マルチメディア通信と分散処理 110－1
（2002．11．21）

カラーペトリネットを用いた分散協調システムの設計とSOAPを用いた一実装法の提案

北野 智広 梅津 高朗 山口 弘純 東野 輝夫

本研究では，グループワークなどのネットワークを介した分散協調システム向けの，カラーペトリネットを用いた仕様記述モデル，ならびに，そのモデルで記述されたシステムのサービス仕様からSOAPを用いた実装を自動的に導出する手法を提案する．提案モデルにおいては，複数のサーバおよび各端末（ノード）からなるシステム全体が提供するサービスの仕様（サービス仕様）が一つのカラーペトリネットとして記述されるとする．与えられたサービス仕様からプロトコル合成を行うことで，ノードの動作仕様群（プロトコル仕様）が自動的に導出される．各ノードの動作仕様はそれぞれJavaのプログラムとして得られ，ノード間の通信はSOAPを利用して実装される．自動導出系を試作し，協調作業管理システムの仕様記述例に適用することで，提案手法による有効性を評価した．

Colored Petri-net Based Model for Designing Distributed Cooperative Systems and Its Implementation Using SOAP

Tomohiro Kitano Takaaki Umedu
Hirozumi Yamaguchi Teruo Higashino

In this paper, we introduce a colored Petri-net based model for designing distributed cooperative systems (concurrent systems) and a method for deriving its implementation. In this model, a specification of a distributed cooperative system is described as a centrarized program in a colored Petri-net (service specification). Then a set of behavior specifications of all the cooperative nodes (protocol specification) is derived from the given service specification automatically. The derived protocol specification is guaranteed to be equivalent to the service specification. In our derication method, the behaviour specifications of those nodes are described in Java, where communication among the nodes is implemented based on SOAP. We have developed an automatic derivation tool and applied our method to some example system specifications by using this tool in order to evaluate the efficiency of our method.

1 はじめに

近年の携帯端末の発達により，インターネットアクセス機能を持つ端末が一般的になってきた．そのような多機能端末の普及により，移動体端末による多人数参加型の分散協調システムなど新しいカテゴリに属する分散アプリケーションの重要性が増している．

従来，信頼性の高い分散システムを簡潔に設計するための手法としてプロトコル合成と呼ばれる方法が研究されている[1]．プロトコル合成法では，設計する分散システムの仕様を，単一の計算機で実行させるような集中型のプログラム（サービス仕様）として記述し，ノード間の通信は記述しない．次に，サービス仕様とは独立にサーバやクライアント用端末など，実際に与えられたプログラムを動作させる環境に関する情報（動作環境情報）を与える．それらからサービス仕様と等価な動作を行うようにそれぞれのノードで実行されるプログラム群（プロトコル仕様）を自動導出する．プロトコル仕様ではサービス仕様と等価な動作をさせるために必要なノード間の同期やデータ交換のための通信動作が追加される．我々の研究グループにおいては負荷分散を考慮したプロトコル合成を行うモバイルアプリケーションの実装法[2]の研究などを行ってきている．

本稿では，多人数参加型の分散協調システムのサービス仕様を記述するためのカラーペトリネットに基づく記述モデルを提案する．提案するモデルでは，アプリケーションで利用するデータ構造はJavaで記述し，アプリケーション全体の制御構造はカラーペトリネットを用いて記述する．Javaは，携帯端末やWebサーバなど多くの端末において実行可能であるため，プロトコル仕様はそれぞれのノードで動作させるべきJavaプログラムの形で導出する（図1）．

本稿において，汎用性とプロトコルの単純さを考慮し，プロトコル仕様におけるノード間の通信にはSOAP[3]を利用する．SOAPはそれに基づく企業間通信商取引支援に関する研究[4]なども行われており，今後の普及が見込まれているプロトコルである．ここでは，SOAPは以下のような特徴を有するため，提案手法において採用した．

- 下位プロトコルとしてHTTPを用いることが一般的であるため，多くのプラットフォームで利用が可能であり，通信時に暗号化を行いたい場合にはHTTPSを用いることも可能
- 異種プラットフォーム間での通信が容易
- XMLでメッセージの交換を行うため，近年データの標準となってきているXMLデータをそのまま送受信可能で

