グループワークを考慮した協調計算システムにおける動作プログラム群の生成と分散実行

伊東達雄[†,☆] 今城広志[†,☆] 岡野浩三[†]
東野輝夫[†] 松浦敏雄[†,☆☆] 谷口健一[†]

グループワークの支援システムを複数の計算機による協調計算システムとしてみなすことができる．そこで，グループワークの各作業をゲートに対するデータの入出力と（ファイル等を表す）レジスタの更新式として抽象的にとらえ，グループワーク全体の作業順を拡張有限状態機械モデルとしてモデル化する．また，その記述（全体動作仕様）と，各計算機へのゲートとレジスタの配置指定等から，グループワークを支援する各計算機の作業手順（該当する作業者の作業手順やデータの入出力のほか，他の計算機とのデータの交換などの通信動作も記述されたプログラム）を自動生成するアルゴリズムを，従来のアルゴリズムを拡張することによって考案した．また，この導出アルゴリズムの処理系（生成系）を作成し，導出時間の評価を行った．また，導出した動作プログラム群を並列に実行する分散実行系を作成し，実行時間等の評価を行った．本アルゴリズムは，全体動作仕様と配置指定等を区別することにより，作業者へのグループワーク内の作業の割当ての変更に柔軟に対応できる等の利点がある．また，各処理系の実験評価から本方法の有用性を確かめることができた．本実行系では，音声や画像情報なども扱える．特に線描画ファイル編集では複数の作業者で同時に共同編集が行える機能を実現した．また作業者間の意志の疎通を容易にするためのサブシステム（画像や音声情報の交換機能）等を用意するなどの工夫をしている．

Derivation of Protocol Entities' Specifications of Distributed System for Groupwork and Their Parallel Execution

TATSUO ITOH,[†] HIROSHI IMAJO,[†,☆] KOZO OKANO,[†] TERUO HIGASHINO,[†]
TOSHIO MATSUURA[†] and KENICHI TANIGUCHI[†]

In this paper, a groupware is treated as a distributed system where the contents and ordering of the actions in a group work are described as a service specification in an extended FSM model. In order to ensure the temporal ordering of the actions described in a service specification, each person in the group must exchange some data values and synchronization messages each other. The description of each person's work flow including both data exchanges and his own work contents is called a protocol entity specification. In this paper, we propose an algorithm for deriving the protocol entities' specifications from a service specification and a resource allocation. We have implemented the algorithm where the derived protocol entities' specifications are given as C programs and they can be executed in parallel on UNIX systems. We have also developed a system to support the group work in those protocol entities' specifications. The system has some facilities to represent sound data and pictures. The users can edit the same figure simultaneously and each modification is shown to the displays of all users where the conflicts of modifications are well solved. The system also has communication and drawing tools so that the users can exchange their ideas in the group conveniently. Our approach is useful especially for the case that the service specification and resource allocation are frequently changed.

1. まえがき

近年，グループウェア[1]に関する設計法や処理系について，多くの研究が行われている[2)~4)]．

グループワークの各作業者に1つの計算機を対応させる．このとき，各計算機は，グループワークの流れにそって作業者に対してデータの入力を促し，またデ

† 大阪大学基礎工学部情報工学科
Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University
* 現在，日本電気株式会社
Presently with NEC Corporation
** 現在，大阪市立大学学術情報センター
Presently with Osaka City University

ータの出力・加工（エディト等）を行い，グループワークを支援する．各計算機はお互いにデータや同期のためのメッセージ交換を行いながら並行に動作するため，全体を複数の計算機からなる協調計算システムと見なすことができる．

このようなグループワーク支援のための協調計算システムを作成する1つの方法として，以下の手順が考えられる．

（1）グループワーク全体としての処理の制御の流れを記述する（以降，全体動作仕様と呼ぶ）．

（2）グループワークのうちどの動作を(協調動作として）多重化するか（以降，多重化指定と呼ぶ），各計算機（ノード）にどの動作やデータを割り当てるかを指定する（以降，リソースの配置指定と呼ぶ）．

（3）上記(1)と(2)からノードの外部から見て全体動作仕様通りに動作するような各ノードの（データの交換や，同期のためのメッセージの送受信を含む）支援プログラム（動作仕様）を導出する．

(1)の全体動作仕様は，幾つのノードでどのように実行するかといった具体レベルの記述は行わず，あたかも1つの計算機で実行するようにモデル化する．本論文では拡張有限状態機械（EFSMモデル）として以下のようにモデル化する．データベースやファイルなどグループワークを通して参照，生成されるリソースを抽象的にレジスタとして扱う．データの表示，入力といった動作をゲートに関する入出力動作とする．ファイルの結合，変更等を（適当な基本関数による）レジスタの更新式として捉える（ここでは「各レジスタが具体的にどのノードにあるか」という情報には無関係に,全てのレジスタを参照,更新できるものとする）．入出力動作と複数のレジスタの更新を1つの状態遷移とし，状態遷移の実行条件とともに各状態遷移で有限状態制御部を記述する．

(2)ではノードの集合を定め，各ゲートの多重化指定と，各ゲートやレジスタをどのノードに割り当てるかを表す配置指定を定める.各ゲートの多重化指定は，複数の作業者による共同作業や，同一データの同時出力，あるいは，データ入力の競合などを表す．各ゲートの配置指定はどの入出力動作をどの計算機(作業者)が行うべきかの指定に相当する．また，各レジスタの配置指定はデータが実際にどのノードにあるかを指定する．

各ノードの動作仕様は，データ交換や同期用のメッセージの送受信動作が含まれるため繁雑なものとなり，自動生成できることが望ましい．近年，協調計算システムの全体動作仕様から各ノードの動作仕様を自動生成する種々の方法が，研究されされている[6]．我々の研究グループでも，EFSMモデルで全体動作仕様を記述し各ノードの動作仕様（EFSMモデル）を自動生成するアルゴリズムを考案した[7]．

本論文ではその導出アルゴリズムのグループワーク支援システム設計への応用について述べる．文献7)ではゲートの多重割り当てを考慮していないが，グループワークでは，複数の作業者が同時に同一のデータを入出力する場合が多いので，これに対応するため，ゲートの多重化の概念を導入する．また，本論文で用いる全体動作仕様のクラスでも自動導出できるよう導出アルゴリズムを拡張した[8]．また，本論文では，このEFSMモデルに基づくグループワーク支援システムとその生成系について述べる．(1)生成系は上述のアルゴリズムに基づき各ノードの動作仕様を自動生成する[8]．(2)グループワーク支援システム（実行系）は生成系から導出された動作仕様群をグループワーク支援のための協調計算システムとして実行する[9]．この実行系ではデータとして数値，文字列などの他に音声や画像情報などが扱える．特に，作業者間の共同作業の1つとして共同編集が重要と考えられる．そこで,同一の線描画ファイルを複数人で同時に共同編集できる機構を実現した．さらに，共同作業を円滑に行うための作業者間のコミュニケーションツールとして，黒板システムと会話システムを作成した．これは実行系のサブシステムとして利用できる．このように実行系では共同作業を考慮した工夫を種々行っている．また，生成系の導出時間や，実行系の実行時間等を計測し，その結果から本方法や，システムの有効性を評価した．

