レジスタ付きペトリネットを用いた全体動作仕様から分散動作仕様の自動合成とその応用

山口　弘純[†]　　岡野　浩三[†]　　東野　輝夫[†]　　谷口　健一[†]

Protocol Synthesis in a Petri Net Model with Registers and Its Application

Hirozumi YAMAGUCHI[†], Kozo OKANO[†], Teruo HIGASHINO[†], and Kenichi TANIGUCHI[†]

あらまし　プロトコル合成法は分散協調システムの設計において，通信動作の記述の誤りをなくし，設計者の負担を軽減する有用な手法の一つであり，これまでにさまざまな計算モデルを用いた手法が提案されている．しかし，システムの状態変数を扱うことができ，かつ並列動作や選択動作などを含む制御構造が記述できるクラスに対する合成法は提案されていない．本論文ではレジスタ付きペトリネットモデルに対する一つのプロトコル合成法を提案する．提案する手法では，p 個のノード（計算機）とそれらの間の信頼できる通信路からなる分散システムを対象とし，そのシステムの全体動作仕様と，各ノードへのゲートやレジスタの配置指定が与えられたときに，全体動作仕様どおりの動作を行う各ノードの動作仕様の組（分散動作仕様）を自動合成する．また，提案した手法に基づく合成系や実行系を試作し，協調活動の支援に適用することにより本手法の有用性を示す．

キーワード　分散協調システム，ペトリネット，プロトコル合成，協調活動支援システム

1. まえがき

分散協調システムの設計においては，設計者がノード（計算機）間の通信動作を含む各ノードの動作仕様を記述する必要がある．しかし，一般にそれらの通信動作は複雑なデータ伝送や同期通信を含むため，誤りが生じやすい．また，設計段階での仕様や環境の変更が生じるたびに通信動作を含む各ノードの動作仕様を指定し直す必要がある．従って，設計時の通信動作の記述誤りをなくし，それらを直接記述することにより生じる設計者の負担を軽減する設計支援法が望まれる．

プロトコル合成法[2]はそのような問題を解決する有用な手法の一つである．プロトコル合成法では，分散システム全体を一つのシステムとみなしたときのシステム全体の動作内容とそれらの実行順序を記述した動作仕様（以下全体動作仕様と呼ぶ）を与える．この仕様から，各ノードが実行すべき動作内容と他ノードとの通信動作，それらの実行順序を指定した各ノードの動作仕様の組（以下分散動作仕様と呼ぶ）が自動合成される．合成された各ノードの動作仕様の組は互いに協調して全体動作仕様どおりの順に動作を実行する．プロトコル合成法では，ノード間の通信動作を直接記述する必要がないため，通信動作の記述誤りをなくし，設計者の負担を軽減することができる．

これまでに FSM や EFSM [5],[6],[8], LOTOS [3],[4], ペトリネット[10]などの計算モデルに基づくプロトコル合成法が提案されている．しかし，FSM や EFSM では並列動作を含む制御構造が記述できないため，複数ノードの並列動作を全体動作仕様レベルで指定できない．また，LOTOS では並列，選択や割込みなどの並列動作を含む制御構造が記述できるが，LOTOS を用いた従来の手法のほとんどは動作の実行順序の制御に焦点を絞り，システムの状態変数を扱えないクラスに制限している．我々は文献[10]で並列動作のみが記述できるマークグラフと呼ばれるペトリネットのサブクラスに対するプロトコル合成法を提案している．しかし，このクラスでは選択動作が記述できないため，外部入力やシステムの状態変数による条件分岐が記述できず，一般の分散協調システムの設計に応用するには十分なクラスとは言えない．

本論文では，既存の手法の制限を取り除くため，ペトリネットの一般クラスに対する一つのプロトコル合

[†] 大阪大学基礎工学部情報科学科，豊中市
Department of Information and Computer Sciences, Osaka University, Toyonaka-shi, 560 Japan

成法を提案する．対象とする分散システムは p 個のノードとそれらを結ぶ信頼できる通信路からなるとし，各ノードはそれぞれいくつかの入出力ゲートとレジスタ（リソース）を保持するものとする．提案する手法は，レジスタ付きペトリネットモデル（Petri net model with registers，以下 PNR モデル）と呼ばれるペトリネットの拡張モデルで記述された分散システムの全体動作仕様と，リソースの p 個のノードへの配置指定が与えられたときに，同モデルで記述された各ノードの動作仕様の組（分散動作仕様）を自動合成する．また，提案した手法に基づき，全体動作仕様から各ノードの動作仕様を自動合成するシステム（合成システム）と合成した各ノードの動作仕様を解釈実行するシステム（実行システム）を試作し，協調活動の計算機支援に適用することでその有用性を示す．

以下，2. ではレジスタ付きペトリネットモデルの定義を与える．3. では全体動作仕様の記述例と合成した分散動作仕様の例を与え，本論文で扱う合成問題を定義する．4. では自動合成アルゴリズムについて述べる．5. では支援システムについて述べる．最後に 6. では結論と今後の課題について述べる．

