マルチメディア通信と分散処理　102－23
（２００１．３．２２）

制約指向モデルで記述された対称性を持つ並行システムの要求仕様に対する形式的検証とプロトコル合成

梅津 高朗† 山口 弘純† 安本 慶一†† 東野 輝夫†

†：大阪大学大学院基礎工学研究科情報数理系

††：滋賀大学経済学部情報管理学科

umedu@ics.es.osaka-u.ac.jp

本研究では、グループワークなどネットワークを介した対称性を持つ並行システムの仕様記述のための制約指向モデルと効率の良いデッドロックの検出法、ならびに、システム全体の動作仕様（サービス仕様）から各ノードの動作仕様群（プロトコル仕様）を自動導出するための一つのプロトコル合成法を提案する。提案する手法では、記述スタイルは制約指向モデルを採用し、並行システム全体の動作仕様を色つきペトリネットの集合とその間の同期指定で記述する。また、状態間の同値関係を自動的に検出して高速に可達性判定を行う検証系を作成した。また、提案モデルにおけるプロトコル合成法を考案し実装を行った。

Deadlock Detection and Protocol Synthesis for Constraint Oriented Concurrent Systems with Symmetries

Takaaki Umedu† Hirozumi Yamaguchi†
Keiichi Yasumoto†† Teruo Higashino†

†: Dept. Info. and Math. Sci., Osaka Univ., Japan

††: Fac. of Economics, Shiga Univ., Japan

In this paper, we introduce a model for designing distributed cooperative systems (concurrent systems) with symmetries. We propose an efficient deadlock detection method and a protocol synthesis method for this model. In our method, we adopt a constraint oriented style, and we describe the specification of the whole system by a set of colored Petri-nets and synchronization among them. We have developed a verification tool, which checks the reachablity of concurrent systems efficiently by using the equivalence relation among states. Then we introduce a protocol synthesis method for this model that derives the specifications of all the nodes (the protocol specification) from a given specification of the whole system (service specification) automatically.

1　はじめに

近年のネットワークの高速化に伴い、ネットワーク電子会議や遠隔授業、マルチメディアを利用したグループワークなどネットワークを介した複数ノード間の並行分散システムが数多く開発されている。これらの並行分散システムでは、毎回参加者の人数が変わったり、参加者の人数やネットワーク環境に応じて異なる複数の制約条件が付加される場合が多い。しかし、複数の制約条件が課されると、それらの制約条件をすべて満たすような実行系列が存在しない（すなわちデッドロックが含まれる）場合がある。そこで、本研究では、このような並行分散システムの仕様記述のためのモデルと効率の良いデッドロックの検出法、ならびに、システム全体の動作仕様から各ノードの動作仕様群を自動導出するための一つのプロトコル合成法を提案する。

提案する手法では、並行システム全体の動作仕様（要求仕様）を色つきペトリネットの集合とその間の同期指定で記述する。記述スタイルは仕様記述言語 LOTOS[1] などで用いられる制約指向モデル [2, 3] を採用する。このような制約指向モデルでシステム全体の動作仕様を指定した場合、システム全体の挙動の変更・制限が容易に行える反面、互いに矛盾するような制約プロセス群を指定した場合、システム全体としてデッドロックなど予期しない事態が生じる可能性がある。本研究では対象とするアプリケーションの性質に適した検証方法として、要求仕様の各ノードの実行プロセス群と全体の実行制御やノード間の制約を記述した制約プロセス群の依存関係から、対称性を持つ実行プロセス群（どのプロセスが実行を行っても全体の可達性解析に影響を与えないプロセス群）を自動的に抽出し、それらの対称性を用いて効率よく可達性解析を行う方法を提案する。

