ヒューマンインタフェース 35−19
（1991. 3. 8）

マルチエージェントモデルに基づく グループウェア構築ツールMicheleの提案

中内 靖　伊藤 嘉邦　安西 祐一郎

慶應義塾大学 理工学部

Micheleは非同期通信に基づいたグループウェアを構築するために開発されたマルチエージェントモデルに基づくグループウェア構築ツールである。本論文ではユーザ間での処理の流れを表すことのできるMicheleのマルチエージェントモデル、エージェントの振舞いを記述するための言語MDL、Micheleの分散環境に対応した実装について説明する。

Michele: A Groupware Toolkit based on Multiagent Model

Yasushi Nakauchi　Yoshikuni Itoh　Yuichiro Anzai

Faculty of Science and Technology, Keio University

3-14-1, Hiyoshi, Kohoku-ku, Yokohama 223, Japan

Michele is a groupware toolkit based on multiagent model. Michele is developed to construct groupware based on asynchronous communication. In this paper we describe that multiagent model of Michele can describe flow of processing in cooperative work, multiagent model description language MDL, and its implementation in the distributed computational environment.

1 はじめに

計算機利用による生産性の向上は、ワードプロセッサや各種ビジネスソフトの開発により個人に閉じた環境では成果を上げてきた。また近年、オフィスや研究機関ではネットワークに接続されたパーソナル・ワークステーションが普及するのにともない、電子メールに代表されるLANによる通信技術を個人が容易に利用できるようになってきた[4]。そこで、このような分散環境を利用して個人だけではなく、組織としての生産性の向上を目的とした計算機による協調作業支援システム(グループウェア)の研究が活発に行なわれてきている[3] [5]。

これまでに組織の生産性を向上させることを目的として、電子スケジューラ・システム、オフィスオートメーション・システム、グループの意思決定支援システムなど様々なアプリケーションが開発されてきている。これらのグループウェアが単一目的であるのに対して、また一方では様々なグループウェアを構築する枠組としてLIZA [2]、Object Lens [6]といったグループウェア構築ツールが提案されている。ところが、LIZAではオフィスの定型業務のように仕事の依頼が書類として届き、ある程度時間が経ってからそれが処理されるような非同期的な通信に対応しておらず、またObject Lensでは、複雑な処理の流れを記述する能力がない。

そこで、我々は非同期通信に基づくグループウェア構築ツールMichele (Multi-agent Interface with Communication by Hectic ELEments) [9]を提案する。非同期通信を分散環境において実現するためにはユーザや他の構成モジュールをうまく表現することのできるモデルが必要である。我々はこの目的を達成するためにマルチエージェントモデルを選択した。このモデルを利用することにより、分散環境において協調作業を行なうユーザ、協調作業に必要とされる知識、ユーザ間で受け渡される書類など、協調作業を構成する任意の要素をエージェントとしてモデル化することができる。Micheleにより提供される幅広い枠組を利用することによりアプリケーションに依存しない様々なグループウェアの構築を可能にする。Micheleは我々が既に開発したグループウェアMulti-Cooperator [8]を分散環境においてより幅広く対応できるようにマシン依存性を軽減し、さらにシステムが学習することによりユーザを支援できるよう、知識獲得の機能を追加して改良したものである。

本論文では以下の第2節でMicheleが提供するマルチエージェントモデルについて説明し、第3節ではエージェントの振舞いを記述するために開発された言語MDL (Multi-agent Description Language)について説明する。そして、第4節では分散環境に対応した実装について説明する。そして、最後にMicheleの将来の展望について述べる。

2 モデリング

2.1 協調作業のモデル

非同期な通信に基づくグループワークには簡単なものではアンケート調査から、オフィスの定型業務の自動化や組織の意思決定支援など様々な形態が考えられる。このような協調作業を表現するモデルはこれらの振舞いをモデル化するのに充分な柔軟性を持っている必要がある。以下に非同期通信に基づくグループウェア構築ツールに要求される機能について述べる。

アンケート調査は一般に図1に示すような手続きにしたがって実行される。これらの手続きは書類に対する手続きを中心に考えると、書類に対する作成・送付・受理・評価の4種類の手続きから構成されていると考えられる。

