共有空間通信を用いたグループウェアの構築

増井俊之 花田恵太郎 音川英之
シャープ株式会社 技術本部
情報技術研究所
{masui,hanada,otokawa}@shpcsl.sharp.co.jp

現存のグループワーク支援ツールの多くは表現力が足りなかったり特定のシステムを想定して作られていたりしており適用範囲が狭く柔軟性に乏しい。本論文では広い範囲のCSCWシステム及びユーザインタフェースソフトウェアの基盤として使用可能な「共有空間通信モデル」を提案し、グループワークへの適用例を示す。本モデルは同期型/非同期型CSCWの両者に対応しており、異なる言語やプラットフォームで共通に使用することができる。

Groupware based on Shared-Space Communication

Toshiyuki MASUI, Keitaro HANADA, Hideyuki OTOKAWA
Information Technology Research Laboratories
Corporate Research and Development Group
SHARP Corporation
{masui,hanada,otokawa}@shpcsl.sharp.co.jp

We propose the *shared-space communication model* for CSCW applications. Unlike current groupware construction tools most of which are limited in their expressive power and flexibility, shared-space communication model is simple, flexible, and able to support both synchronous and asynchronous group communication under different platforms and languages. We show how a CSCW system can easily be built using the model, by making a prototype teleconference system on workstations connected with a TCP/IP network.

1 はじめに

近年計算機のグループワーク (Computer Supported Co-operative Work, CSCW) への応用の有効性が認識されつつあり、グループワークを電子的に支援する各種の電子会議システムや編集システムが提案されており商用となっているものも多い。またCSCWシステムを作成するためのツールも数多く提案されている。

グループワークを支援するソフトウェアではシングルユーザのアプリケーションでは必要のない様々な機能が要求される。例えば複数ユーザが同じデータを見たり操作したりする機能が多くの場合必要であり、その場合ユーザ間の制御の移動を適当に処理するための機能が必要である。またアンケートや投票機能などの機能が必要となる場合も多い。既存のユーザインタフェース構築支援システムはこれらをサポートしていないため上記のような機能を有するCSCWアプリケーションプログラムの作成を支援するためのツールキットが近年多数提案されている [2] [8] [13] [20] [22] [25] [26]。しかしこれらのシステムは以下のいずれかの問題点を持っている。

- Xウィンドウなど特定のシステムの機能に依存しており汎用性に乏しい
- 特定の対話モデルに限定されている
- 同期型CSCWと非同期型CSCWの両者に対応していない
- 既存のインタフェースツールとの整合性がわるい
- 一対一の通信を行なっているため相手について詳しい知識がないと通信ができない
- 通信の制御のような低いレベルからCSCWのアプリケーションの制御までの広い範囲をカバーしようとしている
- シングルユーザ向けシステムを複数ユーザ用に拡張しているため無理がある

本論文では、広い範囲のCSCWシステム及びユーザインタフェースソフトウェアの基本構造として使用できる「共有空間通信モデル」を提案し、本モデルにより以上のような問題が解決されることを適用例を通じて示す。

2 共有空間通信によるCSCW

我々はLinda [5] [7] を拡張した共有空間通信モデルに基づいたユーザインタフェースソフトウェアの構築手法を提案している [15][16]。本節では共有空間通信モデルについて簡単に解説を行ない、本モデルが通常のシングルユーザアプリケーションだけでなく複数ユーザのためのグループワークアプリケーションの構築にも有効であることを示す。

2.1 ユーザインタフェースにおける並列処理

近年の高度なユーザインタフェースソフトウェアでは並行処理機能が必須となってきている。マウスとキーボードのような複数の入出力装置を使用するプログラムでは各入出力装置に対し別々のプロセスを割り当てて処理を行なう方が簡単であるし、テキスト入力枠やボタンなどのインタフェース部品が並んだグラフィックインタフェース画面においては、ユーザが任意の順番で部品を操作可能にするためにはそれぞれの部品を並行に動作させなければならない。またユーザをプログラムと独立に動くプロセスと考えると都合が良い場合もあるし、UIMSの大きな目標であるアプリケーション部とユーザインタフェース部の分離にも有効であるため近年のユーザインタフェース作成ツールのほとんどが明示的または暗黙的に並列処理をサポートしている。

