (2000).
6) Johanson, B., Fox, A. and Winograd, T.: The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms, *IEEE Pervasive Computing* 1:2, pp.67–75 (2002).
7) Myers, B.A., Stiel, H. and Gargiulo, R.: Collaboration Using Multiple PDAs Connected to a PC, *Proc. CSCW'98*, pp.285–294, ACM Press (1998).
8) Munemori, J., Takahiro, N. and Yoshino, T.: Group Digital Assistant: Combined or Shared PDA Screen, *Proc. ICDCS'04*, pp.682–689 (2004).
9) 川喜田二郎：発想法—混沌をして語らしめる，中央公論社 (1986).
10) Munemori, J. and Nagasawa, Y.: GUNGEN: Groupware for new idea generation support system, *Information and Software Technology*, Vol.38, No.3, pp.213–220 (1996).
11) Ohiwa, H., Takeda, N., Kawai, K. and Shimomi, A.: KJ editor: A card-handling tool for creative work support, *Knowledge-Based Systems*, Vol.10, pp.43–50 (1997).
12) 重信智宏，吉野 孝，宗森 純：GUNGEN DX II：数百のラベルを対象としたグループ編成支援機能を持つ発想支援グループウェア，情報処理学会論文誌，Vol.46, No.1, pp.2–14 (2005).
13) Wallace, G., et al.: Tools and Applications for Large-Scale Display Walls, *IEEE Computer Graphics and Applications*, June/August, pp.24–33 (2005).
14) Wallace, G., Peng, B., Li, K. and Anshus, O.: A MultiCursor X Window Manager Supporting Control Room Collaboration, Princeton University, Computer Science, Technical Report TR-707-04 (2004).
http://multicursor-wm.sourceforge.net/
15) http://dmx.sourceforge.net/
16) シュナイダーマン，B.：ユーザーインタフェースの設計，第 2 版，日経 BP (1993).
17) Walrath, K., Campione, M., Huml, A. and Zakhour, S.: *JFC Swing Tutorial, The: A Guide to Constructing GUIs, 2nd Edition*, Addison-Wesley (2004).
18) Arnold, K., Gosling, J. and Holmes, D.: *The Java Programming Language, 3rd Edition*, Addison-Wesley (2000).
19) https://jinput.dev.java.net/

付　　録

A.1 GUI レベルのアクセス制御

GLIA で操作権を実現するプログラム例を**図 9** に示す．これは，GLIA 化した KUSANAGI で実際に使用しているプログラムの断片で，1 つのマウスのみが操作権を持てる意見ラベルである．操作権は変数 authoredMouseNumber にマウス番号を代入したマウスが持つ．マウス番号は，GMouseEvent クラスの，getMouseID メソッドで取得できる．以下詳しくプログラムを説明する．

コンストラクタ内ではマウスリスナを登録している．これにより，KJLabel へマウス入力があったときにマウスイベントが mousePressed メソッドや mouseReleased メソッドに送信されるようになる．

意見ラベルがマウスで押されたときに呼ばれる mousePressed メソッド内の動作は次のようになっている．(1) まず MouseEvent を GMouseEvent へキャストする．(2) 次に，すでに他のマウスが操作権を持っているいる場合（lockedMouseNumber != 0），すぐにメソッドから抜ける．(3) そうでない場合，操作権を取得し（lockedMouseNumber へマウス番号を代入して）処理を続ける．

```
public class KJLabel extends JLabel implements
MouseListener {
  public KJLabel (){
    addMouseListener(this);
  }
  long authoredMouseNumber = 0;

  public void mousePressed(MouseEvent e){
    GMouseEvent ge = (GMouseEvent)e;
    if (authoredMouseNumber != 0){
      //既に他のマウスが操作権を持っている.
      return;
    }
    authoredMouseNumber = ge.getMouseID();
    //マウスイベント処理
    .....
  }

  public void mouseReleased(MouseEvent e){
    GMouseEvent ge = (GMouseEvent)e;
    if (authoredMouseNumber != ge.getMouseID()){
      //他のマウスが操作権を持っている.
      return;
    }
    //マウスイベント処理
    .....
    authoredMouseNumber = 0;
  }
  //その他のメソッド
  .....
}
```

図 9 操作権の仕組みを持った GUI 部品のプログラム例
Fig. 9 A code for an access control of a GUI component.

