

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

今城 広志 岡野 浩三
東野 輝夫 谷口 健一

大阪大学 基礎工学部 情報工学科, 豊中市

あらまし グループウェアのような協調計算システムの設計においては、設計者はシステム全体としての作業順番やデータの依存関係(要求仕様)のみを記述し、その記述から、要求仕様通りに作業を進めるためにノード間でどのようなデータや同期用メッセージを交換する必要があるかを機械的に決定し、各ノードごとの動作仕様を自動生成できれば信頼性の高いシステムを効率よく開発出来る。我々は、有限個のレジスタを持つ拡張有限状態機械で記述された分散システム全体の要求仕様とゲートやリソース(レジスタ)の配置を与えられて、全体として要求仕様通りに動作するような各ノードの動作仕様を自動生成するためのアルゴリズムを提案している。一般に協調計算システムでは入力を行う可能性のあるノードが複数あったり、複数のノードに同時に同一内容のデータを出力したい場合がある。これらは同一ゲートの複数ノードへの多重割り当てに相当するが、従来のアルゴリズムではこれをサポートしていない。本研究では、ゲートの複数ノードへの多重割り当てに対応できるよう従来のアルゴリズムを拡張した。また、与えられた要求仕様から各動作仕様を自動生成する処理系や、各動作仕様をCプログラムに変換するコンパイラ、動作仕様の実行やレジスタの更新状況を図的に表示するツールを作成した。

Design of Distributed Computing Systems using an Extended Finite State Machine Model

Hiroshi IMAJO Kozo OKANO
Teruo HIGASHINO Kenichi TANIGUCHI

Dept. of Information and Computer Sciences, Faculty of Engineering Science,
Osaka University, Toyonaka, Osaka 560, Japan

Abstract In a distributed computing system, each protocol entity must exchange some data values and synchronization messages in order to ensure the temporal ordering of the actions described in a requirement specification for the distributed system. In this paper, we propose an algorithm which synthesizes automatically protocol entities' specifications from a requirement specification described as an extended FSM (EFSM) model with registers. Each resource is treated as a register. The designer gives an allocation of the registers and gates to the protocol entities. Each register and gate may be allocated to some protocol entities. We have developed a system deriving protocol entities' specifications from a requirement specification, a compiler transforming each protocol entity's specification into a C code and a graphic tool indicating the current transition and registers' values dynamically on the X Windows.

1. はじめに

近年、グループウェアのような協調計算システム(分散システム)の設計法や処理系、グループ間通信の方法などについて、多くの研究が行われている^{(1),(8),(9),(10)}。抽象レベルでは協調計算システム全体としての各作業の実行順序や作業内容の記述をそのシステムの要求仕様とみなすことができる。与えられた要求仕様に対して分散システム上の各ノード(計算機)はお互いに協調しながら全体として要求仕様の実行順序を満足するように動作する。このような協調計算システムを設計する一つの方法として、設計者はシステム全体としての要求仕様のみを記述し、その作業手順や作業の依存関係から要求仕様通りに作業を進めるために必要な同期用メッセージやデータの送受信のタイミングを機械的に決定し、各ノードの詳細な作業手順(動作仕様)を自動生成するという方法が考えられる。従来、要求仕様から動作仕様の自動生成の方法について幾つかの研究が行われている^{(2),(3),(4),(5),(6)}。一般に要求仕様を記述するレベルでは、入出力を行うゲートやリソースをどのノードに割当ててかを決定しない場合が多い。また動作仕様作成後もゲートやリソースの割当てが変更される場合が多い。割当てが変更されればデータや同期用メッセージを伝える相手も変更されるため各ノードの動作仕様も大きく変更される。また設計者が自身が同期用メッセージ等の交換相手を決めるのは複雑であり間違いも生じやすい。このような理由から、要求仕様から動作仕様を自動生成することは信頼性の高い協調計算システムを設計開発するための有効な手段の一つである。

我々は文献(5),(6)で、有限個のレジスタを持つ拡張有限状態機械(EFSM)として記述された協調計算システム全体の要求仕様とゲートやリソースの割当ての指定から、全体として要求仕様通りに動作する各ノードの動作仕様を自動生成するためのアルゴリズムを提案している。しかしこの方法では、同一ゲートの複数ノードへの多重割当てを許していない。しかし、グループ作業などではゲートを単一のノードに固定して扱いたくない場合がある。例えば、あるデータを複数のノードの出力ゲートに同時に出力したい場合や、入力を行う可能性のあるゲートが複数ノードにあり、そのうちの何れかのノードからの入力をシステムへの入力としたい場合がある。そこで要求仕様のレベルでは単に論理的な一つのゲートの入出力と記述しておき、各動作仕様を導出する際に、出力に関してはその論理ゲートが割当てられた複数ノードのゲートへ並行して出力を行い、入力に関してはいずれか1つのノードのゲートから入力を行うようにする。

本稿ではゲートの多重割当てを許すようなモデルを提案し、そのモデル上で協調計算システム全体の要求仕様から各ノードの動作仕様を自動生成出来るよう文献(5),(6)のアルゴリズムを拡張した。アルゴリズムの拡張に際しては、まず論理的なゲートに基づいて書かれたシステム全体の要求仕様をゲートの多重割当ての仕方に従って物理的なゲートに基づく要求仕様に変形し、変形後の要求仕様に対して従来のアルゴリズムを適用する。また実際に与えられた要求仕様から各動作仕様を機械的に生成するシステ