明示的にユーザインタフェース専用の並列処理言語を使用した例として、CSP[11] を使ったインタフェース記述言語 Squeak[4], Newsqueak[21] や、ERL というイベント処理言語を使用した Sassafras UIMS [10] などがある。

暗黙的に並列処理を行なっているものとしては制約記述言語やユーザインタフェースツールキットがあげられる。近年のグラフィックインタフェース作成システムでは、グラフィックオブジェクトの間に制約を設定しオブジェクトの位置などの変化により自動的に別のオブジェクトが更新されるというものが数多く提案されているが [3] [9] [12] [23] [24]、オブジェクト間の制約は暗黙的な並列プロセスが制約の数だけ存在することと同等である [1]。

一方Xウィンドウや NeXTstep などのウィンドウシステムでは各種のインタフェースツールキットと呼ばれるライブラリ群がインタフェースプログラムの作成に使用されている。インタフェースツールキットとはテキスト入力枠やボタンなどのグラフィックインタフェース部品の外見及び動作をライブラリとして定義したもので、各部品に対しマウスやキーボードなどから操作を加えると部品に対し定義された「コールバック」関数が呼び出されるようになっている。インタフェースツールキットではシステムが入力装置からの要求をとりまとめてツールキットに渡し、ツールキット側においてその要求がどの部品に対応するのか判断して各部品にイベントを配送するという構造になっているものが多い。この場合ツールキットのイベント配送システムがインタフェース部品の並行動作を支援していることになる。

以上のような並行処理表現は、特定の応用には便利であるものの、汎用のインタフェース記述方式としては不充分である。例えば制約解決システムは図形の配置やスプレッドシートのようなアプリケーションには便利であるがプログラムモジュール間の複雑な通信を表現することはむずかしい。Squeak や ERL のような言語を使用すると明示的に並列動作の記述をきめこまかく行なえるという利点はあるが、並列処理だけのために特殊な言語を使用する必要があるためアプリケーションプログラムの開発の障害となる。インタフェースツールキットはグラフィックインタフェース作成に現在最も広く使用されているが、プログラムの制御移動の枠組が固定的で柔軟性に乏しく、部品は常にイベント配送システムから呼び出されるという形になるためプログラム構造が制約をうけてしまいアプリケーションを作成

しにくくなるという欠点があるし [14] [16] [1]、並行動作する部品間で情報を交換することが簡単でないことが多い。

以上と異なるアプローチとして、既存のアプリケーション記述言語に並行処理機能を追加するという方法が考えられる。言語仕様に並行処理機能を追加する試みは従来より数多く行なわれてきたが[2]、そのいずれもが処理系に大きな変更を加えるものであり広く一般的に使用されているものは少ない。これに対し、言語に並行処理ライブラリを追加することにより既存の言語に並行処理機能を加えるというアプローチがある。この手法では言語仕様はほとんど変更せず、並行処理ライブラリが呼び出されたときのみプロセス切替などの処理を行なう。このようなものの例として Linda がある。Linda は数個のプリミティブから成る並行処理モデルで、既存の言語にライブラリとして追加することにより比較的簡単に言語を並行動作可にすることができる[3]。以下では Linda について簡単に解説し、Linda がユーザインタフェース及びグループワークソフトウェアの構築に都合が良いことを示す。

2.2 共有空間通信モデル Linda

Linda [5] [7] は、タプル空間 (Tuple Space) と呼ばれる共有空間を用いて複数プロセスが通信を行なう並列処理モデルである。一般の共有メモリを用いたプロセス間通信と異なり、Linda では共有空間に対するデータの書込み / 読出し操作においてプロセスの同期が行なわれる[4]。この際共有メモリ上のデータに対しパタンマッチングを行ない、連想メモリのようにデータの内容にもとづいて書込み / 読出しを行なう。タプル空間は全てのプロセスにより共有されており、各プロセスはタプル空間に対してタプルと呼ばれるデータ組の入出力を行なうことができる。タプルはひとつのラベルと任意個のパラメタから構成される。ラベルは任意長の文字列で、パラメタは任意の型の変数または定数である。各プロセスはタプル空間に対し以下の操作を行なうことができる。