大阪大学 大学院情報科学研究科 豊中市待兼山町
Graduate School of Information Science and Technology, Osaka University, Toyonaka, Osaka 560-8531, Japan

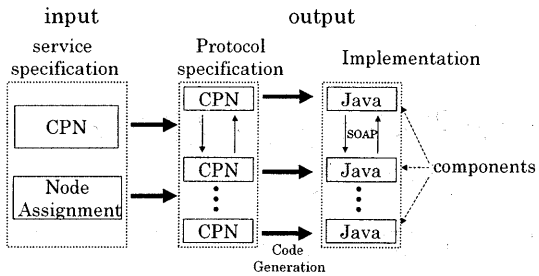

図 1: 提案手法の概要

あり，XML データに電子署名や検証機能の付加，暗号化機能を付与することも容易

- 今後システム間通信の際の標準プロトコルになる可能性が高い
- 既存の SOAP ベースのサービスとの間の整合性が高い

また，一般に，分散アプリケーションで用いられる通信プロトコルとして CORBA や Java RMI がよく研究されている [5, 6]．しかし，CORBA では提供するサービスをあらかじめ IDL により記述する必要があり，Java RMI でも同様にサーバの提供するリモートメソッドはあらかじめ定義されていなければならない．SOAP ではそういった手順を踏まず通信コンポーネントを導出できるため，導出系を単純に実装できるという利点も存在する．

我々は本稿において，多数のユーザが参加する分散協調システムのサービス仕様と分散環境の指定からプロトコル仕様を合成するシステムモデルと実装法の一つを提案し，その実装例を示す．

以下，2 章で仕様記述モデルの提案を行い，3 章で提案モデルを用いた仕様の解析，4 章で提案する実装方法について述べる．

2 分散協調サービス仕様の記述モデル

2.1 カラーペトリネットに基づく記述モデル

提案モデルにおいては，サービス仕様は，Java のクラスのインスタンスをトークンとするカラーペトリネットとして記述する．

ペトリネットはプレースとトランジションの 2 種類のノードからなる重み付き有向 2 部グラフである．各重み付き有向辺はアークとよばれ，それぞれには重み ($W(a)$ で表す) が付与される．各プレース p には複数のトークンを配置でき，この配置をマーキングとよぶ．マーキング m におけるプレース p のトークン数は $m(p)$ で表す．マーキングはペトリネットの状態を表す．トランジション t はその各入力プレース p に，p から t へのアーク a の重み $W(a)$ 以上の個数のトークンが存在する場合のみ発火可能であり，t が発火すると各プレース p から $W(a)$ 個のトークンが取り除かれ，同時に，t の各出力プレース p' に，t から p' へのアーク a' の重み $W(a')$ の個数のトークンが追加される．

提案モデルで用いるカラーペトリネット (Coloured Petri Net，以下 CPN)[7] は，高階ペトリネット (High Level Petri Net)[8] の一つであり，各トークンは色と呼ばれる型 (整数型や実数型，文字列型など) を持ち，その色に属する値を保持する．また，アーク a の重み $W(a)$ は，トークンとバインディング変数の多重集合として定義される．バインディング変数とはそれと同じ色のトークンを代入できる変数であり，多重集合は同一要素を複数個含むことのできる集合である．また，

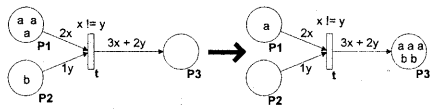

図 2: CPN におけるトランジションの発火

マーキング m によるプレース p へのトークンの配置 $m(p)$ はトークンの多重集合である．以下では，要素 $e_k(k = 1..n)$ が w_k 個含まれる多重集合を $\sum_k w_k e_k$ で表す．さらに各トランジション t には，t の入力アークの重みに利用されるバインディング変数を用いた論理式 (ガード式) を定義する．