2. 記述モデル

PNR モデルは有限個の入出力ゲートとレジスタに対し，トランジションの振舞いとしてゲートへの入出力動作とレジスタ値の更新動作を指定できる．

各トランジションはラベル〈ガード，入出力動作，レジスタ値更新動作〉をもつ．入出力動作は "$a!Exp_1(\ldots), Exp_2(\ldots), \ldots ?x, y, \ldots$" または "$i$" で与えられる．$a$ は入出力ゲート，"$!Exp_1(\ldots), Exp_2(\ldots), \ldots$" は式 "$Exp_1(\ldots), Exp_2(\ldots), \ldots$" の値を出力する動作（出力動作），"$?x, y, \ldots$" は入出力ゲート a からの入力値を入力変数 "$x, y, \ldots$" に割り当てる動作（入力動作）を表す．入力変数はそのトランジション t のみに有効な変数である．各式 Exp_n はレジスタ変数を引数にもつことができる．レジスタ変数はすべてのトランジションで有効な状態変数である．"i" は入出力動作を行わないことを表し，内部動作と呼ぶ．ガードはレジスタ変数または入力変数を引数にもつことのできる述語である．レジスタ値更新動作は "$R_{h_1} \leftarrow f_1(\ldots), \ldots, R_{h_i} \leftarrow f_i(\ldots)$"，若しくは空文で与えられる．各関数 $f_j (1 \leq j \leq i)$ はレジスタ変数または入力変数を引数にもつことができる．レジスタ値更新動作はレジスタ $R_{h_1}, \ldots, R_{h_i}$ の値をそれぞれ同時に $f_1(\ldots), \ldots, f_i(\ldots)$ の値に更新する動作を表す．空文である場合はいずれのレジスタ値も更新しない．

PNR モデルのトランジション t は，(a) t の入力プレースに発火に必要なトークンがそろい，(b) t のガードの値が真であるとき発火可能である．発火可能なトランジションが発火すると，t の入出力動作が実行され，その後レジスタ値更新式が実行される．

PNR モデルは基本的にカラーペトリネットなど従来の一般的な拡張ペトリネットモデルでも表現可能であると思われるが，PNR モデルは各トランジションが参照可能な状態変数（レジスタ）を用いており，制御部とデータ処理部が比較的容易に分離できるなどの特徴がある．本手法ではそのような特徴を利用し，全体動作仕様の制御部を利用して分散動作仕様の制御部を合成するなどアルゴリズムを簡単化している．

3. 全体動作仕様と分散動作仕様

3.1 全体動作仕様

図 1 は簡単なソフトウェアの協調開発の例題の全体動作仕様（以下 *Sspec* で表す）の記述例である．本例題の全体動作仕様 *Sspec* は，ソフトウェア仕様，ソースコード，オブジェクトコード，ライブラリー，テスト仕様，テスト結果の六つのファイルをもつ一つの計算機上で解釈実行されているとみなし，それらの内容はレジスタ $R_{spec}, R_{src}, R_{obj}, R_{lib}, R_{test}, R_{res}$ で保持される．各入出力ゲート QA, SC, TC は作業者と計算機とのインタフェースを表し，作業者の作業はすべてこれらの入出力ゲートを介した計算機との入出力動作として表されるとする．QA, SC, TC 上の入出力動作は，それぞれ品質管理，コーディング，テスト

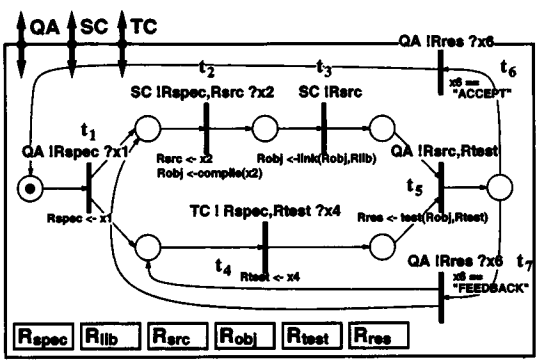

図 1　全体動作仕様 *Sspec* の例
Fig. 1　An Example of a service specification *Sspec*.

仕様生成の作業であることを表す．

トランジション t_1 はソフトウェア仕様の変更，t_2 はソースコードの変更とオブジェクトコードの生成，t_3 はライブラリーのリンク，t_4 はテスト仕様の変更，t_5 はオブジェクトコードの試験，t_6 と t_7 は作業の終了またはやり直しの決定を表す．例えば t_2 が発火すると，入出力ゲート SC（SC作業者）に R_{spec} の値（変更後のソフトウェア仕様）と R_{src} の値（変更前のソースコード）が出力される．入力変数 x_2 の値（SC作業者によって変更されたソースコード）は R_{src} に格納される．同時に，関数 compile(x_2) の値（オブジェクトコード）が R_{obj} に格納される．また，t_6, t_7 は入出力ゲート QA（QA作業者）からの入力によりいずれが発火するかが決定される．以降の図中ではガード "True"，内部動作 i，空文であるレジスタ値更新動作はすべて省略する．