また、オフィスの定型業務も基本的には4種類の書類に対する手続きから構成されていると見なすことができる。ただし、オフィスの定型業務では数種類の書類が同時にやり取りされ、それらのうち特定の書類が揃ってはじめて次の処理に移れる場合がある。そこで、オフィスの定型業務を扱うためには書類の受理の手続きにおいて、特定の書類の到着を待ち合わせる機能が必要である。ま

1. 質問者がアンケート用紙を作成する。
2. 質問者がアンケート用紙を回答者に送付する。
3. 回答者がアンケート用紙を受理する。
4. 回答者がアンケートの内容を評価する。
5. 回答者がアンケートの回答を作成する。
6. 回答者がアンケートの回答を質問者に送付する。
7. 質問者がアンケートの回答を受理する。
8. 質問者がアンケートの回答の統計値を計算 (評価) する。

図 1: アンケート調査の手続き

た、オフィス業務は海外にある支店や提携会社との間で遂行される場合も多い。そのような分散環境にも対応していることが望まれる。

組織の意思決定支援システム [5] では図 2に示すように検案を繰り返すことにより決定事項の質の向上を図ることがよくある。図 2の「好ましい案の作成・提示」と「改訂案の聴取」の繰り返しはアンケート調査の繰り返しと見なすことができる。そこで、このような意思決定支援システムを構築する場合に、アンケート調査モジュールのように既に作成されたモジュールをより高度なシステムを構築する場合のサブモジュールとして利用できると便利である。

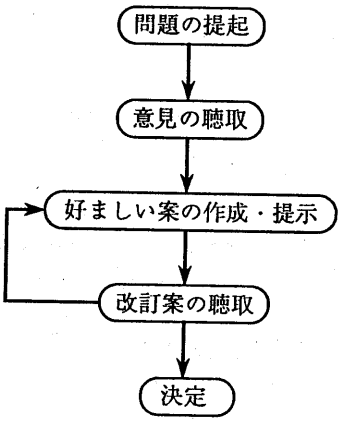

図 2: 組織の意思決定の手続き

以上に見てきたように、非同期通信に基づくグループワークではユーザ間のインタラクションは書類による仕事の依頼として表現することができる。そして、書類による仕事の依頼は基本的には書類の作成・送付・受理・評価の 4 種類の手続きを利用できればよい。またグループウェアでは複数のユーザが複数の仕事を同時に遂行することがあり、その場合、複数の書類は同時にユーザ間で受け渡されて並列処理されることになる。そのため、特に書類の受理に関しては複数の書類を待ち合わせる機能が必要である。

さらに、分散環境に対応したグループウェアとして実用的なシステムを構築するためには、上述した基本的な機能を用意するだけでなく以下に示す 3 つの特性も合わせ持つことが望まれる。

独立性 ワークステーションが疎結合された分散環境では、各ユーザが直接アクセス可能な範囲には制限がある。そこで、システムの構成要素はモジュール化され、ユーザは必要なモジュールだけを参照すればシステムを利用できることが望まれる。

構築性 意思決定支援システムを構築する場合、アンケート調査の手続きを利用できると便利である。このように、システムの拡張を容易にするためには、既に構成されたモジュールを他のモジュールのサブモジュールとして利用できることが望まれる。

適応性 協調作業では人間を相手に仕事が進められるため、一連の作業が必ずしも予め決めておいた手続き通りに進むとは限らない。そこで、ユーザが協調作業の途中経過を知ることができ、状況に応じて手続きに変更を加えられることが望まれる。

2.2 協調作業のマルチエージェントモデル

2.1に述べた 3 種類の特性のうち独立性と構築性を満足するためには、その構成要素はモジュール化されている必要がある。そこで、我々はユーザも含めてシステムを構成する要素すべてをエージェ

ントと呼ばれるモジュールとして表現する。ここでエージェントとはActorモデル[1]でのActorのようなものであり、固有の内部状態を持ち、そのエージェントに関する手続きをメソッドとして持つものである。そして、エージェント間の通信はすべてメソッドコールとして実行される。エージェント内のアクティビティは一つであり、あるメッセージに対する処理中に届く他のメッセージは現在処理中のメソッドの処理が終るまで待たされる。また、エージェントの活動状態にはready・active・deadの3種類があり、エージェントは生成されると最初はready状態になる。そして、他のエージェントによりメソッドの手続きが呼び出されるとそのエージェントはactive状態になる。また、エージェントはdead状態になることにより消滅される。