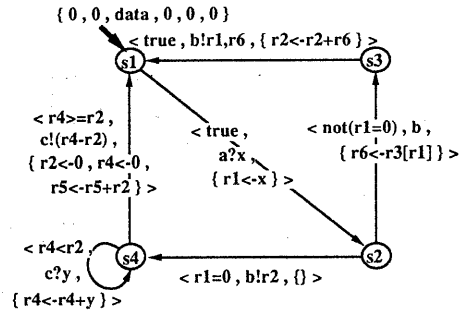


図 1: 要求仕様 (EFSM) の例

ムを UNIX 上で作成した。さらに拡張有限状態機械型の動作仕様を C プログラムに変換するコンパイラや動作仕様の実行やリソース(レジスタ)の更新状況を図的に表示するツールを作成した。本稿ではこれらの処理系の内容についても述べる。

2. 協調計算システムの要求仕様

本稿では協調計算システム全体の要求仕様を EFSM で記述する。また要求仕様のレベルでは各ノード間でのデータやメッセージのやりとりについては一切記述しない。

EFSM は FSM に有限個のレジスタを加えたものである。EFSM の状態遷移図は図 1 のように表す。この図で節点は状態を表し、枝は状態遷移を表す。状態遷移では入力、出力のいずれかを行う。初期状態では各レジスタの初期値が指定されている。状態遷移を表す枝には遷移条件、入出力動作、レジスタ更新式の 3 字組がラベルとして付けられている。例えば $a?x$ はゲート a からのデータ x の入力を表し、 $a!E(\dots)$ で式 $E(\dots)$ の値をゲート a に出力することを表す。入出力を何も行わないときは空値の出力動作を行うと考え、便宜上 a と表す。ある状態において、その状態から出る各状態遷移の実行可能性を判定する述語が遷移条件である。遷移条件の評価には現状態のレジスタ値と入力変数値が用いられる。これが真となる状態遷移から非決定的に 1 つの遷移が選択され、入出力動作が実行される。次にレジスタ値が更新され、次状態に遷移する。レジスタ値の更新を行う代入文の組がレジスタ更新式である。レジスタ値の更新は、各代入文の右辺(引数はレジスタと入力変数)の値を同時に評価した後、その評価値で行う。

2.1 要求仕様の例

図 1 は要求仕様の例である。この例は店のレジの業務をモデル化したものである。レジスタは $r_1 \dots r_6$ の 6 つ、ゲートは a, b, c の 3 つが用いられている。状態 s_1 が初期状態で、各レジスタは $\{0, 0, data, 0, 0, 0\}$ に初期設定されている($data$ は、商品データベースの内容を表す)。商品番号をゲート a から入力し、レジスタ r_1 に保存する($s_1 \rightarrow s_2$)。状態 s_2 で商品番号(r_1 の値)が 0 でなければ、商品データ

ベース r_3 から該当商品の価格 $r_3[r_1]$ を調べ、レジスタ r_6 に代入する ($s_2 \rightarrow s_3$)。ただし、この状態遷移では入出力動作はないので動作は便宜上 b としている。次に表示器 (ゲート b) に商品番号、価格 (r_1, r_6 の値) を表示し、総合計 r_2 を更新し、初期状態に戻る ($s_3 \rightarrow s_1$)。一方、状態 s_2 で商品番号 (r_1 の値) が 0 の場合、商品入力終了と見なし、総合計を表示器に表示し ($s_3 \rightarrow s_4$)、客から受けとった現金を順次レジ (ゲート c) に入れる ($s_4 \rightarrow s_4$)。受けとった金額の合計はレジスタ r_4 に入る。受けとった金額の合計 (r_4 の値) が商品価格の総合計 (r_2 の値) 以上になれば、釣り銭をレジから取り出して客に渡し、店の売上総合計 (r_5 の値) を更新し、各レジスタをリセットして初期状態に戻る ($s_4 \rightarrow s_1$)。

3. 動作仕様の導出問題

3.1 ゲートの複数分散配置

本稿では要求仕様 (以下、単に EFSM) を複数個のノードからなる協調計算システムとして実現する際、各レジスタや入出力ゲートがそれぞれどのノードに属しているかを設計者自身で与えるものとする。同一レジスタや同一ゲートが複数のノードに属してもよい。1つの入力ゲートが複数のノードに割当てられた場合、いずれか1つのノードから入力されたデータをシステム全体の入力と考える。また、1つの出力ゲートが複数のノードに割当てられた場合、すべてのノードに出力を行うものとする。ただし出力の順序は問わない。上のような割当てを Alloc(EFSM) と表す。例えば、図1の EFSM に対して、ノードを3つとし、Alloc(EFSM) を以下のように決める。

	ノード 1	ノード 2	ノード 3
ゲート	a, b	a, b, c	
レジスタ	r_1, r_2	r_2, r_4	r_3, r_5, r_6

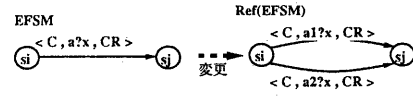
この例ではノード 1, 2, 3 はそれぞれ商品番号自動読みとり装置、キャッシャー、商品データベース管理マシンを想定している。商品番号入力のためのゲート a は商品番号自動読みとり器 (ノード 1)、または数値入力キー (ノード 2) と考える。また表示器 (ゲート b) は客用 (ノード 1) と店員用 (ノード 2) の2つ、ゲート c はレジである。