我々のシステムは，全体動作仕様とその他の指定等から各ノードの動作仕様を自動生成するという手法を用いているので，全体動作仕様の変更や機能の追加などに対しても，再度動作仕様を自動生成することにより容易に対処できるという利点をもつ．また，同じ全体動作仕様であっても，作業分担者やリソースの配置指定のみ変更が生じる場合もある．この場合でも，多重化指定や配置指定のみを変更して容易に異なる協調計算システムを自動生成できる．

以下,2章で記述モデルとその意味,ならびに全体動作仕様の例について述べる．3章で生成系と実行系の機能について述べる．4章では実行系の実行例について述べ，5章でシステムの評価を行う．

2. 記述モデルと全体動作仕様例

2.1 記述モデルと意味

設計者は全体動作仕様SSを拡張有限状態機械

(EFSM)で記述する．EFSMは，有限状態機械に有限個のレジスタを加えたもので，以下の6字組 $M=(S, R, G, V, \delta, init)$ で定義する．

S, R, G, V はそれぞれ有限制御部の状態の有限集合 $\{s_1, \cdots, s_n\}$，有限個のレジスタの集合 $\{r_1, \cdots, r_m\}$，有限個のゲートの集合 $\{a, b, \cdots\}$，入力変数の集合 $\{x, y, \cdots\}$ である．δ は状態遷移関数の集合で，各状態遷移関数は $s-\langle C, IO, CR\rangle\rightarrow s'$ で表す．ここで C はレジスタと入力変数を引数とする述語で，遷移条件と呼ぶ．IO は次の2種類の入出力動作である．$a?x$ はゲート a からデータを入力し，その値を入力変数 x で参照することを表す．$a!E(\cdots)$ は式 $E(\cdots)$ の値をゲート a に出力することを表す．E はレジスタ $r_1, \cdots, r_m$ またはその一部を引数とする関数である．入出力動作を何も行わないときは，内部動作を表すシンボル i を用いる．CR はレジスタ更新のための代入文の集合で，その代入文をレジスタ更新式と呼ぶ．各代入文の右辺の引数には，レジスタ値と入力変数値を表す変数が許される．$init$ は初期状態と各レジスタの初期値を指定する．

EFSMは以下のように動作する．ある状態において，その状態から出ている全ての遷移の遷移条件を評価し，真となるもののうち1つを非決定的に選択する(遷移の選択)．その後，入出力動作，レジスタの更新を行い，次の状態へ移る．レジスタの更新は，全ての代入文の右辺の値の評価を行った後，全ての代入を同時に実行（同時代入）する．

協調計算ではファイルなどの編集作業を簡便に記述できる必要がある．いま f を編集作業のためのツール名とし，あるテキスト t を f を用いて編集し編集後のテキスト $f(t)$ を作成する場合を考える．いま，ツール f にある入力(コマンド) x を与えた時に得られるテキストを $f'(x, t)$ で表すことにする．一般に編集作業では n 回のコマンド入力の実行後最終的なテキスト $f(t)=f'(EXIT, f'(x_{n-1}, \cdots, f'(x_2, f'(x_1, t))\cdots))$ が得られる(ここでは終了時に $EXIT$ を入力すると仮定する)．このような繰返しを我々のモデルで記述するため，編集作業に相当する動作 $a\#$ を用いて以下のように略記する．

$$s-\langle C, a\#f(R), \{R'=f(R), \cdots\}\rangle\rightarrow s'$$
$$\equiv s-t_1\rightarrow s_1,\ (s_1-t_2\rightarrow s_2-t_3\rightarrow s_1)^*,\ s_1-t_4\rightarrow s'$$

ここで

$$t_1\equiv\langle C, a!R, \{tmpR=R\}\rangle,$$
$$t_2\equiv\langle true, a?x, \{tmpR=f'(x, tmpR)\}\rangle,$$
$$t_3\equiv\langle true, a!tmpR, \{\ \}\rangle,$$
$$t_4\equiv\langle true, a?"EXIT", \{R'=tmpR, \cdots\}\rangle$$

であり，レジスタ $tmpR$ や状態 s_1, s_2 はこの略記ごとにあるとする．すなわち，$a\#f(R)$ という動作は，ゲート a にレジスタ R の値を出力し，その値を新たなレジスタ $tmpR$ に保存し，ゲート a からコマンド x を何回か入力し，$tmpR$ を関数 f' で変更し，最後に R' に代入するという遷移系列を表している*．

2.2 全体動作仕様の例

我々の手法では，設計者が全体動作仕様を記述する段階では，いくつのノードで協調計算システムを構成するかを意識する必要はない．従って，ノード間のデータやメッセージの送受信は陽に記述しない．

図1は状態遷移図で表現した全体動作仕様 SS の例である．この例は，建築の設計をモデル化したものである．建築の設計は意匠設計 (Design for Exterior)・構造設計(Strength Analysis)・設備設計(Design for Interior)の3つからなり，それぞれの設計は異なるグループが担当する．レジスタは r_1, r_2, r_3, r_4, r_5, r_6, r_7 の7つ，ゲートは a, b, c, d の4つが用いられている．まず意匠設計グループで設計図を作成する ($s_0\rightarrow s_1$)．この遷移の $a\#Edit\_Key3(r_1)$ は前述の編集作業 $a\#f(R)$ の具体的な表現である．この遷移の意味は，レジスタ r_1 の値（設計図）を編集ツール Key 3[14] を用いて編集することである．次に構造設計グループが構造設計の観点からその設計図を改良する ($s_1\rightarrow s_2$)．次に設備設計グループが設備設計の観点から設計図を改良する ($s_2\rightarrow s_3$)．その次に設計図は再び意匠設計グループに渡され，意匠設計グループは改良する必要があれば意匠設計を行う ($s_3\rightarrow s_4$)．そしてその設計図を構造設計グループに渡し再度同じ作業を行うか，設計を終了するかを決定する ($s_4\rightarrow s_5$)．このときレジスタ更新式 $r_4=r_1$,

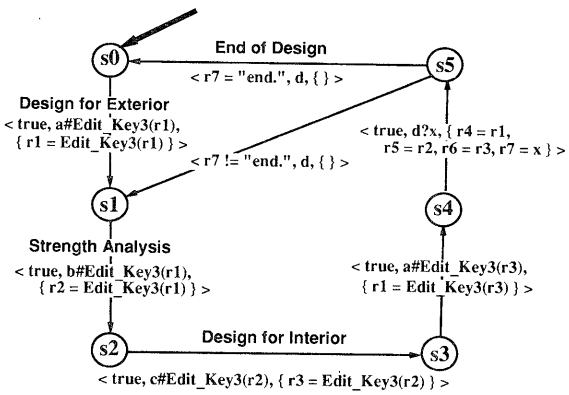


図1 EFSMで記述された全体動作仕様の例
Fig. 1 Service specification.

