Vol.2009-HCI-133 No.1
2009/5/15

ネットワーク対応マルチカーソルを実現したミドルウェアGLIAの応用と今後

由井薗　隆也[†1]　西村　真一[†2]

大画面共同作業環境を構築するためにミドルウェア GLIA を開発してきた．ミドルウェア GLIA は，複数 PC 上に表示されたウィンドウ画面をネットワークマウスによって操作することによって大画面環境を構成可能とすると共に，複数のマウスに対応することによってマルチユーザの操作環境を提供する．その応用として，共同作業支援システム，簡単なゲーム，様々なディスプレイ配置について紹介する．

Development Middleware GLIA for Network Connected Multi Cursors and its Applications

Takaya Yuizono[†1] and Shinichi Nishimura[†2]

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. We introduce examples of GLIA applications, such as collaboration support systems with a big screen, multi-user games, and various display arrangements.

1. はじめに

近年，コンピュータネットワーク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上でも重要である[4)]．

一方、我々は，衆知を集める発想法として知られた KJ 法[5)⋆1]を支援する発想支援グループウェア郡元の研究を長年行ってきた[6)]．その際，開発環境として，開発効率がよくカードシステムと相性のよいメタファを備えた HyperCard（Apple Computer）を用いてきた．そして，分散環境でのグループウェアの効果について明らかにしてきた．しかしながら，その開発環境は，マルチスレッドに未対応であり，単一計算機を用いた場合にマルチカーソルを実現することが困難であった．また，使用できる画面に制限があり物理的に大きな大画面共同作業環境を構築することも困難であった．よって，進んだ共同作業環境を実現するためには新たな開発環境を検討することにした．

そこで，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) を提案し，開発した[7)]．この GLIA を用いると複数のウィンドウ結合が利用でき，平面を３次元空間上に展開する表現を含むアプリケーションなどへの応用も期待できる．現在，発想支援グループウェア郡元を発展させた大画面共同作業インタフェースをもつグループウェア KUSANAGI を実現し，数百のデータを用いた共同作業を支援することを可能にしている[8)]．また，GLIA の複数画面を結合させる機能の可能性を探るための様々な応用を検討している．

本報告では，２．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．で，GLIA を用いたアプリケーション開発について述べ，４．では，共同作業支援システムや単純なゲーム，様々なディスプレイ配置への応用について紹介する．

2. ミドルウェア GLIA

2.1 設計方針

GLIA はマウス等の入力デバイスによる，ネットワークを介した入力操作を可能にするこ

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology
†2 株式会社ブリッジコーポレーション
Bridge Corporation Inc.

⋆1 「KJ 法」は，株式会社川喜田研究所の登録商標である．

とで，複数の計算機で動作しているウィンドウを結合し，大きな共同作業空間の構築を支援するミドルウェアである．GLIA 内部は**図 1** に示ように，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ネットワーク通信処理，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 処理が連携し動作している．このミドルウェア上で動作する GLIA アプリケーションを，GLIA が提供する支援機能を利用して開発することで，大型の共同作業空間を実現できる．**表 1** に実現機能の一覧を示す．