ゲートに関して、ノード 1 とノード 2 のゲート a はどちらも商品番号を入力するゲートという意味で論理的には同じものであるが、物理的には別のものである。そのため、ノード 1, 2 のゲート a をそれぞれ a_1, a_2 と表す。同様に、全く同じ働きをする客用と店員用の表示器 (ゲート b) についても b_1, b_2 とし、ゲート c を c_2 と表す。

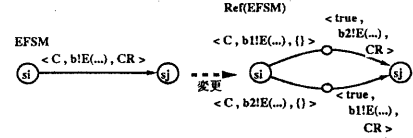
図1の EFSM のように、2章で述べた要求仕様は論理的なゲートを基にその動作が定義されている。しかしゲートの多重割当てを考慮に入れた場合、物理的なゲートに基づいて動作定義をした方がより自然である。以下ではそのための方法について述べる。

3.2 物理的なゲートに基づく EFSM

以下、与えられた要求仕様 EFSM とゲートやレジスタの割当て Alloc(EFSM) に対して、物理的なゲートに基づく要求仕様 Ref(EFSM) を次のように構成する。



(1) 入力動作のある遷移



(2) 出力動作のある遷移

図 2: EFSM から Ref(EFSM) への変形

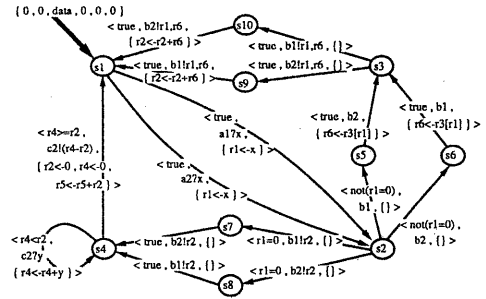


図 3: Ref(EFSM) の例

図2に Ref(EFSM) の構成法を示す。今、上述の例のようにゲート a は a_1, a_2 に、ゲート b は b_1, b_2 に分けられるとする。EFSM の入力に相当する遷移については、この入力ゲートを持つノードのうちいずれか1つのノードで入力が行われればよいことから、図2(1)のように元の遷移の入力ゲート名のみを変えた並列の遷移に置き換える。また出力に相当する遷移については、この出力ゲートを持つすべてのノードで出力動作が行われること、各出力動作の順番は問わないことから、図2(2)のような置き換えを行う。

出力の遷移について、この図の例ではゲート b を b_1 と b_2 の2つに対応づけているので2本の遷移系列に置き換わっているだけであるが、一般に b を $b_1, \dots, b_n$ に置き換えると、これらのあらゆる順序に対応するため $n!$ 本の遷移系列に置き換えられることになる。ただし設計者側が、出力の順序がただ1つに決まっていなくてもかまわないという立場をとるのなら、 $n!$ 本のうちの1本のみに置き換えてもよい (このようにしても以下の導出法に影響はない)。

なお、EFSM から Ref(EFSM) へは自動的に変形できる。図3に、図1の EFSM を変形した Ref(EFSM) を示す。

3.3 導出問題

要求仕様と同様に各ノードの動作仕様も EFSM で記述する。ノード k の動作仕様を EFSM_k で表す。 p 個のノードの動作仕様の組を EFSM^{1-p} と表し (p ノードの) 協調計算

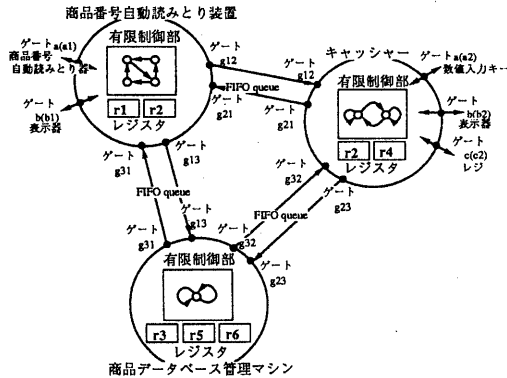


図 4: 協調計算モデル

システムの動作仕様と呼ぶ。EFSM_iからEFSM_jへの通信路は無限の容量を持つFIFOキューで結ばれているとし、両端のゲート名を g_{ij} とする(図4)。EFSM_iからEFSM_jへのメッセージの送受信はゲート g_{ij} を用いたゲート入出力と考える。

[動作仕様の導出問題]

EFSMと協調計算システムのノードの集合{ノード1, ..., ノード p }及びゲートやレジスタの割当Alloc(EFSM)が与えられたとき、Ref(EFSM)と等価な p ノードの協調計算システムの動作仕様EFSM^{1-p}を導出する問題(等価性の定義については後述)。

ただし要求仕様として与えられるEFSM及びAlloc(EFSM)は次のような制約条件を満足するものとする。

初期状態を s_I とする。 $s_I \rightarrow \langle C(\dots), a \dots, CR \rangle \rightarrow s'$ なるすべての状態遷移に対し、論理的なゲート a に対する物理的なゲートを $a_1, \dots, a_n$ とする。任意の $i(1 \leq i \leq n)$ に対して、遷移条件 $C(\dots)$ は、ゲート a_i が属するノードに割当てられたレジスタのみを用いた述語で記述されていなければならない。また、 $a \dots$ が出力動作であれば、この出力動作に必要なレジスタもすべてゲート a_i が属するノードに割当てられていなければならない。