* なお，もし，レジスタ更新式で $R'=f(R)$ がなければ，t_4 のレジスタ更新式で $R'=tmpR$ もなく，したがって，この遷移系列の実行は，レジスタ R を含め（$tmpR$ 以外の）もとのどのレジスタ値にも影響を与えない．

$r_5=r_2$, $r_6=r_3$ によって，構造設計・設備設計・意匠設計終了後の設計図 (r_2, r_3, r_1) がバックアップのためレジスタ r_5, r_6, r_4 に格納される．

2.3 分散システム上の動作モデル

複数のノードからなる協調計算では，ノード間で通信が行われる．ノード間の通信に関して，我々のモデルでは，ノード i からノード j への通信路を無限の容量を持つ FIFO キューとしてモデル化し，その両端をゲート g_{ij} で表す．よって，通信路に対するデータの読み書き（メッセージの送受信）はゲート g_{ij} での入出力動作と考える．これによって，各ノードの動作仕様も全体動作仕様と同様に EFSM モデルで記述できる[7]．各ノードの EFSM は，自ノードに配置されたレジスタを参照，更新することができ，また自ノードに配置されたゲートに関する動作は行えるが，他ノードにあるレジスタを直接参照，更新することはできないし，また，他ノードにあるゲートに関する動作は行えない．全ノード (p 個) の動作仕様を合わせたものを動作仕様群とよび，PE^{1-p} で表す．

2.4 動作仕様群の導出問題

この節では，全体動作仕様と動作仕様群 PE^{1-p} との等価性と導出問題を与える．

一般にグループウェアにおいては，複数人で作業を行う場合，入力は誰から行っても良いし，全員に同じ出力を見せたいことが多い．このことを EFSM で記述するには，以下のような状態遷移を記述することになる．

例えば，入力がゲート a_1, a_2, a_3 のいずれからあっても良いということは，**図2**(1)の左のような状態遷移系列を記述することによって行われる．同様に，ゲート b_1, b_2 に同一出力が連続して起きることは，図2(2)の左のように記述する．

このような状態遷移系列は，図2の(a)の左の状態遷移に関するゲートの集合に対して論理ゲート a を新たに与えて，ゲート a のただ1つの入力動作と見なす．また図2の(b)の場合も同様に論理ゲート b のただ1つの出力動作と見なす．このことより全体動作仕様の記述が簡便化できる（図2の右）．

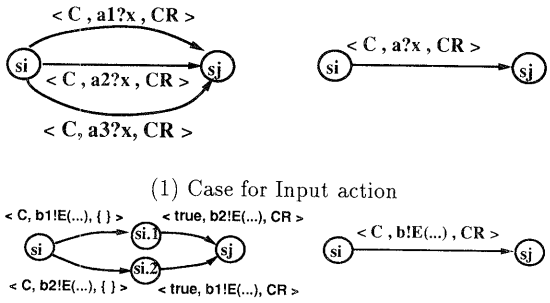


図2 ゲートの多重化指定
Fig. 2 Multi assignment of gates.

逆に，このような略記を用いた全体動作仕様と，論理ゲートから具体ゲートの部分集合への多重化指定（例えば，$\{a\mapsto\{a_1, a_2, a_3\}, b\mapsto\{b_1, b_2\}\}$）から，もとの状態遷移系列を陽に記述した全体動作仕様を表すことができる．略記された全体動作仕様 SS と多重化指定 σ で決まる全体動作仕様を $\mathrm{Ref}(SS, \sigma)$ で表す．

設計者は略記された全体動作仕様 SS を記述用に用いる．この利点は，ゲート多重化指定 σ を変更することにより，SS 自体を変更することなしに，容易に他の $\mathrm{Ref}(SS, \sigma)$ を与えることができることである．

一方，$\mathrm{Ref}(SS, \sigma)$ を直接用いると，以下のような欠点もある．例えば，$\mathrm{Ref}(SS, \sigma)$ に，同一出力動作を行うゲートの数が n 個あれば，図2の(b)の状態遷移の枝の数は $n!$ 個になる．このような複雑な仕様は設計者が記述するのは，繁雑であるし，全体の見通しが悪くなる．

例えば，2.2節の全体動作仕様の例を略記された全体動作仕様 $S1$ とみなすと，この $S1$ と $\sigma=\{a\mapsto\{a_1, a_2\}, b\mapsto\{b\}, c\mapsto\{c\}, d\mapsto\{d\}\}$ に対する $\mathrm{Ref}(S1, \sigma)$ は**図3**のようになる．

$\mathrm{Ref}(SS, \sigma)$ と動作仕様群 PE^{1-p} との等価性は以下のように定義する．

[等価性]

PE^{1-p} で内部通信用のゲート g_{ij} に関する動作をすべて非観測動作とし，$\mathrm{Ref}(SS, \sigma)$ と PE^{1-p} が観測合同[12]であるとき，かつそのときのみ $\mathrm{Ref}(SS, \sigma)$ と PE^{1-p} が等価であるという．

動作仕様群 PE^{1-p} を $\mathrm{Ref}(SS, \sigma)$ から導出する際，各（具体）ゲートや各レジスタがどのノードに配置されるかを指定する必要がある．この指定を（リソースの）配置指定 al と呼ぶ．al として，例えば，$\mathrm{Ref}(S1, \sigma)$ に対して，ノード数を4とし，以下のような配置を考える．

	ノード1	ノード2	ノード3	ノード4
ゲート	a_1, d	a_2	b	c
レジスタ	r_6	r_1, r_6, r_7	r_2, r_4	r_3, r_5

なお，これは，事実上，σ と合わせて，$S1$ の論理ゲートに対して，以下のようなゲートの多重割り当てとゲート名の指定（例えば，ノード2の a_2 を a と読み変える等）を行ったことに等しい．

	ノード1	ノード2	ノード3	ノード4
ゲート	a, b	a	b	c

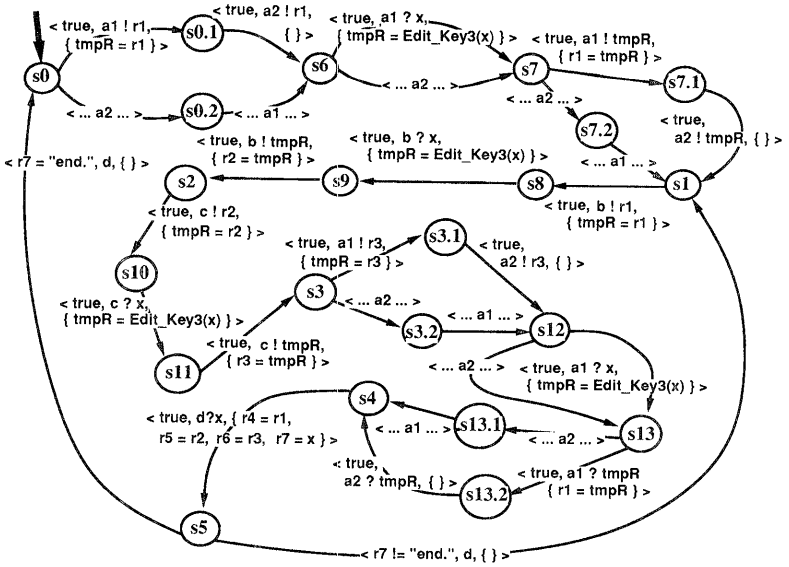


図3 Ref($S1$, σ)
Fig. 3 Ref($S1$, σ).

