

マルチプロセスによるマルチスレッド対話型ユーザインタフェースの設計

宮武明義

詫間電波工業高等専門学校 情報工学科

今宮淳美

山梨大学 工学部 電子情報工学科

近年、マルチスレッド対話型ユーザインタフェースが注目を集めている。これまでユーザインタフェースの設計者は、アプリケーションと、ユーザインタフェースを完全に分離しようとしていた。しかし、迅速かつ有効な意味的フィードバックを提供するためには、アプリケーションが持つデータをユーザインタフェースとアプリケーションモジュールで共有する必要がある。また、マルチスレッド対話を実現するためには、並行処理可能な環境が必要である。

本稿では、マルチスレッド対話を支援するユーザインタフェースの実行時の構成を考えて、マルチプロセスによるユーザインタフェースモデルを提案し、そのモデルに基づくアプリケーションの実現例を示す。

Designing the user interface based on
multiprocess model for multithread dialogue

Akiyoshi Miyatake

Department of Information Engineering,
Takuma National college of Technology

551 Takuma, Mitoyo, Kagawa 769-11, Japan

Atsumi Imamiya

Department of Electrical Engineering and
Computer Science, Yamanashi University

4-3-11 Takeda, Kofu, Yamanashi 400, Japan

Abstract

The important issue of user interface for multithread dialogue has begun to be addressed only recently. Dialogue independence is the key concept that separates design of the interface from design of the computational component of an application system. To support a rapid and a fine semantic feedback, however, it is necessary to share the application data between user interface and application modules. Concurrency is also necessary to implement the multithread dialogues.

In this paper we consider a fine run time structure of user interface to support multithread dialogue, then present a multiprocess model, and the prototype of the user interface based on this model.

1. はじめに

現在、対話型ソフトウェア（アプリケーション）の作成及び実行時にそのユーザインタフェース部を支援する、さまざまなユーザインタフェース管理システム（以後、UIMS と略す）が開発されている。これからの研究課題として、ユーザとの対話形式に次の2つの特徴を持たせることが注目されている。1つは、ユーザがアプリケーションを通して扱う対象とその操作をより現実世界のものに近付けるという直接操作[東87]、もう1つは、さまざまな対象操作が並列に指定可能とするマルチスレッド対話[Tanner87]である。

直接操作をサポートする UIMS は、対象の表示とその操作方法、および意味的フィードバックがアプリケーションによってかなり異なるため、まだまだ実用段階には至っていない。一方マルチスレッド対話を実現する UIMS については、スレッドごとにユーザとの対話が独立しているのので、その対話に従来使用されているコマンド及びメニュー形式を用いることで、かなり汎用性が高くなると考えられる。

本稿では、マルチプログラミング環境（UNIX 4.3BSD）でのマルチスレッド対話のための動作時のユーザインタフェース部とアプリケーションルーチンの構成を提案し、その実現例を示す。

2. マルチスレッド対話

スレッドとは、MACH や OS/2 などのオペレーティングシステムにおいて、UNIX でいう資源管理の対象であるプロセス中に存在し、並列に動作可能なスケジューリングの対象となる単位である。

UNIX System V 系では、ライトウェイトプロセスに対応するが、UNIX BSD 系では1つのプロセスがそのままスケジューリングの単位となる。すなわち、1つのプロセスに対し1つのスレッドしか存在しないと考えられる。それに対し、1つのプロセス内に複数のスレッドが存在する場合をマルチスレッドという（図1参照）。このとき各スレッドはそれが属するプロセス内のデータを共有

することができるが、データのアクセスに対し排他制御が必要となる。

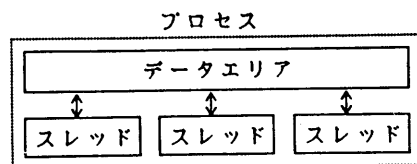


図1 マルチスレッド

本稿では、OS レベルでなくユーザから見たときの、共有データをアクセスしつつ並行実行可能なソフトウェアの単位をスレッドと呼ぶことにする。マルチスレッド対話型ユーザインタフェースとは、複数の並行して操作可能なスレッドを用意し、ユーザはそれらを任意の順序で実行しながら、全体として1つの仕事を達成可能とする。

