# 共有対話オブジェクト方式によるマルチユーザインタフェースシステムの設計と実装

越塚 登[†,*] 坂村 健[†]

近年，高性能なワークステーションやコンピュータネットワークの普及に伴い，グループウェアのような共同作業を支援するシステムが要求されている．しかしグループウェアの実装には，分散処理とユーザインタフェースの複雑な機構が必要なため，その構築は困難である．本論文では，グループウェアの構築に必要な機能を組み込んだ，次世代のウィンドウシステムの実現方式として，共有対話オブジェクト方式を提案する．共有対話オブジェクト方式は，共有ウィンドウを，対話オブジェクトを格納した分散共有記憶によって実現する方式である．この方式は，グループウェアアプリケーション本体とウィンドウシステム間の高いモジュール独立性が実現できる．更に本論文では，本方式を実現するシステムとして実現された，ウィンドウ実身と呼ばれる分散共有記憶システムについて述べる．ウィンドウ実身は，ウィンドウシステムに含まれ，格納された対話オブジェクトの共有を実現する．ウィンドウ実身は，領域ベースのアドレッシングや抽象イベントなどの機能を提供し，格納された対話オブジェクトをアプリケーションから扱いやすくしている．またウィンドウ実身は，領域木や完全複製方式の分散共有アルゴリズムを導入することで，効率良く実装することができる．実際に，評価システムを用いた性能測定によってその実装機構の有効性を検証することができた．

# Design and Implementation of Multiuser Interface System Based on the Shared Interaction Object Model

NOBORU KOSHIZUKA[†,*] and KEN SAKAMURA[†]

On account of the recent progess of high-performance workstations and computer networks, it is required to support our collaboration using computer systems such as groupwares. However, it is difficult to implement groupware since it contains complicated mechanisms of distributed processing and user interface. This paper proposes shared interaction-object model, which is a software architecture model for next generation window systems that supports the implementation of groupwares. The shared itneraction-object model realizes shared windows using distributed shared spaces which store interaction-objects displayed on the windows. This model can accomplish high module independence between groupware application body and the window system. Furthermore, this paper describes a distributed shared memory system, named Window Real-Object (WRO), which implements the shared interaction-object model. WROs store interaction objects in it, and work as a part of window systems. WROs support accesses to the stored objects by providing spatial addressings and abstract event functions. It can be implemented effectively by using region-tree and fully replicated distributed sharing algorithm. The authors have verified the efficiency by the performance measurement on the prototype system. This paper also describes the result of the measurement.

## 1. はじめに

近年，パーソナルコンピュータやワークステーションをコンピュータネットワークで接続し，遠隔地にいる複数の人間の共同作業を支援するシステムに対する要求が高まっている．その中でも特に求められているのが，リアルタイム型グループウェア[1]と言われる，リアルタイムに書類やグラフなどの同一データを各々のコンピュータの画面に表示し，編集などが矛盾なく行えるシステムである．例えば複数のユーザが同時に同じ書類を編集できるマルチユーザエディタや，一つの機器を複数のユーザが協調して制御する共有制御パネルのようなシステムがある．

ところが，リアルタイム型グループウェアの問題は，その実装が極めて困難なことである．特にそのユーザインタフェース（マルチユーザインタフェー

† 東京大学大学院理学系研究科情報科学専攻
Graduate School of Information Science, Division of Science, the University of Tokyo
* 現在，東京工業大学大学院情報理工学研究科
Presently with Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

ス）の実現には複雑な機構が必要とされる．例えば，画面上に表示される書類データを，複数のユーザが同時に編集するためには，データに加えられた操作をすべてのユーザの画面に反映させ，また各ユーザが並行に加える操作からそのデータを保護する機構が必要である．更に，この処理の応答性を向上させるためには，画面に表示されるデータの複製を各ワークステーション上に置き，それらの間の一貫性を保つ機構も必要となる．こうした機構は，最も一般的なユーザインタフェースシステムであるウィンドウシステム[2)~4)]自体では，ほとんど支援されていないのが現状である．

本論文の目的は，これからのコンピュータの重要な応用の一つであるリアルタイム型グループウェアの構築に必要な機構を組み込んだ，次世代のウィンドウシステムの実現方式を提案し，その効率のよい実装方式を明らかにすることである．

従来，マルチユーザインタフェースの構築支援の多くは，ウィンドウシステム上で動作するグループウェアツールキット[5)~7)]によって行われている．それらは，画面上の表示データの共有を，抽象度の高いセマンティックデータで行っている．例えば，画面上で「グラフ」データを共有する場合，「グラフ」の基となる「数値」のデータ構造／操作をツールキットが管理して共有を実現する．この「数値」データのようなアプリケーション本体に含まれるべきセマンティックデータは，ツールキットが管理できる枠組や言語を用いて実装することが必要となる．従って，グループウェアの扱う応用に合わせて，複数の言語，例えばC言語やLisp，Prologなど，を使い分けて実装することなどが困難となる．特に，ウィンドウシステムでマルチユーザインタフェースを支援する場合には，上記のようなアプリケーションの枠組や言語を強く制限する方式は，そのウィンドウシステムが扱える応用を限定するため，望ましくない．

本論文では，上記の問題を解決する実現方式として，共有対話オブジェクト方式を提案する．共有対話オブジェクト方式は，画面に表示されているデータを，従来の方式よりも抽象度の低い，文章や図形のセグメント☆やパーツ，メニューといった，対話オブジェクト（Interaction Objects）で共有する．先の「グラフ」の例の場合，グラフを構成する直線セグメントや文字列セグメントで共有する．

☆ ここでセグメントとは，図形や文章のパラメータ表現されたデータ単位をいう．

この抽象度での共有は，アプリケーション本体よりも下位の階層で実現できるため，アプリケーション本体の記述言語や，モジュール構成，セマンティックデータ構造を自由に構築できる．例えば，現在の我々の実装では，リアルタイム型グループウェアの記述に，C言語やSchemeなどが使え，またセマンティックデータは，C言語の構造体でもリスト構造でも実現可能である．この共有対話オブジェクト方式については，次章で述べる．

更に本論文では，共有対話オブジェクト方式を実現するために我々が設計，実装し，ウィンドウ実身と呼んでいる分散共有記憶システムについて述べる．ウィンドウ実身は，ウィンドウシステムに組み込まれて対話オブジェクトの共有を実現し，更に格納された対話オブジェクトをアプリケーションから扱いやすくするために，領域ベースのアドレッシングや抽象イベントなどの機能も提供する．これらの機能に関しては，第3章で詳しく述べる．

ウィンドウ実身の実装では実行性能を重視し，領域木に構造化された対話オブジェクト集合のデータ構造の複製を各ワークステーション上に置くという，ユニークなデータ構造とアルゴリズムを導入した．この実装機構の詳細は第4章で述べる．

実際に，上記の機構に従ってウィンドウ実身の評価システムを実装し，性能評価を行ったところ，上記の実装機構の有効性を示す結果が得られたので，第5章ではその詳細を述べる．

## 2. 共有対話オブジェクト方式

### 2.1 マルチユーザインタフェース

本節は，我々が構築するウィンドウシステムが支援するマルチユーザインタフェースへの機能要求を明確にするために，代表的なリアルタイム型グループウェアの例を二つ示す．

まず，**図1**は，マルチユーザ図形エディタである．中央のウィンドウは，編集対象の図形集合が配置された共有ウィンドウで，その上と左は，エディタの編集モードや，描画パターンを指定するパレットウィンドウである．共有ウィンドウは，複数のワークステーション上に同時に表示され，複数のユーザは共有ウィンドウ中の絵を同時に編集できる．共有ウィンドウの画面上での表示位置，表示倍率や，スクロール位置は，ユーザごとに別々に設定できる．一方，パレットウィンドウは，ユーザごとにローカルに管理されており，