この配置によって各ノード，つまり各人がどのような仕事を担当しなければいけないかが決まる．この場合ゲート a が意匠設計，ゲート b が構造設計，ゲート c が設備設計，ゲート d がこのグループワークの管理を表し，例えばノード1に相当する人が意匠設計とグループワークの管理を行うことを表す．

［導出問題］

（略記した）全体動作仕様 SS とゲート多重化指定 σ および，配置指定 al を与えたとき，Ref(SS, σ) と等価な動作仕様群 PE^{1-p} を求める問題．ただし，配置指定 al は PE^{1-p} で満たされていること．

2.5 動作仕様群の導出アルゴリズム

我々は文献7)で，全体動作仕様とリソースの配置指定から各ノードの動作仕様を自動生成するアルゴリズムを提案している．その生成法では，全体動作仕様の各状態遷移を，選択動作の通知，更新に必要なレジスタ値の転送，レジスタ値の更新，更新終了の通知などの一連の動作系列で置き換えることにより実現している．状態遷移の決定を各状態での唯一のノード（責任ノード）が行うこととし，また，各ノードでのレジスタの更新終了の通知を次の状態の責任ノードに送信することによって，ノード間の同期を実現し，デッドロック等を回避している．またノード間のメッセージ送受信の総数を0-1整数線形計画問題の手続きを用いることによりできるだけ少なくしている[7]．本論文でも基本的にこのアルゴリズムを採用する．

文献7）のアルゴリズムでは1つの状態において2つ以上の状態遷移がある場合，それらの遷移条件を1つのノードで判定できるようにするため，それらの状態遷移（入出力）を実行する責任ノードは1つでなければならないという制約を課している．全体動作仕様 SS に対してこの制約を課しても，Ref(SS, σ) へ展開することにより，同一状態から異なる入力動作が現れる場合が生じる．そこで，この制約を満たすようにするため，このような状態に対しては，各状態遷移に対する責任ノードの数だけ状態を設け，それらの状態間に図4のようなループ状の非観測動作（これらは操作権を表すメッセージ（トークン）の送受信動作に対応付ける）からなる遷移を設け，責任ノードに順に入出力の実行の機会が回るようにする．これにより1つの状態から出ている全ての遷移の責任ノードを1つにしている．Ref(SS, σ) と al に対して，この変換を行った全体動作仕様を Imp(SS, σ, al) と呼ぶ．Imp(SS, σ, al) のようにループ状の非観測動作を加える変形を行っても変形の前後で観測合同性が保存される[13]．よっ

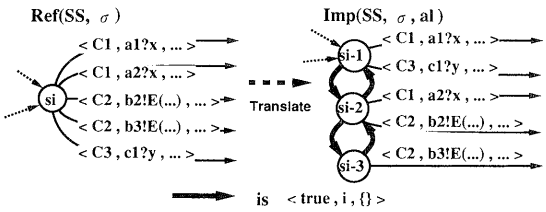


図4 Ref(SS, σ) から Imp(SS, σ, al) への変形
Fig. 4 Translation of Ref(SS, σ) into Imp(SS, σ, al).

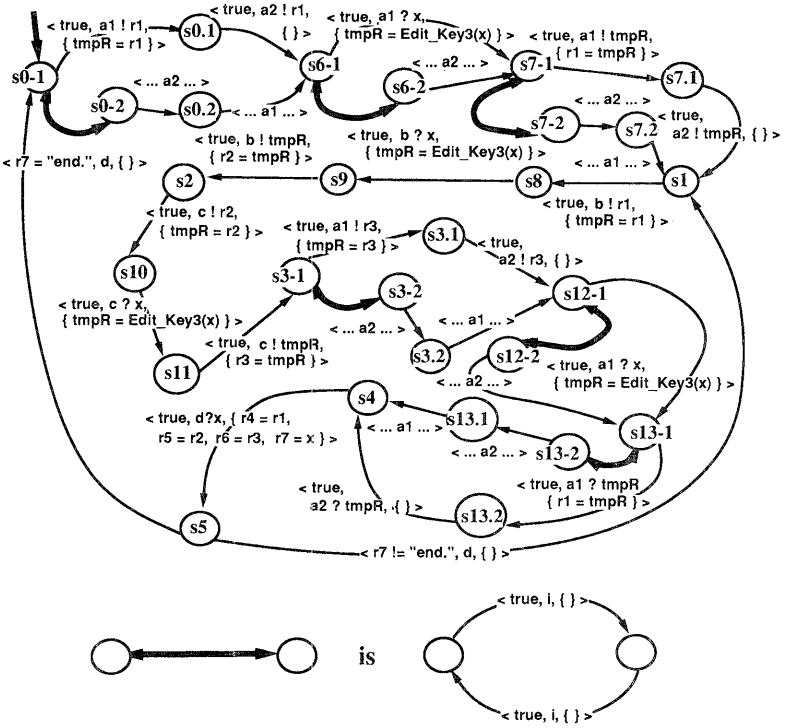


図5 Imp($S1, \sigma, al$)
Fig. 5 Imp($S1, \sigma, al$).