out (*Label*, P_1, P_2, ... P_n)
in (*Label*, P_1, P_2, ... P_n)
rd (*Label*, P_1, P_2, ... P_n)

out (*Label*, P_1, P_2, ... P_n) は新しいタプル <*Label*, P_1, P_2, ... P_n> を生成してタプル空間に格納する。プロセスは **in** (*Label*, P_1, P_2, ... P_n) または **rd** (*Label*, P_1, P_2, ... P_n) によりマッチするタプルを非決定的にタプル空間から読み出すことができる。タプル空間内のタプルと **in** または **rd** のラベルが一致しかつパラメタの型と値が一致するときマッチしたと判断される。パラメタ中の変数と定数は同じ型ならば常にマッチし、マッチング後定数が変数に代入される。**in** は処理後タプルをタプル空間から削除するが **rd** は削除しない。目的のタプルがみつかるまで **in**, **rd** はブロックする。**out** はブロックしない。

Linda のプリミティブは以上のように単純であるが、図 1のように様々な形態のプロセス間通信を行なうことができる。

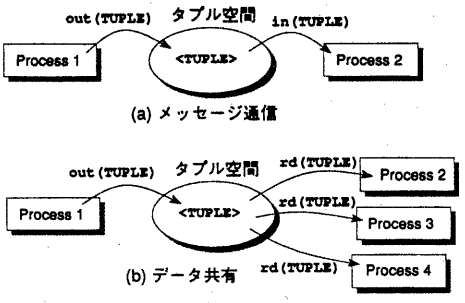

図 1: タプル空間を使った通信

図 1(a) は一対一のメッセージ通信、図 1(b) は複数プロセスによるデータ共有の様子を示している。ここでデータの送信者は受信者を指定していない。また受信者はタプルの内容によって選択的に受信を行なうためメッセージの送信者よりもメッセージの内容にもとづいて通信を行なっていることになる。このように、一般の通信モデルと異なり、相手プロセス名や通信チャネルを意識せずに通信を行なうことができるためプロセスのモジュール化が容易になっている。またタプルは **in** により読込まれない限りタプル空間内に存在し続けるため、送受信者とも相手の状態を考慮せずにタプル空間に対するアクセスを行なうことができ、非同期的情報伝達・情報共有が可能である。

以下では Linda にもとづいたモデルを共有空間通信と呼ぶことにする。

2.3 共有空間通信をユーザインタフェースに適用する利点

従来の方式に比べ共有空間通信方式をユーザインタフェースプログラムに用いることには以下のような利点がある [15] [16]。

- モジュールの独立性

Linda の各プロセスは共有空間内データを介して疎な結合による通信を行なう。例えば入力装置それぞれに対しその処理プロセスを割り当て、結果のみを共有空間に書き出すようにしておくことにより、入力処理部が独立し、キーボードによる文字入力モジュールとペンによる手書き文字入力モジュールを交換して使うといったことが可能になる。また入力モジュールのかわりに自動的に文字列を生成するモジュールを使用すれば、システムの自動テストを行なうこ

[1] 例えば再帰呼び出しの最も深いレベルで結果出力を行なう 8-Queen プログラムを、ツールキットを使ってマウスクリックにより順番に次の解が得られるような形に変更するのは容易ではない [16]。

[2] 言語名の先頭に "concurrent-" という文字列を加えればよい。

[3] この場合言語名の後尾に "Linda" という文字列を加えることになっている。

[4] 例えば UNIX System V では複数プロセスでデータを共有するために共有メモリを使用することができるが、プロセス間の同期をとるためにはセマフォなど別の機構を使う必要がある。

ともできるし自動デモにも有用である。このようにユーザや入出力モジュールをシステム内の単なるモジュールとして他のモジュール同様に独立して扱うことができるため、インタフェース部とアプリケーション本体を分離させることが可能である (図 2)。