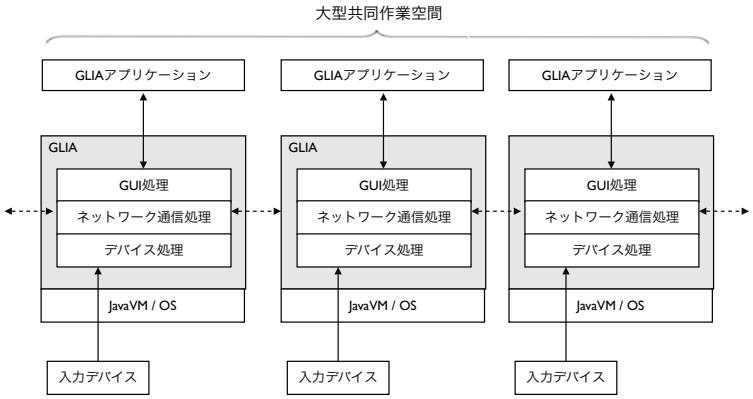

図 1 GLIA のシステム構成

GLIA は Java1.5(Sun Microsystems 製)[9] で開発され，総プログラム行数は約 5 千行，総クラス数 155（その中，内部クラス数 35）であった．その他，OS 依存の処理を用いるために JNI(Java Native Interface)[9] と C++を用いた数十行規模のプログラムが存在する．複数の入力デバイス装置から直接データを取得するために，Java の API である JInput[10] を使用している．また，エージェント移動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[9] のデータ通信を実現している．

GLIA が動作可能な環境は，GLIA の実装言語である Java が動作する MacOSX(Apple Computer 製)，Windows OS (Microsoft 製)，各種 Unix であり，異機種 OS 間で相互にネットワークを介した入力操作ができる．ただし Unix では，デバイス入力を取得する部分の実装が不完全な部分があるため，１台の計算機に複数のマウスを接続することができない．

表 1 GLIA の支援機能一覧

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.
複数マウス接続	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 に Swing を用いたアプリケーションを GLIA へ容易に移植可能にする.
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みえる.
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で複数ウィンドウ間での多様なデータ通信を支援.

2.2 ネットワーク介した入力操作

GLIA の内部構成とデバイス情報送信の様子を図 1 を用いて説明する．最下部にあるデバイス処理層は，１つの入力デバイスにつき１つ生成される ”モニタ ”からなる．モニタは監視している入力デバイスから情報を取得してそれを送信する．モニタにはマウスモニタ，キーボードモニタ，PDA モニタ，ゲームパッドモニタがある．

中間部のネットワーク通信処理は，ネットワーク移動を行うエージェントと，エージェントを管理するエージェントマネージャから構成される．エージェントは，モニタが取得したイベントをネットワークを介して受け取るオブジェクトである．エージェントには，マウスに対応するマウスエージェント，ゲームパッドに対応するゲームパッドエージェントがある．最上部の GUI 処理は，共同作業ウィンドウへのマウスカーソル描画や，GLIA アプリケーションへのデバイス情報送信を行う．

ここでモニタとエージェントの接続関係を**図 2** に示す．図ではマウス入力操作にしぼって示している．GLIA 内部では，各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のペア部品が生成され，マウスモニタからネットワークを介してマウスエージェントへ入力情報の受け渡しが行われる．

ネットワークを介した入力操作を実現するために行っている処理の概要について述べる．

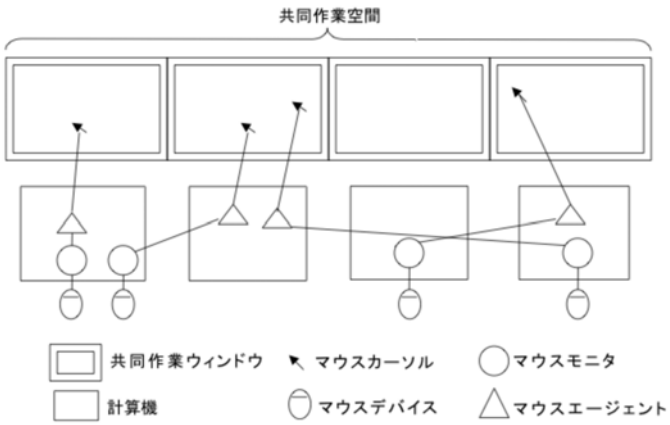

図 2 マウスエージェント，マウスモニタの接続関係

モニタの定期的な起動処理，ネットワークソケットを監視する処理，および，GUI 処理に取りかかる契機となるキューに入れられた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．

2.3 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関係などの情報は XML ファイルに記述する．

GLIA の通信プログラムは常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 >内に各計算機がもつウィンドウの情報を記入する．その中で，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．