て，このような変形を行っても Ref(SS, σ) と Imp(SS, σ, al) は観測合同である．本例題の Ref($S1, \sigma$) に対する Imp($S1, \sigma, al$) は図5のとおりである．Imp(SS, σ, al) と al を文献7) のアルゴリズムの入力とすると，Imp(SS, σ, al) と本論文で仮定している通信環境のもとで等価な動作仕様群 PE^{1-p} を導出することができる．

Ref(SS, σ) と Imp(SS, σ, al) および，Imp(SS, σ, al) と PE^{1-p} が観測合同であり，観測合同性は推移律で閉じているので，Ref(SS, σ) と PE^{1-p} の等価性が保証できる．すなわち，本導出アルゴリズムは妥当である．

2.6 分散制御方式の利点

上記の方法によって導出された各ノードの動作仕様は分散実行される．この分散制御方式に対して，サーバがすべての作業の管理を行う集中制御方式もある．単に実行制御だけ考えるなら集中制御の方が分散制御よりも簡単に対処できる．しかし分散制御方式は以下の利点を持つ．各ノードつまり各個人の動作仕様によって，各個人がどのような作業をどのような順に実行するのか，自分の作成したデータをだれが必要としているか，といった全体のグループワークの中での個人の作業が明確にできる．また，これにより，各個人の作業量の把握，全体動作仕様の修正やリソース配置情報の修正に伴う各人の作業量の変化の把握，などが容易となる．

3. グループワーク支援システムと生成系

3.1 システムの構成

前章で全体動作仕様を記述するためのモデルとそのモデルから各ノードの動作仕様を自動導出するためのアルゴリズムを述べた．我々は，このアルゴリズムをもとに動作仕様群を自動生成する**生成系**と，その動作仕様群を並列に実行して全体として協調計算を実現する**実行系**（グループワーク支援システム）を作成した．我々のシステムはこれら2つによって構成される．

生成系の入力は，(1)全体動作仕様 SS を記述したファイル，(2)ノード数 P やゲートの多重化指定，配置指定，ゲートと実際の入出力デバイスとの対応表を記述したファイル，(3)レジスタと入力変数の型を記述したファイル，(4)全体動作仕様で用いられたユーザ定義関数をC言語で記述したファイル，の4つであり，設計者が各ファイルの内容を記述する．

例えば，(2)のファイルの対応表で抽象的なゲート名と具体的な入出力デバイスとの対応付けが明確になる．また(3)のファイルでレジスタの型が宣言される

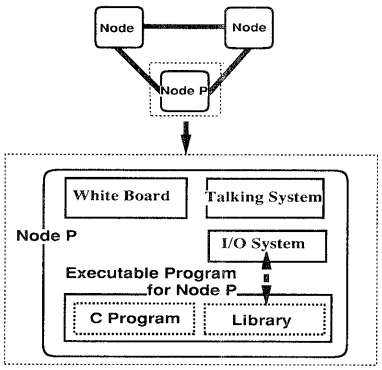


図6 実行系
Fig. 6 Simulation system.

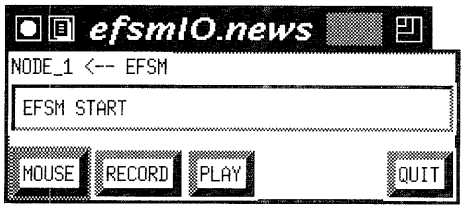


図7 メインウィンドウ
Fig. 7 Main window.

が，その型が，例えば整数等の一般の型の場合，レジスタの値はプログラムに内蔵された変数によって保持される．しかし，ファイル型の場合，レジスタの値自身は外部ファイルで保持され，プログラム内部ではそのファイルの名前のみが保存されており，その名前を通して実際のファイルとアクセスする．

生成系ではこれらの入力に対して構文・意味チェックを行った後，前章のようにEFSMモデルで記述された P 個の動作仕様群 $PE_1, \cdots, PE_P$ を自動生成する．最後に各動作仕様 PE_k はCプログラム（動作Cプログラム $C-PE_k$）に変換される．

実行系は動作Cプログラムを実行して全体として協調計算を実現する．実行系の構成を**図6**に表す．実行系は，以下で述べる実行可能プログラム，入出力システム，ならびに支援ツールである黒板システム・会話システムから構成される．

実行可能プログラムは，動作仕様Cプログラム（$C-PE_k$）と，実行に必要な種々の機能を提供する動作仕様実行ライブラリとをリンク，コンパイルすることによって各ノードごとに得られる．実行可能プログラムとユーザとのインタフェース部として入出力システムがある．これによりXのdisplay上にウィンドウを複数起動し，入出力が行われる．**図7**に入出力システムのメインウィンドウを表す．上から順にノード名，メッセージウィンドウ，操作ボタンである．

表1 データ型とそれに対する機能
Table 1 Data types and functions.

	整数	文字列	テキスト	ビットマップ	線描画
入力	○	○	—	—	○
出力	○	○	○	○	○
演算	○	○	○	○	○
代入	○	○	○	○	○
編集	—	—	○	○	○
共同編集	—	—	—	—	○

3.2 実行系の機能

我々の実行系では，データとして整数や文字列の他に，テキストファイル，ビットマップファイル，線描画ファイル，音声を扱うことができる．これらのデータに対し，データの入出力や（Cの関数として定義できる任意の）数値などの演算・代入が行える．例えば音声の入出力は入力ファイルで指定されたオーディオインタフェースを介して行われる．音声の入力はメインウィンドウのRECORDボタンを押すことによって開始し，ボタンを放すまで入力ができる．音声の出力はメインウィンドウのPLAYボタンをクリックすることにより開始できる．

また，テキストファイル，ビットマップファイル，線描画ファイルの編集が行える．これらの編集の際には，指定されたエディタが自動的に起動される．特に，線描画ファイルの編集では，1つの線描画ファイルを複数人で同時に編集できる線描画共同編集機能を実現した．

以上を**表1**にまとめる．テキストやビットマップの入力を実現していないが，これらはファイル編集操作で代用できるため実用上は問題ない*．

また実行系は各ノードのウィンドウにメッセージを表示し，現在どの動作を行う必要があるかを作業者に明示する（例えばエディタの自動起動など）．このように我々のシステムはユーザに対して作業の誘導を行っている．

3.2.1 共同編集機能

全体動作仕様で $a\#f(R)$ という表現があれば，それはファイル編集を行うことを意味する．例えば全体動作仕様に $a\#Edit\_Key3(R)$ と記述できる．なお，システムでは $Edit\_Key3$ という表現と実際のツールであるKey 3とを対応付けた表を持っており，作業者が使用したいツールをこの対応表に記述することによって自由にツールを使用できる．また，このときゲート a

* 各作業者ごとに異なるファイルを割り当て，個々で独立に編集し，お互いにレビューしあうような作業手順は本モデルでも記述できる．

が複数のノードに多重に割り当てられていたとすると，ゲート a を持つノードがお互いに共同してファイルを編集する．ここではその共同編集について述べる．

この共同編集機能は編集ツール Key 3 の機能を共同編集用に拡張することによって実現した．現在のシステムでは Key 3 でのみ共同編集が可能である．共同編集を行う各ノードでは Key 3 が起動し，それらノード間にはトークンが回っている．各ノードでは入力された変更情報をバッファに溜めておき，トークンが到達したときバッファに入力があれば，その変更情報を共同編集を行う他のノードに伝える．各ノードは変更情報を受信すると Key 3 に対し図形の変更情報を出し，その後もとのノードに更新完了のメッセージを返送する．ほぼ同時に 2 つのノードから同じ図形に対する異なる更新情報が出された場合，先にトークンを確保したノードが更新を行い，残りの更新情報は入力されなかったものとみなされる（再度更新情報を出すことは可能）．このように，共同作業の実現の仕方は，2 章で述べた方法に基づいて行われている．