トランジション t は (i)t とアーク a で接続されている各入力プレース p に $W(a)$ の各バインディング変数に代入できるトークンが存在し，(ii) その代入が t のガード式を満たす場合にのみ発火できる．その結果，t の入力プレース p から $W(a)$ に代入されたトークンが取り去られ，t とアーク a' で接続されている各出力プレース p' に $W(a')$ の変数に代入されたトークンが追加される．発火の際には各バインディング変数への代入が決定される必要があるため，$W(a')$ で用いられているバインディング変数は t のいずれかの入力アークの重みで用いられている変数群の一部である必要がある．以下では，各 $W(a)$ のバインディング変数へのトークンの代入をバインディングとよぶ．

図 2に発火の例を示す．a, b をある色の定数，x, y をそれぞれ a, b と同色のバインディング変数とする．バインディング $[x = a, y = b]$ は t のガード式 "$x! = y$" $(x \neq y)$ を満足するため，t はこのバインディングで発火できる．その結果，$P1$ および $P2$ から，それぞれトークンの多重集合 $2a$ および b が取り除かれ，トークンの多重集合 $3a + 2b$ が $P3$ に追加される．

システムの仕様をカラーペトリネットで記述することで，同じ動作をする多数の端末を含むシステムの記述が容易になり，また，それらの間の協調動作なども簡潔に記述できる．

まず，アプリケーション上で取り扱うデータを格納するためのデータ構造は Java のクラスとして設計し，アプリケーションの動作の流れを表す制御構造をカラーペトリネットを用いて設計する．データを格納するためのデータベースや処理待ちのデータを格納する待ち行列などはカラーペトリネットにおけるプレースとして定義し，データの処理やデータ間の同期などはトランジションを用いて記述する．以下に提案記述モデルを利用したアプリケーションの具体的な記述例を示す．

2.2 サービス仕様の記述例

ここでは，分散タスクマネージメントシステムのサービス仕様を，提案モデルを用いて記述した例を示す．このシステムは，複数のクライアントと複数の作業者の間で仕事の割り当てを行うシステムである．まず，クライアントが仕事を提示し，作業者は提示された仕事のうち，自らの条件に見合う仕事を選択してクライアントとの間で交渉を行う．交渉の結果，仕事が割り当てられた場合には，その内容に応じた作業が行われ，クライアントから作業者に報酬が支払われる．

システムで利用するクラスは Java で定義している．ここでは，仕事を表現する Task クラス (図 3) と，クライアント，作業者を表現する Person クラス (図 4) を定義した．

次に，これらのクラスを用いて，カラーペトリネットによりアプリケーション全体の制御を記述する図 5に，この分散タスクマネージメントシステムのサービス仕様の例を示す．このシステムはプレース *Client* に配置されたトークン (クラス *Person* のインスタンス) によって表現されるクライアントがプレース *Tasks* に列挙された仕事 (クラス *Task* のインスタン

```
class Task{
    //協調作業を表す定数
    public static final int CO = 0;
    //非協調作業を表す定数
    public static final int NC = 1;
    //この仕事が協調作業か非協調作業か
    int tasktype;
    //仕事の名前
    String name;
    //仕事のコスト (報酬)
    int cost;
    //仕事の初期化
    public Task(int t,String n,int c){
        taskType = t;
        name = iName;
        cost = iCost;
    }
    //仕事のコストを得る
    public int GetCost(){
        return cost;
    }
    //この仕事が協調作業かどうか
    public boolean IsCooperativeWork(){
        return tasktype == CO;
    }
    //この仕事が非協調作業かどうか
    public boolean IsNotCooperativeWork(){
        return tasktype == NC;
    }
}
```