マルチスレッド対話の利点としては、あるタスクを実行中に別のタスクを並行して実行できる。これにより、従来処理時間のかかるタスクを実行中は、他のどんなタスクもその実行が終了するまで処理できなかったが、簡単な処理の実行終了を待つことなく行なうことができる。また、あるタスクを実行するためのパラメータ設定中に、前処理のために別のタスクを実行する必要があることに気づいた場合、従来ならばパラメータ設定を中止し、前処理をした後で、再度設定する必要があった。しかしマルチスレッドならば、パラメータ設定を中断し、前処理をした後で、そのまま継続することができる。

以下に、本研究で実現するマルチスレッド対話の特徴をまとめる。

- ① あるコマンド実行のためのパラメータ設定中に、別のコマンドを実行可能
- ② 異なるスレッドのコマンドは、並行実行可能
- ③ 複数のスレッドで同じデータを異なる形式で表示
- ④ ③の場合、一方のデータ変更が他方の表示に反映
- ⑤ 各スレッドは、常に最新の情報を表示

次に、このような対話を効果的かつ容易に実現できるユーザインタフェースのモデルを考える。

3. ユーザインタフェースのモデル

これまで、アプリケーション動作時のモジュール構成を示すいくつかのモデルが提案されている。以下では、代表的なモデルについて考察し、マルチスレッド対話に適する新たなモデルを提案する。ここで、アプリケーションプログラムにおいてユーザとの対話処理を行う部分をユーザインタフェース部、それ以外でアプリケーション固有の機能を実現する部分を計算部と呼ぶことにする。さらに計算部は、各機能の処理に対応するルーチンと、それが扱うデータからなる。

3.1 Seeheimモデル

ユーザインタフェースのモデルとして、1983年にSeeheimモデル[GREEN85]が提案された。これは、ユーザインタフェース部をプレゼンテーション要素、対話制御要素、アプリケーションインタフェース要素の3つのモジュールからなる(図2参照)。

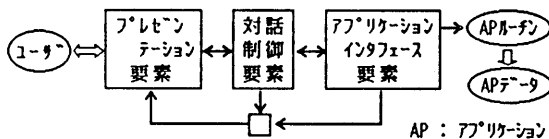


図2 Seeheimモデル

プレゼンテーション要素は、ユーザインタフェースの物理的な提示およびユーザからの入力処理をする。対話制御要素は、ユーザとプログラムとの対話の流れを管理し、アプリケーションインタフェース要素は、ユーザインタフェース部と計算部間のインタフェースを形成する。

たとえば、対話制御要素がプレゼンテーション要素に対しコマンドの入力要求を出すと、プレゼンテーション要素はコマンドのメニューを表示し、ユーザに選択を促す。対話制御要素は、選択されたコマンドを受け取るとアプリケーションインタフェース要素に対し、対応するアプリケーションルーチンの実行を要求する。プレゼンテーション