上の制約条件は初期状態から出る遷移の実行可能性判定及びゲートへの出力をそのゲートを持つノードで行えることを表しており、本質的な制約ではない。この制約条件を満足しない場合は特別な状態を導入し、その状態を仮の初期状態と見なし、ここから元の初期状態へのダミーの遷移を行えばよい。

変形後の要求仕様Ref(EFSM)と動作仕様EFSM^{1-p}が等価であることを以下の様に定義する。

[等価性]

動作仕様EFSM^{1-p}において各EFSM_i, EFSM_j間の通信に用いられる送受信動作 $g_{ij}!x, g_{ij}?E(\dots)$

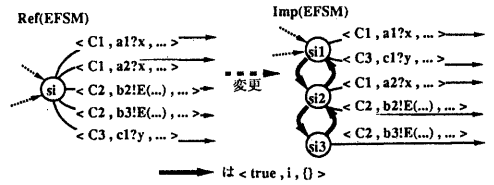


図 5: Ref(EFSM) から Imp(EFSM) への変形

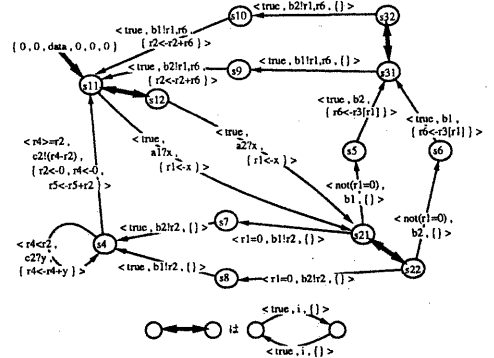


図 6: Imp(EFSM) の例

を観測不可能な動作とし、その他の動作を観測可能な動作とする。このとき、Ref(EFSM)とEFSM^{1-p}が弱双模倣性等価⁽⁷⁾であれば、両者は等価であるという。2つのEFSMが弱双模倣性等価であるというのは、一方のEFSMで実行できる任意の観測可能な動作の系列が他方のEFSMでも実行でき、それがEFSMを入れ替えても成り立ち、かつ任意の実行可能な観測可能動作系列を行なった時点での実行可能かつ観測可能な動作が互いに等しいことである。

4. 各ノードの動作仕様

4.1 動作仕様導出の準備

要求仕様EFSMから各ノードの動作仕様を導出することを考える。文献(5), (6)では要求仕様から各ノードの動作仕様を自動導出するアルゴリズムを提案している。ここではこのアルゴリズムを適用したい。しかし要求仕様に対して同一ゲートを複数のノードに多重割り当てすることを本稿では許しているが、文献(5), (6)ではこれを制約している。そこで同一ゲートの多重割り当てを実現しながら表記上はこの制約を満たすよう、要求仕様EFSMをRef(EFSM)へ変形した。さらに文献(5), (6)では、ある1つの状態から出る遷移の入出力ゲートはただ1つのノードに属さなければならないと仮定しているが、本稿のモデルはこの仮定を満足しない。そこでこの仮定を満足させるため、Ref(EFSM)

を各状態での入出力ゲートがただ1つのノードに属するような要求仕様 $\text{Imp}(\text{EFSM})$ へ変形する。

図5は $\text{Ref}(\text{EFSM})$ の変形の様子である。ゲート a_1, c_1 はノード1, ゲート a_2, b_2 はノード2, ゲート b_3 はノード3にそれぞれ属するとする。このとき $\text{Ref}(\text{EFSM})$ の状態 s_i は3つの状態 s_{i1}, s_{i2}, s_{i3} に分けられ, s_{i1} からはノード1に属するゲートでの入出力動作を持つ遷移のみを出すようにする(同様に s_{i2} はノード2, s_{i3} はノード3)。また s_{i1}, s_{i2} 間, s_{i2}, s_{i3} 間は図5の太線で示したようなダミー遷移で双方向に結ぶ。ここで動作 i は入出力を何も行わない観測不可能な動作とする。

この図でシステムが状態 s_{i1} に遷移したとする。このとき, $a_1?x$ または $c_1?y$ の実行が可能となる。もしシステムが $a_1?x$ または $c_1?y$ を実行せずに太線のダミー遷移を実行すれば, 状態 s_{i2} に遷移し $a_2?x$ または $b_2!E(\dots)$ の実行が可能となる。同様にダミー遷移を行い状態 s_{i3} に遷移すれば $b_3!E(\dots)$ が実行可能となる。入出力動作を行うかダミー動作を行うかは非決定的に選ばれるので, $\text{Imp}(\text{EFSM})$ のように変形しても, もとの $\text{Ref}(\text{EFSM})$ の5つの入出力動作の実行可能性は変化しない。また, このように変形を行うことにより各状態から出る遷移の入出力ゲートはただ1つのノードに属することになる。

以上の変形もまた自動的に行うことが可能である。図6に, 図3の $\text{Ref}(\text{EFSM})$ を変形した $\text{Imp}(\text{EFSM})$ を示す。

4.2 表記法

$\text{Ref}(\text{EFSM})$ における状態 s について, s から出る各遷移の遷移条件に用いられているレジスタ及び出力動作の実行の際に参照されるレジスタのすべてを要素とする集合を $\text{Bset}(s)$ とする。