図 3: Task クラス

ス) の内，依頼したい仕事を選択し，それを提示する (トランジション *t1*)．クライアントとそのクライアントが提示した仕事の組はプレース *p1* にトークンとして置かれ，その仕事を処理したい作業者が現れるまで待機する．作業者を表すトークン (クラス *Person* のインスタンス) はプレース *Workers* に配置され，自分の引き受けたい仕事を *p1* に提示されたトークンから一つ選択する．クライアントと作業者の間で，条件の折り合いがついた場合には，トランジション *t2*, *t6* により作業者に仕事が依頼される．クライアントと作業者が協力して処理すべき仕事の場合はクライアントは作業者と共に処理を行い (トランジション *t6*, *t7*)，処理が終了した後にクライアントは作業者に対して支払いを行う (トランジション *t8*)．作業者のみで完了できる仕事の場合は，作業者に仕事が割り当てられた時点でクライアントは報酬を先払いし，待機状態に戻る (トランジション *t2*, *t3*)．先払いされた報酬はシステムにいったん保持され，作業者が仕事を終了した時点で作業者に支払われる (トランジション *t4*, *t5*)．この場合はクライアントは仕事の終了を待たずに次の仕事のリクエストを発行できる．

記述モデルとしてカラーペトリネットを用いることでユーザ数等の増減にも柔軟に対応できる．例えば，この記述例においてクライアントや作業者の人数を変更したい場合にはプレース *Clients*, *Workers* に配置されたトークン数を変更するだけでよい．

3 サービス仕様の動作解析

提案モデルはカラーペトリネットに基づいているため，カラーペトリネットの解析ツールを用いることでデッドロックの判定や，各プレースのトークン数の上限などを解析できる．

```
class Person{
    //人物名
    String name;
    //所持金
    int money;
    //人物の初期化
    public Person(String n,int m){
        name = n;
        money = m;
    }
    //仕事 t に対する報酬を支払う
    public void Pay(Task t){
        money = money - t.GetCost();
    }
    //仕事 t に対する報酬を受け取る
    public void Receive(Task t){
        money = money + t.GetCost();
    }
    //仕事 t を依頼するのに十分な所持金が有るかどうか
    public boolean EnoughMoney(Task t){
        return money >= t.GetCost();
    }
}
```

図 4: Person クラス

解析により，特定のプレースにトークンが集中しすぎるといった問題を事前に調査できるため，そういった場合には，実装後のサーバに対する過負荷による性能低下なども考えられるため，文献 [2] の手法などにより負荷分散を行うことも考えられる．

ここでは一般的なカラーペトリネット解析ツールを用いた解析を行う方法について述べる．解析ツールとしては Design/CPN[9] を用いた．このツールでは，データ構造は CPN ML と呼ばれる言語で記述するため，検証を行うためにはデータ構造が CPN ML により正しく表現できる必要がある．ここでは，CPN ML により表現可能なデータ構造を表現するため，Java のクラス定義に対して以下の条件を設けた．

(1) 各クラスは以下の 4 つの要素のみで構成される
  (a) 定数宣言
  (b) 保持するデータを格納するメンバ変数
  (c) 保持するデータを修正するメソッド
  (d) 保持するデータや状態を返すメソッド
(2) int や String 型といったプリミティブな型のメンバのみを利用する
(3) 他のクラスのインスタンスへの参照を保持せず，全てのメソッドの動作はインスタンスが保持するデータと，与えられた引数の値にのみ依存する (インスタンス間のインタラクションは全てカラーペトリネットの上に記述された動作式中で行われる)
(4) 全てのメソッド呼び出しは一定時間内に完了する (メソッド内にループを持たない)

データを抽象化したクラスがこれらの条件を満たす場合，そのインスタンスはそのままカラーペトリネットにおけるトークンとして扱うことができるため，文献 [10] において提案した手法などに基づき，カラーペトリネット用の検証ツールを用いて到達可能性検証などを行うことができる．(1)〜(4) の条件を満たさないクラスを用いても本提案モデルを用いて分散協調アプリケーションを実装することは可能であるが，その場合にはカラーペトリネット上での検証は利用できない．また，これらの検証を簡単に行うために，上記の条件を満たす Java で記述されたクラスから CPN ML へと変換する簡単な