複数ユーザからは共有されない．

ここで，ユーザが共有ウィンドウ上に図形を描画すると，各ユーザの共有ウィンドウ上にその図形が表示される．また，画面上の図形はポインティングデバイス（以下 PD）を使って，変形，移動，複写などの編集を行うことができる．個々の操作の間は，操作者以外からは操作できないようにロックされる．このロックは，図形を灰色化表示して各ユーザに通知される．**図 2** は図 1 とほとんど同じであるが，左上の楕円セグメントだけが，他のユーザが操作中のため灰色表示されている．ユーザが操作を終了し，PD をリリースした瞬間に，各ユーザの共有ウィンドウ上の図形は一貫性を保って変更が行われ，ロックも解除される．

次の**図 3** の例は，インテリジェントビルの空調の制御パネルである．このパネルも，共有ウィンドウと同様，あるユーザが温度調節のボリュームパーツのノブを PD でドラッグすると，このパネルを開いているすべてのユーザの画面上で一斉にノブの位置が変化する．この場合も，操作中は他のユーザの操作をロックする．

なおパネル上で，左上を指す「手」のマークが，そのユーザの PD のポインタである．またパネルの左やや上にあり，右下を指している方は，他のユーザの PD ポインタを表すテレポインタである．

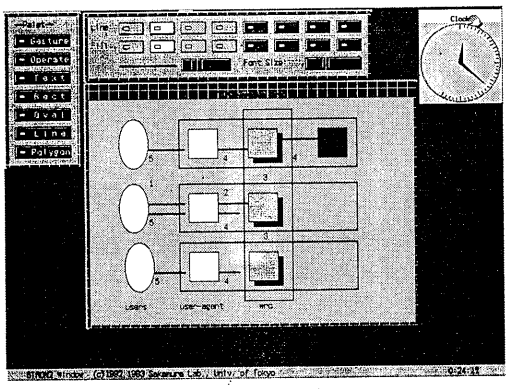



**図 1** マルチユーザ図形エディタ
Fig. 1 Multiuser graphic editor.

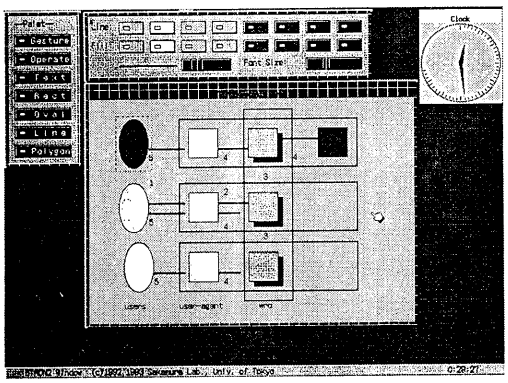



**図 2** 灰色火表示の例
Fig. 2 An example of "graying".

### 2.2 共有対話オブジェクト方式

前節で述べたようなマルチユーザインタフェースの実現方式として，本論文は共有対話オブジェクト方式を提案する．

共有対話オブジェクト方式は，共有ウィンドウや共有パネルを，対話オブジェクトのデータを格納する分散共有記憶で実現する．本稿では，対話オブジェクトを，ウィンドウ上に表示され，各ユーザから共有される，図形やテキスト，スイッチやボリュームなどのパーツ類，更にポップアップメニューなどのグラフィカルオブジェクトとする．また，この分散共有記憶をウィンドウ実身（Window Real-Object：以下 WRO）と呼ぶ．

共有対話オブジェクト方式で実現されたグループウェアは，WRO とそれを共有する複数のモジュールから構成される．それらは，グループウェアの中での役割に応じて，ユーザエージェントとアプリケーション本体に分かれる．**図 4** は本方式のモジュール構成図である．横長の三つの四角形は，ワークステーションを表し，それらは互いにネットワーク接続されている．図 4 では，本論文で述べる実装機構に従い，ウィンドウ実身内の共有データの複製が各ワークステーション上に置かれているが，一つのワークステーションのみにデータを置く実装方式も可能である．

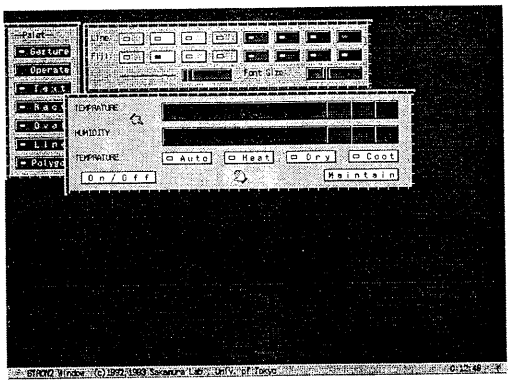



**図 3** 空調制御用分散共有パネル
Fig. 3 Distributed shared panel for air conditioner control.

ユーザエージェントは，各ワークステーション上に一つずつ置かれ，ウィンドウへの図形の描き込みや，編集，操作と，それによって起こるフィードバックなどのユーザインタフェースを実現する．ユーザエージェントはコンピュータの中で，言わばユーザの代理人（エージェント）として振る舞うモジュールである．具体的には，入力デバイスからの入力に応じて WRO に操作を加え，また WRO の変化に応じて画面上に描画処理を行う．通常，ウィンドウシステムの一部として実装される．

一方アプリケーション本体は，ユーザの操作に応じてグループウェアのセマンティックスを実行する．先の図3の空調制御パネルの例では，ユーザのボリューム操作に応じて，実際の空調に制御信号を送る処理，また図1，2のマルチユーザエディタでは，編集結果をファイルに保存する処理などがそれである．グループウェアアプリケーションの開発者は，このアプリケーション本体を実装することになる．

WRO は抽象イベントと呼ばれるイベントメッセージ機能を提供する．抽象イベントは画面上の表示データに加えられたユーザの操作を伝える．例えば，抽象イベントは「ボリューム ID=35 の値が，25 から 36 に変更された」というような高い抽象度の情報を伝える．一方，従来型のウィンドウイベントは，抽象イベントとは異なり，入力デバイスの動作という低い抽象度の情報を通知する．例えば，「マウスがウィンドウ上の点 (134, 827) 上でプレスされた」という情報である．

画面上の表示データに加えられたユーザの操作は，抽象イベントを使って，以下の手順で他のユーザの画面に伝える．

1. ユーザが画面上の表示データを操作する．
2. ユーザエージェントは WRO 内の対話オブジェクトに変更操作を加える．
3. WRO 内で対話オブジェクトの変更操作が処理される．この処理の実装は，第4.2節で詳細に示される．
4. 変更操作の処理後，即座に抽象イベントが発生し，WRO を共有するユーザエージェントやアプリケーション本体に操作結果を通知する．
5. 抽象イベントを受信したユーザエージェントは，もし必要であれば，更に詳細なデータを WRO から read し，ユーザの操作による変更に応じて画面を描画して，操作の伝播が完了する．

なお上記の番号は図4の矢印の番号と対応する．

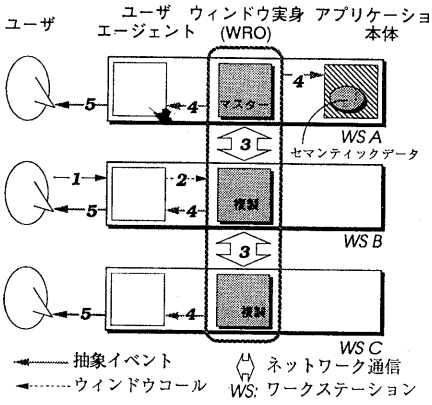



**図 4** 共有対話オブジェクト方式
Fig. 4 Shared interaction object model.

### 2.3 従来の方式との比較

既存のリアルタイム型グループウェアでのマルチユーザインタフェースの実現方式には，大別すると共有ウィンドウ方式と共有セマンティックス方式がある．これらの二方式とも，画面上のデータの共有方式が，本論文で提案した共有対話オブジェクトモデルとは異なる．

共有ウィンドウ方式は，**図5**のように，画面への描画処理と，デバイスからの入力をイベントに変換する処理を行うウィンドウサーバが各ワークステーション上におかれ，対話オブジェクトを実現する部分☆ とアプリケーション本体は，一つのワークステーション上に置かれる．アプリケーションは描画処理要求を各ウィンドウサーバにマルチキャストし，逆に複数のウィンドウサーバから出される入力イベントを集める．この方式のシステムには，XTV[10] や，shX[11] などがあ

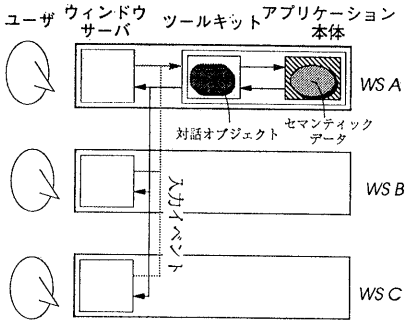



**図 5** 共有ウィンドウ方式
Fig. 5 Shared window model.

☆ 通常はツールキットで実現される．

る.