3.2 分散環境とリソースの配置指定

対象とする分散システムはノード 1, ..., ノード p の p 個のノード（計算機）からなる．ノード i とノード j の間には信頼できる全2重通信路が存在するとし，そのノード i（ノード j）側を入出力ゲート g_{ij}（g_{ji}）で表す．ノード i が出力動作 $g_{ij}!d$ を実行すると，データ d（メッセージ）がノード j に送信される（メッセージ送信）．ノード j が入力動作 $g_{ji}?w$ を実行すると，送信されたメッセージが入力変数 w に読み込まれ，そのデータは通信路から取り除かれる（メッセージ受信）．もし通信路にデータが何もなければノード j は入力動作を実行することができない．また，各メッセージにはそのメッセージ固有の識別子（ID）を与える．メッセージに含まれる識別子を取り出すには関数 ID を用いる．$ID(w)$ は入力変数（メッセージ）w に含まれている識別子を返す．更に，各ノードは局所作業用レジスタ Tmp をそれぞれ保持する．Tmp は他ノードから受け取ったレジスタ値や入力変数値を一時的に保持するのに用いる．$Tmp.R_q$（$Tmp.x$）で Tmp が保持するレジスタ R_q（入力変数 x）の値を表す．

各ノードはいくつかのレジスタや入出力ゲートをそれぞれ保持する．複数のノードが同じレジスタを保持してもよいとする．Alloc ($Sspec$) は $Sspec$ で用いられるレジスタと入出力ゲートの各ノードへの配置指定である．これをリソースの配置指定と呼ぶ．図2は図1の $Sspec$ に対する Alloc ($Sspec$) の例である．なお，図2では各入出力ゲートを一つの計算機（作業者）に割り当てている（3人の協調作業）．SC, TC を同じ計算機に割り当てれば2人の協調作業となる．このように，同じ全体動作仕様に対しリソースの配置指定を変更するだけで，異なる作業割当て（人員変更など）が容易に実現できる．

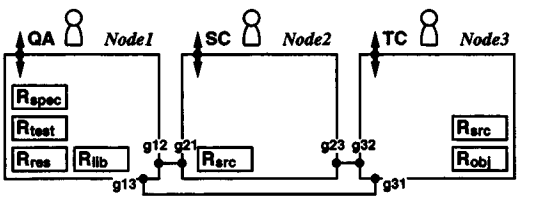

図2 ゲートとレジスタの配置指定 Alloc ($Sspec$)
Fig. 2 A resource allocation Alloc ($Sspec$).

3.3 分散動作仕様

以下，ノード k の動作仕様を $Pspec_k$ で表す．また，p 個のノードの動作仕様の組を $Pspec^{(1,p)}$ で表し，これを分散動作仕様と呼ぶ．図3は図1の全体動作仕様 $Sspec$ と図2のリソースの配置指定 Alloc($Sspec$) から自動合成した分散動作仕様 $Pspec^{(1,3)}$ である．

各ノードは互いにメッセージ交換により同期をとりながら $Sspec$ と同様の動作を実現する．以下，t_i の入出力動作が実行される入出力ゲートが配置されているノードを t_i の責任ノードと呼ぶ．ここで，図1の $Sspec$ のトランジション t_1, t_2 に注目した場合，まず t_1 の責任ノードであるノード1（QAの計算機）が t_1 のガードの真偽値判定を行う．t_1 のガードの値は常に真なので，ノード1は t_1 の入出力動作 "$QA!R_{spec}?x_1$" を実行し，レジスタ値更新動作 "$R_{spec} \leftarrow x_1$" を実行する．その後，ノード1はノード2（SCの計算機），ノード3（TCの計算機）に対し t_1 のレジスタ値更新動作の実行が終了したことを知らせるメッセージ "$\mathrm{Msf}_{1.12}$", "$\mathrm{Msf}_{1.13}$" を送信する．t_1 の次トランジション t_2, t_4 のうち，t_2 の責任ノードであるノード2はメッセージ "$\mathrm{Msf}_{1.12}$" の受信と t_2 のガードの真偽値判定により t_2 が発火可能であると判定し，t_2 の入出力動作 "$SC!R_{spec}, R_{src}?x_2$" を実行する．ここで，ノード2は R_{spec} を保持しないためその値を知らない．従って，ノード1は前述のメッセージ "$\mathrm{Msf}_{1.12}$" にレジスタ R_{spec} の値を含めておく．t_2 の入出力動作の実行後，ノード2でレジスタ R_{src}，ノード3でレジスタ R_{src}, R_{obj} の値の更新動作を行う．この際，ノード3はノード2からの入力変数 x_2 の値を含むメッセージ "$\mathrm{Mrc}_{2.23}$" を受信し，その値を用いて更新動作