要素は、アプリケーションルーチンの実行結果をアプリケーションインタフェースを通して受け取り、ユーザに提示する。

3.2 MVCモデル

Smalltalk には MVC パラダイム[Shan89]に基づくユーザインタフェースのモデル、MVC モデル[前川88, 今宮88]がある。

MVC モデルは、Model, View, Controller の3つのモジュールからなる(図3参照)。

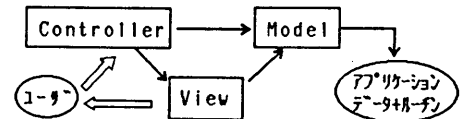


図3 MVCモデル

Model は、計算部の起動、およびアプリケーションデータへのアクセスを行う。これは、Seeheimモデルのアプリケーションインタフェース要素に対応するが、Seeheimモデルでは単にルーチンの呼び出しであるのに対し、Model は、データに対し処理を要求する。View は、ユーザとの対話のための視覚要素(メニュー、プロンプトなど)およびアプリケーションの情報をディスプレイ上に表示するモジュールであり、プレゼンテーション要素の表示機能に対応する。Controller は、ユーザからの入力を監視および対話の流れを管理し、View に対し対話のための視覚要素の表示を要求、Model に対してはアプリケーションルーチンの起動を要求する。これは、プレゼンテーション要素の入力処理機能と対話制御要素に対応する。

3.3 共有データモデル

本稿では、計算部をルーチンとデータに分離し、ユーザインタフェース部とルーチンでそのデータを共有するモデル、共有データモデルを提案する。

共有データモデルは、プレゼンテーション&対話管理要素、アプリケーションルーチン・インタフェース要素、および共有データ・インタフェース要素の3つのモジュールからなる(図4)。

プレゼンテーション&対話管理要素は、ユーザからの入力を受け取り、アプリケーションルーチン

ンの起動要求を行う。また、共有データ・インタフェースからのデータ変更通知によりユーザへのアプリケーション情報の表示を更新する。

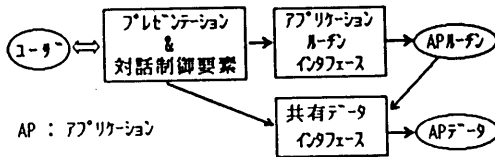


図4 共有データモデル

アプリケーションルーチン・インタフェース要素は、プレゼンテーション&対話制御要素からのアプリケーションルーチン実行要求により対応するルーチンを実行する。これは、Seeheim モデルにおけるアプリケーションインタフェース要素に対応する。

共有データ・インタフェース要素は、アプリケーションのデータに対するアクセス要求を処理する。また、データの更新があった場合は、ユーザへのフィードバックのために、プレゼンテーション&対話制御要素に通知する。

3.4 マルチスレッドへの適用

マルチスレッド対話を実現するためには、前述の各モデルでアプリケーションルーチンが並行して動作できる必要がある。そこで、アプリケーションの各機能をいくつかのグループに分け、各グループをそれぞれスレッドに対応させる。

まず、Seeheim モデルの場合、アプリケーションインタフェース要素から各スレッドに他のスレッドの実行終了を待たずに実行要求を出すことにより、並行実行は可能である。しかし、データへのアクセスは各ルーチンが行うので、データの排他制御機構をルーチン内に持たせなければならない。

次に、MVC モデルでは、データに対し処理を要求するので、データの排他制御は特に必要ない。しかし計算部の実現において、オブジェクト指向プログラミング環境がなければならない。また、データごとに処理ルーチンを用意するため、複数のデータを同時にアクセスするような機能の実現

が困難となる。

最後に、共有データモデルでは、Seeheim のようにルーチンの並行実行を簡単に実現でき、また各モジュールが共有データインタフェースを介してデータにアクセスするため、ルーチン内でなく共有データインタフェースにデータの排他制御機構を持たせることができる。

3.5 マルチスレッド共有データモデル

ユーザインタフェースをマルチスレッド化するために、図4のアプリケーションルーチンをグループ化し、そのグループごとにプレゼンテーション&対話管理要素を持たせる。ここで、アプリケーションルーチンとプレゼンテーション&対話管理要素の対をスレッドと呼ぶことにする。

論理的には、各スレッドが非同期にユーザからの入力を受け取り、対応するルーチンを実行し、結果をユーザに表示すればよい。しかし、実際は各スレッドでマウス、キーボード、ディスプレイなどの入出力デバイスを共有しているため、ユーザの入力に対応するスレッドに配り、また各スレッドからの出力をまとめて管理するための機構、入出力サーバが必要である。したがって、入出力サーバを用いるマルチスレッド共有データモデルを図5に示す。