共有ウィンドウ方式の場合，全く同一のビューを持った共有ウィンドウは簡単に実現できる．しかし，共有ウィンドウに対して，ユーザごとに異なるスクロール位置やウィンドウサイズを設定するといった，柔軟な共有制御ができない．これでは実用的なマルチユーザインタフェースの構築は難しいと報告されている[12]．

もう一つの方法が共有セマンティックス方式で，LIZA[5]，Suite[6]，鼎談[7]など，既存の多くのグループウェアツールキットで採用されている．この方式は，**図6**のようにセマンティックデータをワークステーション間で分散共有することによりマルチユーザインタフェースを実現する．例えば，図3の空調制御パネルは，空調の設定温度を表す変数を，ワークステーション間で共有する．一方，我々の提案する共有対話オブジェクト方式は，設定温度を表示しているボリュームパーツを共有するという点で，この方式とは異なる．

共有セマンティックス方式の利点は，同じセマンティックデータをユーザごとに異なる表現で表示できることである．例えば，空調の設定温度をボリュームパーツで表示したり，数値を直接数字で編集できる数値ボックスで表示したりすることができる．本論文で提案した共有対話オブジェクト方式でも，ウィンドウ実身に格納される対話オブジェクトにローカル属性を設定することで，同様の機能を実現できる．この属性の詳細は，第3章で述べる．

共有セマンティックス方式の問題点は，グループウェアツールキットが，アプリケーション本体のセマンティックデータの構造やアクセスまでも管理していることである．そのためアプリケーション本体部分までも，ツールキットが提供する枠組に従って記述する必要がある．そうした枠組は，例えば，鼎談では拡張MVC モデル，LISA では kernel object, Suite では Shared Active Variable である．従って，グループウェアを実装する場合，ツールキットの提供する枠組が扱える言語しか使うことができず，その応用をプログラムしやすい言語や，枠組，モジュール構成をとって実装することは困難である．

本論文で提案した共有対話オブジェクトモデルは，低い抽象度のデータを共有し，対話オブジェクトの階層だけで分散共有機構を実現することで，アプリケーション本体の独立性を高めた．

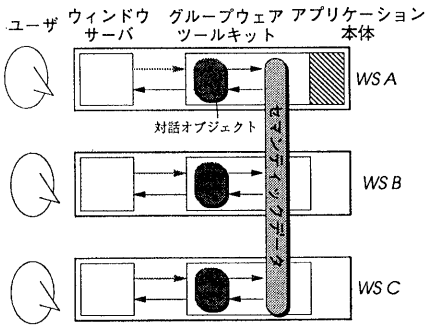



**図 6** 共有セマンティックス方式
Fig. 6 Shared semantics model.

## 3. ウィンドウ実身の設計

我々は，本論文で提案した共有対話オブジェクト方式によるマルチユーザインタフェースを支援するウィンドウシステムを設計，実装した．本論文では，その中でもウィンドウ実身（WRO）に関して，その設計の詳細を述べる．

### 3.1 ウィンドウ実身の機能概要

WRO は，ウィンドウやパネルに配置される対話オブジェクトの集合を格納した分散共有記憶である．対話オブジェクトには，文章や図形データ，ポップアップメニュー，パーツなどがある．WRO は画面上のウィンドウを通して表示されうる平面全体を表し，逆にウィンドウは WRO の一部分を表示する画面上の領域であると考える．

WRO は対話オブジェクト方式を実現するために，以下の特長を持つように設計されている．

1. 対話オブジェクトに対する競合操作を避けるために，操作中の対話オブジェクトを単位にロックできる．
2. ウィンドウの位置，大きさ，スクロール位置等は，ユーザごとに変え，また同一のセマンティックデータを異なる対話オブジェクトで表示するといった，共有データの柔軟な表示が，容易に実現できる．
3. グループウェアアプリケーション本体は，どのようなプログラミング言語でも記述できる．
4. グループウェアアプリケーションの実装の際に，特定のモジュール構成や，セマンティックスデータ構造などの枠組を前提としない．

### 3.2 アクセスプリミティブ

WRO へのアクセスプリミティブセットを，ウィンドウコールと呼ぶ．ウィンドウコールには，対話オブジェクトを操作する以下のプリミティブがある．

1. set/delete：WRO に対話オブジェクトを格納／削除する．
2. read：格納されている対話オブジェクトを読み出す．
3. update：対話オブジェクトにパラメータの変更を加える．例えば，パーツの設定値の変更，対話オブジェクトの移動，変形などを実現する．
4. operate：対話オブジェクトにパラメータの変更を伴わない操作を加える．例えば，ボタンをプレスするような，値の変更を伴わないパーツ操作，メニュー選択操作，対話オブジェクト上のクリック，ダブルクリック操作を実現する．この操作は以下で述べる抽象イベントを発生させ，操作されたことを他のモジュールに通知するという効果がある．
5. lock／unlock：対話オブジェクトにロック属性（第3.5節参照）を設定／解除する．

個々のウィンドウコールは，WRO を共有するモジュールから並行に呼び出されるが，各ウィンドウコールが SWMR (Single Writer/Multi Reader)[13] で実行されるように制御される．その具体的な実装機構は，第4.2節で述べる．

### 3.3 アドレッシング

ウィンドウコール内で，操作対象の対話オブジェクトを指定する方法がアドレッシングである．アドレッシングは，ウィンドウコールで指定されたキーから，一つまたは複数の対話オブジェクトを，WRO から取り出す．WRO では，このキーとして，対話オブジェクトの識別子（ID）だけでなく，ウィンドウ平面上の点や領域を指定することができる．点をキーとしたアドレッシングは，その点の直下の対話オブジェクトを取り出す（点検索）．また，領域をキーとしたアドレッシングは，その領域と重なりを持つ対話オブジェクトを取り出すか，またはそれに完全に含まれる対話オブジェクトを取り出す（領域検索）．そのほかにも，ユーザ操作によって選択状態にあるオブジェクトを取り出すキーも提供している（選択検索）．

このアドレッシングは，GUI における対話オブジェクトへのアクセスのプログラミングを支援する機能である．WRO のアドレッシングは，GUI で使われる，対話オブジェクトへのほぼすべてのアクセスパターンを含んでいる．例えば，PD によるピッキング処理には，PD 位置をキーとする点検索が使われる．ウィンドウの再描画処理で，再描画されるオブジェクトを取り出すには，再描画領域をキーとした領域検索が使われる．また選択状態にある対話オブジェクトに同じ操作を加えるには，選択検索が使われる．

更に領域検索や選択検索は，一回のウィンドウコールで，複数の対話オブジェクトに対して同時に操作を加えることを可能にする．これは，ウィンドウコールの発行回数を減らし，WRO との通信の回数やその通信で使われるメッセージ数も減らすため，効率面でも有効である．例えば，$N$ 台のマシンから共有されているマルチユーザ図形エディタで選択中の $K$ 個の図形セグメントを同時に削除する場合を考える．選択検索アドレッシングを用いれば，ウィンドウコール発行数は1回，ネットワークメッセージ数は，第4.2節で述べる分散共有アルゴリズムのための $N$ メッセージ（$48N$ バイト☆）のみで実行できる．ところが，ID のみを使い図形セグメントごとに delete ウィンドウコールを発行すると，ウィンドウコール発行数 $K$ 回，ネットワークメッセージ数 $KN$（$48KN$ バイト）となる．