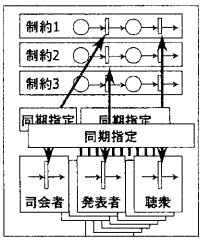

図 1: 提案モデルによるネットワーク会議の仕様

また、本研究では上述のモデルで書かれた制約指向スタイルの要求仕様に対し、その要求仕様を複数ノードで動作させるプログラム群(各ノードの動作仕様群)を自動導出するアルゴリズムを提案する。一般に、分散システムの信頼性を高め、効率よくシステムを設計するための手法としてプロトコル合成法に関する研究が活発に行われている[5]。本モデルに対するプロトコル合成法を考案し、そのアルゴリズムを用いて導出された各ノードの動作仕様群が全体として与えられた要求仕様と等価になることを示している。また、提案手法に基づくプロトコル合成系を試作した。合成に際しては、全体の動作仕様を独立に同期判定が可能な自由選択ネット群に分割することにより、同期判定に要するメッセージ数を削減する工夫を行った。

以下では、2章で本論文で対象とする制約指向モデルとカラーペトリネットの詳細について述べ、3章で提案モデルに対する対称性を利用した可達性解析の手法を説明する。4章で本モデルに対するプロトコル合成法を説明し、5章でまとめと今後の課題を述べる。

2 記述モデル

提案手法における記述モデルについて説明する。提案手法では並行システムの仕様はカラーペトリネット[4]のサブセットにペトリネット間での並列同期等を記述できるよう拡張したモデルを用いる。

2.1 制約指向

多人数が参加するグループワークなどのシステムの仕様を単一つのシステムの仕様として記述することは、ユーザー数の増加に従ってユーザー間のインタラクションや排他制御などの制約条件の付与が増大するため、非常に煩雑な作業となる。そこで、本稿では制約指向に基づく並行システムの記述モデルを提案する。

提案モデルによる簡単なネットワーク会議の仕様の概要を図1に示す。提案モデルにおいては、各々のユーザーの仕様とその間の関係を別々に与え、それらの組でシステム全体の仕様を記述する。システム全体の仕様は、(a)システムに含まれる各参加者の動作を他ノードとの関係を含まない形で色つきペトリネットを用いて個別に定義した**動作ネット**(実行プロセス)群と、(b)複数ノード間の動作の実行順や排他制御などの相互関係やシステム全体として満たすべき制約を色つきペトリネットを用いて定義した**制約ネット**(制約プロセス)群、および、(c)その間の**同期指定**、で記述する。まず、動作ネットとして各参加者の具体的な動作をそれぞれ独立した形で記述する。その際に同じ動作をする参加者(この例では聴衆や発表者)は一つのペトリネットによってその動作をモデル化する。そして、各参加者を表す複数の色のトークンを定義し、同じペトリネットの仕様を共有する各参加者の区別はそれによって行う。一つの仕様を複数人で共有することによってシステムの対称性を明示的に記述できる。動作ネット間の関係は全て制約ネットとの同期指定を用いて定義する。制約ネットでは動作ネット間の関係の他にシステム全体が満たすべき性質などを指定する。複数の制約を組み合わせることで容易にシステム全体の動作の変更を行うことができる。ただし、提案モデルを構成するそれぞれの要素において用いるトークンの色、変数の型等は以下のように限定する。

- 動作ネット：列挙型の有限カラーのみ
- 制約ネット：列挙型の有限カラー、ヴァリアント型、上下限値付整数
- 同期指定：制約ネットと動作ネットのトランジションの組の集合。各組にはそれらのトランジションの入力変数に関する同期条件。ヴァリアント型にバインドされたトークンの数を得る関数#()。

ここで、ヴァリアント型とは任意のカラー、個数のトークンとバインド可能な変数とし、複数の参加者による同期の指定などで利用する。

図2にこのようにしてモデル化を行った具体的な仕様を示す。この例は1人の発表者と複数人の聴衆からなるごく簡単なネットワーク会議の仕様を表している。この図において、上段の2つのペトリネットが動作ネットを表し、下段の2つのペトリネットが制約ネットを表している。この例において、PresenterにはSTART(プレゼンテーション開始)とEND(プレゼンテーション終了)の2つの動作が定義されており、AudienceにはQUESTION_START(質問開始)とQUESTION_END(質問終了)の2つの動作が定義されている。ここでは、参加者が独立して行う動作とそれらの実行制御のみを定義する。

この例ではRestrict1、Restrict2の2つの制約ネットが定義されており、これらは以下のような制約を表している。

(i) 質問はプレゼンテーションの開始後でなければ行えない。
(ii) 質問は各人高々1回しか行えない。
(iii) 質問中は終了できない。
(iv) 質問は必ずちょうどk回行われる。