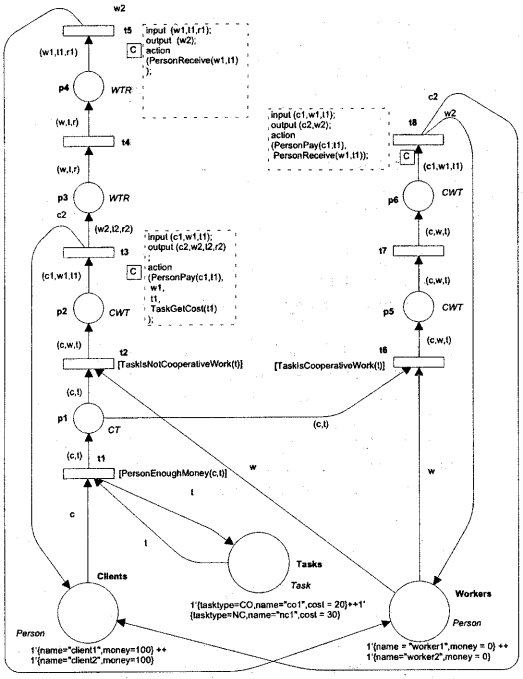

図 5: 制御仕様例

変換系を用意した.

図 5の例題に対してこの手法を適用し，設計意図に従った動作が行われることを確認した．すなわち，全てのクライアントの所持金が不足となった場合にはそれ以上の仕事を発注できないためシステムはデッドロック状態に陥るが，それ以外にはデッドロック状態が存在しないことを確認できた.

4 サービス仕様から Java プログラムの導出

本稿で提案する記述モデルにおいては，システム全体は 1 つのカラーペトリネット (サービス仕様) として記述され，複数のノードによる分散協調アプリケーション (Java) として実装される．まず，サービス仕様に含まれる各プレースに対して，そのプレースに配置されたトークンの示すデータを保持するノードを決定し，各トランジションに対してそのトランジションの発火可能性の判定および発火時のデータ処理を行うノードを決定する．但し，データベースなど恒久的 (永続的) なデータ格納場所を表すプレースや，ユーザに対する入出力処理などを行うトランジションに関しては，それぞれデータベースサーバや，ユーザのノードなどといった，それらの機能を提供できる特定のノード上に実装される必要がある．そのため，そういった一部のプレース，トランジションは実装されるノードを動作環境情報として明示しておく．提案手法に与える入力は以下のようになる.

(1) Java で記述されたデータ構造
(2) カラーペトリネットで記述された制御構造
(3) 特定の機能を持つプレース，トランジションのノードへの配置

ここで，配置するノードが指定されなかったプレース，トランジションはタスクの負荷分散などを考慮し，導出アルゴリズムにより自動的に適切なノードに割り当てられ，システム全体が分散環境上で実装される．その際，トランジションの発火はそのトランジションの入力プレースのデータを利用した動作と見なせるため，各ノードは自身のプレース上のデータのみからトランジションの発火可能性判定及び発火を行うことができるように，各トランジションはその入力プレースと同じノードに配置する．ただし，複数プレースからのトークンを複数のトランジションが奪い合う複雑な競合構造については，それを満足する配置が存在しない場合もある．この存在判定方法及び配置の決定のアルゴリズムは次節で述べる.

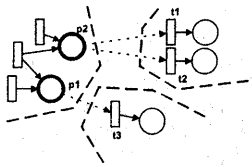

図 6: 通信モデル

全てのプレース，トランジションが，各ノードに対して配置された後，それに従ってカラーペトリネットの分割が行われ，プロトコル仕様が導出される．プロトコル仕様においては，サービス仕様には無かったノード間の通信が必要となるため，以下のような通信モデルを定義する.

ここではノード間の通信をモデル化するため，送信プレースと受信トランジションを定義する．各ノードは，送信プレースを通して他ノードの受信トランジションを発火させることができるものとし，これを用いてメッセージの送信をモデル化する．送信プレースのトークンの型は相手先ノードで実行される受信トランジション名とその引数 (トークン値) の並びとする．送信プレースに置かれたトークンは相手側の指定されたトランジションをそのトークンを用いて発火させる．図 6に例を示す．図は破線で示された仕様による 3 つのノード間のメッセージ通信を示したもので，太線で示されたプレース $p1$，$p2$ が送信プレースであり，破線矢印の通信が定義されている．ただし，送信プレースによって発火される受信トランジション $t1$，$t2$，$t3$ は，必ず単一の送信プレースからのみ発火される必要がある.