### 3.4 抽象イベント

抽象イベントは，ウィンドウコールによって起きた WRO 内のデータの変更の結果を，それを共有する各モジュールに即座に通知するための非同期メッセージである．抽象イベントは，操作対象オブジェクトの ID，操作（発行されたウィンドウコール）の種類，操作者の ID，操作以前の値，操作後の値などの情報が含まれる．抽象イベントを受け取ったアプリケーション本体は，ユーザの操作に応じた機能を即座に実行する．一方，抽象イベントを受け取ったユーザエージェントは対話オブジェクトの変化に応じた再描画を即座に行う．つまり，抽象イベントは WRO を共有するユーザエージェントとアプリケーション本体の間で，制御を移動する機能でもある．

抽象イベントは，通常のウィンドウイベントと比べ，通知内容の抽象度が高く，PD やキーボードなどのデバイスの種類や数から独立である．更に，WRO を共有するユーザの数からも独立である．

また抽象イベントと同程度の抽象度の制御の移動機構としてコールバックがある．コールバックの場合，呼び出されるモジュールを記述する言語は，関数へのポインタを言語仕様に持つ必要がある．またリモートマシン上に置かれているモジュールの起動も困難である．一方，抽象イベントは，非同期メッセージを受理する関数さえ追加できる言語ならば，どの言語でも扱

☆ 現在の我々の実装では，対話オブジェクトの削除要求メッセージは48バイトである．

うことができ，アプリケーション記述言語への制約が少ないという利点がある．また，リモートマシンのモジュールの起動も，WRO を分散化することで容易に実現できる．

### 3.5 柔軟な共有表示の制御機能

これらの基本機能に加えて，WRO にはマルチユーザインタフェースの実現を支援する特別な機能がある．それは，WRO に格納される個々の対話オブジェクトの，複数のユーザに対する表示，また複数のユーザから可能な操作を制御する機能である．その制御のために，WRO の中の各対話オブジェクトには，1) ロック属性と 2) ローカル属性が設定できる．

ロック属性は，その属性を設定したモジュール以外のユーザエージェントからは，操作ができないことを示す．ロック属性は lock/unlock で制御され，複数のユーザ間の操作の衝突を回避することを目的に使用される．更にユーザエージェントは，ロック属性が設定された対話オブジェクトを灰色化表示することで，各ユーザは操作できないオブジェクトを知ることができる．ロック属性を用いた衝突の回避機構は，第 4.3 節で詳細に述べる．

ローカル属性は，属性を設定したモジュール以外のユーザエージェントからは，全くアクセスできないことを示す．ローカル属性とロック属性との違いは，ローカル属性は，アドレッシングの検索対象からもはずれるので，変更を伴わない read や operate も行えない点である．従ってローカル属性が設定された対話オブジェクトは，一人のユーザにしか表示されない．

ローカル属性は，共有ウィンドウ中の対話オブジェクトの柔軟な共有表示の実現に使われる．例えば図 3 のような空調制御の応用で，設定温度データが，ユーザエージェント A にはボリュームパーツで，ユーザエージェント B には数値ボックスパーツで表示されるには，次のようにローカル属性が使われる．まず，ユーザエージェント A に対するローカル属性を持ったボリュームと，ユーザエージェント B に対するローカル属性を持った数値ボックスを同じ WRO に格納する．その間の一貫性はアプリケーション本体が管理するが，抽象イベントによる変更通知とウィンドウコールを利用すれば，**図 7** で示すように簡単に実現できる．このように，ローカル属性により，共有セマンティックス方式のような表示の柔軟性を，共有対話オブジェクト方式でも実現できる．

## 4. ウィンドウ実身の実装機構

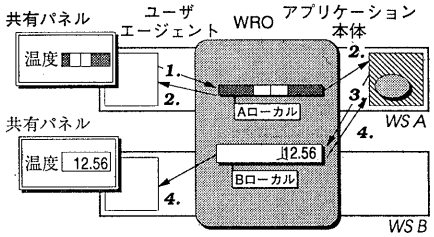
本章では，WRO の実装機構を述べる．本章で述べる機構の実現に必要な計算機環境の条件は，1) マルチタスク（プロセス）環境，2) タスク（プロセス）間共有記憶，3) 非同期メッセージ通信機構，4) セマフォアなどの相互排除機構，5) 信頼性のあるネットワーク通信の機構が提供されることである．この条件は，一般的な分散 OS であれば満たすことができる．

実装機構を述べるに先立ち，マルチユーザインタフェースの性能の重要な指標である応答時間と通知時間を，14) に従い次のように定義する．応答時間は，操作者が操作をしてからその操作が自ウィンドウに反映されるまでの時間である．また，通知時間は，その操作が他者のウィンドウに反映されるまでの時間である．

WRO を実装する際の目標は，まず応答時間を短くすることである．加えて，WRO に格納されている対話オブジェクト数，共有ユーザ数，ユーザの操作位置などの実行時の条件に，操作の応答時間がなるべく依存しないような，安定した応答時間も重要である．他方，通知時間は，ユーザの操作と同じ順序で通知される限りさほど短縮する必要はない．なぜならば，各ユーザは他のユーザの操作を直接見ることはないので，通知時間の長さを意識することはないからである．

### 4.1 対話オブジェクト集合を格納するデータ構造

WRO に多数の対話オブジェクトが格納された時に応答時間を短く保つためには，WRO 内部の対話オブジェクト集合へのアドレッシングを効率良く処理できるデータ構造を導入することが有効である．そこで，WRO には，点検索や領域検索といった，WRO に対




1. ボリュームの変更操作
2. ボリュームの変更を通知する抽象イベント
3. 数値ボックスの変更操作
4. 数値ボックスの変更を通知する抽象イベント

**図 7** ローカル属性を使った表示制御実現
Fig. 7 View control using local attributes.

して頻度の高いアクセスパターンを高速に処理できる対話オブジェクト集合のデータ構造として，R木[15]を導入した．

R木は，図形集合に対する領域検索処理に向いたデータ構造の一つで，B木を多次元へ拡張したバランス木である．対話オブジェクトは葉ノードにおかれ，中間ノードは，それより下のノードをグループ化する．そのグループ化は，各ノードの占める領域によって決まり，親ノードはすべての子ノードの領域を包含する最小の矩形を表し，根ノードがウィンドウ平面全体を表す．同様の応用に向いたデータ構造はほかにもあるが，ユーザの操作によるデータ構造の動的な変化に効率良く対応できる点に着目して，R木を採用した．

R木を用いることで，対話オブジェクト数$N$に対する領域検索を，木の構成状態にも依るが，約$O(\log_m N)$時間で処理できる☆．これが仮に，単純リストの場合，$O(N)$必要である．これは，例えばマルチユーザ図形エディタで，ウィンドウ中の図形数が数百個や千個もの多数になった場合に有効性が発揮される．

### 4.2 対話オブジェクト集合の分散共有機構

次に重要になることは，R木によって構造化されたWROを，ネットワークを介したリモートマシンから効率よく共有できることである．そのためには，WROを共有する各ワークステーション上に，その完全な複製を置く分散共有方式を用いることが有効である．

我々の導入したアルゴリズムでは，WROへのreadウィンドウコールは，ローカルなワークステーション上の複製から読み出す．その時，読み出されているローカルな複製はread以外の，変更を伴ったアクセスが加えられないように保護される．

他方，read以外のウィンドウコールはローカルには処理できない．それらはすべてWRO中に変更を残すため，各ワークステーション上の複製間の一貫性を保って変更する必要があるからである．分散化サーバは，各ワークステーション上に置かれるこれらの複製間の一貫性を保つ分散共有アルゴリズムを実現する．

ここでは分散化サーバが実現する分散共有アルゴリズムを，次のシナリオに沿って説明する．まず，現在3つのワークステーション，A，B，C上でグループウェアが利用されている．ここで最初にグループウェアが起動されたワークステーション上のWROをマスタと呼び（A），それ以外を複製と呼ぶ（B, C）．ここで，例えばB上で発行されたupdateウィンドウコールは次のように処理される（**図8**）．