(i)は、Restrict1のトランジションRES1_STARTがPresenterの動作ネットのSTARTと同期し(図2の矢印1)、RES1_QUESTION_STARTがAudienceの動作ネットのQUESTION_STARTと同期するため(同2)、それらの組み合わせによりこの条件が満たされる。(ii)はRestrict1のRES1_STARTによって全てのAudienceを表すトークンがRES1_PLACE1に置かれる。そして各AudienceのQUESTIONはRES1_QUESTIONと同期することが指定されているため(同2)、RES1_PLACE1に置かれたトークンは発言が行われるたびに消費される。ここでは、同期の条件として"y=z"が指定されている。この指定によりRES1_QUESTION_STARTとQUESTION_STARTの同期は変数y、zにバインドされるトークンが等しい場合にのみ制限される。よって、各Audienceは高々1回しか発言することができない。(iii)

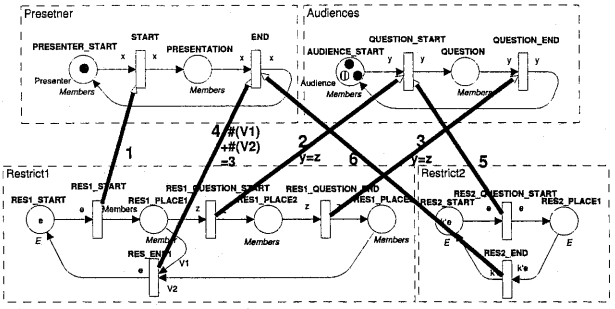

図 2: 提案モデルによるネットワーク会議の仕様記述

も同様に Restrict1 によって条件付けられていおり (同 3,4)、RES1_END と END の同期の指定には、ヴァリアント型として使用する変数 $V1$,$V2$ および、同期条件 "$\#(V1)+\#(V2)=3$" が設定されている。よって、$V1$ と $V2$ にバインドされたトークン数が聴衆の人数、3 人に一致する場合のみ同期可能で、結果としてすべての聴衆が質問中でない場合のみ終了が可能となる。最後の (iv) は Restrict2 によって定められる制約である。まず、初期マーキングにおいては RES2_START のプレースに k 個のトークンが置かれる。これらは QUESTION_START と同期が指定されている RES2_QUESTION_START の発火によって消費され (同 5)、RES2_PLACE1 にトークンが移動する。END と同期が指定されている RES2_END は、k個のトークンが無ければ発火できないため、ちょうど k回の質問が成されるまでは発火できない (同 6)。

これらの仕様群は等価的に一つのシステム全体の動作の仕様に変換することができる。検証はシステム全体の仕様を一つにまとめた上で行う。

3 検証

提案モデルにおいては複数の制約を組み合わせて仕様を定義するため、それらの組み合わせによっては予期せぬ Deadlock 等に陥る可能性がある。Deadlock や Liveness 等の性質を可達グラフの一種である Occurrence グラフ [4] を作成することで検証を行う。そのために、与えられた仕様を合成して一つのシステム全体の動作記述に変換する。まず、その変換手順を示した後、検証の方法について述べる。

3.1 変換手順

提案手法においては、まず、並列有限カラーペトリネットで与えられた仕様の集合を単一の有限カラーペトリネットに変換し、その後、変換されたペトリネットに対して検証を行う。

変換は以下のように行う。

(i) 同期指定のある制約ネットのトランジションをそれぞれ同期指定先の動作ネットのトランジションと一つにまとめ、同期条件が指定されていればそれをガード関数に追加する (図 3)。