$\text{Imp}(\text{EFSM})$ における状態 s について, s から出るどの遷移の入出力動作が行なわれるゲートもただ1つのノードに属する。このノードを状態 s の責任ノードと呼び, $\text{Snode}(s)$ と表す。

$\text{Ref}(\text{EFSM})$ から $\text{Imp}(\text{EFSM})$ への変形の際, $\text{Ref}(\text{EFSM})$ における状態 s が状態 $s_1, \dots, s_n$ に分割されたとする。このとき, $\text{Bset}(s)$ のすべての要素(レジスタ)及び $\text{Alloc}(\text{EFSM})$ によって $\text{Snode}(s_k)$ (ただし, $1 \leq k \leq n$) に割当てられたレジスタのすべてを要素とする集合を $\text{Cset}(s_k)$ とする。

以上の定義は, 文献(5), (6)の動作仕様自動導出アルゴリズムを適用する際に必要となる。図6の $\text{Imp}(\text{EFSM})$ について, 状態 $s_{11}, s_{21}, s_{31}, s_6, s_8, s_{10}$ の Snode はノード1, 他の状態の Snode はノード2である。また, 状態 s_{11}, s_{21}, s_6, s_8 の Cset は $\{r_1, r_2\}$, 状態 s_{12}, s_4, s_5, s_7 の Cset は $\{r_2, r_4\}$, 状態 s_{22} の Cset は $\{r_1, r_2, r_4\}$, 状態 s_{31}, s_9 の Cset は $\{r_1, r_2, r_6\}$, 状態 s_{32}, s_{10} の Cset は $\{r_1, r_2, r_4, r_6\}$ である。

4.3 各ノードの動作の概略

導出される各ノードの動作の概略について図7の例を用いて説明する。図7(1)は, ノードへのレジスタやゲートの割当てと, $\text{Imp}(\text{EFSM})$ のある一つの遷移を示したものである。ノード1はレジスタ r_1 とゲート a_1 , ノード2はレ

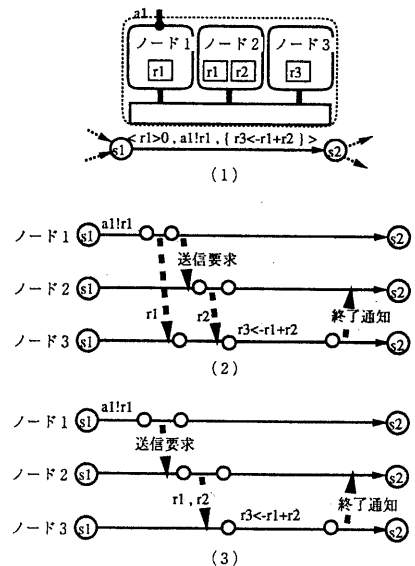


図7: 各ノードの動作例

ジスタ r_1, r_2 , ノード3はレジスタ r_3 をそれぞれ持っている。また $\text{Snode}(s_1)$ はノード1, $\text{Snode}(s_2)$ はノード2であるとする。このような状況で図7(1)の遷移の実現の1つとして, 図7(2)のタイミングチャートで表すような各ノードの動作系列がある。以下これを説明する。なお破線はメッセージの送受信を表す。

まず $\text{Snode}(s_1)$ であるノード1が状態 s_1 から出る遷移の実行可能性の判定をし, いずれかの遷移の実行を選択する。ここでは, 遷移条件 $r_1 > 0$ が成り立ち, ノード1がこの遷移を非決定的に選択したとする。このときノード1は出力動作 $a_1!r_1$ を実行し, その後, 各ノードでレジスタの更新を行う。レジスタの更新の方法は色々考えられるが, 更新すべきレジスタを持つ各ノードが更新に必要なレジスタ値を他ノードから受けとり, 自ノードで更新式を計算したのち更新を行うとする。このときノード3はレジスタ r_3 の更新のために, ノード3自身は持っていないレジスタ r_1, r_2 の値を必要とする。そこで責任ノードであるノード1が自らノード3へレジスタ r_1 の値を送るとともに, レジスタ r_2 を持っているノード2に対して, ノード3へレジスタ r_2 の値を送るように要求する(メッセージを送信する)。レジスタ r_2 の値の送信要求を受けたノード2は, ノード3へレジスタ r_2 の値を送る。ノード3はレジスタ r_1, r_2 の値を受けとった後, レジスタ r_3 を更新する。レジスタの更新を終えたノード3は遷移先の状態 (s_2) の責任ノード(ノード2)へレジスタ更新の終了を知らせるメッセージを送る。これを受けたノード2は, 今行われた遷移が各ノードで終了したことを知り, 次の遷移の実行を行っていく。

4.4 導出アルゴリズムの概略

動作仕様の導出の主な部分は、レジスタ更新の際にやりとりされるメッセージについて、どのノードからどのノードへどのような内容のメッセージが送受信されるのかを決定することである。

一般に、ある1つのレジスタ更新の際のメッセージの送受信の方法は一通りであるとは限らない。例えば図7(1)の例では、ノード3へレジスタ r_1, r_2 の値を送る方法は図7(2)の方法以外に図7(3)のような方法もある。図7(3)では、まず $Snode(s_1)$ であるノード1が実行可能性の判定ののち、図7(1)の遷移を選択したとし、出力動作 alr_1 を行う。ここでノード2はレジスタ r_1, r_2 の両方を持っているので、ノード3が必要としているこれらのレジスタの値をノード2が共に送るように、ノード1からノード2へ送信要求をする。これを受けたノード2がノード3へレジスタ r_1, r_2 の値を送り(ただし、これは1つのメッセージとして送られる)、ノード3がこれを受け、レジスタ r_3 を更新し、次の責任ノード(ノード2)へ終了通知のメッセージを送る。ノード2はこれを受け、この遷移は終る。