1. B上で，updateウィンドウコールが発行される．
2. updateは，WROのマスタがあるA上の分散化サーバに，updateウィンドウコールの実行要求メッセージを送信する．
3. A上の分散化サーバは，updateの実行要求を，要求のあったB上の分散化サーバへ最初に回送する．
4. 次に複製があるC上の分散化サーバに回送する．
5. 要求を受け取ったB，C上の分散化サーバと，A上の分散化サーバは即座にupdateをローカルに加える．その時，各複製／マスタは，排他的にアクセスされるように保護される．
6. 各WROの複製／マスタは，抽象イベントでこの変更を通知する．

このアルゴリズムはウィンドウコールを，その処理要求メッセージがマスタ上の分散化サーバに到着した順序に応じて，シリアライズして処理する．なお，このアルゴリズムを実現するプログラムは，付録Aのようになる．

本アルゴリズムは，$N$台のワークステーション間で共有されているWROのupdateウィンドウコールを$N$メッセージで実現できる．ネットワークにマルチキャスト機能があれば，最小の場合2メッセージで実現できる．

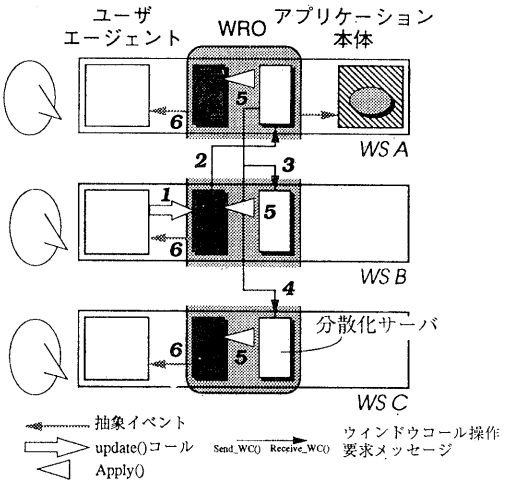



**図8** ウィンドウ実身における分散共有アルゴリズム
Fig. 8 Distributed shared memory algorithm in window real-objects.

☆ $m$はノードに格納するエントリーの最小数を表す．

このアルゴリズムの主な利点は，以下の三点である．第一に，read ウィンドウコールをローカルに処理できるため，read に対する短い応答時間が得られる．我々の実装の経験では，最も発行される頻度の高いウィンドウコールは read であること．read を使って実装されるユーザインタフェースの機能は，再描画やピッキングのフィードバックであり，これらは操作性の観点から高性能が必要とされること．これらのことより read の応答性が高いことはマルチユーザインタフェースの操作性向上に大きく寄与すると考えられる．

第二に，応答時間が WRO を共有するユーザ数に依存せず一定する利点がある．これは本アルゴリズムの3で，ウィンドウコールの実行要求メッセージを，操作要求をしたワークステーションへ最初に回送するためである．

最後に，本アルゴリズムは，ウィンドウコールの実行を取り消したり，ロールバックして再実行するようなことが起こらない．これは，画面上で一旦受け付けられた操作は取り消されないことを意味する．この性質は確実感ある操作性を与えるために重要である．分散している複製データの間の一貫性をとるだけならば，本アルゴリズム以外にも，タイムスタンプを用いた方式など効率的なものもある．しかしこれらは取消やロールバックが起こるため，採用しなかった．

### 4.3 操作の競合の回避

次に，複数のユーザが同じ対話オブジェクトに対して同時に操作を加え，操作の競合が起きた時の動作について述べる．以下で述べるように操作の競合は，WRO のプリミティブを使って適切にユーザエージェントを実装することで回避できる．

ここでユーザの操作の競合を考える時，短い操作と長い操作に分けて議論する．短い操作とは，クリックやダブルクリックのように，論理上一瞬の操作である．一方長い操作は，ボリュームのノブの移動，図形の変形・移動のように，一定の期間対話オブジェクトをドラッグする操作である．

短い操作は，その操作の検出時に，操作に対応したウィンドウコールを一回発行することで処理される（**図 9**）．従って同一オブジェクトに対する短い操作同士の競合は，WRO の分散共有機構によって解決される．なぜならば，各ワークステーション上で発行されたそのウィンドウコールの処理を要求するメッセージが，マスタ上の分散化サーバに到着する順序に応じて処理されるからである．

例えば，二人のユーザが同一のスイッチをほぼ同時にクリックした場合，各ユーザエージェントから，クリックを実現する update ウィンドウコール要求メッセージが送信される．これらはマスタマシン上の分散化サーバへの到着順序に応じて実行される．

一方長い操作は，ユーザがドラッグを開始してから，それをリリースするまでの長い期間に渡る．この長い処理は一つのウィンドウコールで処理せず，まず操作開始時に lock コールで操作対象のオブジェクトのみをロックする．これは，一つの WRO では read 以外のウィンドウコールは同時に一つしか実行できないため，長い操作中に WRO を独占せず，他のオブジェクトへの操作が同時に処理できるようにするためである．そして操作終了時に，operate や update コールで操作の本体を実行し，unlock コールでロックを解除する（**図 10**）．

ロックされているオブジェクトに対して，他のユーザが操作をすると操作の競合が起きる．この場合は，ロックされているオブジェクトに対する read 以外の操作は失敗する（**図 11**）．つまり長い操作との競合は，先に操作をしたユーザがそのオブジェクトの操作権を得ることで競合が回避されるのである．

また，長い操作を競合して，操作を拒否されたユー

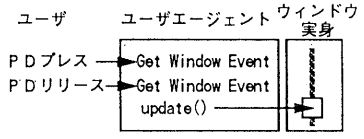



**図 9** 短い操作に対するユーザエージェントの一般的な処理
Fig. 9 An action of user-agent in processing short-term operations.

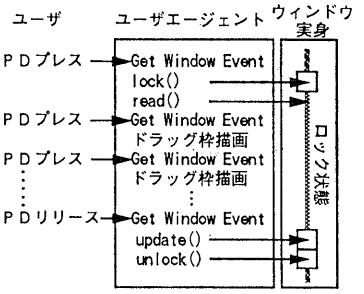



**図 10** 長い操作に対するユーザエージェントの一般的な処理
Fig. 10 An action of user-agent in processing long-term operations.

ザへは，そのオブジェクトは灰色化表示されるため，自分の操作が他のユーザとの操作の競合によって拒否されたという理由も知ることができる．

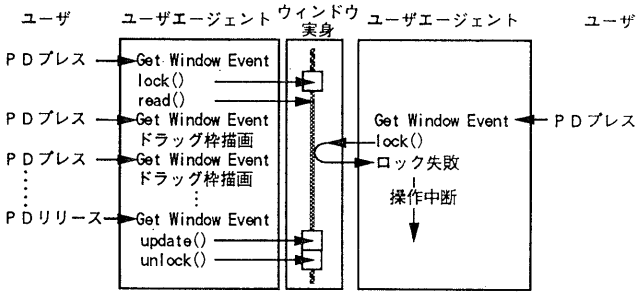



**図 11** 長い操作同士の競合の解決
Fig. 11 Resolving a collision between long-term operations.

## 5. 評　　価

我々は，本論文で述べた設計と実装機構の有効性を検証するために，WRO とユーザエージェントの評価システムを作成し，以下の二点に関して評価を行った．まず第一に WRO の機能評価として，対話オブジェクト方式を実現した WRO が，マルチユーザインタフェースのプログラミングの労力をどの程度軽減するかを検証した．これは実際に WRO を使って，マルチユーザ図形エディタを実装した経験から定性的に述べる．第二に，本論文で提案した WRO の実装機構の有効性を評価するために，WRO の評価システム上で応答時間を定量的に計測した．