例えばエージェントは、「部長」や「秘書」といったワーカ、アンケート調査での「アンケート用紙」やオフィスワークでの「請求書」といった書類、または「アンケート用紙を回収し集計を行なう」といったある機能単位を一つにまとめたものを表現することができる。書類に相当するエージェントでは、その所有者が現在その書類に関する仕事に責任があることを表している。したがって、書類の所有者を明確にし、処理の流れを明示的にする必要がある。既存のマルチエージェントモデル[1]ではユーザを処理の対象として考慮されておらず、したがってユーザ間での処理の流れを明示的にモデル化することはできない。そこで、我々はエージェントの所有権を明確にするためにユーザ環境と呼ばれる概念をモデルに導入する。

ユーザ環境とは協調作業に参加するユーザに1つずつ存在するもので、そのユーザ環境内にあるエージェントは対応するユーザの所有物となり、他のユーザは直接アクセスすることはできない。また、エージェントは一度にはどれか一つのユーザ環境内にしか存在できず、エージェントがユーザ環境間を移動することによりその所有権が移譲される。

したがってエージェント間の通信には、同一ユーザ環境内の場合と他のユーザ環境間との場合の二通りがあり、それぞれの通信は以下のように行なわれる。同じユーザ環境内に存在するエージェントに対して通信を行なう場合、エージェントはメソッドコールにより直接通信を行なうことができる。このときの呼び出しは遠隔手続き呼び出し(Remote Procedure Call)[11]と同様であり、呼び出した側のエージェントの処理は呼び出されたエージェントの処理が終了するまで待たされる。一方、異なるユーザ環境に存在するエージェントに対して通信を行なう場合、通信元のエージェントが通信先のエージェントの存在するユーザ環境に移動して通信を行う。このとき、通信元のエージェントの処理のために新たにプロセスが生成される。すなわち、通信元のエージェントを呼び出していたエージェントが存在した場合、そのエージェントの処理と通信元のエージェントの処理は並行実行される。またこのとき、通信元のエージェントの所有権はがユーザ環境間を移動することにより移動先のユーザに移譲される。

システムで用意されたエージェント(以下システム・エージェントと呼ぶ)とユーザの間の通信では以下に示すような問題がある。まず、システム・エージェントは常にメッセージを受理できてその処理を行なうことができるのに対して、[1]ユーザはlogin時しかメッセージを受理することができない。また、エージェント間のメソッドコールをシステム・エージェントとユーザが同じ表現を用いているのでは、ユーザにとって理解するにはあまりにも煩雑である。そこで、システム・エージェントとユーザ間でのメッセージの表現の差異を埋めるためにMI(Message Interpretor)を導入する。MIはウィンドシステムによって実現さる。MIによりユーザとシステム・エージェントの間でのメソッドコールは、ユーザとシステム・エージェントの双方から見て理解し易い表現に翻訳される。ここで、ユーザにMIをかぶせたものを特にユーザ・エージェントと呼んでシステム・エージェントと区別する。図3にアンケート調査のマルチエージェントモデルを示す。

[1]ただし、排他制御のため一度には一つのメッセージだけが処理される。

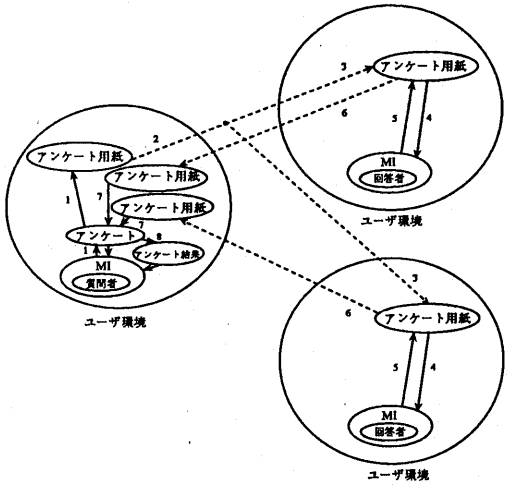

図 3: アンケート調査のマルチエージェントモデル

3 エージェント記述言語

Michele のマルチエージェントモデルによりモデル化された協調作業を計算機に支援させるためには、マルチエージェントモデルにより概念レベルで表現されたエージェントの振舞いを計算機に理解できる表現に素直に変換できる言語が必要である。本節で Michele のマルチエージェントモデルを記述するために開発されたエージェント記述言語 MDL (Multi-agent Description Language) について説明する。