3.2.2 コミュニケーションツール

グループワークでは全体動作仕様に記述した共同作業以外にもメンバー間での相談のようなインフォーマルな会話機能があれば，作業者間でより意志の疎通がとれ，その作業効率が上がる場合がある．しかしそのような動作をあらかじめ全体動作仕様に記述するのは困難であり，またたとえそのような記述が可能であっても，その分だけ状態と遷移が増えて全体動作仕様が理解しにくくなる可能性が高い．よってそのような動作は，全体動作仕様には記述せず，全体動作仕様の枠組から外れた特別な動作としてとらえることにする．その結果，全体動作仕様の動作と独立した，ノード間の意志交換を支援する補助ツールとして，黒板システムと会話システムを作成した（図 6 参照）．

黒板システムは，上述の Key 3 による共同編集機能を実行系とは独立なシステムとして常時立ち上げておき，複数人が同一のキャンバスに図形や文字を読み書きできるようにしている．このシステムにより作業者間で図形や文字などの情報交換を行える．

会話システムは，複数人で同時に会話を行えるシステムである．会話権利を得たノードの利用者がマイクを使って話し，その音声が他ノードに送信され，出力される*．この機能によって作業者間で音声によるインフォーマルな情報交換が行える．

* 同時に話すことのできる話者の数は一人に限定されており，音声を入力したいノードの利用者は会話権利を得る必要がある．

4. システムの実行例

ここではシステムの実行例について述べる．図 1 の全体動作仕様と 2.4 節のリソース配置指定に対して 4 つのノードに対する動作仕様が生成される．

今，図 1 の状態 s_0 にいるとする．ゲート a はノード 1 とノード 2 に配置されているので，それぞれのノードにおいて線描画編集エディタ Key 3 が自動的に起動し，レジスタ r_1 の値を読み込んで共同編集を行う．このときのノード 1 の画面は図 8 のようになる．共同編集ではノード 1 とノード 2 の間でトークンが回され，各ノードはトークンを受け取った時に責任ノードとして画面の変更要求を出すことができる．編集が終了すると，レジスタ更新式で $r1 = Edit\_Key3(r1)$ とあるので，編集した値がレジスタ r_1 に代入される．レジスタ r_1 を持つノード 2 は代入が終了すると終了合図となるメッセージを次の状態 s_1 の責任ノードであるノード 3 に送信する．ノード 3 はそのメッセージを受信して初めて次の遷移の選択と実行を開始する．

状態 s_1 では構造設計が行われるが，ゲート b はノード 3 にしか配置されていないので単独の編集作業となる．状態 s_2，s_3 でも同様の動作が行われる．

状態 s_4 ではゲート d が配置されているノード 1 が責任ノードとなり，データ x を入力する．入力が終了すると各ノードでレジスタ更新が行われる．しかし更新を行うノードが更新に必要なレジスタ値をすべて持っているとは限らない．よって更新に必要なレジスタ値を他のノードから転送しなければならない．その場合，責任ノードはレジスタ値を持っているノードに対してレジスタ値の転送要求メッセージを送信し，そのメッセージを受けたノードは必要とするノードにレジスタ値を転送する．また，責任ノードが持つレジスタ値や入力値については責任ノードから直接送られる．この例では，ノード 1 からノード 3, 4 にレジスタ値の転送要求メッセージが送信され，ノード 2 に入力データ x の値が送られる．

これらのメッセージ（または x の値）を受信すると，ノード 2 はレジスタ r_1 の値をノード 3 に，ノード 3 はレジスタ r_2 の値をノード 4 に，ノード 4 はレジスタ r_3 の値をノード 1, 2 にそれぞれ送信する．これらのメッセージの送信は並列に行われる．次に，必要なレジスタ値を受信したノードはレジスタを更新し，更新終了メッセージを次の状態 s_5 の責任ノードであるノード 1 に送信する．これらの動作も並列に行われる．ノード 1 は更新終了を知らせるメッセージをすべて受信してはじめて次の遷移の選択と実行を開始する．この

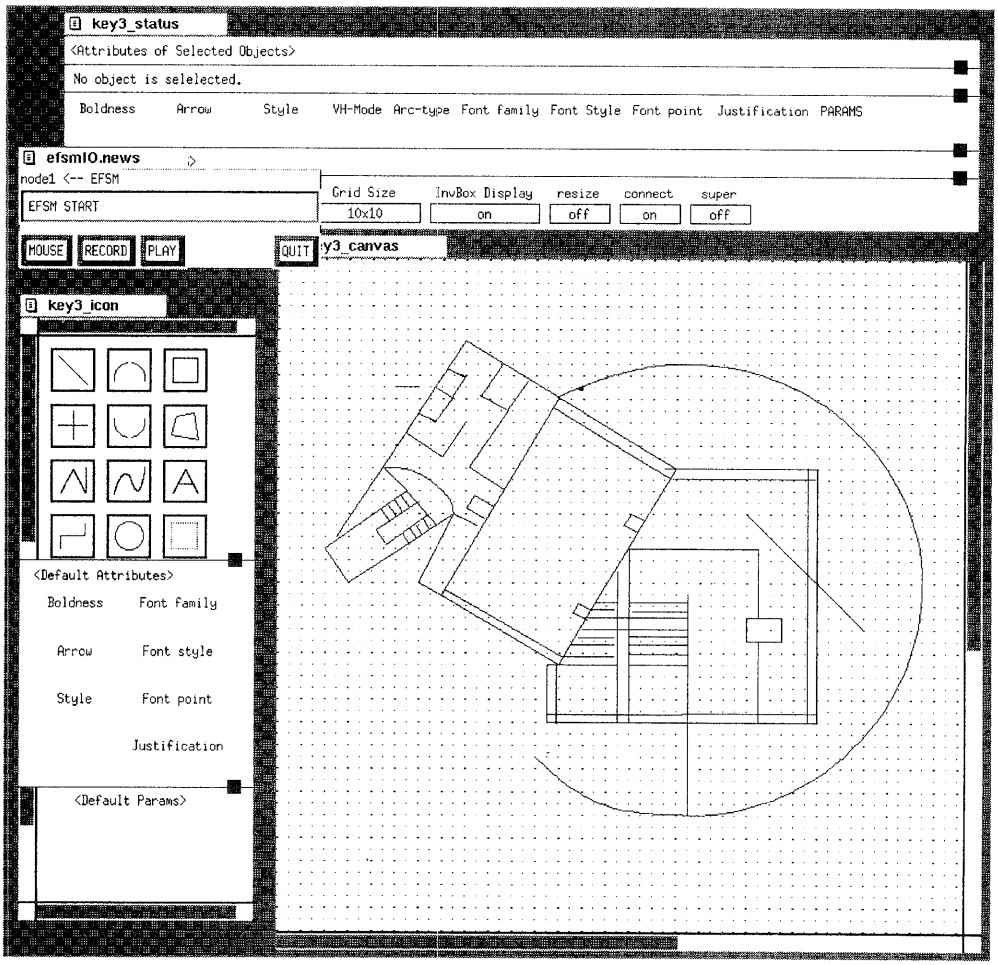


図8 システムの画面例
Fig. 8 Screen example of our system.