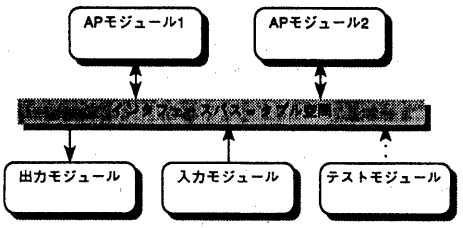

図 2: タプル空間によるモジュールの分離

• 異なる言語の共存

ユーザインタフェースソフトウェアには様々な機能が要求され単一の汎用言語で全てを処理することはむずかしいため様々な言語や手法が提案されているが [18]、Linda は数個の簡単なプリミティブで構成され、既存のいろいろな言語上でライブラリの形で融合して使うことができるため、複数の言語を協調させて使うことが容易にできる [16]。たとえばインタフェース記述専用言語とアプリケーション作成用言語の共用が可能になる[5]。

• 制御の主体の柔軟性

各プロセスはそれぞれ独立した制御の主体となって動作するため 2.1節で述べたような制御移動の問題は発生しない。プロセスは処理を他プロセスに依頼した後そのまま実行を続けることもできるし、他プロセスからのイベントを待ち続けることも可能である。このように、現存ユーザインタフェースツールキットで採用されているようなイベントとコールバック関数を用いた型式にもアプリケーションがインタフェースを呼びだす型式にも対応可能である。

• 他システムとの協調

制約解決システムのように特定分野で有用な別システムをモジュールとして独立に動作させ共有空間を介して通信することが可能なので、既に開発されている各種の有用なツール資産を活用することができる。

• データの粒度、レベルの柔軟性

共有されるデータの型式は任意である。単純な文字列や座標のようなものでもよいし高度な内容をもつメッセージでもよい。複数の「エージェント」が協調して「知的」作業を行なうというモデルが最近流行しているが (例えば [19])、これは共有空間通信の枠組で実現できる。

[5]複数の言語を使用してユーザインタフェースを構成している例として SUN の NeWS ウィンドウシステムがある。NeWS では画面への表示やユーザ入力の処理には拡張 PostScript を使用しアプリケーションの記述には C を使うことができるが、両者の整合性が良いとはいえない。

2.4 共有空間通信の CSCW への応用

共有空間通信方式は以下の点においてグループワーク支援ソフトウェアの構築にも有用である。

• モデルの適合性

複数プロセスのデータ共有にもとづいたモデルであるため、アプリケーション間でのデータ共有が必須であるグループワークと相性が良い。またデータ共有とプロセスの同期を同時に実現できるため共有データの扱いが楽である。

• シングルユーザのアプリケーションと複数ユーザのアプリケーション

シングルユーザのアプリケーションがすでに共有空間通信方式を用いている場合、これを複数ユーザ用に変更することが簡単にできる。

• 異なるプラットフォームの共存

共有データの型式を定めておけば異なるプラットフォームのマシン間でもグループワークが可能である[6]。共有データ構造とその操作方法さえ決まっていればそれを使うアプリケーションの外見はどのようなものでもかまわない。例えばデータを共有するエディタにおいて、一方のシステムが文字端末を使い、他方のシステムがグラフィック端末を使うといったことも可能である。共有データのネットワーク上の実現方式は様々な手法が考えられるが、どの方式を採用する場合でもアプリケーションプログラム本体は変更しなくてよい。

• 共有データの集中管理方式

ウィンドウシステムベースのいくつかの CSCW システムでは効率の改善のために複製方式 (replicated architecture) を採用している [6]。複製方式とはネットワークで接続された全てのマシン上で同じアプリケーションプログラムを並列に実行させ、それら全てに同じ入力が伝わるように制御することにより結果的にグループメンバ全員に同じ画面が見えるようにする方式である。この方式ではアプリケーションの状態の整合性を保つことが困難であるが、共有空間通信方式ではつねにユーザが同じデータを共有するためこのような問題が生じない[7]。

• 非同期グループワークへの対応

CSCW システムは、電子会議のようにグループメンバが同時に仕事を行なう同期型システムと、電子メールのように各人が別のタイミングで仕事を行なう非同期型システムの 2 種類に大きく分類することができる。既存の CSCW ツールキットの多くはこのどちらかのみしか支援していないが、共有空間通信モデルでは共有空間中にデータが残るため同期 / 非同期両方のシステムに対応可能である。