MDL のプログラムはエージェントの宣言の並びから記述される。エージェントの宣言は図4に示すように、エージェントの内部状態を表わすフィールドの並びと他のエージェントから呼び出すことのできるメソッドの並びから構成される。フィールドの記述では初期値をとることもできる。そして、メソッドの記述では引数をとることができ、メソッドの手続きの記述なかで最後の評価式がそのメソッドの戻り値となる。ここで、“エージェントの説明” と “メソッドの説明” はそれぞれエージェント名とメソッド名の別名であり、図5に示すようにユーザが MI を通してエージェントを見た時のそれぞれエージェントの名前と実行可能な

```
(DEFAGENT ≪エージェント名≫ "エージェントの説明"
    (FIELD ≪フィールド名:≫ [≪初期値≫])
        ⋮
    (METHOD ≪メソッド名≫ ≪引数の並び≫ "メソッドの説明"
        ≪メソッドの手続≫)
        ⋮
)
```

図 4: エージェントの宣言

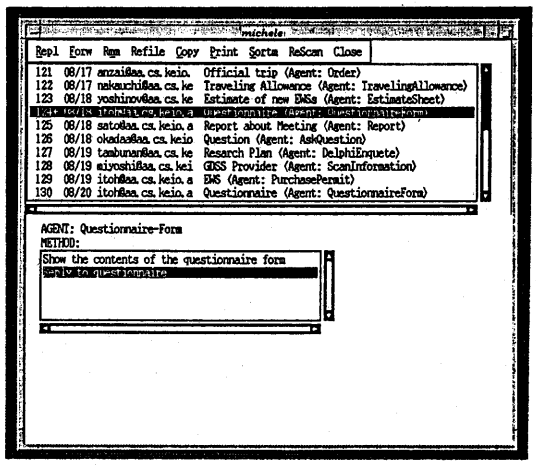

図 5: MI を通して見たシステム・エージェント

手続きの名前として現れる。このようにエージェント記述では、エージェントの振舞いについての記述だけでなく MI の仕様についても記述される。

エージェントのフィールドにはエージェントの状態を表すためにフィールド “activity:” がデフォールトで付けられる。このフィールドの値を “dead” とすることによりエージェントを消滅させることができる[2]。メソッドコールは一般に

(≪メソッド名≫ ≪IP[3]≫ ≪引数の並び≫)

と記述される。2.2にも述べたように並行動作するエージェントの間の通信の同期は遠隔手続き呼び出しと同様で、起動したエージェントから起動されたエージェントに制御権が移り、その実行終了を伝える確認情報（戻り値）が返されるまで起動側のエージェントは待つことになる。

[2]正確には、あるメソッドの手続き実行終了時に “activity:” フィールドの値が “dead” となっているとそのエージェントは消滅させられる。

[3]インスタンス・ポインタ (Instance Pointer)

2.1に示した組織の協調作業を支援するために必要な機能を実現するために、各エージェントには基本的なメソッドがシステムであらかじめ用意されている。

書類の作成は用紙の用意と書き込みによって行なわれる。これらの操作はそれぞれ、エージェントのインスタンスの生成と、インスタンスのフィールドに値をセットによって実行される。

エージェントのインスタンス生成は関数Newを用いて

(New ≪エージェント名≫)

と記述され、戻り値として生成されたインスタンス・ポインタが返される。また、C++ [12] と同様にエージェント名と同じ名前のメソッドは構築子と呼び、インスタンス生成と同時に呼び出される。さらに、Micheleではこの構築子を持つエージェントだけがユーザから直接インスタンスを生成できるエージェントと定義し、これを基本エージェントと呼ぶ。したがって、基本エージェントだけがユーザが最初に起動できるサービスとしてMIに登録される。

そして、フィールドの値の操作はフィールドの値を読み出すメソッドGetとフィールドに値をセットするメソッドPutによって実行される。

メソッドコールGetとPutはそれぞれ

(Put ≪IP≫ ≪フィールド名:≫ ≪値≫)

(Get ≪IP≫ ≪フィールド名:≫)

と記述される。

エージェント間の通信は、同じユーザ環境内では上に示したメソッドコールとして実行されるが、ユーザ環境間の通信の場合は以下に示す関数Migrateによって実行される。

(Migrate ≪ユーザ環境名≫ ≪手続きの並び≫)