### 5.1 評価システムの実装

本節では，実際に構築した評価システムの実装について述べる．評価システムを構築した実装環境は，**表 1** で示したハードウェア仕様を持った PC 9340 である．OS は ITRON/FILE[16] 仕様のマルチタスクカーネルを搭載している．ネットワークには，リアルタイム型グループウェアなどの応用を考慮した，リアルタイム通信用のネットワークシステム NIEP[17] を新たに開発し，それを搭載している．WRO の実装で必要なネットワーク通信は，NIEP の提供する返答・リトライ機能を持ったデータグラム機能を使って実装した．

WRO の実装では，対話オブジェクトデータ構造を共有メモリプール上に置き，そこへのアクセスを提供するウィンドウコールや抽象イベントの機能は，C言語で実装されたライブラリによって実装した．また，scheme インタプリタからのアクセスインタフェースも用意した．ユーザエージェント，アプリケーション本体といった，WRO を共有する各モジュールと，分散化サーバは，それぞれ独立したタスクとして実装した．現在のバージョンの実行コード量が，WRO のライブラリ部分が約 89 K バイト，ユーザエージェントが約 265 K バイト，分散サーバが約 138 K バイトである．

**表 1** 実装環境仕様
Table 1 Hardware specification of target machine.

| | |
|---|---|
| CPU | Gmicro/100 (トロン仕様 CPU)<br>クロック：20 MHz，最大 10 MIPS |
| 主 記 憶 | 16 M バイト |
| 二次記憶 | 100 M バイトハードディスク |
| ネットワーク | イーサネット (10 Mbps) |

### 5.2 リアルタイム型グループウェアの構築例

我々は，WRO がマルチユーザインンタフェースのプログラミングを軽減し，更にアプリケーション本体の記述言語やモジュール構成を制限しないという要求を満足することを検証するために，実際に WRO 上にリアルタイム型グループウェアアプリケーションを構築した．評価のためのアプリケーションとして，第2章で述べた機能を持った，マルチユーザ図形エディタを構築した．

本エディタは，単一の WRO を共有し並列に動作する複数のタスクによって実現されている．例えば，編集結果をファイル保存するタスク，対話オブジェクトを削除するタスク，線幅など画面上の図形オブジェクトを変更するタスクなど，エディタの機能ごとにタ

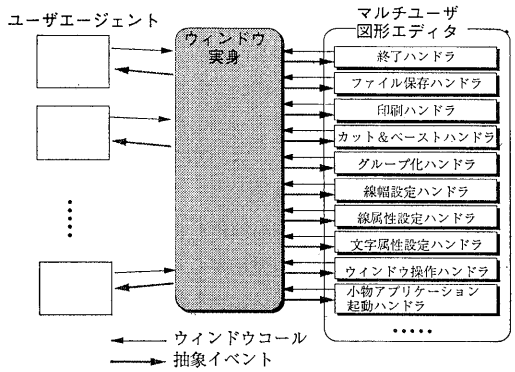



**図 12** マルチユーザ図形エディタのモジュール構成
Fig. 12 Module organization of multiuser graphic editor.

スク分割されている（**図 12**）．

個々のタスクは，抽象イベントとウィンドウコールを使った，高い抽象度のプログラミングインタフェースによって，短く記述される．例えば，線域を1ポイントに設定するメニュー項目を実行するタスクは，付録Bのように記述された．エディタを構成する各タスクは，いずれもこの程度の短いプログラミングで実装できた．従って，エディタは，必要な機能の数だけこうしたタスクを実装すれば良い．更に，これらのタスクは，その記述に適した言語で実装できる．例えば，付録Cは先程と同様のプログラムを scheme で記述したものである．

このように，WRO による支援によって，実装に適した言語で，高い抽象度で記述された，非常に小さいタスクを組み合わせて構築できることがわかる．

### 5.3 ウィンドウ実身の性能評価

対話オブジェクト方式とウインドウ実身の実装機構の有効性を評価するために，以下の性能を測定した．

1. update のコールの処理時間
2. read コールの処理時間

#### 5.3.1 update コールの処理時間

WRO 上に構築されたマルチユーザ図形エディタの共有ウィンドウ中の矩形セグメントを PD で変形し，その変形結果が操作をしたワークステーション上のデータに完全に反映されるまでの時間を測定した．ここで測定した時間は，PD リリース後に，update コールが発行されてから，その update を通知する抽象イベントが自ユーザエージェントに受理されるまでの時間を計測した．つまり，図8の処理の流れで，ステップ1から6の処理時間を計測したことになる．共有ウィンドウには互いに重ならない図形セグメントが 12 個配置され，変形した矩形セグメントは，30×30 ピクセルで，50% グレーパターンで塗りつぶされている．

ウィンドウを共有しているワークステーションを同時に1台から5台のそれぞれの場合に関して，マスタの WRO があるワークステーションから操作を加えた場合と，複製の WRO があるワークステーションからの場合と双方で測定した．その結果が**図 13, 14** である．実験の結果，1台だけの場合の処理時間は約 2.4 ミリ秒である．この 2.4 ミリ秒は WRO のデータ構造の変更，抽象イベントの送付，タスク切替えなどのローカルな処理の時間である．複数台接続した場合に，複製の上での WRO の更新処理の時間は，接続台数によらずほぼ一定の 10.0〜10.2 ミリ秒程度の処理時間である．マスタ側は，メッセージのマルチキャストのコストのため，接続台数に比例して処理時間が増加する．

**図 15** は2台接続した時の複製側の実験データに基づき処理時間を分析した図である．処理時間 10.05 ミリ秒のうち，update コールの実行に 3.99 ミリ秒．update コールがメッセージを送信してから，マスタマシン上の分散化サーバからメッセージが戻ってくるまでに，3.50 ミリ秒．更に，その返ってきたメッセー

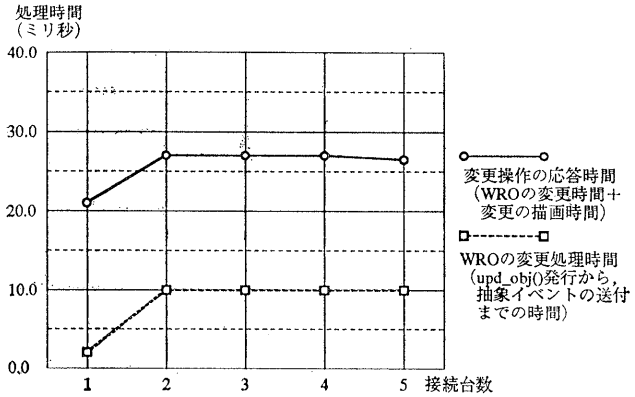



**図 13** update ウィンドウコールの処理時間（複製マシン上）
Fig. 13 Processing time of an update window call (On a replica machine).

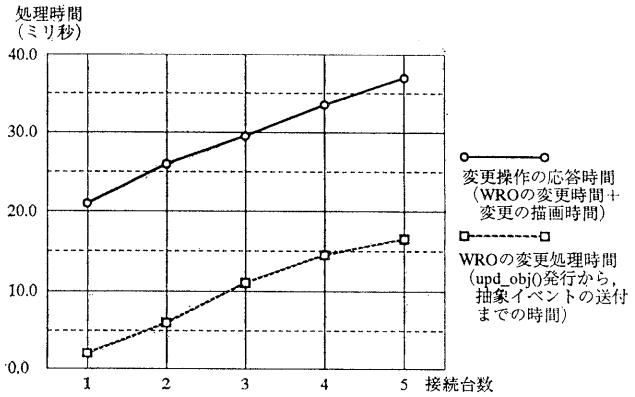



**図 14** update ウィンドウコールの処理時間（マスタマシン上）
Fig. 14 Processing time of an update window call (On a master machine).

ジによって起動されるローカルな分散化サーバの処理に2.56ミリ秒かかっている．

また，応答時間全体に占めるWROの処理コストを明らかにするために，ユーザがPDボタンを離してから，その操作を反映するる描画が終わるまでの応答時間も測定した．結果は図13, 14にWROの更新処理時間と共に示す．ここで2台接続した時の複製側の応答時間は26.8ミリ秒である．上記のWROの更新処理後に抽象イベントが発生し，それにより起動されたユーザエージェントが画面に描画する時間は，16.7ミリ秒である．従って，全体の応答時間のうち62%が描画時間である．また，残りの38%がWROの更新処理である．しかもその中でも，updateコールの中と，マスタマシンの分散化サーバの中で実行されるNIEPメッセージ送信には，各3.5ミリ秒，合計7.0ミリ秒(26%)かかっている．従って，ネットワーク送信，描画以外の，ウィンドウ実身本体の処理時間は全体の応答時間の中の約12%であり，相対的に小さい負荷で実装できていることがわかる．従って，マルチユーザインタフェースを支援するために，ウィンドウシステム中にWROを組み込んで実装することで，性能を大きく低下させるものではないことがわかる．