[6]この点は実際の運用においては特に重要である。共有データの表現には ASN.1 や SUN RPC の XDR のような標準を使用することができる。

[7]複製方式の利点は効率のみであるが、複製方式が必ずしも良い性能を示すとは限らないという報告もある。

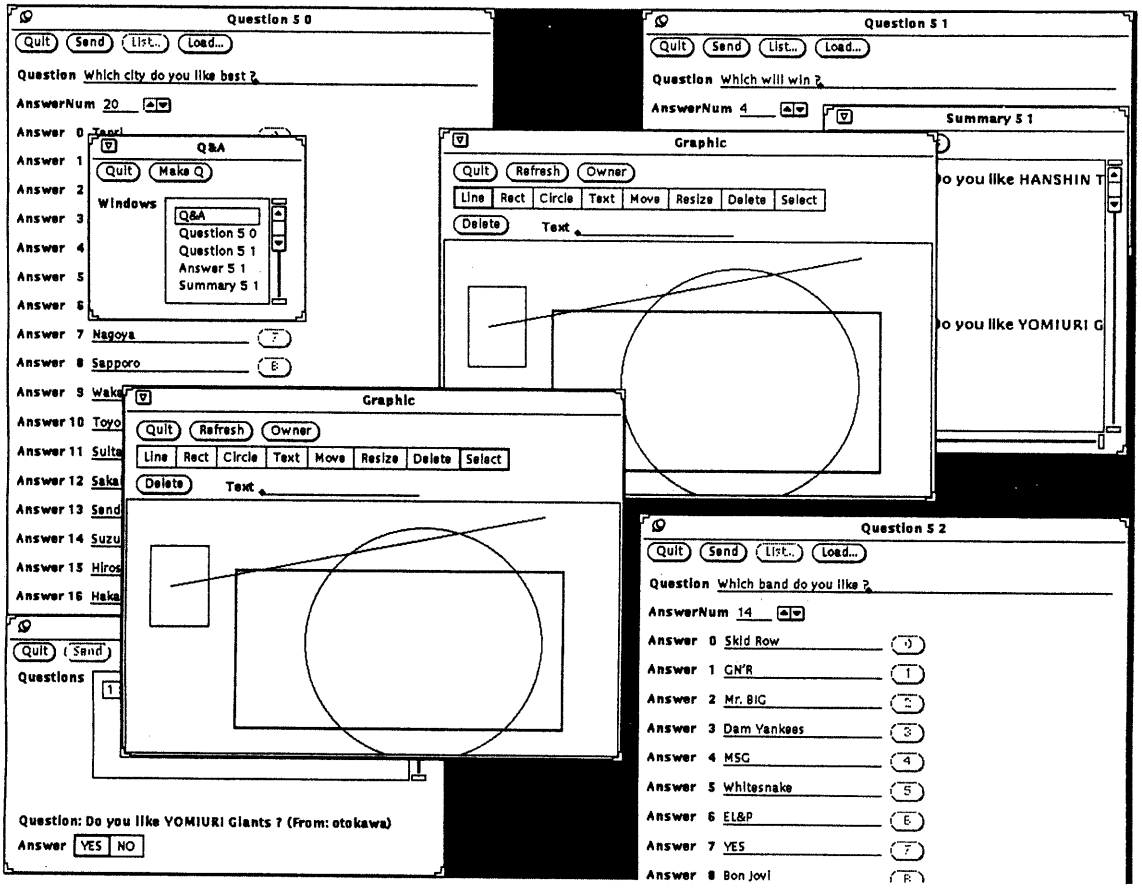

図 3: 図形エディタの実行例

3 共有空間通信を使用した電子会議システム

共有空間通信方式によるCSCWシステムの例として電子会議システムを実装、評価した。システムはTCP/IPで接続された複数のUNIXワークステーション上のXウィンドウシステムで動作する。

3.1 Lindaの拡張

CSCWアプリケーションを構築しやすくするため、Lindaに以下のような若干の拡張を加えた。

1. ブロック/ノンブロック入出力

2.2節で述べたように、Lindaの出力プリミティブ **out** は常にブロックせず、入力プリミティブ **in**、**rd** は常にブロックする。ところがユーザインタフェースで使用する場合、入力の有無を調べるためのノンブロック入力があると便利であるし、ブロック出力が可能だとブロック入力と組みあわせてパイプライン処理ができるようになるので、入出力両方においてブロック/ノンブロックを指定できるように拡張を行なった。