エージェントは関数Migrateを実行することにより、第一引数で指定されたユーザ環境[4]に移動する。そして、第二引数以降に記述された手続きが移動先のユーザ環境内で実行される。関数Migrateの呼び出しでは新たにプロセスが生成され、そのメソッドを呼び出したエージェントと関数Migrateの処理は並行実行される。したがって、関数Migrateは戻り値をとることはできない。

関数Migrateにより書類の送付と受理が記述される。書類の送付は対象となる書類が関数Migrateを実行することにより実行される。また、書類の受理は到着した書類が受理される相手に対してメソッドコールすることにより実行され、メソッドコールされたエージェントは書類の受理にともなう手続きを実行する。そして、複数の書類の選択的な受理はエージェントの内部状態および届いた書類の内容にしたがって処理を分岐することによって記述される。また、複雑な手続きも記述できるように、メソッドの手続きの記述にはLispの豊富な関数群を利用することができる。そして、書類の評価はユーザが定義するメソッドに手続きの並びとして記述される。

2.2で述べたように、ユーザ・エージェントはユーザにMIをかぶせたものである。したがって、ユーザ・エージェントとシステム・エージェント間のメソッドコールでは、システム・エージェントからはMIに対するメソッドコールとして、また、ユーザからはMIにより示されるメニューによる手続きの選択として実行される。システム・エージェントとユーザ・エージェント間のインタラクションは主にユーザに対する情報の提示と問い合わせの2通りである。そこで、MIには質問を受け付け、これに答えるための2種類のメソッドShowとAskを用意した。メソッドShowはユーザへの情報の提示を、そしてメソッドAskはエディタまたはメニューを用いてユーザに情報の問い合わせを行う。メソッドShowとAskはそれぞれ、

(Show ≪IP≫ "フォーマット文字列" ≪引数の並び≫)

(Ask ≪IP≫ [:editor | :menu "メニューの並び"])

と記述される。

4 処理系

4.1 電子メールの利用

Micheleの実装ではエージェントの内部状態を保持する方法とエージェント間の通信を分散環境

[4]ユーザ環境名はネットワーク内でのユーザの識別名、すなわちメールアドレスが指定される。

において実現する方法が重要である。以下に、それぞれの実現方法について説明する。

エージェントの内部状態は複数のフィールドとその値から構成されており、アンケート調査などの協調作業では一連の仕事が終了するまで存在し続けなくてはならない。そこで、Michele ではエージェントの内部状態をフィールドとその値から構成される半構造化メッセージ [6] [7] として表現する。半構造化メッセージはメイルのヘッダの構造と同じ形をしており、メイルシステムによるメッセージの転送を利用し易い。

エージェント間の通信には同一ユーザ環境内の場合と異なるユーザ環境間の場合の2通りがある。同一ユーザ環境内でのエージェント間の通信はメソッドコールによる手続きの呼び出しであり、これはメソッドに対応した関数を呼び出すことにより実現する。また、同一ユーザ環境内に存在するエージェントは実際にはあるユーザのホーム・ディレクトリ内ファイルとして存在させることにより、ワークステーション間の通信機構を必要とはしない。

一方、異なるユーザ環境間での通信は通信先のエージェントが相手先のユーザ環境に移動し、移動先のユーザ環境においてユーザ環境内のメソッドコールを実行する。エージェントの移動はエージェントの内部状態を保持した半構造化メッセージをメールシステムを利用して転送することにより実現する。したがって、エージェント間の通信はメールの届く環境であれば実行可能であり、Michele は TCP/IP で通信できる環境よりもさらに広い環境において利用することができる。

4.2 ユーザ環境間の通信機構

異なるユーザ環境間の通信では他の環境からのメッセージが到着すると、関数 Migrate の第二引数以降に記述された手続きを起動し実行する必要がある。この働きをするのが ICS (Inter-user-environment Communication Support system) である。Michele ではエージェントの移動にメールシステムを利用するため、到着するメッセージが普通のメールであるかエージェントであるかを判別しなくてはならない。ICS はユーザ環境に到着するメッセージを監視しており、それがエージェントの内部状態を持つものであった場合、関数 Migrate に第二引数があるかどうかを判断して対応する手続きを実行する。また、普通のメールが到着した場合には、ICS は Information Lens [7] と同様にユーザによって記述されたルールにしたがい、フォルダに分類して格納する。