3. GLIA アプリケーションの開発方法

GLIA のアプリケーション開発環境は，Java の標準 GUI 開発環境である Swing[11] を拡張したため，そのアプリケーション開発は Swing を用いた開発とほぼ同等な方法で行える．Swing と GLIA の GUI プログラム開発における第一の違いは，Swing では最上位ウィンドウの作成は JFrame クラスを使用するが，GLIA では共同作業ウィンドウの実装である CollaboFrame クラスを使用することである．CollaboFrame は JFrame を拡張したクラスで，JFrame と同等な機能を持ち，さらに複数マウスを扱うための様々な機能が追加されている．

GLIA アプリケーションには，Swing のほぼすべての GUI 部品を利用できる．ただし一部の GUI 部品は共同作業ウィンドウの外に表示されるため，GLIA のマウスを利用した操作に注意が必要である．これを避けるには，これらの GUI 部品を CollaboFrame 内に表示される部品として新たに再実装する必要がある．

```
1  import glia.*;
2  ...
3  class GLIASample implements MouseListener {
4
5    GLIASample(){
6      CollaboFrame frame = new CollaboFrame("CollaboFrame");
7      JButton button = new JButton("swing button");
8      button.addMouseListener(this);
9      frame.getContentPane().add(button);
10     ...
11 }
12
13   public void mouseClicked(MouseEvent e) {
14     System.out.println("clicked");
15   }
16 ...
17 }
```

(a) GLIAプログラム例

```
1  ...
2  class SwingSample implements MouseListener {
3
4    SwingSample(){
5      JFrame frame = new JFrame("JFrame");
6      JButton button = new JButton("swing button");
7      button.addMouseListener(this);
8      frame.getContentPane().add(button);
9      ...
10   }
11
12   public void mouseClicked(MouseEvent e) {
13     System.out.println("clicked");
14   }
15   ...
16 }
```

(b) Swingプログラム例

図 3 GLIA と Swing によるプログラム例

(a) GLIAプログラム例　　(b) Swingプログラム例

図 4　GLIA プログラム例と Swing プログラム例の実行画面

GLIA のプログラム例として，GUI ボタンを押すと標準出力へ文字列を表示するプログラムを**図 3**(a) に，またこれと同じ動作をおこなう Swing プログラム例を図 3(b) に示す．さらにこれらの実行画面を**図 4** に示す．

GLIA プログラム例の実行は，計算機にマウスを 3 つ繋げて起動したため，図 4(a) の実行画面にはマウスカーソルが 3 つ表示されている．GLIA プログラム例と Swing プログラム例とで異なる部分は，GLIA プログラム例 (図 3(a)) における 1 行目と 6 行目である．1 行目は GLIA を利用するために glia パッケージを読み込んでいる．6 行目は GLIA の共同作業ウィンドウを利用するために，JFrame クラスではなく CollaboFrame クラスを使用している．また同じく 7 行目では GLIA プログラム内で，Swing の GUI 部品である JButton を再利用している．このように GLIA プログラム内に Swing の GUI 部品を再利用できるため，Swing 資源の有効活用ができ，さらに Swing アプリケーションを GLIA へ移植するのが容易である．

また複数のマウスで操作される GLIA アプリケーションは，操作するマウスによって動作を変えることができる．これによって GUI レベルのアクセス制御を実現できる．操作するマウスによって動作を変更する GLIA プログラム例を**図 5** に示す．この図はマウスイベント処理部分にしぼって示している．1 行目の mouseClicked メソッドがマウスでクリック動作を行ったときに呼ばれる．2 行目では，受信した Swing のマウスイベント (MouseEvent) を GLIA のマウスイベント (GMouseEvent) へ型変換している．GLIA のマウスイベントには，操作したマウスの情報が付加されているため，3 行目のように GLIA のマウスイベントからマウス番号を取り出せる．

```
1 public void mouseClicked(MouseEvent e) {
2   GMouseEvent ge = (GMouseEvent)e;
3   System.out.println("clicked: mosueID:" + ge.getMouseID());
4 }
```