(ii) ヴァリアント型の変数が使用されているトランジションは、ヴァリアント型を使用しないトランジションに変換する (図 4)。変換は次のようにして行う。

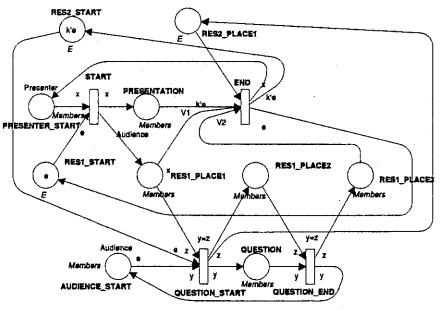

図 3: 図 2 より得られるシステム全体の仕様

(a) 各ヴァリアント型の変数に対するバインディングとして正当であり、かつ、指定されたガード関数を満たすことができるトークンや変数の組み合わせを調べる。

(b) それらに対応するトランジションをそれぞれ作成し、具体的にトークンや変数を割り当てる。

(a) が可能であるためには、ヴァリアント型の変数に対する割り当てのパターンが有限に制限されていなければならない。このようにして図 2のペトリネット群から変換されたものが図 4である。(ii) の処理の例として、この図のトランジション END1〜END4 があげられる。これらのトランジションは、図 2の END と RES2_END1 を組み合わせた結果生じたもので、RES2_END1 に含まれるヴァリアント型の変数は、変数 $x1$〜$x3$ に置き換えられている。即ち、"$\#(V1)+\#(V2)=3$" なる条件より、$V1$、$V2$ が含みうるトークンの数は、$\{(\#(V1),\#(V2))\}=\{(0,1),(1,2),(2,1),(3,0)\}$ の 4 通りとなり、END1〜END4 はそれぞれのトークン数の組み合わせに対応したトランジションとなっている。例えば END2 では、$V1$ は 1 つの変数 $x3$ に置き換えられており、また、$V2$ は 2 つのトークンがバインドできるように、$x1+x2$ に置き換えられている。

このようにしてヴァリアント型を含む並列有限カラーペトリネットは単一の有限カラーペトリネットに変換で

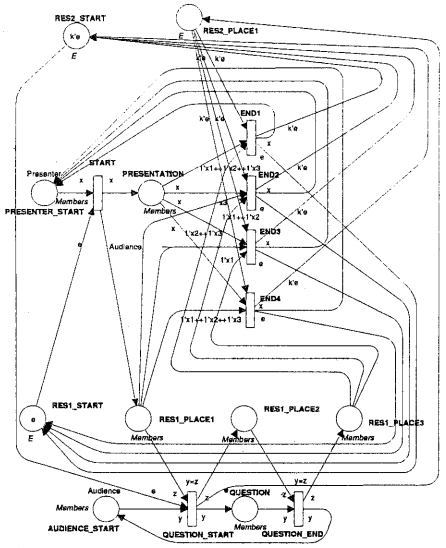

図 4: 変換して導出した全体の仕様

きる。

3.2 OS グラフを用いた検証

例えば、1 人の司会者、k人の発表者と、n 人の聴衆からなるネットワーク会議の例を考えた場合、このネットワーク会議は各々のユーザーの仕様を表す$k+n+1$個のペトリネットの集合としてモデル化できる。しかし、その仕様から合成したカラーペトリネットの検証を行う場合に、そのまま単純に$k+n+1$種類のトークンとして扱うとその Occurrence Graph(Reachability Graph) の状態数は非常に大きくなる。しかし、実際は「聴衆」のうちのいずれかが質問を行う、といった仕様である場合には、質問をした人間が誰であっても、システム全体の動作は同じであると見なすことができる。この場合は、Occurrence Graph を作成する際に「聴衆」を区別する必要はなく、代わりに対称性を用いてサイズを小さくした OS Graph(Occurrence Graph With Symmetries)[4] と呼ばれるグラフを作成することで全状態数を削減し可達判定を高速に行うことができる。

OS Graph は、可達グラフのノードを与えられた対称性に基づく同値関係を用いた同値類にしたものである。OS Graph においては 1 つのノードでそのノードの示す状態と同値関係にある全ての状態を表現することができるため、状態間で多くの同値関係が存在するようなペトリネットの検証に用いれば、大幅にグラフのサイズを縮小することができる。