4.3 メソッドの手続きの実行

Michele のマルチエージェントモデルではエージェントが複数のユーザ環境間を移動するため、一つのエージェントの手続きが異なるユーザ環境で実行されることがある。分散環境では異機種の計算機が存在するため、エージェントのメソッドの手続きはどの計算機上でも実行可能な形になっている必要がある。Multi-Cooperator [8] ではエージェントの手続きは実行前にあらかじめコンパイルして計算機の機種毎に実行オブジェクトを作成しておき、これを各ユーザ環境から参照できるようにしていた。そのため、分散環境において計算機の機種毎に異なるオブジェクト管理することは非常に繁雑であった。そこで Michele ではエージェントの手続きを Lisp のコードにコンパイルしておき、Lisp の処理系の上で実行できるようにした。また、エージェントの実行コードの管理についてはエージェントの内部状態を保持する半構造化メッセージに附属させ、エージェントの移動と同時に実行コードも移動されるようにした。

エージェントのメソッドの呼び出しには3通りの場合がある。それらはユーザが MI を介して呼び出す場合、ICS がエージェントの到着を検知して呼び出す場合、エージェントがメソッドの手続きの中で他のエージェントを呼び出す場合である。上述したメソッドの手続き実行の手法を用いることにより、3通りいずれの場合においても Lisp の処理系が実装されている計算機上であれば計算機の機種に依存することなくメソッドの手続きを実行できるようになった。また、分散環境において実行オブジェクトの管理が容易になった。現在、Michele の処理系は複数機種のワークステーショ

ン (SONY NEWS、SUN-3、OMRON LUNA) 上で稼働中である。

5 おわりに

本論文では非同期通信に基づくグループウェア構築ツール Michele を提案した。Michele のマルチエージェントモデルでは、分散環境において協調作業を行なうユーザ、協調作業に必要とされる知識、ユーザ間で受け渡される書類など、協調作業を構成する任意の要素をエージェントとしてモデル化することができる。さらに、Michele はグループウェアに要求されると考えられる独立性・構築性・適応性の 3 つの特性を持つ。

また、本論文ではエージェントの振舞いを記述するために開発された言語 MDL について説明し、さらに、Michele の分散環境に対応した実装について説明した。我々はさらに Michele を発展させるべく、扱えるデータのマルチメディア化・学習機構の導入 [10]・ユーザに利用しやすいプログラミング環境の開発を試みている。

参考文献

[1] Agha, G. : *ACTORS: A Model of Concurrent Computation in Distributed Systems,* MIT Press, 1986.

[2] Gibbs, S. J. : Liza: An Extensible Groupware Toolkit, *CHI '89*, (1989), pp.29-35.

[3] 石井 裕: グループウェア技術の研究動向, *J. Inf. Proc. Soc. Japan*, Vol.30, No.12, (1989), pp.1502-1509.

[4] 板倉 節男: LAN とワークステーションを用いた電子メール通信, *J. Inf. Proc. Soc. Japan*, Vol.28, No.8, (1987), pp.1030-1037.

[5] Kraemer, K. L. and King, J. L. : Computer-Based Systems for Cooperative Work and Group Decision Making, *Comput. Surv.*, Vol.20, No.2, (1988), pp.115-146.

[6] Lai, K. Y. and Malone, T. W. : Object Lens: A "Spreadsheet" for Cooperative Work, *CSCW '88*, (1988), pp.115-124.

[7] Malone, T. W., Grant, K. R., Lai, K. Y., Rao, R. and Rosenblitt, D. : Semi-Structured Messages are Surprisingly Useful for Computer- Supported Coodination, *CSCW '86*, (1986), pp.102-114.

[8] Nakauchi, Y., Itoh, Y., Sato, M. and Anzai, Y. : Modeling and Implementation of Multiagent Interface System for Computer-Supported Cooperative Work, *Ergonomics*, 1991. (in press)

[9] Nakauchi, Y., Itoh, Y., Sato, M. and Anzai, Y. : Michele: A Multi-agent Interface Architecture for Distributed Open Environments, *Tools'91*, 1991.

[10] Nakauchi, Y., Okada, T., Itoh, Y. and Anzai, Y. : Groupware that Learns, *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1991. (in press)

[11] Nelson, B. J. : Remote Procedure Call, Ph.D.dessertation, Carnegie-Mellon University, (1981).

[12] Stroustrup, B. : *The C++ Programming Language*, Addison-Wesley, 1986.