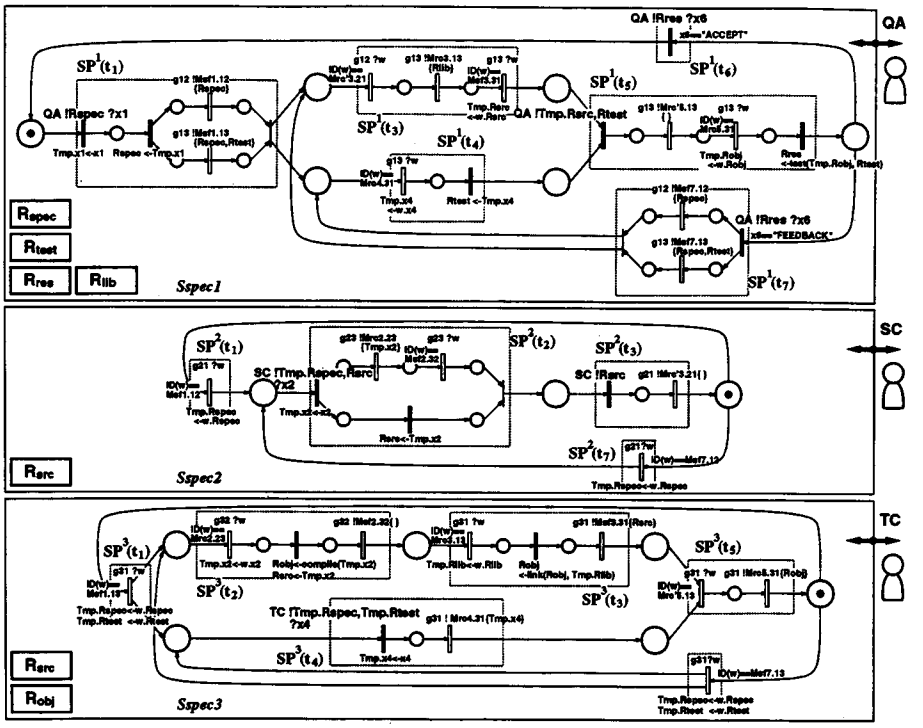

図3 分散動作仕様 $Pspec^{\langle 1,3\rangle}$
Fig. 3 Derived protocol specification $Pspec^{\langle 1,3\rangle}$.

"$R_{src} \leftarrow x_2, R_{obj} \leftarrow compile(x_2)$" を実行する．実行後は次トランジション t_3 の責任ノードであるノード2にレジスタ値更新動作が終了したことを知らせるメッセージ "$\mathrm{Msf}_{2.32}$" を送信する．

3.4 合成問題

本論文では，PNRモデルで記述された全体動作仕様 $Sspec$ と p 個のノードに対する $Sspec$ のリソースの配置指定 Alloc($Sspec$) が与えらえたとき，$Sspec$ と等価な分散動作仕様 $Pspec^{\langle 1,p\rangle}$ を合成する問題を扱う．ここで $Pspec^{\langle 1,p\rangle}$ において通信路の両端を表す入出力ゲートに対するすべての入出力動作（$g_{ij}?x$ や $g_{ij}!E(\ldots)$ など）は観測不能，その他の入出力動作は観測可能であるとしたとき，$Sspec$ と $Pspec^{\langle 1,p\rangle}$ が観測合同であれば[7]，両者は等価であると言う．

$Sspec$ と Alloc($Sspec$) に以下の制約を仮定する．

(制約1) $Sspec$ を記述するペトリネットは活性かつ安全[1]でなければならない．

(制約2) $Sspec$ のトランジション t_i と t_j の各組に対し，t_i と t_j が入力プレース p を共有しているなら，t_i と t_j の責任ノードは同じでなければならない．

(制約3) 同時発火可能である $Sspec$ のトランジション t_i と t_j の各組に対し，(a) t_i, t_j 上で同一レジスタ R の値が更新されるか，または (b) t_i 上でレジスタ R の値が更新され t_j 上で R の値が参照されるか，いずれかが成り立つとき，これをレジスタ競合と呼ぶ．$Sspec$ はレジスタ競合を含んではならない．

(制約4) $Sspec$ は内部動作 "i" を含んではならない．

提案する手法では制約1の仮定のもとで，各ノードの動作仕様に含まれる冗長な動作を除去する．

制約2を仮定すると，選択構造をもつプレース p に対し，どの出力トランジションを発火させるかを一つのノード（p の出力トランジションの責任ノード）だけで決定できる．例えば，t_6, t_7 の責任ノードはいずれもノード1であるため，ノード1はどちらを発火させるかを他のノードに関係なく決定できる．

制約3は複数トランジションの同時実行による同一レジスタへの書込みを禁止している．本手法では，全体動作仕様レベルでは不可分な一トランジションの動作を分散動作仕様レベルでは複数トランジションで実現しているため，同一レジスタへのアクセスの競

合（レジスタ競合）を生じる可能性がある．我々は文献[10]において，分散排他制御アルゴリズムを用いてこの問題を解決する方法を述べている．

制約4は本質的な制約ではないが，以下の議論の簡単のため仮定する．

4. 分散動作仕様合成アルゴリズム

基本的には，(1) $Sspec$ の一トランジション t_i ごと独立に，その動作を複数ノード上で模倣することを考え，(2) $t_1,\ldots,t_n$ の模倣を $Sspec$ でのそれらの実行順序どおりに実行することで $Sspec$ の動作を $Pspec^{\langle 1,p\rangle}$ で実現する．(1) では，Alloc ($Sspec$) で指定されるレジスタやゲートの割当てのもとでどのノードが $Sspec$ の t_i の入出力動作やレジスタ値更新動作を行い，どのようなメッセージを交換すべきかを定めた一定の方針（これを一遷移模倣方針と呼ぶ）に従って t_i の動作を実現する．一遷移模倣方針は付録で述べる．

具体的には以下の手順で $Pspec^{\langle 1,p\rangle}$ を合成する．

(手順1) $Sspec$ の各 t_i の動作を Alloc ($Sspec$) のもとでノード 1, 2, ..., p 上で模倣する全ノードに対する部分動作仕様群 $SP^1(t_i),\ldots,SP^p(t_i)$ を一遷移模倣方針に従い構成する．ここで $SP^1(t_i),\ldots,SP^p(t_i)$ は各ノード k が $SP^k(t_i)$ を実行すると t_i の動作がそれらのノード上で模倣されるようなサブペトリネット群である．