3.2.1 対称性の自動導出

本手法においては、以下の検証の高速化のために以下のような方法を用いる。

- シーケンシャルな動作は一つにまとめる。明らかに一定の順番で発火することが分かっている同期指定のされていないトランジション群はそれらをまとめて一つのトランジションに置き換える。
- 制約ネットと同期指定に動作ネットのトークンの具体名が書かれいない色のトークンに関してはそれらは対称なものとして扱う。
- あるトークンの集合に対して、それに含まれるすべてのトークンが制約ネット、動作ネットの仕様中に具体名が書かれておらず、かつ、そのトークンがバインドされる可能性のある変数が各制約ネットの同期判定において高々1 回しか、用いられていない場合には、その集合に属すトークンの区別を完全に取り除く。すなわち、その集合に属すトークンすべてを代表する色のトークンを新たに 1 つ付け加え、元のトークンすべてをそれに置き換える。

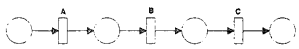

図 5: シーケンシャルに発火するトランジション群

例えば、図 5のように 2 つのトランジション A,B が分岐やその他の同期指定なども無く定義されている場合、A が発火すれば必ずその後 B が発火可能であり、また、B が発火する前には必ず A が発火している必要がある。よって、可達性判定を行う場合にはこれら 2 つの発火を別々に計算する必要は無く一つにまとめて計算を行っても等価な結果が得られる。さらに、同様に図 5のトランジション C もまとめることができる。このようにシーケンシャルな動作に関してはそれらを一つのトランジションに置き換えることができ、それによって、可達性判定の高速化が望める。

次に、制約ネットに具体的なトークンの名前が記載されていない色のトークンに関しては対称性を持つとして扱うことができる。この条件を満たすトークンを互いに置き換えたとする。可達グラフの全てのアーク (すなわち、トランジションの発火) に関して、その発火条件には具体的なトークンの名前は記載されていないため、そのようなトークン同士が入れ替わったことによって発火可能性が変化することは無い。すなわち、トークンの入れ替えを行う前と全く同じ (トークンが入れ替わっただけの) 発火可能性が常に満たされるため、可達グラフはトークンの入れ替わりを除いて全く同じ形になる。また、入れ替わっても発火可能性に変化が無いということから、トークンの入れ替わりのある 2 つの状態は可達性判定において等価と見なせる。例えば、ネットワーク会議のようなマルチメディアプレゼンテーションシステムの仕様においては、しばしば「聴衆」のようにある特定のユーザーに関する動作が一切定義されない仕様が用いられる。よって、こういった対称性を OS グラフを作成する際に利用することで、提案モデルの可達性判定を高速に行うことができる。

さらに、具体的な名前が記載されていない上に、トークンの色による発火条件判定などが一切無い場合には、それらのトークンは全く同じと見なすことができる。すなわち、これらのトークンは実行時の発火可能性判定において一切その具体的な名前が用いられることはなく、システムの動作はこれらのトークンの有無の情報しか扱わない。よって、そういった場合には、トークンの色を

完全に無視しトークンの数だけを扱うことで可達性判定が行える。また、各制約ネットで高々1回しか色に関する条件が用いられていない場合には、そこで使用されたトークンの色に関する情報が他の場所では用いられないことを意味するので、その場合にも色を無視して可達性判定を行うことができる。また、この方法は対称性の計算を可達グラフ作成中に行う代わりに仕様の段階で縮退させることを意味するため、この方法を用いてトークンの色の区別を無くした状態で作成したOSグラフは色の区別を残したまま対称性を用いて作成したOSグラフと同じ形になる。

3.2.2 検証系

提案手法における検証を行うために、動作ネット群、制約ネット群とその間の同期指定を自動的に単一のカラーペトリネットに変換する変換系を作成した。この変換系は、与えられた動作ネット群と制約ネット群を3.1節の手順に基づき、同期指定のされたトランジションを重ねることにより単一のカラーペトリネットへと変換する。また、同時に前述の手順で、ペトリネットの簡約化、対称性を持つトークンの自動抽出と、区別の必要のないトークンの色の削除を行い、検証を行う準備をする。そうして作成したカラーペトリネットおよび対称性を一般のカラーペトリネット検証系にかけることでデッドロック判定を行う。