ように全体動作仕様の1つの遷移は複数個のメッセージの送受信と並行に複数ノードでのレジスタ更新を行うことにより実現されている．一般に我々のEFSMモデルでは本質的な並列動作を記述することはできないが，上述のようなレジスタの並行更新のような並列性は記述できる．なお，この遷移で送受信されたメッセージ数の合計は10個である．

この例では4つのノードで設計を行ったが，ノード数を例えば5つにしたり，多重化指定や配置指定を変えることによって異なる協調計算を容易に実現することができる．また簡単な医療診断システムを我々のシステムで作成した例もある[9]．

5. システムの評価

5.1 生成系の実行時間

我々の導出法では，メッセージ長よりもメッセージ数を少なくする方が全体の効率が良くなると考え，以下のように工夫している．ノード u からノード v にレジスタ r_n の値を送信する必要があるとき真となる命題変数 uv_r_n を導入し，これらの命題変数に関する制約条件を線形不等式の形で記述し，メッセージの総数を目的関数として，0-1整数線形計画問題を解くアルゴリズムを用いてメッセージ総数が小さくなるようにしている[7]．

一般に0-1整数線形計画問題はNP完全である[15]．そこで0-1整数線形計画法で最小解を求める部分の実行時間について計測を行った(SONY NEWS 5000 *)．**表2**における実行時間は，1つの遷移に対してその解を求めるのにかかった時間を表し，割合はその実行時間内に解が得られた遷移の割合を表している．

今回の実験では，99%以上の遷移に対して60秒以内

* CPU R4000，MIPS値100 MIPS

表2 導出時間
Table 2 Derivation time.

導出時間(秒以内)	1	2	5	10	30	60
割合(%)	70	87	95	97	98	99

10種類の配置指定を用いて30個の状態遷移を試行
条件：ノード数5，レジスタ数10

(ただし最長9分)に解が得られたが，場合によっては有効な時間内で解が得られない場合もありうる．有効な時間内で解が得られるかどうかは，作った制約式のうち，$m_1+m_2+\cdots+m_k \geq 1$ の形をした式の大きさ(kの値)や数からおおよそ判断できる．そこで生成系では制約式を作った段階で，0-1整数線形計画法を用いて最適解を求めると有効な時間内に解が得られないと判断した場合，ヒューリスティックな多項式時間アルゴリズムを用いる．もちろん，時間をかけても最小解を求めたい場合，そのように生成系を実行させることも可能である．

なお，2.2節の全体仕様動作と2.4節の配置指定を生成系に入力として与え，全ての遷移に対して0-1整数線形計画法を用いてメッセージを最小にする送受信方法を決定した場合，各ノードの動作仕様を生成し，C言語に変換するまで約0.10秒のCPU時間を要した(SONY NEWS 5000)．

5.2 実行系の実行時間

一般に全体動作仕様の1つの遷移は4章で述べたように複数個のメッセージ交換を行うことにより，実現されている．ここでは実行系において，ノード数10個で，1遷移の実現に最もメッセージ交換の回数が多くなる場合(各ノードが平均10個のメッセージ送信と10個のメッセージ受信を並行して行うような場合)を考え，その際のメッセージ交換に要する時間の計測を行った．このときのネットワーク環境はイーサネット上に30台程度の計算機が接続しているローカルなネットワークであり(以下の実験も同様)，送受信されるメッセージはサイズが10 byteのものを用いた．

その結果，全体動作仕様の1遷移の実現に要する平均時間はターンアラウンド時間で1.2秒であった(SONY NEWS 5000 3台とNEWS 3470* 7台を接続した場合を計測．以下の実験も同様．そのうち0.1秒はレジスタの更新に利用されている)．このことよりシステムを使用するグループの所属しているネットワークの応答時間が適当な速度であれば，十分実用に耐える速度で1つの遷移の処理が終了するといえる．

次に10個のノードで共同編集を行ったときのトークンが1周する平均速度を計測した．その結果はターンアラウンド時間で0.05秒である．また共同編集で交換されるデータ長は，図形の変更情報のみを送信しているので比較的短い(平均50 byte)．また図形の変更を各ノードに伝え，各ノードが図形変更要求をKey 3に送り，もとのノードに変更終了のメッセージを返送するのにターンアラウンド時間で0.13秒要する．これらの結果より，我々のシステムの共同編集機能が実用に耐える速度で動作するといえる．

5.3 グループウェアとしての機能に関する実現度

グループワークを支援する上で有用な機能は次のようなものが挙げられる[5]．同期型機能として，共用ウィンドウ機能・グループエディタ機能・会議機能，非同期型機能として，共同文書作成機能・共用データベース機能などである．

同期型機能は，我々のシステムの線描画共同編集や黒板システム，会話システムによって実現している．複数人に対し共同作業空間を与える共用ウィンドウ機能に関しては線描画共同編集の画面や黒板システムが対応する．複数人によるリアルタイム共同編集を行うグループエディタ機能に関して，我々のシステムでは線描画に限定されているが，その機能を実現している．会議機能に関しては簡単な場合では黒板システムや会話システムが利用可能である．

我々のシステムは全体仕様の記述の仕方でさまざまな形態のグループワーク支援システムとなる．さらにユーザがCで独自に関数を定義することが可能であるのでその柔軟性をより高めている．これらのことから共同文書作成機能を備えたシステムの構築も可能である．共用データベース機能に関しては，我々のモデルではグループワークの結果生じる文書情報などをレジスタに代入することにより共有することができる．また，レジスタの複数ノードへの分散配置が可能であり，これは簡単な分散データーベースの機能に相当する．よって我々のシステムはグループウェアとしての機能をある程度備えているといえる．

6. まとめ

本論文では，グループワークのための協調計算システムの全体動作仕様をEFSMモデルで記述し，その全体動作仕様から動作仕様群を自動生成する方法を提案した．また，これらを複数の計算機上で分散実行させるシステムの機能と評価等について述べた．その結果，全体動作仕様やリソースの配置等が頻繁に変更されるようなシステムに対しては，本方法のような自動生成法は有効である．

* CPU R3000，MIPS値17 MIPS

なお，本論文の手法は仕様の動的変更，すなわち，動作仕様群の実行中に全体動作仕様を変更することには直接対応していない．しかし，次のような方法で実質的に動的変更を行うことができる．まず，実行を中断したいと思うノードが実行を中断することを依頼するメッセージを全ノードに送る．これらのメッセージを受けたノードの中のいずれかのノードが責任ノードになった時点で引き続く動作を中断すれば，動作仕様群の全体の実行を中断できる．その後，変更後の全体動作仕様から動作仕様群を自動生成し，各ノードのレジスタ値を変更前のレジスタ値にセットし，それぞれ変更前の状態に対応する新しい状態から動作を再開すれば，変更後の全体動作仕様どおりに動作が進行する．本論文の導出法では，全体動作仕様の1つの遷移を対応する幾つかの遷移系列に置き換えることにより動作仕様群を生成しているので，このような手法が可能になる．