図7(2)の方法ではノード間でやりとりされるメッセージの総数は4つであるが、図7(3)の方法では3つですむ。導出される動作仕様としては、やりとりされるメッセージの総数をできるだけ少なくしたい。

いま、責任ノードからのレジスタ値の送信や送信要求のためのメッセージをAメッセージ、Aメッセージを受けたノードが他のノードへレジスタ値を送信するためのメッセージをBメッセージ、レジスタ値を更新したノードが次の責任ノードへ送る終了通知メッセージをCメッセージとそれぞれ呼ぶことにする。またノード i からノード j へA, B, Cメッセージを送る必要があるときに1となる変数をそれぞれ A_{ij}, B_{ij}, C_{ij} とする。また、各 A_{ij}, B_{ij}, C_{ij} メッセージでレジスタ r_k の値を送る必要があるときに1となる変数をそれぞれ $A_{ij-r_k}, B_{ij-r_k}, C_{ij-r_k}$ とする。このとき各変数間の関係を線形不等式で記述できる。

例えば、図7(1)の例に対しては、ノード3へレジスタ r_1 の値をノード1, 2のどちらから送ってもよいことから、 $A_{13-r_1} + B_{23-r_1} \geq 1$ でなければならない。他にも、レジスタ r_2 の値はノード2からしか送れないので、

$$B_{23-r_2} \geq 1$$

である必要がある。また、ノード1からノード3へ、ノード2からノード3へのレジスタ値の送信は1つのメッセージとしてまとめて送ることが可能であるので、

$$B_{23} \geq B_{23-r_1}, B_{23} \geq B_{23-r_2}, A_{13} \geq A_{13-r_1}$$

である。さらに、ノード2がノード3へメッセージを送るためにはノード1からの送信依頼が必要なので、

$$A_{12} \geq B_{23}$$

でなければならない。また、レジスタを更新したノード(ノード3)は次の責任ノード(ノード2)へ終了通知を送る必要がある、

$$C_{32} \geq 1$$

でなければならない。

上のような不等式と、1つの遷移でやりとりされるメッ

セージの総数を表した目的関数

$$\sum(A_{ij} + B_{ij} + C_{ij})$$

とを0-1線形計画問題として解き、メッセージの総数が最小となるようなメッセージの送受信方法を決定する。この例では $A_{12}, B_{23}, B_{23-r_1}, B_{23-r_2}, C_{32}$ が1, 残りが0となり、図7(1)に対しては図7(3)がメッセージ総数最小の動作仕様として得られる。動作仕様の導出に関する詳細は文献(5), (6)を参照のこと。

4.5 動作仕様の例

前述アルゴリズムによってImp(EFSM)から導出された動作仕様のうちのEFSM₁(ノード1の動作仕様)を図8に示す。ラベル付けされた状態がImp(EFSM)に対応する状態である。このうちノード1が責任ノードと成るべき状態は網がけで表している。各状態遷移のラベルとして入出力動作とレジスタ更新式が付けられている。遷移条件は基本的に省略している。遷移条件の与え方は後述する。

各ノード間で交換されるメッセージは $M_i (1 \leq i \leq 21)$ で表されている。各メッセージには(対応する)状態遷移名、メッセージのタイプがそれぞれ与えられているが、幾つかのメッセージにはレジスタ値(変数値)の情報も含まれる(メッセージ M_3 に変数 x 、メッセージ M_4, M_{11}, M_{14} にレジスタ r_1 、メッセージ $M_{15}, M_{16}, M_{17}, M_{19}, M_{20}, M_{21}$ にレジスタ r_6 の情報が含まれる)。

図8で省略されている遷移条件は以下のように与える。責任ノードとなるべき状態から出る遷移についてはImp(EFSM)に対応する遷移条件が与えられる。一方ラベル付けされた状態でかつ責任ノードとなるべき状態でない状態(以降、非責任状態と呼ぶ)から出る遷移は常に受信動作となっており、受け取るべきメッセージかどうかを確認する遷移条件が与えられる。例えば、状態 $s_{22} \rightarrow s_8$ の遷移の式は正式には $\langle m = M7, g21?m, \{\} \rangle$ となる。

状態 s_{12} から s_{21} への状態遷移系列は以下のような動作を(他のEFSM_kと協調して)行う。まずEFSM₂から(EFSM₂が入力動作を行った)変数 x の値をメッセージ M_3 として受取り、 x の値を作業用レジスタ r_0 に代入している($put(r_0, m)$ はメッセージ m 中の変数やレジスタの値を r_0 に保管することを表す)。このような作業用レジスタは各EFSM_kに用意されている。次に r_0 に保存された x の値をレジスタ r_1 に代入している($get(r_0, x)$ は r_0 に保存されたレジスタ(変数) x の最新の値を取り出す関数)。なおEFSM₁にまったく関係のない状態遷移は ϵ 遷移に置き換えている($s_7 \rightarrow s_4, s_4 \rightarrow s_4$ など)。