(手順2) 各ノード k において，$Sspec$ の各 t_i を対応するサブペトリネット $SP^k(t_i)$ で置換して得られたネットを $Pspec_k$ とする．各ノード k においてサブペトリネット群 $SP^k(t_1),\ldots,SP^k(t_n)$ は $Sspec$ の $t_1,\ldots,t_n$ の実行順序どおりに実行されるため，$Sspec$ 全体の動作が $Pspec^{\langle 1,p\rangle}$ で実現される．

4.1 全体仕様の一トランジションを模倣する部分動作仕様群の構成

手順1において，$SP^k(t_i)$ は，(a) t_i のガードと入出力動作をもつトランジション，(b) t_i のレジスタ値更新動作（の一部）をもつトランジション，(c) メッセージ送信のための出力動作をもつトランジション，(d) メッセージ受信のための入力動作とメッセージに含まれるレジスタ値や入力変数値を作業用レジスタ Tmp に格納する動作をもつトランジションを付録の一遷移模倣方針により決定される実行順序の依存関係に従って組み合わせることで得られる．例えば $Sspec$ のトランジション t_2 に対し，ノード2のサブペトリネット $SP^2(t_2)$ は入出力動作を実行し，ノード3に RC メッセージ "$\mathrm{Mrc}_{2.23}$" を送信し，ノード3からの SF メッセージ "$\mathrm{Msf}_{2.32}$" を受信する三つのトランジションと，レジスタ R_{src} の値の更新動作を実行するトランジションからなり，ノード3のサブペトリネット $SP^3(t_2)$ はノード2からの RC メッセージ "$\mathrm{Mrc}_{2.23}$" を受信し，レジスタ R_{src}, R_{obj} の値の更新動作を実行し，ノード2に SF メッセージ "$\mathrm{Msf}_{2.32}$" を送信する三つのトランジションからなる（図3参照）．ノード1は t_2 の模倣に関係しないので $SP^1(t_2)$ は空である．なお，メッセージの識別子 "$\mathrm{Mrc}_{i.xy}$" ("$\mathrm{Msf}_{i.xy}$") はトランジション t_i の模倣のためのノード x から y への RC (SF) メッセージであることを表す．

ここで，$Sspec$ の各 t_i ごと，それを模倣するための実行コストを最小化することを考える．分散システムの実行コストとしては一般に動作の処理コストと通信コストとが挙げられる．本手法では一定の方針（一遷移模倣方針）に従って t_i の模倣の仕方を決定するが，この方針に従った場合には t_i の入出力動作やレジスタ値更新動作に関しては，それらの実行に必要な計算をどのノードが行うかといったことはすべて一意に決定される．これに対し，メッセージ交換の仕方には自由度があり，一意に決定されない場合がある．例えば，責任ノード u とノード v' がともにレジスタ R_q を保持しており，ノード v ($\neq u,v'$) がその値を更新動作に必要とする場合を考える．このとき，ノード u が R_q の値を送信することにすれば RC′ メッセージは不要であるため，ノード v' が送信する場合と比べてメッセージ総数は1少なくなる．しかし，ノード v' が別のレジスタ R'_q も保持しており，ノード v ($\neq u,v'$) が R_q, R'_q の値をともに必要とする場合，ノード v' がそれらの値を送信することにすればノード u が送信する RC メッセージは不要であるため，ノード u，ノード v' がそれぞれ R_q, R'_q の値を別々に送信する場合と比べてメッセージ総数は1少なくなる．上述のように，本手法では一遷移の模倣の実行コストのうち動作の処理コストは一定であるとみなせるため，通信コスト，特にメッセージ総数をコスト基準として採用しこれを最小化する．最小化では，例えば "レジスタ値更新動作を実行する各ノード v には必ず RC メッセージが送信されなければならない" など，模倣方針に従った場合に満たすべき制約を以下のような 0-1 整数線形不等式として表す．

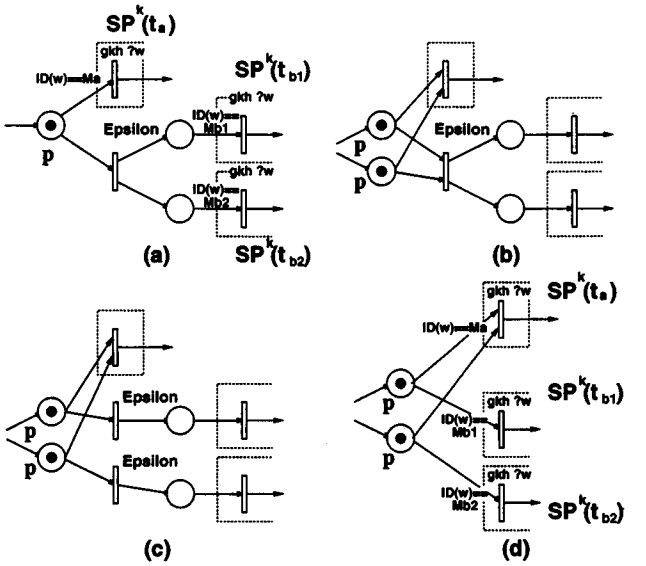

図4　ε-トランジションの除去
Fig. 4　Removal of an ε-transition.

$$\mathrm{RC}_{uv} + \sum_{v' \neq u} \mathrm{RC}_{v'v} \geqq 1$$

ここで，RC_{xy} は RC メッセージがノード x からノード y に送信される場合は 1，そうでない場合は 0 である 0-1 整数変数である．それらのような 0-1 整数線形不等式に対し，すべての変数の総和を目的関数としてそれを最小とする解を 0-1 整数線形計画問題の解法を用いて得る．得られた解より，t_i の模倣に必要なメッセージ総数が一遷移模倣方針のもとで最小であるメッセージ交換の仕方を決定することができる(注1)．なお，紙面の都合上その他の 0-1 整数線形不等式の立て方については省略する [8],[10].