2. タプル出現順序の保存

Lindaでは、マッチするタプルが複数存在した場合、どれが選択されるかは非決定的である。しかし非同期通信を行なうふたつのプロセスにおいて、ひとつのプロセスがタプル空間内に出力したタプルをもうひとつのプロセスが出現順に取得することができると便利である。このため、マッチするタプルが複数ある場合は一番最初に出現したタプルを使用するモードを追加してある[8]。

3. 同一タプル **rd** の制限

タプル空間内のマッチするタプルを全て **rd** により処理しようとする場合、**rd** の前後でタプル空間の状態は変化しないので、何度 **rd** を発行しても同じタプルが選択されてしまう

[8]通常のLindaでもタプルのパラメータに番号をふっておくことでタプルに順序付けすることは可能であるが、順序付けが必要になる場合は多いためこのような機能を追加した。

可能性がある。このような要求もよく発生するため、一度 **rd** により選択されたタプルは再度マッチしないようにするモードを設けた。

3.2 共有空間通信サーバ

共有空間はUNIXのひとつのサーバプロセスとして実装している。各アプリケーションプログラムはTCP/IPでサーバプロセスと通信することにより拡張Lindaの各プリミティブを実行する。

3.3 複数ユーザ図形エディタ

複数のユーザが同時に操作可能な図形エディタを共有空間通信サーバを用いて実装した例を図3に示す。全ての図形データはサーバ上のタプル空間内に存在しどのプロセスからでも参照可能である。ユーザはシングルユーザ用のエディタと同様に図形の編集を開始するが、現在の状態はすべて共有空間内に保持されているため、後で別のユーザが編集作業に加わっても古いユーザと全く同じ画面から開始できる。このように同期型/非同期型両方の性質をもつシステムが同じ枠組で実現できる。

図形の追加の際のタプルのやりとりを図4に示す。

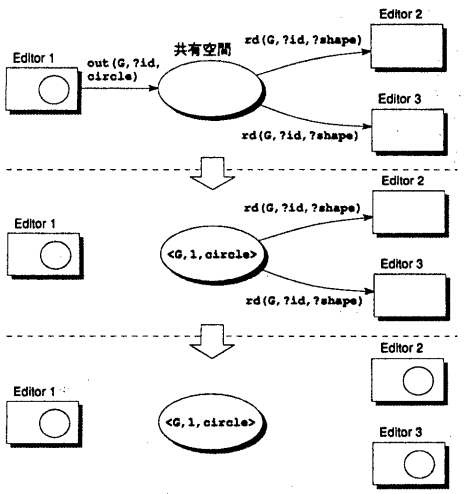

図4: 図形の追加時のタプルのやりとり

ユーザはエディタの"Circle"ボタンを選択し、画面上でマウスをドラッグすることにより円を描く。描画が終了するとそのプロセスは新しい円のデータを示すタプルを **out** により共有空間内に出力する。別のエディタプロセスは **rd** によりこれを検出し、自分の画面にも同じ円を描く。

また図形の選択の際のタプルのやりとりを図5に示す。

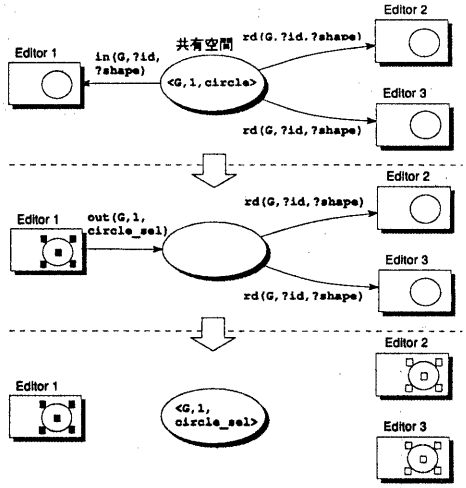

図5: 図形の選択時のタプルのやりとり

ユーザはエディタの"Select"ボタンを選択してから図形上でマウスをクリックすることにより図形を選択する。このとき **in** により共有空間内からその図形を示すタプルを除去し、それが成功するとそのタプルのかわりにその図形が現在選択されていることを示すタプルを出力する。他のエディタプロセスは **rd** によりこれを検出するとその図形が他のエディタにより選択されていることを表示する。**in** は排他的操作であるため複数のプロセスが同時に同じ図形を選択することはない。