4.1 導出アルゴリズム

次に，プロトコル仕様導出アルゴリズムについて述べる．サービス仕様は一つのプログラムとして与えられるため，分散環境上で実装するためには，サービス仕様のトランジションの一部をこれらの送信プレースと受信トランジションに置き換えることで仕様を分割し，各ノードの動作仕様 (プロトコル仕様) を得る.

ただし，その際にノードに対するプレースやトランジションの配置の割り当て方によっては，トランジションの発火可能性の判定が，異なるノード上に配置された複数のプレースのトークンに依存し，それらのトークンを利用して発火可能性を判定する競合トランジションとの間での排他制御が必要となる．提案手法ではそのような排他制御を行う必要のない配置の仕方が存在する場合はそれを求めることができる.

具体的には，カラーペトリネットを構成する全てのプレースおよびトランジションの集合を，各ノードに対して自由に割り当てが出来るサブペトリネット群に分割する．すなわち，異なるサブペトリネットに含まれるプレースおよびトランジションの間では互いに発火可能性判定において競合しないよう分割を行う.

このようにして得られた，各サブペトリネットは任意のノー

ドに配置可能である．ここで，トランジション t，プレース p に対して，$\bullet t$ で t の入力プレースの集合，$p\bullet$ で p の出力トランジションの集合を表すとする．ペトリネットに含まれるすべてのプレースの集合 P およびすべてのトランジションの集合 T を，同じノードに配置すべき範囲によって分割する．即ち，以下を満たす $P_1,...,P_k$，$T_1,...,T_k$ の k 個の部分集合を求める．

$$\bigcup_{1\leq i\leq k} P_i = P$$

$$\bigcup_{1\leq i\leq k} T_i = T$$

$$\forall i,j : i \neq j \rightarrow P_i \cap P_j = \emptyset$$

$$\forall i,j : i \neq j \rightarrow T_i \cap T_j = \emptyset$$

$$1 \leq i \leq k : \bigcup_{p\in P_i} p\bullet = Ti$$

$$1 \leq i \leq k : \bigcup_{t\in T_i} \bullet t = Pi$$

これらの条件式は T_i に含まれるすべてのトランジションの発火条件の判定は P_i に含まれるプレースに存在するトークンのみで行え，また，P_i に含まれるプレースに存在するトークンを必要とするトランジションはすべて T_i に含まれている，という条件を表している．よって，この各分割に含まれるプレースとトランジションは同じノードに配置し，それ以外のトランジションとは独立して発火可能性判定を行えばよい．両端が異なるノードに配置されるアークを持つトランジションは送信プレースと受信トランジションとすることで，ノード間でのトークンの移動を行う．

極めて複雑なトランジション発火が定義されているペトリネットの場合には $k = 1$ すなわち，一切，分割がされない可能性がある．そのため，複雑なクラスのペトリネットを与えた場合，このアルゴリズムを用いて任意のノード割り当てに対してプロトコル仕様が必ず導出されるとは限らない．この問題に対する解決法は今後の課題である．

以下にこの分割を求めるアルゴリズムについて説明する．

(i) まず，すべてのトランジションの集合 $T = \{t_i | 1 \leq i \leq |T|\}$ に対して，$k = |T|, T_i = \{t_i\}, P_i = \emptyset (1 \leq i \leq k)$ を初期値として，以下の (ii)〜(iv) を集合に変化が現れなくなるまで繰り返す．

(ii) すべての $t \in T_i$ に対して $\bullet t$ を P_i に加える．

(iii) すべての $p \in P_i$ に対して $p\bullet$ を T_i に加える．

(iv) 同じ要素を含む分割はまとめる．すなわち，ある 2 つの分割が，$P_i \cap P_j \neq \emptyset$ もしくは，$T_i \cap T_j \neq \emptyset$ である場合には，P_i を $P_i \cup P_j$ に置き換え，T_i を $T_i \cup T_j$ で置き換え，P_j と T_j を削除して，k の値を更新する．