4.2　各ノードの動作仕様の合成

手順 2 では各ノード k において，$Sspec$ の各トランジション t_i をサブペトリネット $SP^k(t_i)$ で置換する ($SP^k(t_i)$ が空でない場合)．$SP^k(t_i)$ が空である場合は，まず t_i を ε-トランジション（ラベルをもたないトランジション）で置換し，次いで後述の方法で ε-トランジションの除去を行う．この方法をとる理由も後述する．以上により得られるネットを $Pspec_k$ とする．

ε-トランジションを除去する方法は以下のとおりである．まず，$Sspec$ を記述する活性かつ安全なペトリネット PN から，それを覆う強連結な SM-成分 [1] $SM_1,\ldots,SM_m$ を見つける手続き Find を導入する．ここで，SM-成分とはその各トランジションが入出力プレースをたかだか一つずつもつような PN の部分ネットである．

[手続き Find]

与えられたペトリネットを PN とし，$i=1$ とする．

（1）（$i = j$ のとき）既に得られた $SM_1,\ldots,SM_{j-1}$ のいずれにも含まれないアーク e が存在すれば，e を通る PN の有向ループを一つ見つけ，それを SM_j とする．存在しなければ $m=j-1$ として手続きを終了する．

（2）SM_j 上の二つのプレース p_x, p_y の各組に対し，p_x から p_y への SM_j を通らない有向経路があればそれを SM_j に追加する．SM_j にそれ以上新しい経路が追加されなくなるまで（2）を繰り返す．

（3）$i=i+1$ とし，（1）に戻る．　□

次に，得られた $SM_1,\ldots,SM_m$ を使って ε-トランジションを除去する手続き Remove を各 $Pspec_k$ に適用し，すべての ε-トランジションを除去する．この際，各サブペトリネットは一トランジションとして扱う．なお，手続き Remove に関しては紙面の都合上，以下の適用例をもとにアイデアを述べ，詳細は省略する [11].

例えば，図 4(a) のノード k の動作仕様において，

(注1)：そのような解は複数存在する可能性もあるが，それらのいずれを用いてもメッセージ総数は一遷移模倣方針のもとで最小であることを保証できる．

ノード k は受信したメッセージのIDによりサブペトリネット $SP^k(t_a)$ を実行するか，サブペトリネット $SP^k(t_{b_1})$ と $SP^k(t_{b_2})$ を実行するかを決定しなければならないが，このように ε-トランジションが同期点を表している（複数の入力プレースまたは出力プレースをもつ）場合は，上述のような選択構造を保持したまま，有限オートマトンにおける状態縮約のように前後の状態（プレース）をまとめて ε-遷移を除去できない．従って，手続き Remove では手続き Find に従って見つけたSM-成分を利用し，まず ε-トランジションの入出力プレースをそのプレースを含むSM-成分の数だけ複製する．例では二つのSM-成分がプレース p を含むため，p が二つとなるように複製する（図 4(b)）．これを行う理由は次で述べる．次にそれらのSM-成分がそれぞれ別々に ε-トランジションを含むようにその ε-トランジションを分割する．例では p を含む二つのSM-成分がそれぞれ独立に ε-トランジションを保持している（図 4(c)）．上述のプレースの複製により，ε-トランジションの入出力プレースはちょうどそれを含むSM-成分の数だけ存在するため，分割された各 ε-トランジションの入出力プレースは一つずつとなる．従って，その入出力プレースをまとめて一プレースとし，ε-トランジションとその入出力アークを除去（状態縮約）し，ノード k の動作仕様（図 4(d)）を得る．この動作仕様では上述の選択動作が実現される．

なお，サブペトリネットへの置換において，$SP^k(t_i)$ の先頭または終端のトランジションが一つでない場合は，それらの同期をとるための ε-トランジションを $SP^k(t_i)$ に追加した後置換する（それらも Remove により除去される）．

5. 支援システム

我々は，(a) 分散システムの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合成し，(b) 合成した各動作仕様を計算機上で解釈実行するシステムを試作した．(a)，(b) を実行するシステムをそれぞれ合成システム，実行システムと呼ぶ．以下ではシステムの概略とシステムの協調活動支援機能について述べる．