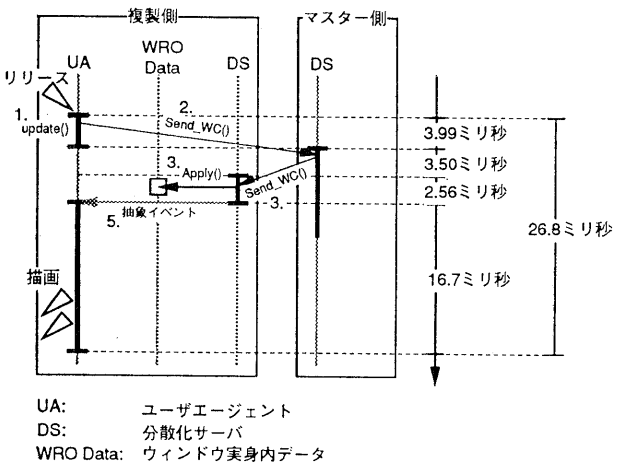



**図15** updateウィンドウコールの応答時間の内訳
Fig. 15 Precise contents of an undate window call processing.

**表2** readウィンドウコールの処理時間（単位：ミリ秒）
Table 2 Processing time of the read window call (msec).

| サンプルデータ名 | オブジェクト数 | R木実装 | | 単純リスト実装（先頭～最後尾） |
|---|---|---|---|---|
| | | 検索A（最大，最小） | 検索B | |
| サンプルパネル | 6 | 0.58-0.60 | 0.18 | 0.030- 0.135 |
| サンプル矩形集合 | 15 | 0.75-0.95 | 0.18 | 0.030- 0.343 |
| トロンの概念図 | 118 | 0.99-1.38 | 0.99 | 0.085- 6.706 |
| 日本地図 | 227 | 0.79-1.59 | 0.31 | 0.110-53.606 |

検索A：対話オブジェクト上の点で点検索
検索B：対話オブジェクト以外の点で点検索

なお，実験を行った台数よりもはるかに多く，例えば30や40台を接続した場合を考える．この場合，複製側の応答時間は一定であると思われるが，付録に示したアルゴリズムから推定し，通知時間とマスタ上での応答時間は台数に比例して増加する．通知時間も高性能が必要ないとはいえ限度があり，多くの台数の接続はこの実装のままでは困難になる．これは，ネットワークシステムにマルチキャスト機構を導入し，それを利用して本アルゴリズムを実装すれば解決できる．

我々以外のグループウェアシステムでは，7)で性能が報告されている．7)では，Sparc Stationを用いpingコマンドで64Kバイトメッセージが1ミリ秒以下で往復するネットワーク環境上で，シングルユーザインタフェースをマルチユーザ化した場合にかかる余分な処理時間を計測し，約30ミリ秒程度という結果が報告されている．この値に対応する我々のシステムの性能は，(複数台数の時の応答時間)－(1台の時の応答時間)で表すことができる．すると，約5.5～6.0ミリ秒程度で，我々のシステムの方が約5倍程度性能がよい．更に，我々のターゲットマシンであるPC 9340は，Sparc Stationと比べ処理が5倍程度遅い☆ため，実際には上記の差以上の性能差があることが推測される．

### 5.3.2 readコールの処理時間

readコールはそれを要求したワークステーション上で完全にローカルに行われるため，その応答性はウィンドウを共有するワークステーションの数には，ほとんど影響されない．むしろ応答性は，WROに格納される対話オブジェクトの数や，その平面配列に依存

☆ 著者らの手元のSparc Station 1+のドライストーンは，レジスタ未使用で，約45000．更に同様の条件で，PC 9340上で測定したドライストーン値は6250であった．

する．

そこでオブジェクト数や配置に関して様々なサンプルデータを，点検索を用いた read コールの処理時間を測定した．処理時間の測定に使ったシステムは，R木を使って実装された WRO と，評価実験用に特別に単純リストで実装された WRO である．その測定結果が**表 2** である．この表から，R木を使った場合は，対話オブジェクト数が増加してもさほど応答性が低下しないことがわかる．またR木実装の場合，処理時間の最大値と最小値の差が少ないこともわかる．これは，R木がバランス木であるためで，ユーザがウィンドウ上のどの場所を操作しても，均等な応答時間を提供できるという利点につながる．

## 6. おわりに

本論文では，マルチユーザインタフェースの構築を支援する機能をあらかじめ装備したウィンドウシステムの構成方法として，共有対話オブジェクト方式を提案した．その方式に従って我々が設計，実装した分散共有記憶システム，ウィンドウ実身について述べた．本方式の導入によって，マルチユーザインタフェース支援システムとグループウェアアプリケーション本体の間の高いモジュール独立性を達成することができた．

また，短く安定した応答性を実現する実装機構として，R木を使った内部データ構造と，共有データの複製を各ワークステーションに置くという分散共有アルゴリズムを提案した．この機構の有効性は，WRO の評価システムを用いた評価実験によって確かめることができた．この実装機構は，共有対話オブジェクト方式の実装のみに有効なのではなく，より一般的に，グループウェア全般，GUI ツールキットの実装にも適用可能である．

本論文では，共有対話オブジェクト方式を，分散された複数のユーザエージェントが単一のウィンドウを共有する方式として論じてきた．実は，この方式を逆から適用すると，分散された複数のアプリケーション本体が単一のウィンドウを共有する枠組にもなる．

例えば，第5.2節のマルチユーザエディタの例のように，GUI アプリケーションを協調動作する複数のタスクに分割して実現できる．この分割開発によって，GUI アプリケーションの開発効率と保守性の向上がのぞめる．各タスクは，WRO の複製があるどのマシン上でも実行できるので，これらのタスクを複数のマシン上で並列に処理できる．更に，各タスクをその実行に適したマシン上で実行できる．例えば，エディタ中で編集結果を保存するタスクは，ファイルサーバマシン上で処理し，また，印刷を行うタスクは，印刷データをスプールするマシン上で処理することなどができる．この分散処理によって，GUI アプリケーションの実行性能の飛躍的向上も期待できる[18]．

このような考察から我々は，共有対話オブジェクト方式は，単にマルチユーザインタフェースだけを実現する方式としてでなく，より広く先進的な GUI を実現する次世代のウィンドウシステムの基本アーキテクチャとして有効であると考えている[19]．現在我々は，共有対話オブジェクト方式に基づき，本論文で提案した実装方法によって BTRON 2 ウィンドウシステム[8),9),20)]の開発を進めている．

## 参考文献


1) 石井 裕：コンピュータを用いたグループワーク支援の研究動向，コンピュータソフトウェア，Vol. 8, No. 2, pp. 110-122 (1991).
2) Apple Computer, Inc.: *Inside Macintosh*, Addison-Wesley (1985).
3) Scheifler, R. W. and Gettys, J.: The X Window System, *ACM Trans. Gr.*, Vol. 5, No. 2, pp. 79-109 (1986).
4) Webster, B. F.: *The NeXT Book*, Addison-Wesley Inc. (1989).
5) Gibbs, S. J.: LIZA: An Extensible Groupware Toolkit, *ACM CHI '89 Proc.*, pp. 29-35 (1989).
6) Dewan, P. and Choudhary, R.: A High-Level and Flexible Framework for Implementing Multiuser User Interfaces, *ACM Trans. Inf. Syst.*, Vol. 10, No. 4, pp. 345-380 (1992).
7) 暦本純一：CSCW 基盤システム「鼎談」とその同時実行制御方式，コンピュータソフトウェア，Vol. 10, No. 1, pp. 51-62 (1993).
8) Sakamura, K.: Design Policy of the Operating System Based on the BTRON 2 Specification, *TRON Project 1990*, pp. 103-118, Springer-Verlag (1990).
9) 坂村 健監修：BTRON 2 カーネル標準ハンドブック，p. 479，パーソナルメディア (1992).
10) Cecil, M. J.: XTV: A User's Guide—Revision 2.1— (1991).
11) Altenhofen, M. P.: Using and Porting shX (version 1.0), Digital Equipment Corporation (1990).
12) Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S. and Suchman, L.: Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meeting, *Comm. ACM*, Vol. 30, No. 1, pp. 32-47 (1987).