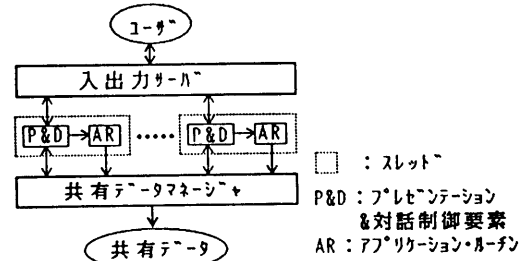


図5 マルチスレッド共有データモデル

ここで共有データは、単にユーザインタフェース部とアプリケーションルーチン間だけでなく、複数のスレッド間で共有される。そこで、各スレッドからの非同期なデータアクセス要求を一括して処理し、複数のスレッドが同時に同じデータへアクセス要求したときの排他制御機構として共有データマネージャを用意する。

4. マルチスレッド共有データモデルの実現

4.1 実現環境

マルチスレッド対話を実現するには、マルチプログラミング環境が不可欠である。そこで、OSとしてUNIX 4.3BSD, ユーザからの入力イベントの受け取りおよび表示管理のためにX Window System V11を使用する。したがって、入出力サーバとしてXサーバが対応する。さらに、プレゼンテーション要素の実現のために、X Window Systemに付随するXツールキットおよびウィジェットを用いる。Xツールキットは、C言語で書かれたライブラリの形式で提供されており、メニューやスクロールバーなどの対話技法を容易に実現することができる。

また開発言語としては、プロセス間通信の実現容易さおよびXツールキットを使用するためにC言語を用いることにする。

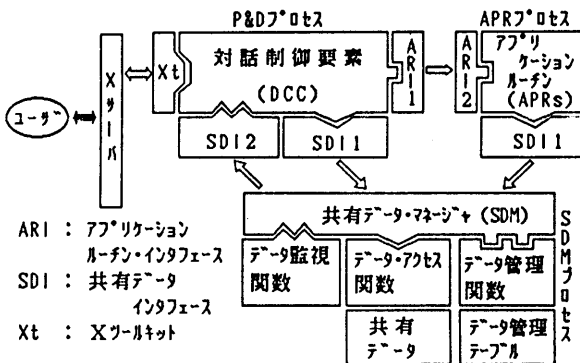


図6 マルチプロセスによる共有データモデル

以上の環境のもとで、マルチスレッド共有データモデルを図6に示すようにXサーバ以外に3種類のプロセスで実現する：プレゼンテーション&対話制御プロセス、アプリケーションルーチン・プロセス、共有データ管理プロセス。以下に、各プロセスごとの構成を詳細に説明する。

4.2 プレゼンテーション&対話制御(P&D)

プロセス

スレッドごとに存在し、次のモジュールからなる。

プレゼンテーション要素(Xt) :

Xツールキットおよびウィジェットを用いてユーザとの対話を実現する。Xサーバからユーザの入力イベントを受け取り、入力トークン(ある意味を持つ入力イベント列)を対話制御要素(DCC)に伝える。また、DCCからの出力トークンにより対応する表示要求をXサーバに送る。
対話制御要素(DCC) :

アプリケーションルーチン(APRs)の実行、および表示の更新を管理する。Xtからの入力トークンと共有データインタフェース(SDI)2からのデータ変更通知を監視し、アプリケーションルーチンインタフェース(ARI)1に対しAPRsの実行要求、SDI1に対しデータアクセス要求、およびXtに対し画面更新要求を関数呼出し形式で行なう。

アプリケーション・インタフェース1(ARI1) :

DCCからのAPRsの実行要求を、メッセージに変換してARI2に送る。

共有データ・インタフェース1(SDI1) :

DCCからのデータアクセス要求をメッセージに変換して共有データマネージャ(SDM)に送る。

共有データ・インタフェース2(SDI2) :