3.3 検証結果

図2であげた例題に対して提案手法の実験をフリーウェアのカラーペトリネットのデザイン・シミュレーションツールである Design/CPN[6] を用いて行った。例題に対して聴衆の人数と質問の回数を変化させ、デッドロックの有無を判定した。その結果を表1に示す。この表はそれぞれ、対称性を利用せず Occurrence Graph 作成した場合、対称性を利用して OS Graph を作成した場合、および、対称性を利用し、さらに色情報を無視することのできるトークンの色情報を削除した場合についての得られた可達グラフのサイズと計算時間を示している。対称性を用いて可達グラフを作成した場合、可達グラフのサイズは非常に小さくなるが、対称性の判定の計算量の増加のためこの処理系においては単純に計算時間の低減には繋がらなかった。そこでさらに色情報の削減を行った結果、問題規模が大きくなった場合の計算時間を大幅に短縮できた。

4 プロトコル合成

2章で述べたシナリオ群では複数のペトリネットが自由に同期を行うことが仮定されている。一方、実際のネットワーク環境ではこれらを非同期通信で実現する必要がある。そこで、同期を含む複数の有限カラーペトリネットで記述された仕様 (サービス仕様と呼ぶ) からプロトコル合成を行い、実装レベルのプログラム (プロトコル仕様と呼ぶ) を自動導出する。プロトコル仕様に関してもメッセージの送受信を行えるペトリネットとして記述する。

4.1 制約ネットの実装

提案モデルにおいては、各参加者の仕様はそれぞれ独立して動作を行うペトリネットとして表現されている。

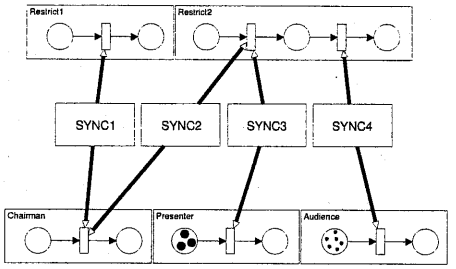

図 6: サービス仕様のイメージ

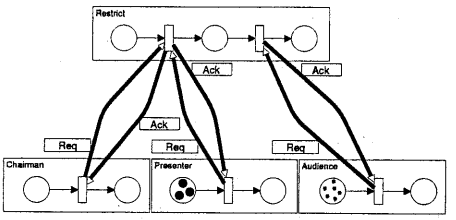

図 7: サービス仕様の同期指定をメッセージ交換を用いて実現

よって、各ノードの動作仕様としては動作ネットをそのまま用いる。また、制約と同期指定によって定義される全体の実行制御を実現するためそれらのノードの間の相互動作を制御する制約ネットを別に導出する。与えられた制約ネットを同期指定に基づいて単一のペトリネットへと結合し、与えられた同期指定を満たすよう動作ネットと制約ネットの間のメッセージ交換を追加する。同期を行いたいノードが制約ネットの仕様に対して同期依頼メッセージ (Req) を送信し、制約ネットはそれらのメッセージから同期の判定を行い、その結果を同期許可メッセージ (Ack) として返信するようメッセージ交換を追加する。

例えば、図6のようなサービス仕様が与えられた場合、制約ネットをすべて1つにまとめ、同期制御メッセージの追加を行い、図7のような仕様が得られる。このようにして与えられたサービス仕様と等価に動作するペトリネット群を導出することができる。

4.2 分散制御方式による同期の実現

得られた制約ネットをネットワーク上の単一の制御ノードの仕様として用いる場合、同期の際には必ず同期に参加するすべてのノードからの同期依頼とそれらのノードへの同期許可メッセージが交換される。この実現はメッセージ交換の数が多く、効率的ではない。そこで、得られた制約ネットの仕様を複数のノードに分散配置をし、元の仕様を満たす範囲でできる限りローカルに同期判定を行う方法を提案する。

まず、制約ネットを複数のノードに分散配置を行うために分割する。その際に、競合する同期の判定を複数ノードで別々に行うとそれらのノード間で判定結果に不整合

	Occurrence Graph			OS Graph			OS Graph(色削減)
聴衆	ノード数	アーク数	計算時間	ノード数	アーク数	計算時間	計算時間
3人	28	61	1秒	11	14	1秒	1秒
4人	66	177	1秒	11	14	1秒	1秒
5人	132	451	1秒	11	14	1秒	1秒
6人	234	1333	7秒	11	14	3秒	1秒
7人	380	6063	212秒	11	14	586秒	1秒