3.4 アンケート調査システム

図3内には共有空間通信サーバを用いたアンケート調査システムも同時に示されている。アンケート調査システムとは、各ユーザが選択したデータが集計されて表示されるというものである。アンケート調査システムにおけるタプルの流れは図6のようになっている。アンケート調査システムで使っている共有空間通信サーバは図形エディタで使っているものと同じであり、アプリケーションの数が増えても共有空間通信はひとつだけでよい。

4 評価

前節の例で示したように、共有空間通信方式を使用することにより、単純な通信プリミティブを使用してCSCWアプリケーションを容易に作成することができる。**out** や **in** はメモリへの書込み/読出しと似た操作なので、従来のシングルユーザアプリケーションと同様の感覚で複数ユーザアプリケーションを作成することができる。このような簡便さにもかかわらず、図形エディタの例では「図を描く度にその情報を共有空間に書く」というだけの手間によりグループワークに必要な排他制御などの機能がすべて処理可能で

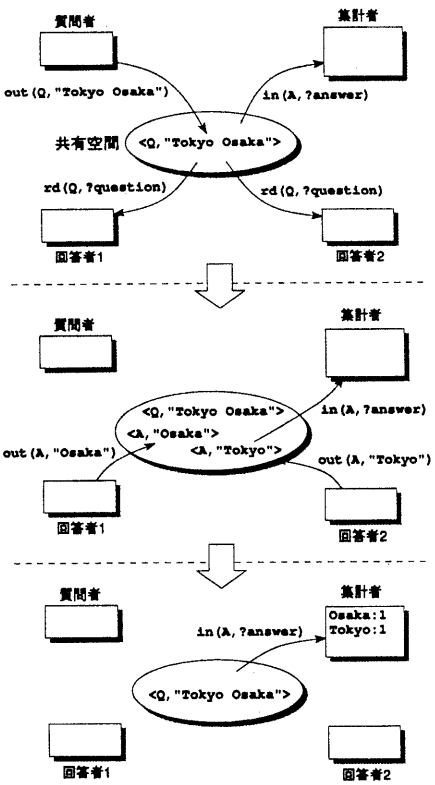

図 6: アンケート調査システムのタプルのやりとり

ありグループワークに特化した処理が必要無いという点が重要である。

今回のシステムではグラフィックユーザインタフェース部はXウィンドウの汎用のツールキット (XView) を使用しており、CSCW 対応にするための変更は不要である。またXウィンドウ以外のウィンドウシステムや文字端末インタフェースを用いて同じアプリケーションを作成することも簡単にできる。

Linda を拡張した共有空間通信モデルとして拡張 Linda[17] や Cellula[27] が提案されている。Linda では全てのタプルが全てのプロセスから大域変数のように参照可能であるため、通信するプロセスの数が増えた場合問題となるが、Cellula や拡張 Linda では局所的なタプル空間を使用することができるため CSCW アプリケーションへの応用において Linda よりも有利と考えられる。

5 結論

Linda を拡張した共有空間通信方式を使用することにより、CSCW 専用の特殊なツールキットを使用することなく、シングルユーザ用のアプリケーションとほとんど変わらない手間で簡単に CSCW アプリケーションを作成できることを実証した。共有空間通信方式は、ここで解説したようなツールキットへの応用や複数ユーザの協調といった用途以外にも、オンラインヘルプシステムの作成やアニメーションのオーサリングなど様々なユーザインタフェースシステムへの応用が可能であり、現在そのようなシステムを試作評価中である。

謝辞

本研究を援助していただいているシャープ株式会社情報技術研究所の大崎幹雄所長及び小淵保司主任研究員に感謝いたします。

参考文献

[1] D. Baldwin. Consul: a parallel constraint language. *IEEE Software*, Vol. 6, No. 4, pp. 62–69, July 1989.

[2] A. Bonfiglio, G. Malatesa, and F. Tisato. Conference toolkit: A framework for real-time conferencing. In *Proceedings of the 1st European Conference on Computer Supported Cooperative Work (EC-CSCW '89)*, Gatwick, U.K., September 13-15 1989. Computer Sciences House, Sloug, UK.