SMDからのデータ変更通知メッセージを受け、DCCに知らせる。

4.3 アプリケーションルーチン(APR)プロセス

P&Dプロセスと同様に、スレッドの数だけ存在する。

アプリケーション・インタフェース2(ARI2) :

アプリケーションの各機能を実現する関数の集合である。ARI2から起動される。

アプリケーション・インタフェース2(ARI2) :

ARI1からメッセージを受け、APRs内の対応する関数を呼び出し、その結果をARI1に返す。
共有データ・インタフェース1(SDI1) :

APRsの関数実行に伴うデータアクセス要求をメッセージに変換してSDMに送る。

4.4 共有データ(SD)管理プロセス

1つのアプリケーションに対し、1つ存在する。

共有データマネージャ(SDM)：

SDI1 からのメッセージにより、データ管理関数を用いて排他制御を行い、データアクセス関数を呼び出して値を返す。またデータに変更があった場合は、データ監視関数を通して対応する SDI2 に変更通知を行う。

データアクセス関数：

共有データの読みだし、書き込み等を行う関数の集合で、アプリケーション固有である。

データ管理関数：

共有データの排他制御を行う関数の集合で、共有データの情報を登録した管理テーブルを参照しながら動作する。この関数自体は、アプリケーションに依存しない。排他制御には、各データにキーを対応づけ、そのキーを獲得したものがアクセスできるという集中ロック方式 [STEPIK87]を採用している。

データ監視関数：

共有データの変更を監視し、そのデータに関係している P&D プロセスの SDI2 に対し変更通知を行なう。

4.5 プロセス間通信

ソケットシステムコールを用いてプロセス間通信を実現している。各プロセス間は、次の形式のメッセージ単位で通信を行う。

(command h'ラメ-タ1 h'ラメ-タ2) ①

これは、C言語における関数呼び出し

command(h'ラメ-タ1, h'ラメ-タ2,) ②

に対応する。したがって各インタフェースでは、①の形式のメッセージを受け②の形式で関数を呼び出す。また送信では、②の関数呼び出しを①のメッセージに変換して対応するインタフェースに送る。

各インタフェース間の通信内容は；■ SDI1 から SDM：共有データの使用およびアクセス要求。■ ARI1 から ARI2：アプリケーションルーチンの実行要求。■ SDM から SDI2：共有データの更新通知。

現在これらの各モジュールは、C言語を用いて

実現してある。関数呼び出しを共通にすることによって、各インタフェースを取り除き、全体を1つのプロセスとして構築することも可能である(図7)。これにより、プロトタイプ作成・テストが容易となる。

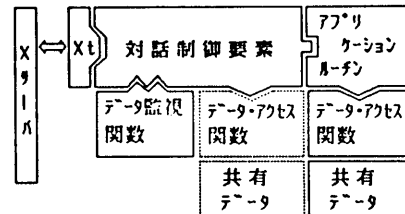


図7 テスト時の構成

以下では、共有データ管理プロセスで行なう管理方法についてもう少し詳細に説明する。

5. 共有データ管理

1つのアプリケーションを複数のスレッドで構成するため、各スレッド間で同じデータをアクセスできなければならない。また、2つ以上のスレッドが並行して同じデータを処理しようとするとき、あるスレッドが読み出す前に他のスレッドがそのデータを変更してしまうと、そのデータの処理結果は間違ってしまう。この現象を防ぐためにも、ひとつのデータへの同時アクセスを防止する「排他制御」が必要である。

このため、各スレッドからの共有データへのアクセスは、SD管理プロセスへのメッセージ送信により一括して行う。また、各データの排他制御の必要な単位(以後、資源と呼ぶ)ごとにキーを用意し、データ管理関数を用いて管理する。

5.1 管理関数と管理テーブル

2つの管理テーブルと、5つの管理関数を用意する。

まず、データ管理テーブルは、

(1) 資源ロックテーブル(図8a)：

各資源に対し、その資源が使用中かどうかの状態(Locked：使用中, Unlocked：使用可能)と、使用中の資源に対しそれを要求している資源の待ち行列を持つ。

(2) 資源要求テーブル (図 8 b) :

各スレッドが、必要としている (まだ獲得していない) 資源を記録 (Wait : 資源要求) .
からなる.