以上のようなアルゴリズムで導出されたサブペトリネットを任意のノードに割り当てることによって，サービス仕様からプロトコル仕様を導出することができる．

4.2 サービス仕様分割によるプロトコル仕様導出例

上記のアルゴリズムを図 5に対し適用した結果を図 7に示す．まず，プレース Client と Tasks は共に T1 を出力トランジションとして持つため，この 2 つのプレースは上記のアルゴリズムに従い，同一のサブペトリネットに加えられる．また，同様にトランジション T2，T6 は共にプレース P1，Workers を入力プレースとして持つため，同一のサブペトリネットにまとめられ，さらにその結果，P1，Workers も同一の出力トランジションを持つため，これらも一つのサブペトリネットに集められる．このように図 5のサービス仕様は前節の条件を満足する 7 個のサブペトリネット群に分割できる．従って，

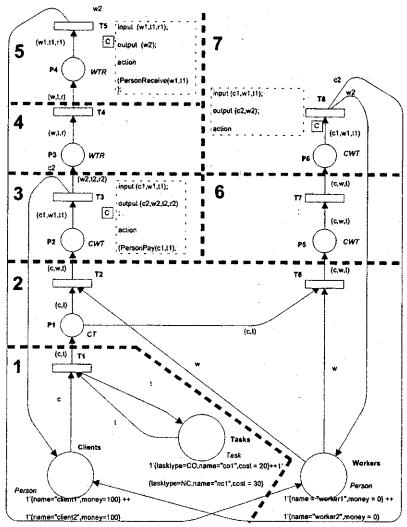

図 7: 発火可能性判定の関連による分割の例

これらのサブペトリネット群の任意のノードに対する割り当ては発火可能性判定の競合に関する問題を引き起こさない．

この例においてはノードを 3 つとする．各ノードの役割はそれぞれ，クライアントの入力を受け付けるノード A，作業者の入力を受け付けるノード B，作業の実行を制御するノード C とする．それぞれの役割に従って，ノード A には図 7のサブペトリネット 1 を割り当て，ノード B にはサブペトリネット 2，3，6 を，残りのサブペトリネット 4，5，7 をノード C に割り当てる．結果として得られる各ノードの動作仕様は図 8のようになる．各ノードの役割を以下に示す．

ノード A はクライアントがタスクからの要求を受け付け，作業者を探す準備段階までが行われていることを示している．ノード B にはクライアントからの要求と作業者との折り合いがついた際に仕事の依頼が終了し，協調作業が必要な場合と非協調作業での場合との動作が記述されている．また，非協調作業の場合はクライアントの解放を行う動作も含む．ノード C は作業者が実際にタスクの処理を行う部分であり，協調作業を行う場合と非協調作業を行う場合で 2 通りの遷移が存在している．

4.3 SOAP を用いた動作仕様の実装

得られた各ノードの動作仕様 (カラーペトリネット) を Java のプログラムに変換する．この変換は，与えられたカラーペトリネットの動作をエミュレートする Java プログラムを導出することで実現する．エミュレートはプレースに配置されたトークンをそれぞれプロセスとし，トランジションの発火をプロセス間の同期によって実現することで行う．

カラーペトリネットの分割の過程で追加された送信プレースおよび受信トランジションは，SOAP を用いて実装したオブジェクト転送メソッドにより実装する．転送メソッドにより，カラーペトリネットのトークン (Java のオブジェクト) は XML データにエンコードされ，HTTP によりノード間を転送される．