[3] A. Borning. The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 4, pp. 353–387, July 1989.

[4] Luca Cardelli and Rob Pike. Squeak: a language for communicating with mice. *Proceedings of SIGGRAPH*, Vol. 19, No. 3, pp. 199–204, July 1985.

[5] Nicholas Carriero and David Gelernter. *How To Write Parallel Programs*. The MIT Press, Cambridge, MA, 1990.

[6] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, and Raymond Tomlinson. Mmconf: An infrastructure for building shared applications. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, Los Angeles, California, October 7-10 1990. ACM Press.

[7] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, pp. 80–112, January 1985.

[8] S. J. Gibbs. LIZA: An extensible groupware toolkit. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'89)*, pp. 29–35. Addison-Wesley, May 1989.

[9] R. D. Hill. *A 2-D Graphics System for Multi-User Interactive Graphics Base on Objects and Constraints*. Springer-Verlag, Berlin, 1990.

[10] Ralph D. Hill. Supporting concurrency, communication, and syncronization in human-computer interaction: The sassafras uims. *ACM Transactions on Graphics*, pp. 179–210, July 1986.

[11] C. A. R. Hoare. Communicating sequential process. *Communications of the ACM*, Vol. 21, pp. 666–677, 1978.

[12] Scott E. Hudson. Graphical specification of flexible user interface displays. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '89)*, pp. 105–114. ACM Press, November 1989.

[13] Michael J. Knister and Atul Prakash. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, pp. 343–355, Los Angeles, California, October 7-10 1990. ACM Press.

[14] Jonas Lowgren. An architecture for expert systems user interface design and management. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '89)*, pp. 43–52. ACM Press, November 1989.

[15] Toshiyuki Masui. User interface construction based on parallel and sequential executiln specification. *IEICE Transactions*, Vol. E 74, No. 10, pp. 3168–3179, October 1991.

[16] Toshiyuki Masui. *User Interface Programming with Cooperative Processes*, chapter 15, pp. 261–277. In Myers [18], 1992.

[17] Satoshi Matsuoka and Satoru Kawai. Using tuple space communication in distributed object-oriented languages. In *Proceedings of the 1988 ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, Vol. 23, pp. 276–284, November 1988.

[18] Brad A. Myers, editor. *Languages for Developing User Interfaces*. Jones and Bartlett, Boston, MA, 1992.

[19] Hajime Nonogaki and Hirotada Ueda. FRIEND21 project: A construction of 21st century human interface. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, pp. 407–414. Addison-Wesley, April 1991.

[20] J. F. Patterson, R. D. Hill, S. L. Rohall, and W. S. Meeks. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, pp. 317–328, Los Angeles, California, October 7-10 1990. ACM Press.

[21] Rob Pike. Newsqueak: a language for communicating with mice. Computing Science Technical Report 143, AT&T Bell Laboratories, Murray Hill, New Jersey, 1989.

[22] Mark Roseman and Saul Greenberg. Groupkit: A groupware toolkit for building real-time conferencing applications. Technical Report Research Report, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, March 1992.

[23] Pedro A. Szekely and Brad A. Myers. A user interface toolkit based on graphical objects and constraints. In *Proceedings of the 1988 ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA88)*, pp. 36–45, August 1988.

[24] Bradley T. Vander Zanden. Constraint grammars - a new model for specifying graphical application. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'89)*, pp. 325–330. Addison-Wesley, May 1989.

[25] 葛岡英明, 三井博隆, 広瀬通孝, 石井威望. プラガブルなネットワーク・アプリケーション・ツールの開発. 情報処理学会ソフトウェア工学研究会研究報告 73-19, No. 73, pp. 147–154, July 1990.

[26] 中内靖, 伊藤嘉邦, 安西祐一郎. Michele: マルチエージェントモデルに基づく協調作業の新しい枠組. コンピュータソフトウェア, Vol. 9, No. 5, pp. 403–415, September 1992.

[27] 吉田紀彦, 植崎修二. 場と一体化したプロセスの概念に基づく並列協調処理モデル cellula. 情報処理学会論文誌, Vol. 31, No. 7, pp. 1071–1079, July 1990.