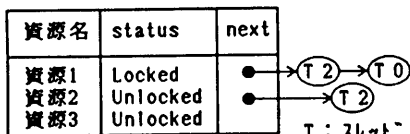


図 8 a 資源ロックテーブル

	資源1	資源2	資源3
T0	Wait		
T1			
T2	Wait	Wait	

図 8 b 資源要求テーブル

次に、データ管理関数を示す.

(1) CHECK 関数 :

スレッドからのメッセージを処理可能かどうかをチェックする.

(2) LOCK 関数 :

必要な資源のキー獲得 (ロック) を試み、獲得できれば、資源ロックテーブルを更新し OK を、そうでなければ FAIL を返す.

(3) WAIT_LOCK 関数 :

必要な資源のロックを試み、獲得できなければ資源要求テーブルと資源ロックテーブルの NEXT 欄に、情報を追加する.

(4) UNLOCK 関数 :

資源のロックを解除する.

(5) ACTIVATE 関数 :

資源待ちの適当なスレッドに資源を与える.

ここで、共有データマネージャが CHECK と ACTIVATE 関数を使用し、他は各スレッドが SDI を通して使用する.

5.2 アプリケーションルーチン実行手順

各スレッドは、5.1 節に示した関数をデータアクセス関数と組み合わせて処理を進める. 各アプリケーションルーチンの実行前に、必ず使用する資源のキーを獲得し、実行終了後解除することにより、他のスレッドとの関係を気にすることなく

処理することができる.

図 9 に、その手順の例を状態遷移図で示す. 図 9 a では、実行しようとしたときに、必要な資源が使用中ですぐには実行できなかった場合、ユーザに待つかどうかの確認をとる. また図 9 b では、すぐ実行できるかどうかに限らず、実行をする場合である.

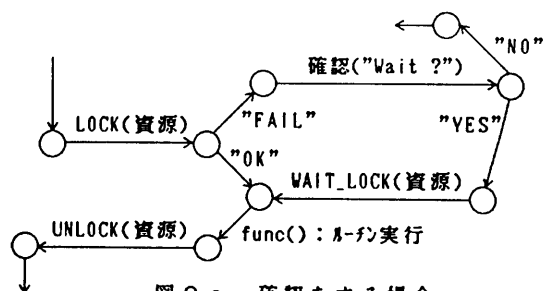


図 9 a 確認をする場合

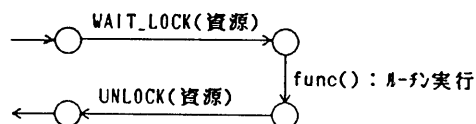


図 9 b 確認なしの場合

それぞれ、円 (○) が状態、矢印 (→) が遷移を表す. 矢印に付けられたラベルは、遷移に伴う関数呼出し、またはその関数の返り値による遷移条件 (" " で囲まれた文字列) を示す.

ここで、func() はアプリケーションルーチンの呼出しを意味し、その中では自由にデータのアクセス要求を行うことができる.

5.3 管理アルゴリズム

共有データマネージャは、各スレッドからのメッセージをポーリングして待ち、1 メッセージ単位で処理を行う. 図 10 にそのアルゴリズムを示す. まず 現在のスレッド (Ti) に対し、CHECK 関数により Ti がすでに必要な資源を獲得しているか、または何も資源を要求していないことを確認し、メッセージを処理してよいかどうか調べる. OK ならばメッセージを読み込み、もしメッセージがあれば処理し、次のスレッドの処理に移る.

```

while (1) {
    for (i=0; i<スレッド数; i++)
        if (check(Ti) == OK)
            if (Ti からのメッセージ != NULL)
                switch(要求) {
                    case LOCK :   lock(Ti, 資源);
                                   break;
                    case WAIT_LOCK : wait_lock(Ti, 資源);
                                   break;
                    case UNLOCK : unlock(資源);
                                   activate(資源);
                                   break;
                                   :
                                   :
                }
            }
}