5.1 合成システム

ここでは，提案する自動合成の手法が現実的な時間で合成可能であるかどうかを確認するため，自動合成システムを作成し，それを用いて合成に要する時間を測定した．測定対象として，(a) 本論文で用いた簡単な例題の $Sspec$（トランジション数 7，ノード数 3，レジスタ数 6）と Alloc($Sspec$)（図 2），(b) 実際のソフトウェア開発により近い工程数，人数，レジスタ数での例題の $Sspec'$（トランジション数 35，ノード数 10，レジスタ数 50）と Alloc ($Sspec'$)（一般的な分散配置，一部のレジスタはフォールトに備え多重配置）を用い，PC/AT 互換機（CPU: Pentium Pro，クロック: 200 MHz，メインメモリ 96 Mbyte）上で合成システムを実行した結果，合成に要した CPU 時間は，(a) 0.4 秒，(b) 1031 秒であった．なお，(b) においてメッセージ数の最適化を行わず，最適解の近似解を求めるアルゴリズムを用いた合成システムが合成に要した CPU 時間は 39.2 秒であった．なお，一つの全体動作仕様に対しても異なるリソース配置指定を与えれば合成に要する処理時間も異なり得る．これらの例題より実用規模の協調活動に対しても十分実用的な時間で自動合成が可能であると思える．

また，異なるリソース配置指定に対しては，合成した分散動作仕様の実行処理時間も異なる．本手法では設計者が一つの全体動作仕様に対しリソース配置指定を変えて自動合成したいくつかの分散動作仕様を比較評価することにより，実行処理時間が他より短い分散動作仕様を得ることもできる．

5.2 実行システム

実行システムは，各計算機上におかれ，ユーザ k が使用する計算機 k 上で合成システムにより自動合成された動作仕様 $Pspec_k$ を解釈実行する．実行システムは実行制御部，ネットワーク制御部，ユーザ制御部からなる．なお，ユーザ制御部は X window 上で OSF/Motif を用いて実装した（図 5）．

各実行システムにおいては，実行制御部が発火可能なトランジションを計算し，それらの入出力動作を抽出する．ユーザ制御部では，それらの入出力動作のリストを表示し，ユーザに選択を促すと共にペトリネットのグラフ表現による動作仕様の表示を行う．実行制御部はユーザが選択した入出力動作に対し，指定されたレジスタ名（ファイル名）を引数として，入出力に必要なツールを起動する．また，実行制御部はレジスタ値更新動作を自動実行し，実行可能な通信動作をネットワーク制御部を介して自動実行する．

本実行システムは，協調活動向けの以下の支援機能を提供する．

- 作業活動の管理および作業補助：主に，(a) 作業（作業者との入出力動作）の誘導，(b) 通信動作の

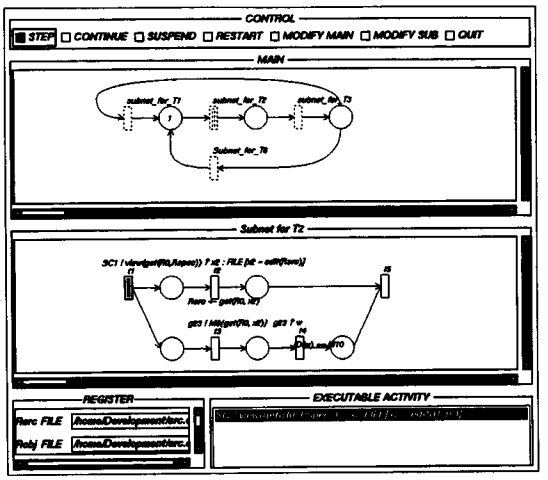

図5　実行システムの実行画面
Fig. 5　Window of execution system.

自動実行，(c) ファイルなどを表すレジスタの値の自動更新，が挙げられる．(b)，(c) により，本来の作業以外の作業の実行誤りを防ぎ，繁雑さを解消することができる．(a) では実行可能な作業の抽出，作業に必要なツールの自動起動などがある．これらにより作業順序の実行誤りや，ファイル操作時の対象ファイルの指定誤りなどを防ぐことができる．

- 作業記述，作業割当ての実行時の変更要求（動的変更）への対応：支援システムでは，分散制御されている実行中の作業活動を矛盾なく停止させる機能をもつ．従って，作業記述や作業割当ての動的変更要求に対し，それぞれの作業活動を停止させ，全体の活動記述または作業割当て（リソースの配置指定）を変更し，合成システムを起動して新しい作業記述を再合成し，新しい作業記述を各作業者の計算機に配布することで作業の動的変更に対応することが可能である．これにより，例えばソフトウェアの開発過程で頻繁に生じる開発プロセスの変更や人員変更などにも柔軟に対処できる．

- 作業記述の視覚化：ユーザ制御部はペトリネットのグラフ表現によりその作業者の作業記述を表示する．従って作業者は現在や今後どのような作業が可能かを容易に把握できる．また，作業記述における各作業の詳細をサブペトリネットとして別ウィンドウに階層化表現できる．このため，作業者は作業全体の流れと各作業の詳細とが容易に把握できる．

6. むすび

本論文では，PNR モデルで記述された分散システムの全体動作仕様と，各ノードへのリソースの配置指定が与えらえたときに，各ノードの動作仕様の組（分散動作仕様）を自動合成する手法とその応用について述べた．今後の課題は，試作した支援システムを実際の協調活動支援に適用し，手法の有用性を評価することなどである．

文　献

[1] T. Murata, "Petri nets: properties, analysis and applications," Proc. of the IEEE, vol.77, no.4, pp.541–580, 1989.

[2] R. Probert and K. Saleh, "Synthesis of communication protocols: Survey and assessment," IEEE Trans. Comp., vol.40, no.4, pp.468–476, 1991.