非責任状態は(そこから出る状態遷移は遷移条件から区別できることなどから)一つの状態にまとめることが出来る。この様に状態簡約を行い ϵ 遷移を取り除いて得られるEFSMが最終的なEFSM₁となる(図は省略)。

5. 協調計算モデルの実現

前章までに議論した協調計算モデルを実際のシステム上で実現する計算システムを作成した。ただしゲートの多重割当ての部分に関しては現在作成中である。

計算システムは、要求仕様及びリソースの配置の指定か

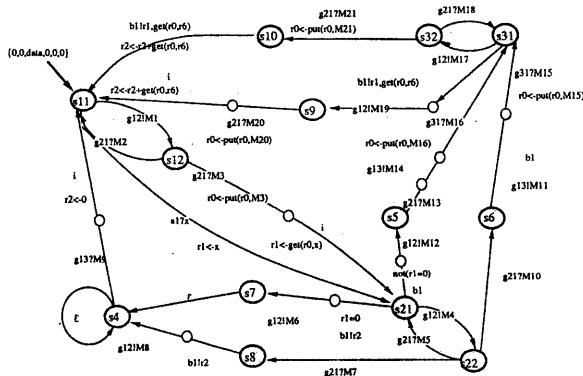


図 8: 動作仕様の例

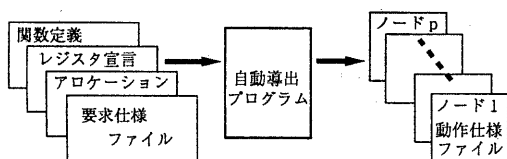


図 9: 生成系

ら各ノードの動作仕様を自動導出する生成系と、自動導出された動作仕様を実行するための実行系及び実行の様子を視覚的に確認するための動作表示プログラムから成る。生成系では、要求仕様 EFSM を定められたフォーマットに従って記述したテキスト (EFSM 型プログラムと呼ぶ) からノード間でのメッセージのやりとりをも含めた各ノードの動作仕様 (EFSM 型プログラム) を自動導出する。実行系では、各ノードの動作仕様 (EFSM 型プログラム) を C プログラムに変換し、これをコンパイルして実行可能プログラムにする。これとは別に実行可能プログラム同士の通信を実現するための通信プログラムを用意している。

5.1 生成系

生成系は要求仕様 EFSM 型プログラムから各ノードの動作仕様 EFSM 型プログラムを自動導出するシステム (自動導出プログラム) である (図 9 参照)。

自動導出プログラムに対する入力となるファイルは、図 10 のような形式の要求仕様 EFSM 型プログラムを書いたファイル (要求仕様ファイル) の他、ゲートやレジスタがどのノードに属しているかの情報を書いたファイル (アロケーションファイル)、レジスタや入力用変数の型や構造を書いたファイル (レジスタ宣言ファイル) 及び設計者が要求仕様内で用いた関数を C 言語で書いたファイル (関数定義ファイル) の 4 つである。自動導出プログラムは上のような入力に対して、要求仕様 EFSM プログラムの構文・意味チェックを行なった後、4 章や文献 (5), (6) で述べられて

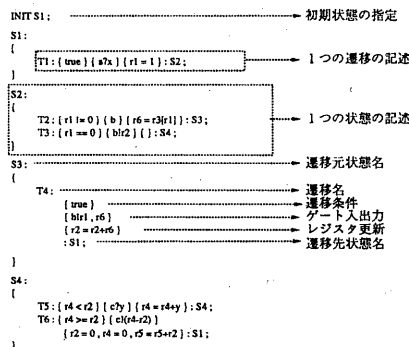


図 10: 要求仕様ファイル

いる動作仕様の自動導出法を用いて、各ノードの動作仕様 EFSM 型プログラムを書いたファイル (動作仕様ファイル) を出力する。動作仕様ファイルでは、図 10 の要求仕様ファイルの 1 つの遷移の記述にメッセージの送受信に関する記述が挿入されている。

5.2 実行系

実行系は動作仕様 EFSM 型プログラムから C プログラムへ変換する変換プログラムとノード間の通信を実現するための通信プログラムから成る。

変換プログラムへの入力は生成系によって得られた各ノードの動作仕様 EFSM 型プログラムで、出力は各ノードの C プログラムである。これらの C プログラムをコンパイルし、通信プログラムを利用するためのライブラリとリンクさせ、各ノードの実行プログラムを得る。

1 つの実行プログラムは 1 つのノードでの実際の動作 (状態遷移) を実現したものである。各ノードでそれぞれ実行プログラムを実行して協調計算を実現するが、このとき別に通信プログラムを走らせておきノード間の通信は通信プログラムを介して行う。

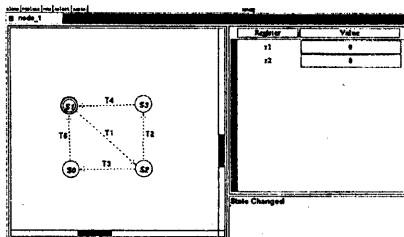
5.3 動作表示プログラム

実行プログラム内部で行われている状態遷移やレジスタの値などを可視化するために動作表示プログラムを用意した。実行プログラムの実行の際、動作表示プログラムを利用することもできる。