意見ラベルからマウスが離されたときに呼ばれる mouseReleased メソッド内の動作は次のようになっている．(1) まず MouseEvent を GMouseEvent へキャストする．(2) 次に，すでに他のマウスが操作権を持っている場合（lockedMouseNumber != ge.getMouseID()），すぐにメソッドから抜ける．(3) そうでない場合は処理を続け，最後に操作権を手放す（lockedMouseNumber の値を 0 にする）．

A.2 複数の共同作業用ウィンドウ間でのデータ送受信

複数の共同作業用ウィンドウ間でデータを交換するプログラムを記述したい場合がある．たとえば，複数のウィンドウを用いて動作するテニスゲームの場合，テニスボールの画像を計算機間で移動させる必要がある．このようなことは，複数のウィンドウを使用しているときに頻繁に発生するので，GLIA は各計算機上で動作しているアプリケーションどうしがオブジェクトデータを交換できる機能を用意した．

図 10 に複数のウィンドウ間でオブジェクトデータを交換するプログラムについて説明する．送信側プログラムで，CollaboFrame オブジェクトの migrageObject メソッドを用いてボールオブジェクトを送信している．送信方向は，migrageObject の第 1 引数へ Direction クラスを渡すことで指定する．この例では，Direction.LEFT が渡されているので，左に接続されたウィンドウへそのオブジェクトを送信する．

受信側のプログラムでは，まずコンストラクタで CollaboFrame オブジェクトへ Receiver オブジェク

```
  CollaboFrame frame = new CollaboFrame();
  TennisBall ball =   …    //送りたいボールオブジェクト
  frame.migrageObject(Direction.LEFT, ball);

  (a)ボールオブジェクトを送信するプログラム

public class Receiver implments MigrateListener {
    public Receciver(){
      CollaboFrame frame = new CollaboFrame();
      frame.addMigrateListener(this);
    }
    public void objectMigrated(MigrateEvent e){
      TennisBall ball= (TennisBall)e.getMigratedObject();
      Direction from = e.getFromDirection();
      //送られてきたボールオブジェクトを処理する.
    }
}
  (b)ボールオブジェクトを受信するためのプログラム
```

図 10 共同作業用ウィンドウ間でのオブジェクト通信のプログラム例
Fig. 10 A code for exchanging an object data between windows for a collaboration space.

トを MigrateListener として登録している．これによって接続されたウィンドウからボールオブジェクトを受信できる状態になる．GLIA が隣のウィンドウからそのオブジェクトを受信すると，Receiver オブジェクトの objectMigrated メソッドが呼び出される．objectMigrated メソッド内では，MigrateEvent オブジェクトが持つ getMigratedObject メソッドによる受信オブジェクトの取得，getFromDirection メソッドでによる送信方向の取得を行う．その後，取得情報をもとに適切な処理を記述する．

(平成 18 年 10 月 31 日受付)
(平成 19 年 4 月 6 日採録)

西村　真一（正会員）

2005 年島根大学総合理工学部数理・情報システム学科卒業．同年同大学院総合理工学研究科博士前期課程入学，2007 年同課程修了．現在，株式会社ブリッジコーポレーションに勤務．2006 年 DICOMO 優秀論文賞を受賞．ミドルウェアの研究に従事．

由井薗隆也（正会員）

1999 年鹿児島大学大学院理工学研究科システム情報工学専攻博士課程修了．同年同大学工学部情報工学科助手．2002 年島根大学総合理工学部数理・情報システム学科講師，同学科助教授を経て，2006 年より，北陸先端科学技術大学院大学知識科学研究科准教授．博士（工学）．2005 年 KES'05 Best Paper Award，2006 年 DICOMO2006 優秀論文賞をそれぞれ受賞．グループウェア，知識メディア，システムソフトウェア等の研究に従事．ACM，IEEE，電子情報通信学会，ソフトウェア科学会各会員．

宗森　純（正会員）

1984 年東北大学大学院工学研究科電気及通信工学専攻博士課程修了．工学博士．同年三菱電機（株）入社．鹿児島大学工学部助教授，大阪大学基礎工学部助教授，和歌山大学システム情報学センター教授を経て，2002 年同大学システム工学部デザイン情報学科教授．2005 年システム情報学センター長（兼務）．1997 年度本会山下記念研究賞，1998 年度本会論文賞，2002 年 IEEE-CE Japan Chapter 若手論文賞，2004 年度本会学会活動貢献賞，2005 年，2006 年 DICOMO 優秀論文賞，2005 年 KES'05 Best Paper Award をそれぞれ受賞．本会論文誌編集委員会ネットワークグループ主査等を歴任．現在，グループウェアとネットワークサービス研究会主査．グループウェア，形式的記述技法，神経生理学等の研究に従事．IEEE，ACM，電子情報通信学会，人工知能学会，オフィスオートメーション学会各会員．