13) Maekawa, M., Oldhoeft, A. E. and Oldhoeft, R. R.: *Operating Systems: Advanced Concept*, p. 414, The Benjamin/Cumming Publishing Company, Inc. (1987).
14) Ellis, C. A., Gibbs, S. J. and Rein, G. L.: Groupware: Some Issues and Experience, *Comm. ACM*, Vol. 34, No. 1, pp. 38-58 (1991).
15) Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching, *ACM SIGMOD Conf. Proc.*, pp. 47-57 (1984).
16) 坂村 健監修: ITRON/FILE 標準ハンドブック, パーソナルメディア (1992).
17) 前沢浩明, 高田広章, 坂村 健: 効率的リアルタイム通信を実現するライトウェイトプロトコル NIEP の設計と実装, トロン技術研究会資料, Vol. 3, No. 5, pp. 51-65, トロン協会 (1993).
18) Koshizuka, N.: Window Real-Objects: a Distributed Shared Memory for Distributed Implementation of GUI Applications, *Proc. ACM Symp. on User Interface Software and Technology '93*, pp. 237-247, ACM (Nov. 1993).
19) 越塚 登: BTRON 2 Window System: A Window System Facilitating Cooperation among GUI Applications in Distributed Environments, 学位論文, 東京大学大学院理学系研究科 (1994).
20) Koshizuka, N. and Sakamura, K.: Highly-Responsive Implementation of the BTRON 2 Window System, *Proc. 10th TRON Project Symp.*, pp. 82-93, IEEE Computer Society Press (1993).

## 付録A 分散共有アルゴリズム

### A.1 ローカルに用いられる手続き

プログラム中の番号は，図8の同じ番号の処理部分を示す．

```
procedure P(kind: READ/WRITE, wro: WRO の複製);
procedurt V(kind: READ/WRITE, wro: WRO の複製);
{ローカルに置かれた wro のデータを MRSW ロックするセマフォア}
procedure Apply(op: 操作, wro: WRO の複製);
    ……5.
{op の操作を wro に適用}
begin
  if op=READ then begin
    P(READ, wro);
    wro から READ する;
    V(READ, wro)
  end
  else begin
    P(WRITE, wro);
    op を wro に加える;
    抽象イベント発行; ……6.
    V(WRITE, wro)
  end
end.
procedure Send_WC (op: 操作, wro: 操作対象の WRO, to: 送付先);
procedure Receive_WC(op: 操作, wro: 操作対象の WRO);
{ウィンドウコールの要求メッセージの送受信}
```

### A.2 分散化サーバの動作

```
program 分散化サーバ {Distribution Server: 以下 DS};
var replica: ウィンドウ実身複製;
    op:      操作;
    wro:     ウィンドウ実身
begin
  while forever do begin
    Receive_WC(op, wro); ……2., 4.
    if 自分が wro のマスタ do begin
      if 要求メッセージの送り元=自ホスト then
                              ……3.
        Apply(op, wro)
      else
        Send_WC(op, 送り元, replica のあるホスト上の DS);
      foreach replica in 送り元以外の WRO の複製
        do ……4.
        if replica は自ホスト上にある then
          Apply(op, replica)
        else
          Send_WC(op, replica, replica のあるホスト上の SD);
    end
    else
      Apply(op, wro); ……5.
  end {while}
end.
```

### A.3 ウィンドウコールの動作

```
procedure ウィンドウコール (op: 操作, wro: 操作対象 WRO) ……1.
```

```
begin
  Send_WC(op, wro, wro のマスタホスト上の DS)
      .....2.
end.
```

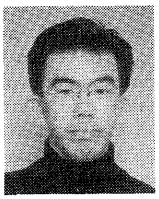
## 付録B　線幅を1ポイントに変更するメニューハンドラのCプログラム例

```
#include "BGSHELL.h"
main()
{
  ABSTRACT_EVNT ev;  /*抽象イベント*/
  unsigned int  l_atr=0x01 ;/*線幅指定    */
  ERROR         err;
  InitBGSHELL(MBF_ID, MACHINE_ID) ;
                    /*初期化              */
  GiveMeAEVT(ACCESSWROID) ;
                    /*抽象イベント送信要求*/
  while((err=Get_AEVT(&ev, TMO_FEVR))
     >=0)  {
    /*抽象イベント取得*/
      if((ev.wo_class    ==MENU_OBJ)&&
                    /*メニューへの操作    */
         (ev.wo_id    ==MENUOBJID)&&
                    /*オブジェクト ID はメニュー*/
         (ev.wc_class   ==OPE_WO)&&
                    /*operate された      */
         (ev.wc_subclass==CLICK)&&
                    /*操作内容はクリック  */
         (ev. info1     ==1PT_ITEM))
                    /*線幅 1pt にするメニュー項目*/
    /*選択状態の対話オブジェクトの，線幅を 1 pt
      に変更*/
    update(aev.wro_id, SELECTION_SEARCH,
      NULLKEY, L_ATTR, &l_atr) ;
  {
  QuitBGSHELL() ;
  exit(0) ;
}
```

## 付録C　線幅を1ポイントに変更するメニューハンドラの scheme プログラム例

```
(define(make-latr-1pt)
 (do(InitBGSHELL MBF_ID MACHINE_ID)
   (GiveMeAEVT ACCESSWROID)
   (repeat
     (let*((ev          (Get_AEVT TMO_FEVR))
           (woclass     (get_wo_class ev)MENU_OBJ)
           (woid        (get_wo_id ev) MENUOBJID)
           (wcclass     (get_wc_class ev)OPE_WO)
           (wcsubclass (get_wc_subclass ev)CLICK)
           (info        (get_infol    ev)1PT_ITEM))
      (cond((and        (= woid       MENU_OBJ)
                        (= woclass    MENUOBJID)
                        (= wcclass    OPE_WO)
                        (= wcsubclass CLICK)
                        (= info       1PT_ITEM))
           (update(get_wro_id ev)
           '(SELECTION_SEARCH NULLKEY)
           'L_ATTR
           info)))))))
```

(平成5年7月12日受付)
(平成6年6月20日採録)



**越塚　登**（正会員）

1966年生．1989年東京大学理学部情報科学科卒業．1994年同大学院理学系研究科情報科学専攻博士課程修了．博士(理学)．現在，東京工業大学大学院情報理工学研究科数理・計算科学専攻助手．現在，トロンプロジェクトにおいて，ウィンドウシステムやUIMS等のGUIの基盤システムの構成法の研究，障害者支援ユーザインタフェースシステムであるイネーブルウェアの構築などに従事．IEEE，ACM各会員．

**坂村　健**（正会員）

1951年生．東京大学大学院理学系研究科情報科学専攻助教授．工学博士．1984年から21世紀を目指した新しい考え方に基づくコンピュータアーキテクチャ体系の構築プロジェクト，トロンプロジェクトを興し，同プロジェクトリーダーを務める．IEEE MICRO Magazine編集委員，(社)トロン協会プロジェクト推進委員長．情報処理学会，電子情報通信学会，IEEE等から論文賞を受賞．主著に「TRONからの発想」(岩波書店)，「電脳社会論」(飛鳥新社)，「電脳未来論」(角川書店)など多数．