図 5　GLIA のマウスイベントを使用したプログラム例

4. GLIA による応用例

4.1 発想支援グループウェア KUSANAGI

複数の参加者が同時に共有データに対する操作ができるとともに，画面切り替えを行わずに数百枚のデータを眺めることができる発想支援グループウェア KUSANAGI を GLIA によって実現した[8]．その使用風景を**図 6** に示す．

図 6　分散協調型 KJ 法支援ソフト KUSANAGI の使用風景

この GLIA を用いた開発には，KUSANAGI で使用している JFrame クラスを CollaboFrame クラスに置き換えるだけでなく，マウスカーソルが隣の計算機画面に移動しても継続的な GUI 操作を実現する仕組みが必要であった．

一般的に GUI のプログラムを組む場合は，ユーザがどのような操作状態であるかの情報（以下，コンテキストと呼ぶ）を GUI 部品のプログラム内に混在していることが多い．そして，マルチユーザを想定せずにプログラム開発が行われており，1 つしかコンテキストを保持していないことも多い．そして，各計算機は，それぞれに接続されたマウスのコンテキストのみを保持している状態が多いと推測される．この状態で，ある計算機に隣の計算機で活動していたマウスエージェントが移動した場合（GUI 操作を別の計算機に移動した場合）に問題が生じうる．そのマウスエージェントが移動先にある別マウスのコンテキストを変更してしまう可能性や，また，そのエージェントのコンテキストを知らない移動先の計算機が適切な反応を行えない可能性がある．そこで，各マウスに操作コンテキストを付加すると共に，操作対象となる GUI オブジェクトはマウスのコンテキストに反応するプログラムに変更した．

この開発した KUSANAGI を約 300 枚の意見データを整理する作業に適用し，性能評価を行ってきた[8]．その際，従来システムを用いた作業結果や紙面上の作業と比較し評価した．その評価実験より得られた結果は次の通りである．（1）マルチカーソル機能によって，参加者による共有画面への操作密度が高くなり，数百枚のデータに対する島作成時間が短縮される．（2）大画面環境では，島の数は増加し，島に含まれる意見の一覧性は確保され，かつ，島作成に関する共有画面操作が増加しており，丁寧な島作成が行われる．（3）会話量は紙面上の共同作業と同等であると共に，島作成の時間効率が紙面上と比べて優れているという過去の発想支援グループウェアでは得られていない傾向がみられた．

また，この KUSANAGI を水平面上の共同作業に適用することを検討した様子を**図 7** に示す．その実現のためのハードウェアには小型 PC と三脚，液晶プロジェクタの計算機セットを 2 組使用している．この水平面型共同作業のための空間はプロジェクタと投射領域を離すことによって画面の大きさを拡大できる．一方，その画面解像度を用いた場合，支援対象として適切なラベルデータ数は百枚規模と推測される．

水平面上に画面を投射した場合，ユーザが画面の手前にいる場合だけでなく四方向にいる場合も検討する必要がある．この際，ペンやタッチスクリーンによる入力を用いる場合と異なり，マウスによる入力の場合では，マウスの移動方向をユーザの向きに対応させる必要がある．そこで，GLIA にマウスの移動方向を 90 度刻みの特定角度で，自動加算する機能を追加している．

4.2 ゲーム作品への応用

GLIA を使用したリアルタイム性を要求されるアプリケーションの例として，テニスゲーム（**図 8**）及びアクションゲーム（**図 9**）を作成した．

図 7 水平面共同作業環境の検討

テニスゲームは複数の共同作業ウィンドウを利用して複数人で遊べるゲームである．マウスカーソルは長方形のラケット画像に変更しており，それを利用して画面上を移動するテニスボールを打つことができる．またラケットはディスプレイ上のどこにでも移動できるため，大画面の全体を使ったプレイができる．図では，左側のプレイヤーが真ん中のディスプレイにラケットを移動して積極的な攻撃を行っている様子である．