表 1: デッドロック判定結果

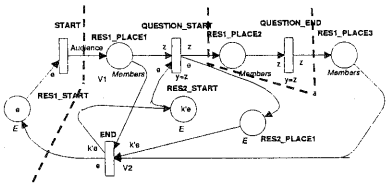

図 8: 発火可能性判定の関連による分割の例

が生じる可能性がある。よって、競合する同期判定群に関してはそれぞれ1つのノードで行う必要がある。そのために、同期が競合する範囲に基づいて制約ネットを分割する。制約ネットが自由選択ネットと呼ばれるクラスに属する場合には、この分割は単純に同じプレースからの入力トークンを持つトランジションごとに分割するだけでよい。制約ネットが複雑な構造のペトリネットである場合には一切分割できない場合もある。

そして、各サブペトリネットに対し、そこに含まれる全ての同期判定を行う責任ノードを定める。責任ノードは必要なメッセージ交換が最小となるように割り当てる。責任ノードが変わるトランジションにはメッセージ交換を追加して、責任ノードの切り替えとトークンの移動を実装する。

以上のようにして制約ネットの仕様を分散制御することができる。

図8に制約ネットの分割の例をあげる。この例は図2の2つの制約ネットを同期指定に基づいて結合し、同期の競合するトランジションを元に分割を行ったものである。この例では3つのサブペトリネットに分割される。

5 まとめと今後の課題

制約指向を用いた対称性を持つ分散アプリケーションの設計法を提案した。提案手法では各ノードの動作をそれぞれ独立したプログラムとしてあるクラスのカラーペトリネットで表現し、それらのノード間の関係を制約という形でトランジションの同期を用いて表現する。

また、この手法で設計を行った分散アプリケーションの可達性判定の手法を提案した。この手法は、与えられた仕様から自動的に対称性を検出し、OS Graphを作成することで高速に可達性判定を行う。いくつかの例題に対してこの方法を適用した。作成した例題については、可達グラフのノード数を大幅に減らすことができ、検証に要する計算時間も短縮できた。

さらに、この手法で設計を行った分散アプリケーションを実際のネットワーク環境で実行するためのプロトコル合成法の提案を行った。このプロトコル合成法では設計段階における同期制御をメッセージ交換に置き換えることで実環境における動作仕様を導出する。メッセージ交換に関しては全ての同期制御を一つのノードで処理する集中制御型の動作仕様に変換した後、その仕様をいくつかのサブネットに分割してそれぞれ別々のノードで同期制御を行う分散制御の方法に変換することによりノード間で交換されるメッセージ数を減らす工夫をしている。なお、本研究に関する詳細は文献[7]にある。

今後の課題は、本研究で考案したモデルを時間制約を記述できるように拡張し、そのモデル上での可達性判定を行う方法を考案することなどが考えられる。

参考文献

[1] ISO : "Information Processing System, Open Systems Interconnection, LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807 (1989).

[2] Bolognesi, T. : Toward Constraint-Object Oriented Development, *IEEE Trans. on Software Eng.*, Vol. 26, No. 7, pp. 594 – 616 (2000).

[3] Vissers, C. A., Scollo, G. and Sinderen, M. v. : "Architecture and Specification Style in Formal Descriptions of Distributed Systems", *Proc. 8th Int. Symp. on Protocol Specification, Testing, and Verification (PSTV-VIII)*, pp. 189–204 (1988).

[4] K. Jensen. : "Coloured Petri Nets", *EATCS Monographs in Theoretical Computer Science* Vol. 2. Springer-Verlag, (1997).

[5] K. Saleh : "Synthesis of Communication Protocols: an Annotated Bibliography", *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 5, pp.40–59 (1996).

[6] CPN group at the University of Aarhus, Denmark : "Design/CPN", Ver. 4.0.4, *http://www.daimi.aau.dk/designCPN/*

[7] 梅津高朗 : "制約指向モデルで記述された対称性を持つ並行システムの形式的検証とプロトコル合成", 大阪大学大学院基礎工学研究科情報数理系専攻修士学位論文, (2001).