[3] R. Gotzhein and G. v. Bochmann, "Deriving protocol specifications from service specifications including parameters," ACM Trans. Comp. Syst., vol.8, no.4, pp.255–283, 1990.

[4] C. Kant, T. Higashino, and G. v. Bochmann, "Deriving protocol specifications from service specifications written in LOTOS," Distributed Computing, vol.10, no.1, pp.29–47, 1996.

[5] P.-Y.M. Chu and M.T. Liu, "Synthesizing protocol specifications from service specifications in FSM model," Computer Networking Symp. '88, pp.173–182, 1988.

[6] Y. Kakuda, H. Igarashi, and T. Kikuno, "Automated synthesis of protocol specifications with message collisions and verification of timeliness," Proc. Int. Conf. on Network Protocols (ICNP '94), 1994.

[7] R. Milner, "Communication and concurrency," Prentice-Hall, 1989.

[8] 岡野浩三，今城広志，東野輝夫，谷口健一，"拡張有限状態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，" 情処学論，vol.34, no.6, pp.1290–1301, 1993.

[9] 安本慶一，東野輝夫，谷口健一，"LOTOS によるソフトウェアプロセスの記述とその実行，" コンピュータソフトウェア，vol.12, no.1, pp.16–30, 1995.

[10] H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi, "Synthesis of protocol specifications from service specifications of distributed systems in a marked graph model," IEICE Trans. Fundamentals, vol.E77-A, no.10, pp.1623–1633, 1994.

[11] H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi, "Synthesis of protocol entities' specifications from service specifications in a Petri net model with registers," Proc. 15th Int. Conf. on Distributed Computing Systems (ICDCS-15), pp.510–517, 1995.

付　　録

[一遷移模倣方針]

● *Sspec* のトランジション t_i（模倣対象のトランジション）の責任ノードをノード u とする．ノード u は（a）t_i のガードの値が真であり，かつ（b）t_i の前トランジションの模倣が完了したことを確認（後述する SF メッセージを受信）すれば，t_i の入出力動作を実行する．但し，ガードの真偽値判定および出力動作の実行に必要なレジスタ値のうちノード u が保持しない値は（b）の SF メッセージにより受け取るとする．なお，初期状態では各ノードは各レジスタの初期値を知っているものと仮定する．

● レジスタ値更新動作は，値を更新されるレジスタを保持するノードがそれぞれ独立に実行する．この各ノードをノード v とする．ノード u は入出力動作の実行後に，以下のようなメッセージを送信する．

－ ノード v が更新動作の実行に入力変数値若しくはノード u が保持するレジスタ値を必要とするなら，ノード u はそれらの値を含むメッセージ（RC メッセージと呼ぶ）をノード v に送信する．

－ ノード v が更新動作の実行にノード v'（$v' \neq u$）が保持するレジスタ値を必要とするなら，ノード u は RC メッセージの送信を依頼するメッセージ（RC′ メッセージ）をノード v' に送信する．ノード v' は依頼されたレジスタ値を含む RC メッセージをノード v に送信する．

－ ノード v が自ノードの保持するレジスタ値のみで更新動作を実行可能であるなら，ノード u は値を含まない RC メッセージをノード v に送信する．これにより，各ノード v（$\neq u$）に少なくとも一つの RC メッセージが送信される．

● RC メッセージを受信した各ノード v はレジスタ値更新動作を実行する．$v = u$ である場合，ノード u は入出力動作の実行後に実行する．

● 各ノード v は更新動作の実行後，次トランジションの各責任ノード z に対しメッセージ（SF メッセージと呼ぶ）を送信する．その際，次トランジションのガードの真偽値判定や出力動作の実行に必要なレジスタ値をノード v が保持しているなら，そのメッセージにその値を含めておく．

● もしノード z が次トランジションのガードの真偽値判定や出力動作の実行に必要なレジスタ値を，トランジション t_i のレジスタ値更新に無関係なノード z'（$\neq v$）が保持しているなら，ノード u は入出力動作の実行後，SF メッセージの送信を依頼するメッセージ（SF′ メッセージ）をノード z' に送信する．ノード z' は依頼されたレジスタ値を含む SF メッセージをノード z に送信する．

● ノード u が上記のいずれのメッセージも送信しないなら，ノード u は SF メッセージをノード z に送信する．

（平成 8 年 12 月 9 日受付，9 年 3 月 28 日再受付）

山口　弘純

平 6 阪大・基礎工・情報卒．平 8 同大大学院博士前期課程了．現在，同大大学院博士後期課程在学中．分散システムの設計法等の研究に従事．平 8 より日本学術振興会特別研究員．情報処理学会会員．

岡野　浩三　（正員）

平 2 阪大・基礎工・情報卒．平 5 同大大学院博士後期課程中退．同年同大情報工学科助手，現在に至る．工博．代数的手法によるプログラム開発，分散システムなどの研究に従事．情報処理学会会員．

東野　輝夫　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大助手．平 2，6 モントリオール大学客員研究員．平 3 阪大・基礎工・情報工学科助教授．工博．分散システム，通信プロトコル等の研究に従事．情報処理学会，IEEE，ACM 各会員．

谷口　健一　（正員）

昭 40 阪大・工・電子卒．昭 45 同大大学院博士課程了．同年同大・基礎工・助手，現在，同情報工学科教授．工博．この間，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．