アクションゲームでは，入力デバイスはマウスではなくゲームパッドを使用し，ゲームパッド入力に応じてキャラクターが複数のウィンドウ間を移動できる．図はゲーム開始前の様子で，ゲームステージが 4 つのディスプレイで構築されている．またここではゲームパッドは 3 つ接続されており，3 人でプレイすることが可能である．

これらゲームは，ゲームクリアの条件を含むシナリオ設定されておらずエンターティメント設計が不十分である．ただし，対戦方式のプレイは行いやすいため，複数人参加による楽しみの効果が期待できる．

Vol.2009-HCI-133 No.1
2009/5/15

図 8 テニスゲーム

図 9 アクションゲーム (複数ゲームパッド使用可能)

図 10 計算機７０台を用いたスライディングパズル

スライディングパズルを 70 台の計算機，横 10 列＊縦７台の組み合わせで動作させた様子を**図 10** に示す．図 10 中央から右に「16」と表示されたディスプレイが見えるが，その右横にあるディスプレイに表示された黒丸はマウスカーソルである．このように GLIA では応用に合わせてカーソル画像を変更することができる．この応用例より，GLIA を用いると数十台の計算機を用いた画面構成を利用できることがわかる．今後は，パズルを巨大化しただけでなく，空間配置を活かした応用を検討する予定である．

4.3 ディスプレイ構成の検討

様々なディスプレイ構成を検討するために，多人数で操作できる描画アプリケーションを用いた．このアプリケーションを用いた場合，複数の計算機上で動作する共同作業ウィンドウを結合して，いろいろな形状のキャンバスを構成できる．各ユーザは接続された複数のマウスを用いて，このキャンバス上に独立かつ同時に線を描画できる．

L 字型のディスプレイを構成した様子を**図 11** に示す．このようなディスプレイ構成は，2.3 で述べた接続指定方式と行列指定方式のどちらでも可能である．ただし，接続指定方式のほうが，より柔軟な接続が可能であり，左上にあるウィンドウの上部分と右下にあるウィンドウの下部分を関連付けることができる．

GLIA の通信プログラムは，その起動時には隣に関連づけられたウィンドウに通信接続を

図 11 L 字型ディスプレイ

試みる．よって，隣同士の接続状態さえ記述できれば動的なディスプレイ配置も可能である．図 11 に示すディスプレイ配置の場合，隣同士にあるウィンドウの接続情報さえ XML 設定ファイルに記述してあればよく，左上の描画領域は，右下部分の描画領域に関する計算機情報を知っている必要はない．この機能を用いれば，物理資源を問題なく接続できる限り，描画領域を上下左右どこにでも拡張することができる（もちろん，ディスプレイを物理空間上に自由配置するためには適切な機材が必要となる.）.

描画領域を平面ではなく，３次元的に展開した例としてディスプレイを三角形状に配置した様子を**図 12** に示す．このケースの場合，有線マウスではディスプレイを一周することはできず，無線マウスを使用する必要があった．また，各計算機の配線が邪魔にならないように隠す必要もあった．一方，ディスプレイの投射は水平面でなく，垂直方向であるので，ユーザの移動方向とマウスの方向の整合性に問題は起こらなかった．もちろんユーザはすべての面を同時に見ることはできない．

メディアート作品として，PC 画面をロボット形状に並べたディスプレイ配置を検討したものが**図 13** である．複数の参加者が各マウスカーソルを用いてロボット形状の描画領域に絵を描くことができる．また，マウスのカーソルには感情表現に用いられるエモティコンを使用し，その感情に相応しいと思われる色を対応づけている．例えば,（^^）の場合は黄色で,(TT) の場合は青色（ブルー）である．この作品は GLIA を用いると，ロボット形状の

図 12 トライアングル配置ディスプレイ

ディスプレイ配置を実現可能なことを示している．ただし，作品としての更なる改善が必要である．具体的には，作品がもつコンセプトとそれを支えるインタラクティブな表現を明確にすることが重要と考える．