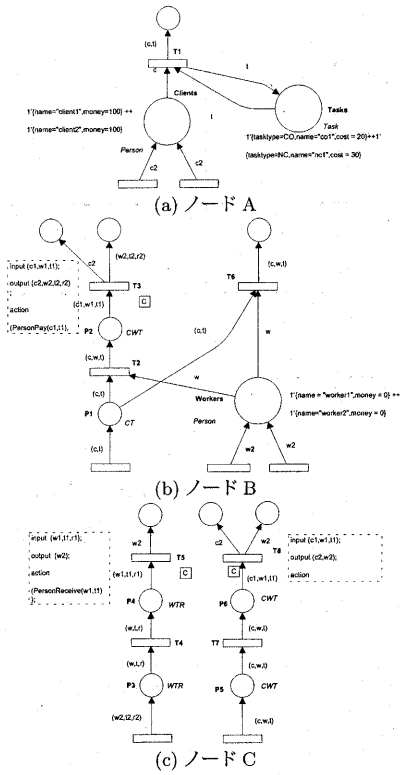

図 8: 分割後のペトリネット

5 まとめ

本稿ではカラーペトリネットを用いた分散協調アプリケーション設計のためのモデルを提案し，そのモデルからのプロトコル合成法に基づく実装法を提案した．提案手法により，カラーペトリネットを用いて単体のアプリケーションとして記述した仕様から簡単に分散協調アプリケーションとして動作する動作仕様群が得られる．実装は Java 言語を用い，通信プロトコルとしては，他のフレームワークとの親和性も高く，拡張性にも優れている，SOAP を用いた．

また，本研究グループにおいては，カラーペトリネットに基づく効率的なシステム設計モデルおよび，そのモデルで記述されたデッドロック可能性判定法や，負荷分散を考慮に入れた実装法の提案を行ってきている [2, 10]．本稿における提案モデルも同様のモデルに基づいているため，それらの提案手法を用いることで，より簡潔にデッドロックフリーなシステム設計を行うことができる．

今後の課題としては，カラーペトリネットの分散化の一般化を含め，より利用しやすい実装方法について検討することが挙げられる．本提案手法によるカラーペトリネットの分割方法では，発火可能性の判定の競合を単純にそれらを単一ノードに割り当てる事で行っているため，任意の仕様に対して自由にノードへの機能割り当てを行うことが出来ない．この問題に対応するためには，分散トランザクション処理の手法 [11] などを導入することが考えられる．また，本稿において提案した処理系においてはユーザーインターフェイスを含む入出力などに関しては全て設計者が Java を用いて記述する必要があり，記述によっては 3 章の手法による検証が利用できない可能性が存在する．こういった問題に対処するために，ユーザーインターフェイスなども含めたシステム全体の記述のためのフレームワークを提供することで利用可能性を高めることも検討している．

参考文献

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

[2] 梅津 高朗, 山口 弘純, 中田 明夫, 安本 慶一, 東野 輝夫 : “複数の移動端末を扱う分散協調システムの一設計法”, 情報処理学会モバイルコンピューティングとワイヤレス通信研究会報告, Vol. 2001, No. 83, pp.105-112 (2001).

[3] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, D. Winer : “Simple Object Access Protocol (SOAP)”, Ver. 1.1, *http://www.w3.org/TR/2000/NOTE-SOAP-20000508/*

[4] H. Kuno and M. Lemon : “A Lightweight Dynamic Conversation Controller for E-Services”, *Proc. of the Third Int. Workshop Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'01)*, pp. 90-99 (2001).

[5] L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, R. R. Koch, K. Berket : “Multicast Group Communication for CORBA”, *Proc. of Int. Symp. on Distributed Objects and Applications*, pp. 98-107 (1999).

[6] J. Rodrigues Nt., V. Ulm de G. Lima, G. Lima, M. Ferreira, J. Alves de Almeida, S. de Oliveira e Cruz, R.Cerqueira, and C. Martins : “A Command and Control Support System Using CORBA”, *Prof. of the 21st Int. Conf. on Distributed Computing Systems (ICDCS'01)*, pp. 735-738 (2001).

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

[8] K. Jensen and G. Rozenberg (eds.) : “High-level Petri Nets. Theory and Application”, Springer-Verlag, (1991).

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

[10] 梅津 高朗, 山口 弘純, 安本 慶一, 中田 明夫, 東野 輝夫 : “制約指向モデルで記述された対象性を持つ並行システムの形式的検証”, 情報処理学会論文誌, Vol.42, No 12, pp. 3054-3062 (2001).

[11] K-Y Lam, J. Cao, C-L Pang and S. H. Son : “Resolving Conflicts with Committing Transactions in Distributed Real-time Databases”, *Proc. of the Third Int. Conf. on Engineering of Conplex Computer Systems (ICECCS'97)*, pp. 49-58 (1997).