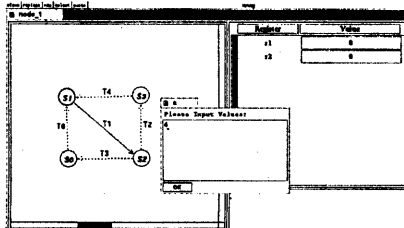
動作表示プログラムは状態遷移図、レジスタ表を常時表示している (図 11)。状態遷移図の状態は丸で、遷移は破線で描かれていて、現在の状態は二重丸、現在実行中の遷移は実線で表示される。レジスタ表はレジスタ名とその値とが表示されている。ただし状態遷移図に表示されている一本の遷移は実際には、使用者とのインタラクションを伴う入出力動作とそれに引き続くメッセージの送受信やレジスタ変更などのシステム内で自動的に行われる細かい遷移の系列を表している。

動作表示プログラムは実行プログラムからの要求を待っており、現在の状態や遷移、更新されたレジスタ値などの

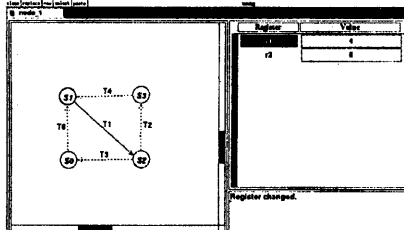
表示の要求があれば、状態遷移図やレジスタ表を更新し、実行プログラムへ図表の更新の終了通知を行うというように設計した。このため、変換プログラムが出力するCプログラムは動作表示プログラムを利用するためのコードを挿入できるようにしている。このため動作表示プログラム側が状態遷移のスピードをコントロールすることが可能である。また動作表示プログラムが必要とする状態遷移図の情報は自動導出プログラムの出力から得られる。



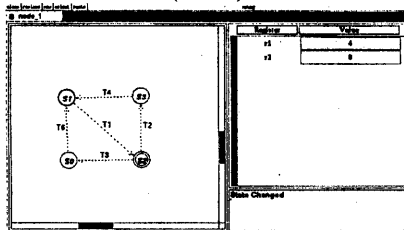
(実行 1)



(実行 2)



(実行 3)



(実行 4)

図 11: 実行系の動作

5.4 動作例

図 10の要求仕様を 3 ノードの協調計算システムとして実現したときの 1 つのノードの実行系の動作の様子 (動作

表示プログラムの表示) を紹介する (図 11)。

(実行 1) 現在のこのノードは状態 s_1 にあり、 s_1 が二重丸になっている。このノードが持っているレジスタは r_1, r_2 の二つであり、それぞれ値は 0 になっている。

(実行 2) 現在の責任ノードであるこのノードが、遷移 $T1((true, a?x, \{r_1 \leftarrow x\}))$ の実行を選び ($T1$ の矢印が実線になっている)、ここでのゲート入力動作 $a?x$ が実行される。ゲート入力用のウィンドウにはゲート名 a が示されており、使用者が値 (4) を入れている。

(実行 3) 入力動作が終わった後、各ノードでレジスタの変更が行われる。レジスタ名 (r_1) が反転表示されており、値が更新されたことを示す。

(実行 4) 遷移 $T1$ の実行が終り、このノードは状態 s_2 に移る (s_2 が二重丸になる)。

6. おわりに

本稿では、EFSM モデルで記述された協調計算システムの要求仕様から各ノードの動作仕様を導出する方法を与え、これらを実際のシステム上で実現した計算システムを紹介した。計算システムに関しては、ゲートの多重割り当てへの対応を完了させること、データとして画像や音声なども扱えるようにすることが今後の課題である。

文献

- (1). C. A. Ellis, S. J. Gibbs and G. L. Rein : "Groupware - Some Issues and Experiences", Communication of the ACM, Vol. 34, No. 1, pp.38-58, 1991.
- (2). R. Probert and K. Saleh : "Synthesis of Communication Protocols: Survey and Assessment", IEEE Trans. Comput., Vol. 40, No. 4, pp. 468-475, 1991.
- (3). G. v. Bochmann and R. Gotzhein : "Deriving protocol specifications from service specifications", Proc. of ACM SIGCOMM '86, pp. 148-156, 1986.
- (4). C. Kant, T. Higashino and G. v. Bochmann : "Deriving Protocol Specifications from Service Specifications Written in LOTOS", Proc. 12th Int. IEEE Phoenix Conf. on Computers and Communications, pp.310-318, 1993.
- (5). 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, 第 36 巻, 第 6 号, 1993.
- (6). T. Higashino, K. Okano, H. Imajo and K. Taniguchi : "Deriving Protocol Specifications from Service Specifications in Extended FSM Models", the 13th IEEE Int. Conf. on Distributed Computing Systems (ICDCS-13), pp.141-148, 1993.
- (7). R. Milner : "Communication and Concurrency", Prentice-Hall, 1989.
- (8). I. Greif and S. Sarin : "Data Sharing in group work", Proc. of the first ACM Conf. on Computer-Supported Cooperative work, pp. 175-183, 1986.
- (9). S. R. Ahuja, J. R. Ensor and D. N. Horn : The Rapport Multimedia Conferencing System", Proc. the ACM Conf. on Office Information Systems, pp. 1-8, 1988.
- (10). K. Watabe, et. al. : "A Distributed Multi Party Desktop Conferencing System and its Architecture", Proc. of 9th Int. IEEE Phoenix Conf. on Computers and Communications, pp.386-393, 1990.