最後に，GLIA を利用したアプリケーションでは，使用する計算機ごとにそれぞれのアプリケーションを起動する必要があり，数十台規模のディスプレイを使用する場合，台数に応じた手作業に時間がかかるという問題がある．また，一端，アプリケーションを起動した場合，その接続を変更する方法が提供されていないために，動的なウィンドウ結合の制限となっている．これら課題を改善し，GLIA をより使いやすくするために，分散資源管理のための名前サーバ開発などの分散システムの基本技術[12] を追加することが必要である．

5. おわりに

大画面共同作業環境を構築することを目的としてミドルウェア GLIA を開発してきた．ミドルウェア GLIA は，複数 PC 上に表示されたウィンドウ画面をネットワークマウスによって操作することによって大画面環境を構成可能とすると共に，複数のマウスに対応することによってマルチユーザの操作環境を実現している．そのミドルウェア GLIA を用いて大量データをまとめる共同作業を支援する発想支援グループウェア KUSANAGI を実現し，従来の発想支援グループウェアと比べて優れた共同作業環境を実現した．

一方，GLIA の柔軟な画面結合を活かした応用についても検討を進めてきた．具体的には，テニスゲーム，アクションゲームやパズルなどを複数画面で行えるものを開発した．ま

Vol.2009-HCI-133 No.1
2009/5/15

図 13 インタラクティブアート作品への応用

た，様々なディスプレイ配置（L 字型，三角筒形状，ロボット形状）を実現することを検討した．これらより，多くのディスプレイ使用や様々なディスプレイ配置を実現できることを提示できた．

今後は，GLIA を使いやすくするために数十台の計算機および入出力装置を資源管理するための名前サービスを付加する予定である．また，ディスプレイ配置を活かしたコンテンツを検討することも課題である．

参 考 文 献

1) Stefik, M., Brown, J.S.: Toward Portable Ideas, in Technological Support for Work Group Collaboration, Edit. by Olson, M.H., Lawrence Erlbaum Associates, pp.147–165 (1989) .
2) Russel, D., Streitz, N. and Winograd, T. : Building Disappearing Computers, Com. of the ACM, Vol. 48, No. 3, pp.42–48 (2005).
3) Streitz, N. et al.: i-LAND: An interactive Landscape for Creativity and Innovation, Proc. of CHI'99, pp.120–127 (1999).
4) Greenberg, S. :Enhancing Creativity with Groupware Toolkits, Proc. of CRIWG'2003, LNCS, Vol. 2806, pp.1–9, Springer-Verlag (2003).
5) 川喜田二郎：発想法-混沌をして語らしめる, 中央公論社 (1986).
6) 由井薗隆也，宗森　純： 発想支援グループウェア郡元の効果 -数百の試用実験より得たもの-, 人工知能学会論文誌, Vol. 19, No.2, pp.105–112 (2004).
7) 西村真一, 由井薗隆也, 宗森 純: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, 情報処理学会論文誌, Vol.48, No.7, pp.2278–2290 (2007).
8) 由井薗隆也，宗森 純，重信智宏:大画面共同作業インタフェースをもつ発想支援グループウェア KUSANAGI が数百データのグループ化作業に及ぼす効果, 情報処理学会論文誌, Vol.49, No.7, pp.2574–2588 (2008).
9) Arnold, K., Gosling, J. and Holmes, D.: The Java Programming Language, Third Edition, Addison-Wesley (2000).
10) https://jinput.dev.java.net/.
11) Walrath, K., Campione, M., Huml, A., Zakhour, S. : JFC Swing Tutorial, The: A Guide to Constructing GUIs, 2nd Edition, Addison-Wesley (2004).
12) Tanenbaum, A.S. and Steen, M.V.: Distributed Systems -principles and paradigms, 2nd Edition, Addison-Wesley (2007).