```

図10 共有データ管理アルゴリズム

また図8は、3つのスレッド、T0 T1 T2 から非同期に

```

T0 : (LOCK 資源1)(WAIT_LOCK 資源1) ..
T1 : (LOCK 資源1) .....
T2 : (WAIT_LOCK 資源1 資源2) .....

```

のメッセージを受け、それぞれメッセージの到着順に処理したときの管理テーブルの状態を示している。各スレッドごとのメッセージのずれは、到着時刻の差を示している。

6. 実現例

通産省の情報処理技術者試験で用いられている仮想計算機 COMET のシミュレータを、本モデルに基づき4つのスレッドで構成する。以下に各スレッドの機能を示す。

- ①仮想端末スレッド：プログラムのロード，実行および CASL の I/O 命令に対する入出力
- ②メモリ・スレッド：メモリ内容の表示，変更
- ③レジスタ・スレッド：レジスタ値の表示，変更
- ④デバッグ・スレッド：トレース，ブレークポイントの設定，実行。および行単位のアセンブル，逆アセンブルと結果の表示。

また，COMET 計算機のメモリとレジスタを共有データとし，データアクセス関数として，メモリとレジスタの読みだし，書き込みを用意する。

個々のスレッドでの対話は，メニュー形式で行なう。

7. おわりに

本稿では，ユーザインタフェースの動作時のモデルとして，マルチスレッド共有データモデルを提案し，マルチプログラミング環境での実現例を示した。このモデルに基づきアプリケーションシステムを実現することにより，マルチスレッド対話を容易に実現できる。現在のところ，まだ COMET のシミュレータを試作した段階で，具体的にモデルの評価をするに至っていない。そこで今後，様々なタイプのアプリケーションを実現することにより，本モデルの有効性，適応性そして動作時の性能評価を行う予定である。

また現在，このモデルに基づくユーザインタフェースの生成を支援するユーザインタフェース管理システム (UIMS) の開発を行っている。そこでは，図9に示すように状態遷移で対話の流れを記述することで対話制御プログラムの生成を可能とする。さらに，ARI などの各種インタフェースプログラムの作成支援のツールを開発中である。

謝辞

本研究において，多大なる助言を頂いた山梨大学工学部 電子情報工学科の渡辺喜道 助手，坂本忠明技官，また，プロトタイプ作成において協力してくれた同大学4年生の山本，柴田両君に感謝致します。

参考文献

- [東87] 東基衛 他訳：ユーザー・インタフェースの設計，日経マガロ社，1987，155-187
- [Green85] Green M. : The University of Alberta User Interface Management System, ACM SIGGRAPH'85, Vol.19, No.3, July 1985, 205-213
- [今宮88] 今宮，関村：ウィンドウ指向対話管理システム，情報処理学会 研究会報告とCADシンポジウム，1988年10月，43-51
- [前川88] 前川守 監訳：ウィンドウ指向のプログラミング，第9章，トナリ，1988
- [Shan89] Yen-Ping Shan : An Event-Driven Model-View-Controller Framework for Smalltalk, ACM OOPSLA'89 Proceedings, October 1-6, 1989, 347-352
- [STEFIK87] MARK STEFIK, et al : BEYOND THE CHALKBOARD: COMPUTER SUPPORT FOR COLLABORATION AND PROBLEM SOLVING IN MEETINGS, CACM, Vol.30, No.1, 1987, 32-46
- [Tanner87] Tanner, P.P. : Multi-Thread Input, ACM SIGGRAPH'87, Vol.21, No.2, 142-145