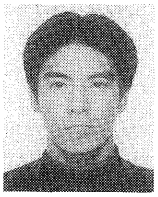
モデルの記述能力に関して，本論文のEFSMモデルで，例えば文献9)で述べたような簡単な医療診断システムのようなグループワークの仕様が記述できる．一般に医療診断は，複数の医者や検査技師による並行作業とみなせるが，その医療のターゲットとなる患者は1人であり，患者が並行に診断を受けることはなく，その診断プロセス（診療の過程）はEFSMモデルとして記述できる．このように，我々のEFSMモデルでは，全体の作業の流れに本質的な並行動作が含まれないシステムを対象としているが，それでもかなりのグループワークの全体仕様をこのモデルで記述できる．勿論，本質的な並行動作が含まれるグループワークも数多くあり，そのような並行動作を含むモデルに対して全体動作仕様から動作仕様群を自動生成できることがより望ましい．このため，我々は本論文のEFSMモデルを並行動作の記述できるレジスタ付のペトリネットモデルに拡張し，そのモデル上での動作仕様群の自動生成法を考案している[10),11)]．将来的には，このモデルへのシステムの拡張を検討している．

システムの拡張と実際的な応用例による本手法の有効性の検討などが今後の課題である．

参考文献

1) Ellis, C. A., Gribbs, S. J. and Rein, G. L.: Groupware—Some Issues and Experiences, *Comm. ACM*, Vol. 34, No. 1, pp. 38-58 (1991).
2) Greif, I. and Sarin, S.: Data Sharing in Group Work, *Proc. the First ACM Conf. on Computer-Supported Cooperative Work*, pp. 175-183 (1986).
3) Ahuja, S. R., Ensor, J. R. and Horn, D. N.: The Rapport Multimedia Conferencing System, *Proc. the ACM Conf. on Office Information Systems*, pp. 1-8 (1988).
4) Watabe, K., Sakata, S., Maeno, K., Fukuoka, H. and Maebara, K.: A Distributed Multi Party Desktop Conferencing System and Its Architecture, *Proc. 9th Int. IEEE Phoenix Conf. Computers and Communications*, pp. 386-393, (1990).
5) 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. 14-26 (1991).
6) Probert, R. and Saleh, K.: Synthesis of Communication Protocols: Survey and Assessment, *IEEE Trans. Comput.*, Vol. 40, No. 4, pp. 468-475 (1991).
7) 岡野浩三，今城広志，東野輝夫，谷口健一：拡張有限状態機械モデルを用いた分散システムの要求仕様から各ノードの動作仕様の自動導出，情報処理学会論文誌, Vol. 34, No. 6, pp. 1290-1301 (1993).
8) 今城広志，岡野浩三，東野輝夫，谷口健一：拡張有限状態機械を用いた協調作業向きの計算システム，情報処理学会研究会資料, 93-OS-60, 93-DPS-61, pp. 147-154 (1993).
9) 今城広志，尹 達雄，岡野浩三，東野輝夫，谷口健一：協調計算システムの動作仕様群の分散実行系，情報処理学研究会資料, 94-GW-6, pp. 19-24 (1994).
10) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K.: Synthesis of Protocol Specifications from Service Specifications of Distributed Systems in a Marked Graph Model, *IEICE Trans.*, Vol. E77-A, No. 10, pp. 1623-1633 (1994).
11) 山口弘純，岡野浩三，東野輝夫，谷口健一：ペトリネットモデルを用いたソフトウェアプロセスの記述とその分散実行制御，信学技報, SS94-38, pp. 25-32 (1994).
12) Milner, R.: *Communication and Concurrency*, Prentice-Hall (1989).
13) Langerak, R.: Decomposition of Functionality; a Correctness-Preserving LOTOS Transformation, *Proc. Tenth IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification*, North Holland, pp. 229-242 (1990).
14) 松浦敏雄，直田 創，中村 眞：図形の部品化および接続包含関係の保存機能を持つ作図ツールKey 3，信学論, Vol. J 73-D, No. 11, pp. 864-872 (1990).
15) Garey, M. R. and Johnson, D. S.: *Computers and Intractability*, Freeman (1979).

（平成6年9月2日受付）
（平成7年1月12日採録）



伊東　達雄

昭和45年生．平成5年大阪大学基礎工学部情報工学科中退．平成7年同大学院基礎工学科研究科博士前期課程修了．現在，日本電気(株)勤務．グループウェア，通信プロトコルの検証等の研究に興味を持つ．

今城　広志（正会員）

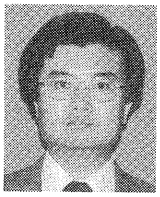
昭和44年生．平成4年大阪大学基礎工学部情報工学科を中退し，同大学院に進学．平成6年同大学院博士前期課程修了．在学中は，分散システム，グループウェア等の研究に従事．現在，日本電気(株)パーソナルコンピュータ事業部製品技術部所属．

岡野　浩三（正会員）

昭和42年生．平成2年大阪大学基礎工学部情報工学科卒業．平成5年同大学院博士後期課程中退．同年同大学基礎工学部情報工学科助手，現在に至る．博士(工学)．代数的手法によるソフトウェア設計開発法，分散システム等の研究に従事．電子情報通信学会会員．

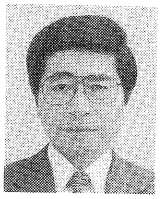
東野　輝夫（正会員）

昭和31年生．昭和54年大阪大学基礎工学部情報工学科卒業．昭和59年同大博士後期課程修了．工学博士．同年同大学助手．現在，基礎工学部情報工学科助教授．平成2，6年モントリオール大学客員研究員．分散システムや通信プロトコル等の研究に従事．ACM，IEEE-CS，電子情報通信学会各会員．



松浦　敏雄（正会員）

昭和27年生．昭和50年大阪大学基礎工学部情報工学科卒業．昭和54年同大大学院基礎工学研究科（情報工学専攻）博士後期課程退学後，大阪大学基礎工学部情報工学科助手．大阪大学情報処理教育センター助教授を経て，現在，大阪市立大学学術情報センター教授．工学博士．ユーザインタフェース，マルチメディア，情報教育等に興味をもつ．ACM，IEEE，電子情報通信学会等会員．



谷口　健一（正会員）

昭和17年生．昭和40年大阪大学工学部電子工学科卒業．昭和45年同大学院基礎工学研究科博士課程修了．工学博士．同年同大学基礎工学部助手．現在，同情報工学科教授．計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法および支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法などに関する研究に従事．